Impact of parallelism using multithreading on program execution time.

Project report version 1.0

For class CSE 3320

University of Texas at Arlington

Created by : Naveen Sokke Nagarajappa

Date : 9-Oct-19

Objective :

To create a program to generate images in Mandelbrot set, use multithreading in the program to create separate band of the image and to investigate how the number of threads affects the execution time of the program.

Theory :

Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer. A **multithreaded program** contains two or more parts that can run concurrently. Each such part of a program called thread.

The main purpose of multithreading is to provide simultaneous execution of two or more parts of a program to maximum utilize the CPU time and reduce program run time.

Mandelbrot set, which is a well known
fractal structure. The set is interesting both mathematically and aesthetically because it has an infinitely recursive structure. we can zoom in to any coordinates and find and find swirls, spirals, snowflakes, and other fun structures, as long as we are willing to do enough computation.

Description :

In order to study parallelism, we must have a problem that will take a significant amount of computation. For this purpose, we used c programs Mandel.c and Bitmap.c to generate image of Mandelbrot set and saves them as BMP files. If we run the program with no arguments,

it generates a default image and writes it to mandel.bmp, The program takes below arguments

"m" - The maximum number of iterations per point.
"x" - X coordinate of image center point.
"y" - Y coordinate of image center point.
"s" - Scale of the image in Mandelbrot coordinates.
"W" - Width of the image in pixels.
"H" - Height of the image in pixels.
"o" - Set output file name.

You can see all of these command line options with -h, and use them to override the defaults.
"h" - Show the help text.

This program uses the escape time algorithm. For each pixel in the image, it starts with the x and y position, and then computes a recurrence relation until it exceeds a fixed value or runs for max iterations.

Then, the pixel is assigned a color according to the number of iterations completed. In this program color scheme is to assign a gray value proportional to the number of iterations.

The max value controls the amount of work done by the algorithm. If we increase max, then we can see much more detail in the set, but it may take much longer to compute. For example, here is the same area in the set computed with four different values of max
It can take a long time to compute a Mandelbrot image. The larger the image, the closer it is to the origin, and the higher the max value, the longer it will take.

We have modified Mandel.c program to use an arbitrary number of threads to compute the image. Each thread should compute a completely separate band of the image. For example, if you specify three threads and the image is 500 pixels high, then thread 0 should work on lines 0-165, thread 1 should work on lines 166-331, and thread 2 should work on lines 332-499. Add a new command line argument -n to allow the user to specify the number of threads. If -n is not given, assume a default of one thread.
Program takes new argument. "n" - Number of threads created to compute the image.

We have used Pthreads for threads in our programing as it is simple to use, and readily available in GCC. And it is a kernel thread so OS will take care of scheduling and resource locks

Procedure :

- Compiled and built Mandel.c and Bitmap.c codes using command "gcc mandel.c bitmap.c -o mandel"
- Ran the code for configuration A. "time mandel -x -.5 -y .5 -s 1 -m 2000 -W 2048 -H 2048" with threads (i.e. -n) 1, 2, 3, 4, 5, 10, and 50.
- As the execution time of these experiments may be upset by other things going on in the machine, repeated each measurement ten times, and noted down the execution time to get the fastest time achieved.
- Ran the code for configuration B. "time mandel -x 0.2869325 -y 0.0142905 -s .000001 -W 2048 -H 2048 -m 1000" with threads (i.e. -n) 1, 2, 3, 4, 5, 10, and 50.
- As the execution time of these experiments may be upset by other things going on in the machine, repeated each measurement ten times, -x 0.2869325 -y 0.0142905 -s .000001 -W 2048 -H 2048 -m 1000 to get the fastest time achieved.
- Created execution time vs number of threads graph for fasted time achieved.

Results :

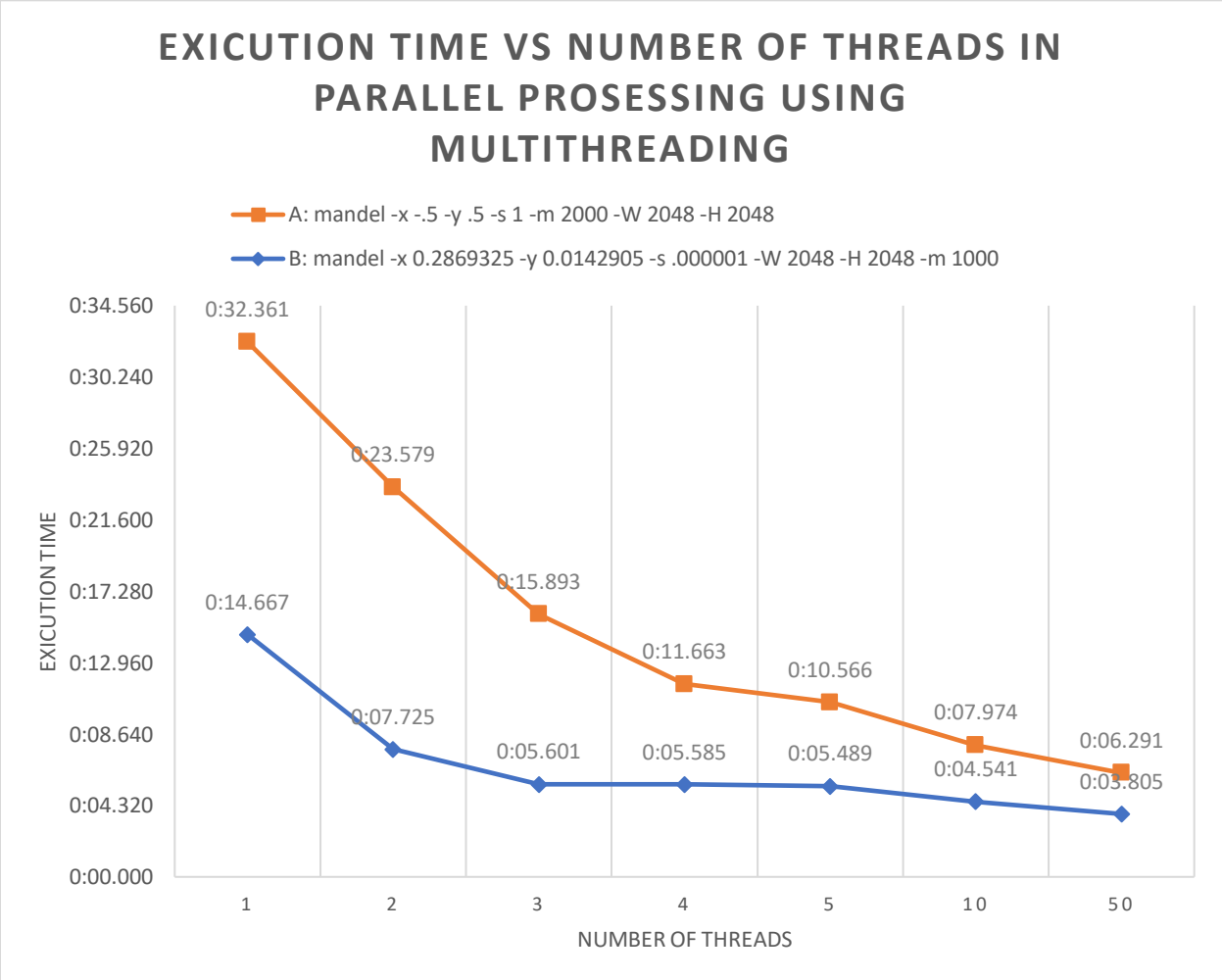
Configuration A : mandel -x -.5 -y .5 -s 1 -m 2000 -W 2048 -H 2048 with changing -n

Execution time for configuration A based on number of threads							
Number of Threads / Execution number	**1**	**2**	**3**	**4**	**5**	**10**	**50**
1	1:01.143	0:27.646	0:23.894	0:12.129	0:20.177	0:11.420	0:06.291
2	1:52.989	0:24.354	0:39.002	0:13.962	0:16.557	0:10.479	0:10.014
3	2:04.242	0:25.688	0:53.487	0:34.326	0:15.621	0:07.974	0:08.440
4	0:40.880	0:23.579	0:31.435	0:19.097	0:10.881	0:08.460	0:07.045
5	0:40.961	0:25.931	0:23.206	0:15.330	0:10.566	0:11.190	0:09.563
6	0:45.171	0:30.962	0:22.045	0:11.663	0:11.584	0:10.462	0:09.569
7	0:32.361	0:31.495	0:16.195	0:12.281	0:11.425	0:08.851	0:07.410
8	0:40.131	0:26.832	0:16.569	0:46.686	0:11.923	0:08.421	0:11.889
9	1:17.327	0:48.949	0:16.148	0:32.757	0:10.628	0:10.251	0:07.785
10	0:34.225	0:52.554	0:15.893	0:20.981	0:15.602	0:08.203	0:07.490

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

Execution time for configuration B based on number of threads							
Number of Threads / Execution number	**1**	**2**	**3**	**4**	**5**	**10**	**50**
1	0:23.203	0:10.107	0:06.716	0:05.585	0:13.103	0:06.708	0:03.805
2	0:14.806	0:41.116	0:17.847	0:21.952	0:07.129	0:05.104	0:12.171
3	0:15.510	0:09.048	0:07.805	0:10.382	0:08.832	0:23.640	0:12.146
4	0:21.395	0:10.995	0:07.778	0:06.922	0:09.641	0:05.547	0:12.684
5	0:20.529	0:14.451	0:10.337	0:17.348	0:07.931	0:06.551	0:09.900
6	0:18.232	0:07.877	0:05.601	0:14.666	0:11.011	0:04.889	0:04.599
7	0:22.666	0:07.725	0:06.820	0:18.691	0:10.680	0:10.011	0:06.253
8	0:39.905	0:08.387	0:05.897	0:11.604	0:06.080	0:09.775	0:06.348
9	0:14.667	0:09.976	0:06.175	0:10.667	0:05.489	0:04.585	0:05.443
10	0:16.333	0:17.281	0:08.931	0:07.376	0:08.649	0:04.541	0:05.002

A: mandel -x -.5 -y .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	
Number of threads	**Execution time**	**Number of threads**	**Execution time**
1	0:32.361	1	0:14.667
2	0:23.579	2	0:07.725
3	0:15.893	3	0:05.601
4	0:11.663	4	0:05.585
5	0:10.566	5	0:05.489
10	0:07.974	10	0:04.541
50	0:06.291	50	0:03.805



- In configuration A and B, a total of 10 trials were executed for each number of threads. (i.e. 10 trials were taken for each instance of the number of threads)
- Execution time has lot of variation for each number of threads. As execution time is also dependent on number of other factors like number of processes running on the system, number of CPUs available etc.
- Used minimum execution time of each number of threads to plot time vs number of thread graph.
- As the number of threads are increased execution time has decreased
- Curve looks like there is inverse relation between time and number of threads. Initially as we increase threads time decreases drastically but gradually as the number of threads increases the rate of change of time decreases.
- Curve A and Curve B has almost same shape. But for 1 thread Curve A lot more time then curve B but as the number of threads increases this difference in time also decreases.

Conclusion :

- Increase in number of threads decreases execution time.
- As the number of threads increases rate of decrease in time also decreases.
- Execution time also depends on other factors of the system like number of users, number of processes running on the system, number of CPUs available etc. So even far same number of threads execution time may vary.