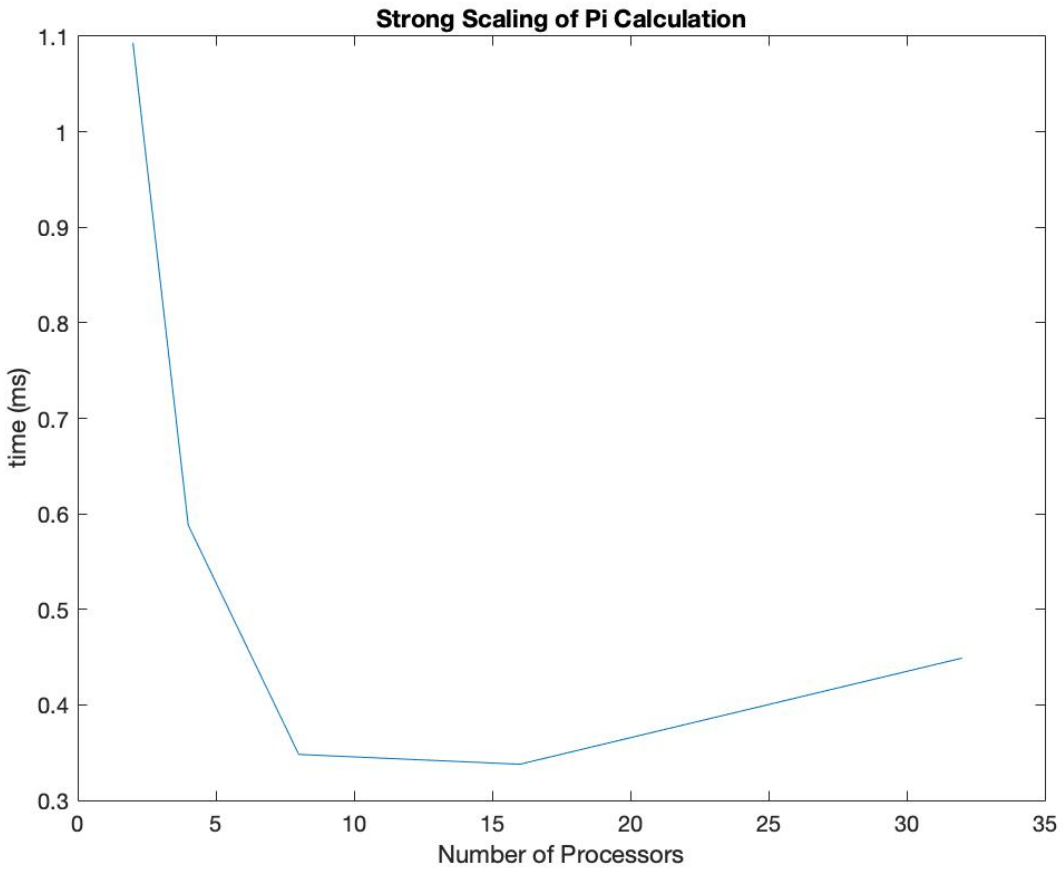
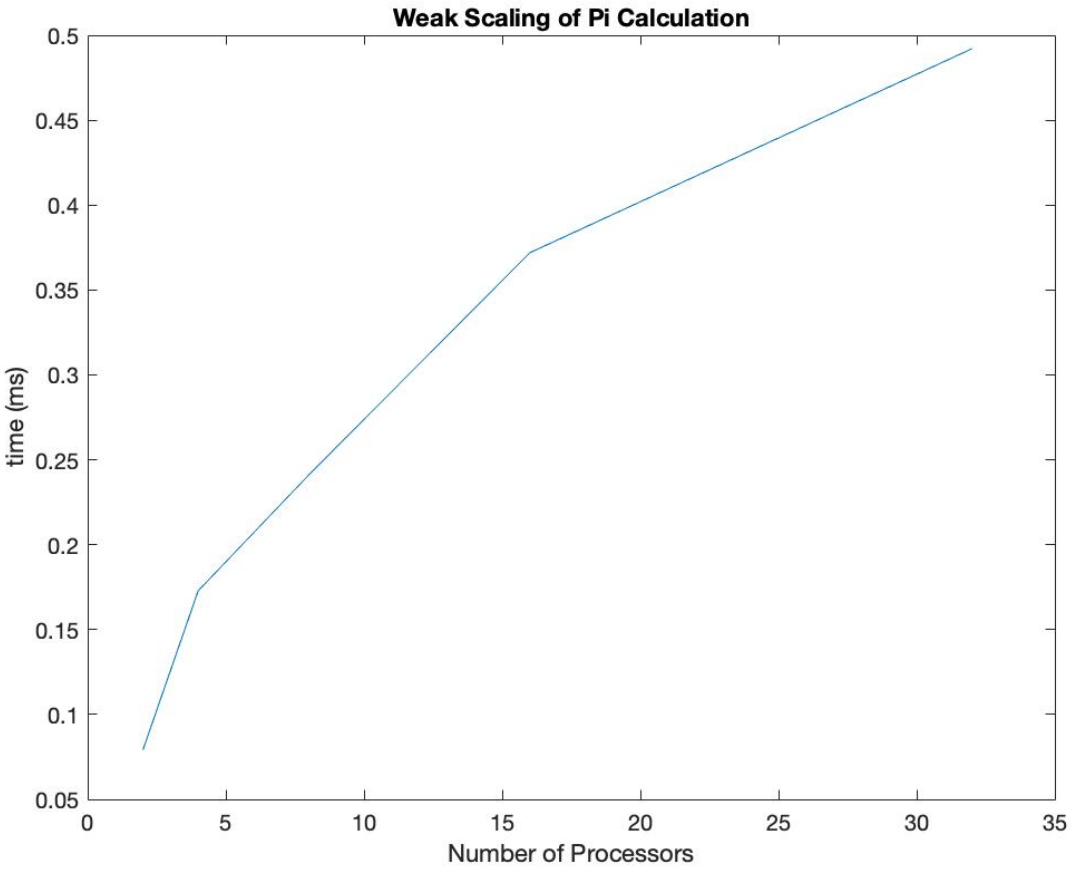
HPC 1 Homework #2 Ezra Rowitsch

Problem 1:

1. The serial code is located in the submitted folder under directory ../Problem_1/ with the filename “pi_num_serial.c”. You can submit the number of bins you wish to calculate with and will get different answers based on your submission. With 20,000 bins you get a solution of 3.141593 which is very close to the actual solution.
2. The parallelized code is in the same directory under “pi_num_parallel_ver2.c”. This has been checked and you get a similar result to the serial code. That can be seen in the various output files I have included in the directory.
3. Below you can see the plot of the strong scaling on the pi numerical calculation problem. You can see that the fastest time was when there were 16 processors doing the calculation.



4. Below you can see a chart of the weak scaling of the numerical calculation. You can see that as the number of processors (and calculations) increase the time increases as well.

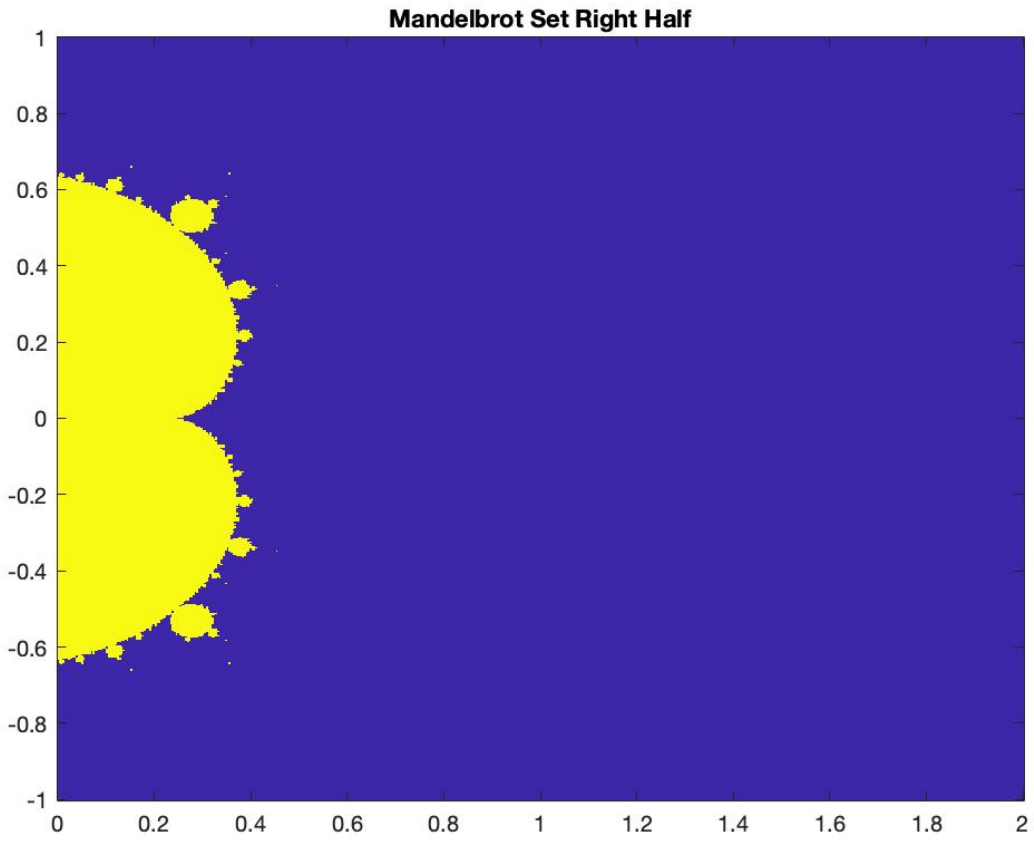


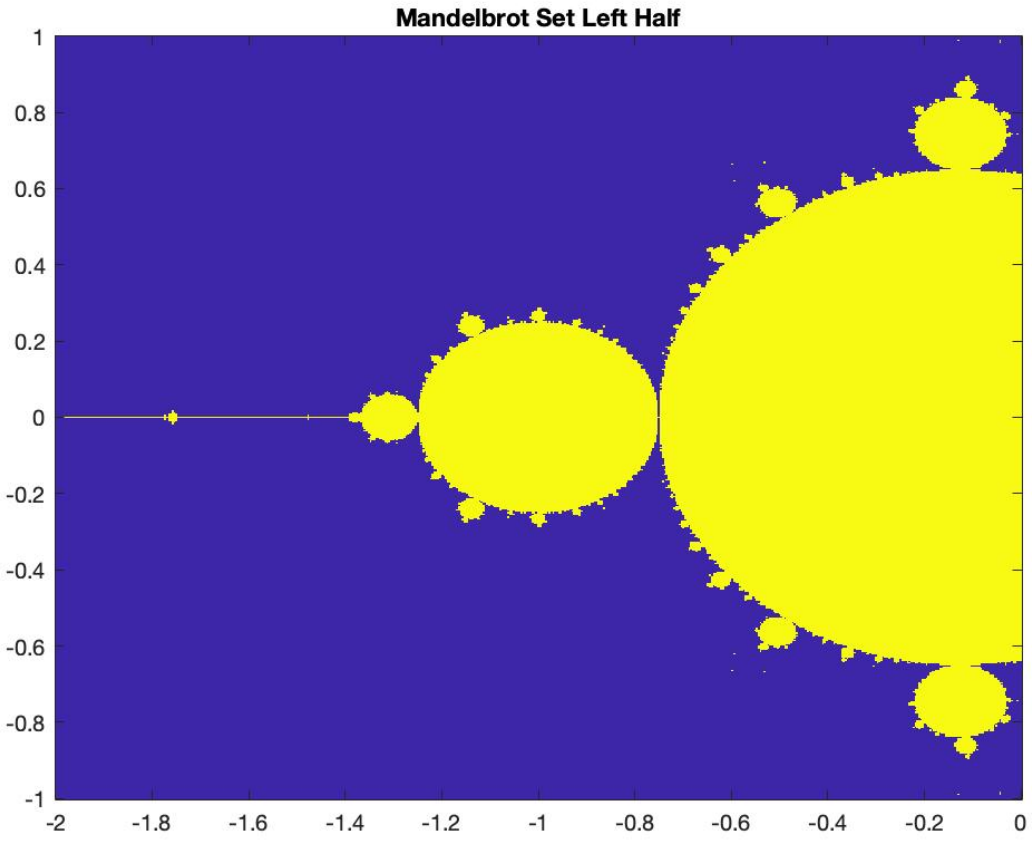
5. From the above strong scaling chart we can see that as the number of processors gets to be over 32 we start to lose efficiency from running a higher number of processes. This indicates that there is some time of communication or memory bandwidth issue that is causing this calculation to slow down. From the output file “pi_1_strong.out" we can get the time to run the program as .00181 seconds (T1). We know from the “pi_16_strong.out” the time we got was 0.0003378 (Tp). By taking T1/Tp we can get the strong speedup which is approximately 5.358. The ideal speedup is close to the number of processors which is 16, this is not close and shows that there were significant losses due to communication and other issues.

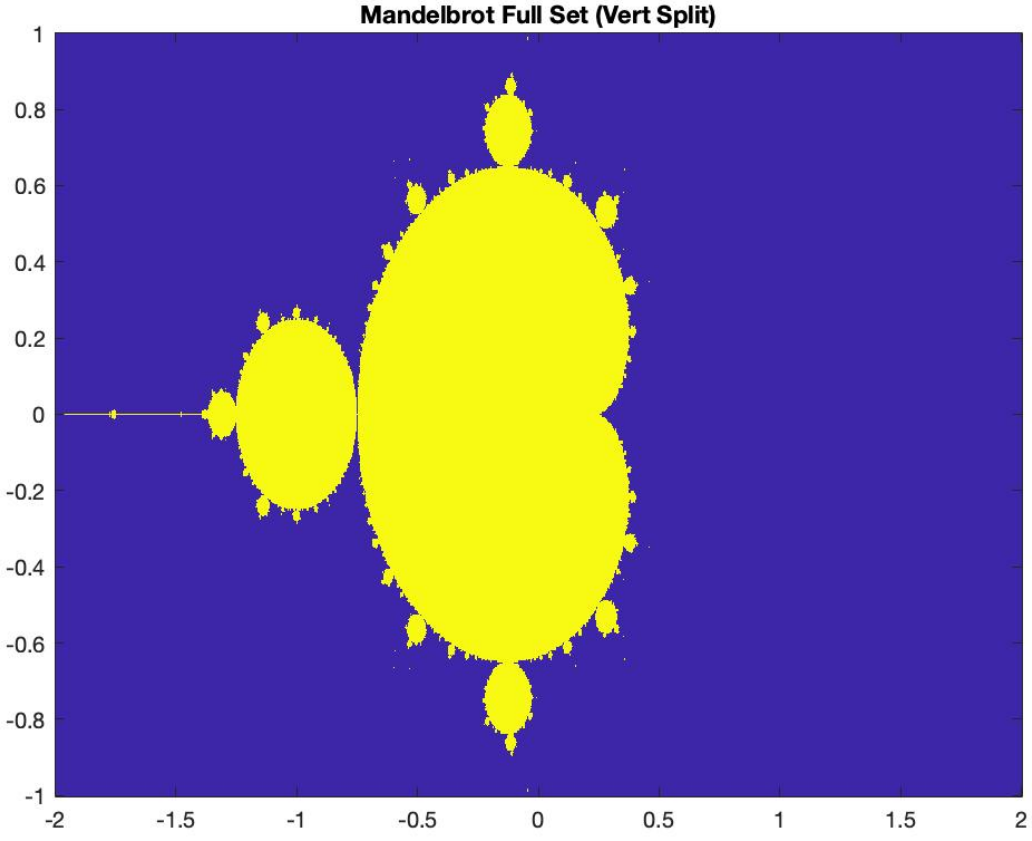
Problem 2:
A:

1. The code for splitting the Mandelbrot domain in two along the vertical axis can be found in the Problem_2 directory under the “Man_ver.c” code.
2. The area for this calculation comes to approximately 1.5081. This is found by counting up all the 1’s in the matrix and multiplying that number by 8/(# of columns * # of rows). This calculation can be seen in detail in the attached Figures_and_calcs.m M-file.
3. For the left side of the Mandelbrot set it took .318116 s and for the right it took .171029. This means the total time it took was .489145 s. If you take .171029/.318116 you get approximately .538. This means that it was approximately half the time to calculate the right side of the set compared to the left, indicating that the load balancing for this problem was not great.

   This makes sense because the left side had most of the points in the mandelbrot set which require the full amount of iterations to confirm, compared to the right where most of the points might become unbounded pretty quickly.
4.




Mandelbrot Set Left Half
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-2
-1.8
-1.6
-1.4
-1.2
-1
-0.8
-0.6
-0.4
-0.2
0

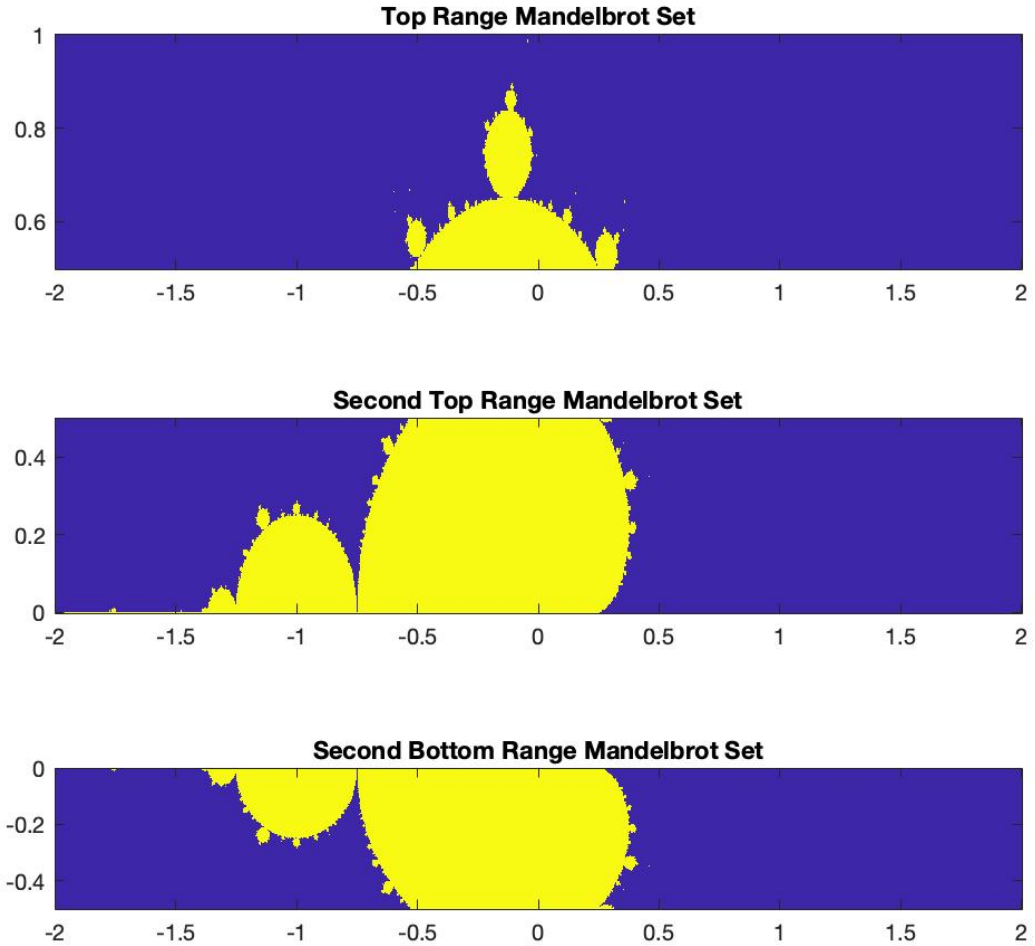

Mandelbrot Full Set (Vert Split)
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

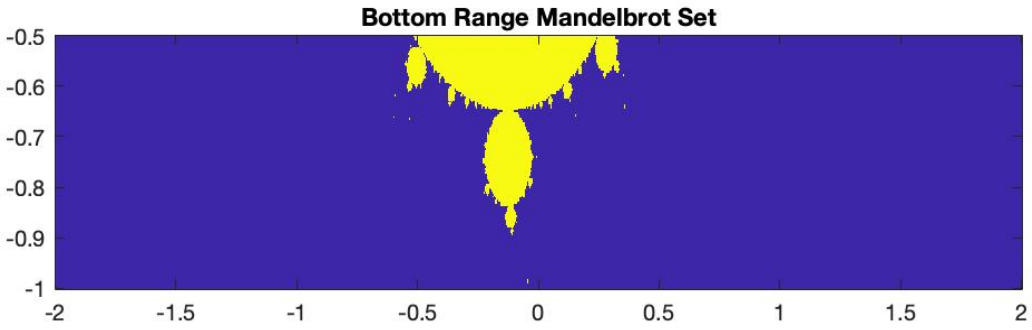
B.

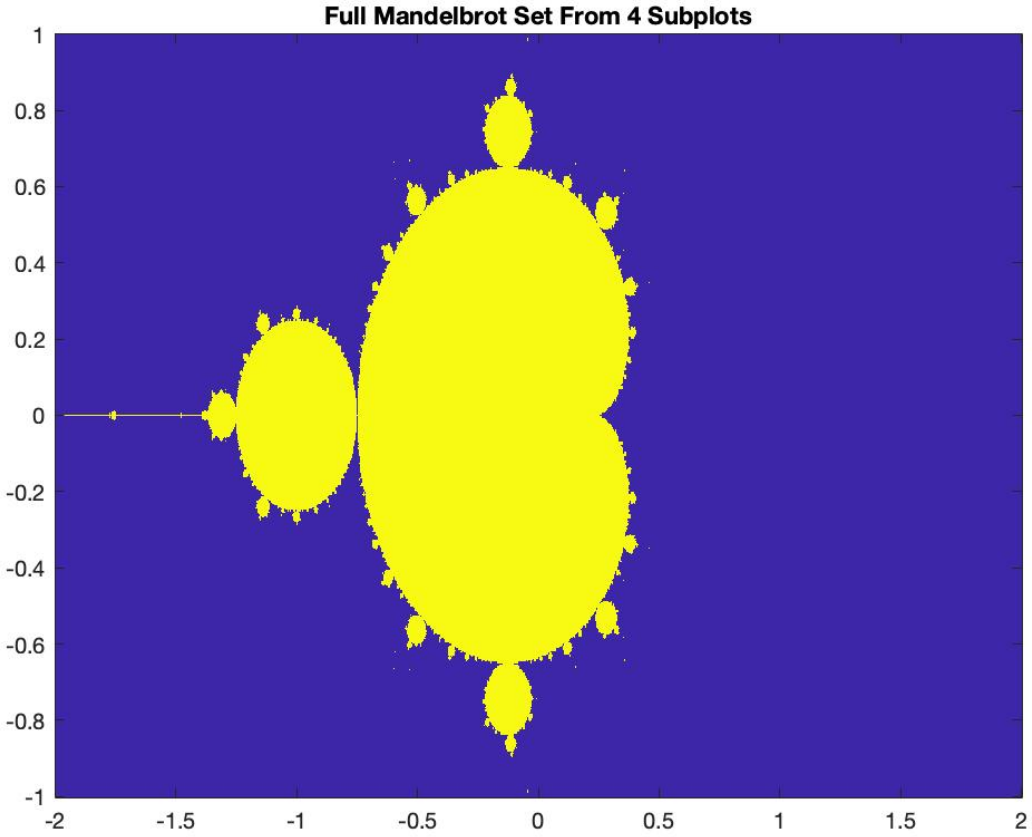
1. The last code for splitting the Mandelbrot set in 4 slices is in the code “Man_final.c”.
2. The area is estimated in the same was as in part A and results in a very similar area of 1.5081.
3. The times we got in calculating this from “Mandelbrot_4_final.out“:
   Rank 0, Total Time: 0.067395
   Rank 1, Total Time: 0.240772
   Rank 2, Total Time: 0.227313
   Rank 3, Total Time: 0.073341

   This leads to a total time of .608821.

   The minimum time divided by the maximum time is .067395/.240772 which leads to approximately .2799 or 28%. This is an even worse problem loading than in part A. Although this makes sense as when you look at the graphics the topmost and bottommost have an even greater difference between which percentage is in the Mandelbrot set.
4.




Bottom Range Mandelbrot Set
-0.5
-0.6
-0.7
-0.8
-0.9
-1
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


Full Mandelbrot Set From 4 Subplots
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2

Problem 3:

1. $T_p = T_1 F_s + \frac{T_1}{p} F_p$

   $T_p = 4000 * .10 + \frac{4000}{96} * .90$

   $T_p = 400 + 37.5$

   $T_p = 437.5$

2. Perfect speedup is not achieved as 10% of the program is serial and therefor not affected by the speedup. Perfect speedup would lead to:

   $T_p = \frac{T_1}{p} = \frac{4000}{96} = 41.6667$