Jacob Tyree
Lab12

Multiple Threads:

Brief Overview of Implementation:

The modifications made to mandel.c were mainly in the compute_image function. Additionally, a structure was added to easily access variables like the minimum and maximum values for x and y, color schemes, thread ids, and the number of threads. A new function “threat_compute(void *arg)” was made as a helper function for the program. This function took the second half of compute_image and modified it so that it would work with a number of threads. More specifically, it takes each thread and then calculates their portion of the image on that thread alone. This helps with parallel computing because multiple threads are working on a specific portion of the image instead of 1 thread working on the entire image.

Table of Results:

# Processes	Time(s)	# Threads
1	62.302	5
2	33.898	6
3	30.909	7
4	29.795	8
5	27.477	9
6	25.866	10
7	59.216	1
8	36.667	2
9	35.427	3
10	30.24	4

Discussion of Results:

As shown from the graph and the table, as the number of processes and threads increased, the time that it took to complete the program decreased until it reached an asymptote of around 55 seconds.

Although the number of processes helped decrease the runtime, the number of threads made the most significant difference when it came to runtime. This could be because it

allowed for more parallel computing where sections of the code could be run at the same time. Additionally, it allowed for each thread to take a small portion of the image, calculate it, and then piece each portion together at the end of the program. This is faster than taking the whole image, trying to calculate it, and then returning the image one process after another.

There was a “sweet spot” where optimal runtime was achieved. This was around 9 threads and 5 processes, which achieved a run time of about 27 seconds.