Introduction To Scalable Systems Assignment

By Muttaqi Ahmad Alladin

Aim: To see the effect of parallel programming on the execution times of OpenMP and MPI programs.

Methods: The execution times were obtained on the IISc Turing Cluster that, as per the best of my knowledge, is composed of NVIDIA Kepler K-40 GPU's. All the plots in this assignment were made using various libraries like matplotlib and seaborn in Jupyter Notebook.

Note: The source code used is heavily influenced by the slides of Prof. Sathish Vadhiyar. Moreover, I would also like to cite various sources like stack overflow, geeksforgeeks, and various other online sources.

Note: For the exact execution times please refer to the **RAW Results section that countians the

Results and Discussions:

Q1) Write a sequential program that multiplies a lower triangular matrix, L, with a vector, X, to produce a vector Y, i.e, LX=Y. Parallelize the loop(s) using OpenMP. Use both the static and dynamic scheduling options for the OpenMP for construct.

Use a lower triangular matrix of size 3000x3000, and a vector X of size 3000, initializing both with random values. Execute your OpenMP program with 2,4,8,16 and 32 threads, and report execution times and speedups. Report the execution times in a table, and report speedup as a graph with the x-axis for number of threads, and y-axis for speedups over sequential program. For each experiment with a fixed number of threads, and for the sequential program, run 5 times, and obtain the average execution time across these 5 runs. Report times and speedup for both the static and dynamic scheduling options.

Give a report on the results and observations.

A1) Each program was executed 10 times(instead of 5 required in the question) to increase confidence in values and the plots in this section are made using the average value of these 10 executions. For more information on the exact and average times, please refer to the RAW Results section.

Static Case:

Table 1 shows the average execution times across the 10 runs carried out. Table 2 shows the speedups obtained while executing the programs. Figure 1 shows a line plot of the speedups vs number of threads. Figure 2 shows a bar plot of the speedups vs the number of threads.

Table 1: shows the average execution times under static scheduling.

Threads	Avg Execution Times (10 runs)
1(sequential)	0.16265110
2	0.11474340
4	0.04849139
8	0.04199683
16	0.14521570
32	0.25554960

Table 2: Speedup obtained under static conditions while executing the OpenMP program

# Threads	Speedup
2	1.4175203105363792
4	3.3542263894683155
8	3.8729375526676657
16	1.1200655301045275
32	0.6364756587370906

Figure 1: Line plot of speedups vs number of threads under static scheduling.

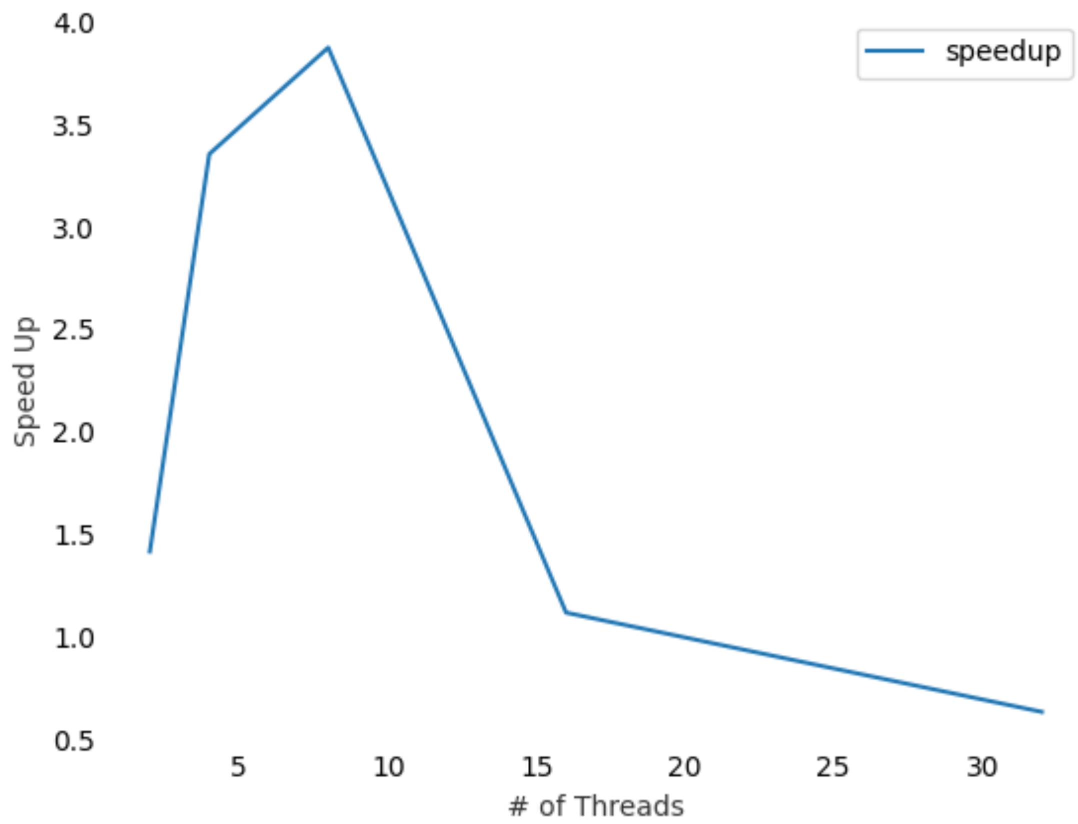
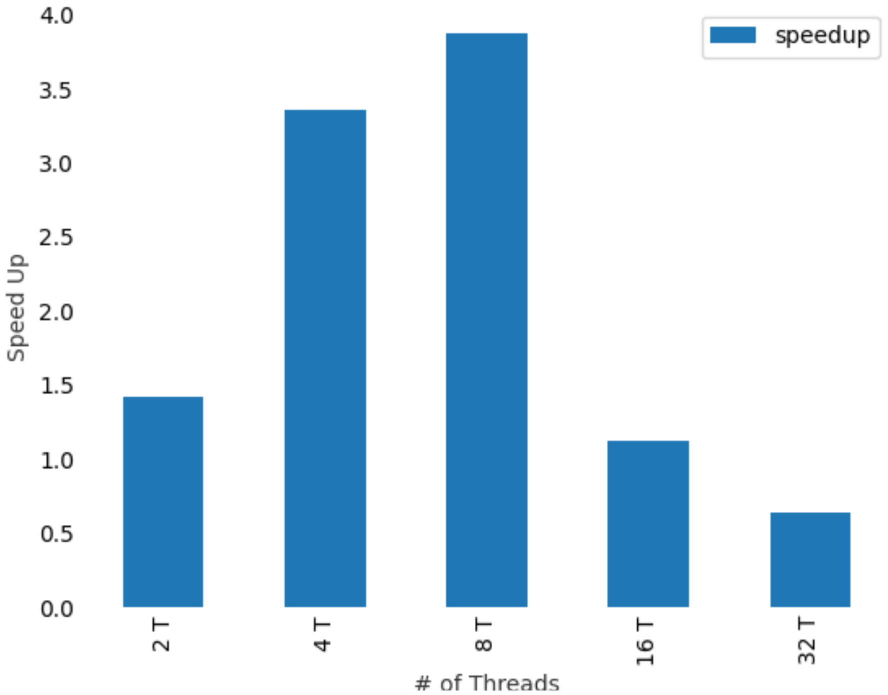


Figure 2: Bar plot of speedup vs number of threads under static scheduling.



Brief Summary of what is observed:

As can be seen from the plots, the speedup initially increases and is maximum at 8 threads. Moreover, we can see that the speedup is massive from 2 threads to 4 threads as the speed more than doubles from 2 threads to 4 threads while the number of threads doubles and, although from 4 threads to 8 threads we do see a speedup, it is not nearly as massive. Then the speedup tends to decrease and at 32 threads, we actually get a slowdown. This is showing that at 32 threads, the overheads of parallelism outweigh the advantages of parallelism.

The above observations show us that optimal parallelism is all about balancing the overheads of parallelism and the benefits of parallelism.

1. In the above example, from 2 threads to 4 threads, we get massive speed up where the threads double (from 2 they become 4) and the speedup more than doubles (from about 1.4 to about 3.35). In this case the overheads of parallelism are completely outweighed by benefits of parallelism and we get a massive speed bump.
2. From 4 threads to 8 threads, the threads again double, but we only get a slight speedup (from about 3.35 to about 3.87). In this case, even though, the benefits of parallelism continue outweigh the costs, we can clearly see that the overheads are becoming somewhat significant and, although we do get a speed bump, compared to the previous case, it is only slight.
3. From 8 threads to 16 threads, we see a decrease in speed up it goes from about 3.87 to about 1.12. Here, we can tell that the overheads of parallelism have become significant and, although we see some performance increase as compared to sequential execution, it is very little as compared to some of the other cases described above.

4. From 16 to 32 threads, we see that any benefit of parallelism is completely outweighed by its costs as we get an execution time that is lower than that of sequential program

Conclusion: Till 8 threads we see the speedups behave somewhat in accordance with Amdahl's Law(which is ideal and assumes no parallelization overheads). After that we see a decrease in speedup, as in practice, there are parallelization overheads and these overheads become significant and tend to dominate with increase in cores.

Dynamic Case:

Table 3 shows the average execution times across the 10 runs carried out. Table 4 shows the speedups obtained while executing the programs. Figure 3 shows a line plot of the speedups vs number of threads. Figure 4 shows a bar plot of the speedups vs the number of threads.

Table 3: shows the average execution times under dynamic scheduling.

Threads	Avg Execution Times (10 runs)
1(sequential)	0.16265110
2	1.2520260
4	1.7592370
8	1.8111590
16	1.9958447
32	2.0855390

Table 4: Speedup obtained under dynamic conditions while executing the OpenMP program

# Threads	Speedup
2	0.12991032135115405
4	0.09245547927880098
8	0.08980498123025088
16	0.08149486781210984
32	0.07798995847116741

Figure 3: Line plot of speedups vs number of threads under dynamic scheduling.

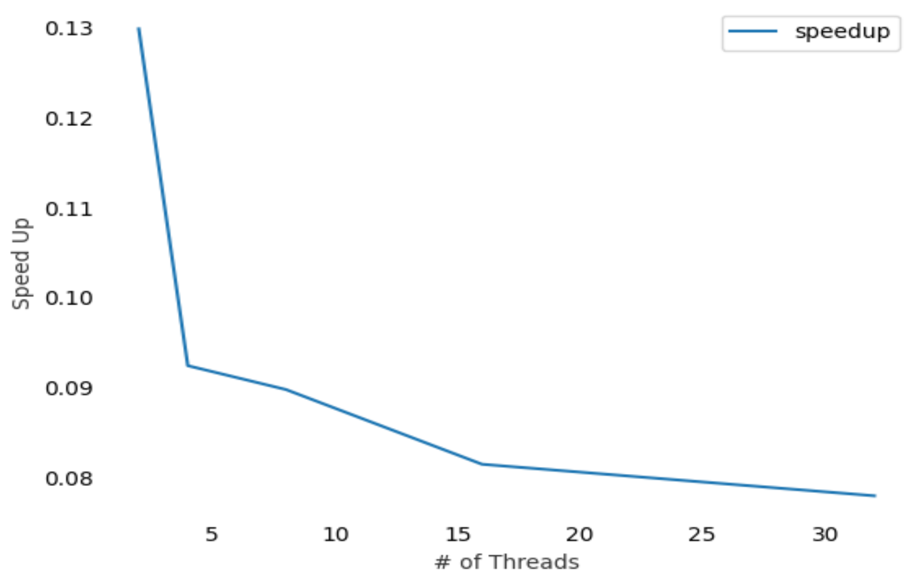
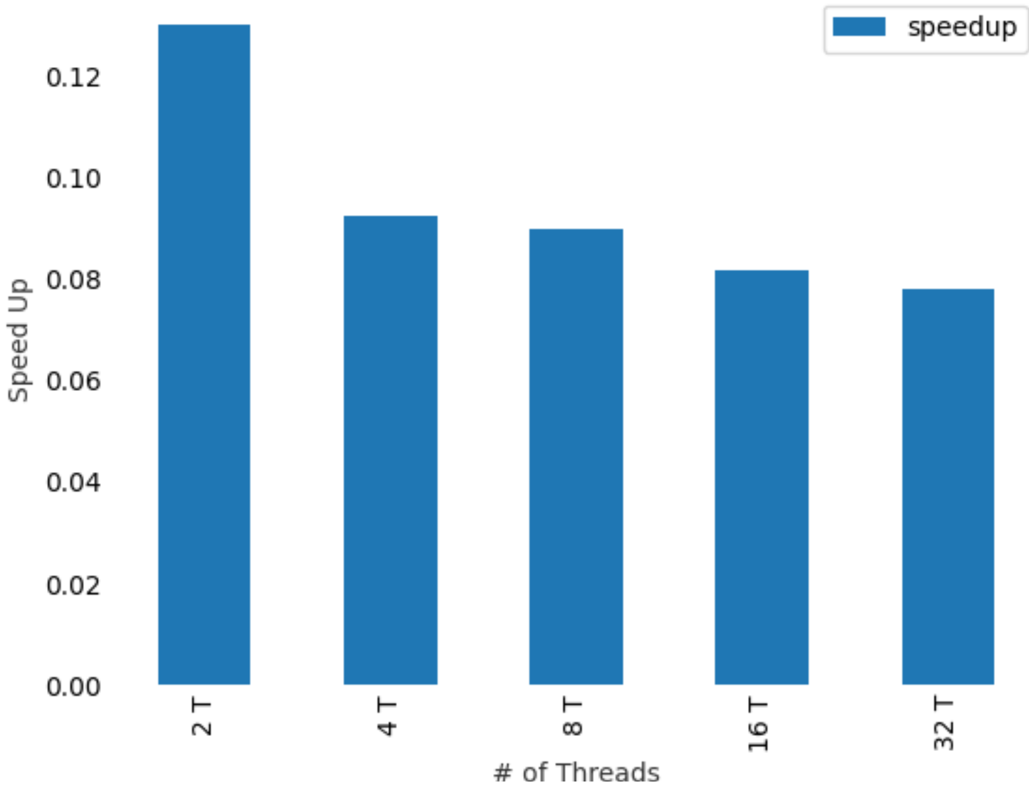


Figure 4: Bar plot of speedup vs number of threads under dynamic scheduling.



Brief Summary of what is observed:

Right off the bat we can see that, in dynamic scheduling, the cost of parallelism outweighs the benefits of parallelism in all cases in this particular example. This is the reason that all the speedups in the above examples are fractions .i.e. The execution time of parallel program is greater than the execution time of sequential program. Moreover, as the number of threads increase, we see that the overheads also tend to increase and we get lower and lower execution times.

OVERALL SUMMERY (comparison of static vs dynamic scheduling):

Before we try to understand why we are getting a better speedup in the case of static scheduling and worse speedup in case of dynamic scheduling, let us try to understand what is static scheduling and what is dynamic scheduling. Static schedule means that iterations blocks are mapped statically to the execution threads in a round-robin fashion. The nice thing with static scheduling is that OpenMP run-time guarantees that if you have two separate loops with the same number of iterations and execute them with the same number of threads using static scheduling, then each thread will receive exactly the same iteration range(s) in all parallel regions. Dynamic scheduling works on a "first come, first served" basis. Two runs with the same number of threads might (and most likely would) produce completely different "iteration space". In general dynamic scheduling has more overheads but ensures better load distribution than static case. Static Scheduling in general ensures better locality as compared to dynamic scheduling.(as everything is statically mapped)

Now, in the above example, the benefit of load balancing in dynamic case doesn't seem to offset the higher overhead of dynamic scheduling and the loss of locality. In my opinion, the bigger factor may have been the loss of locality of reference in dynamic scheduling. Due to this the parallel program became even slower than the sequential program. (I had done an entire assignment about the importance of locality of reference in Prof. Matthew's section of the course) Hence, we see a decrease in performance in dynamic scheduling. While in static scheduling, given the right number of threads, we do see an increase in performance.

Q2) Refer to the MPI class lecture slides that has a MPI program involving two processes that tries to find a particular element in an array distributed across the two processes. Now write a MPI program that works with any number of processes.

For this program, generate a random double array of size 1000000 (i.e., 1 million) with random integer elements between 1 to 5000000 (5 million). You can either generate this array in process 0 and distribute it equally across all the processes or generate this array to a file and make the processes read from different portions of the file (e.g., using fseek). Now an integer element is given as input to the program and the processes start searching for this element in their local sub arrays. As soon as a process finds the element, it informs the other processes and all processes will stop searching. Process 0 should print the global index of the overall array in which the element was found.

Execute the program with different number of processes including 1 (sequential), 2, 4, 8, 16, 32 and 64. For each number of processes, execute with 50 different random input numbers for searching and in each case, measure the time taken for the search. Report the average time (across the 50 instances) taken for each number of processes. Plot the execution times and speedups for different number of processes. Ensure that you execute different processes on different cores.
Prepare a report with the methodology, execution times, and speedup graphs

Ans 2:
Methodology:
A c program was created that carried out the experiments; This c program was heavily inspired by the slides of Prof. Sathish Vadhiyar and the resources provided by Dartmouth College. The program was executed at the Turing cluster of IISc and I executed the program at around 1:30-3:00 AM. Before executing I also ran "qstat" to ensure that no other person was executing anything at the Turing cluster.

Care was taken to ensure that the workload of different threads was well balanced and that if any thread found the integer value, all threads would stop searching. To ensure this by using non-blocking receive calls and at the end of the loop the processor would check if any other processors had found the elements.(similar to example in class) In case the a processor found the element, it would immediately send a message to all other processors and exit the loop. I used the "file" approach in this program. The time was calculated using "MPI_Wtime()". My entire code can be seen using the Turing cluster. It is in the directory "MPI_Assignment".

In general, the OS would take care and execute all the threads optimally in different cores but as this problem explicitly asked us to ensure this, this can be done using the command switch "--bind-to core" while using "mpiexec"

Table 5 shows the average execution times across the 50 runs. Table 6 shows the speedups. Figure 5 shows a line plot of the speedups vs number of threads. Figure 6 shows a bar plot of the speedups vs the number of threads.

Table 5: shows the average execution times for the MPI program.

Threads	Avg Execution Times (10 runs)
1(sequential)	0.16676494
2	0.10686822
4	0.05364030
8	0.02452324
16	0.01173188
32	0.00935516
64	0.00878192

Table 6: Speedup obtained while executing the MPI program

# Threads	Speedup
2	1.560472701800404
4	3.108948682240779
8	6.800281691978710
16	14.214681704892996
32	17.825984804108110
64	18.989576311330556

Figure 5: Line plot of speedups vs number of threads for MPI Program.

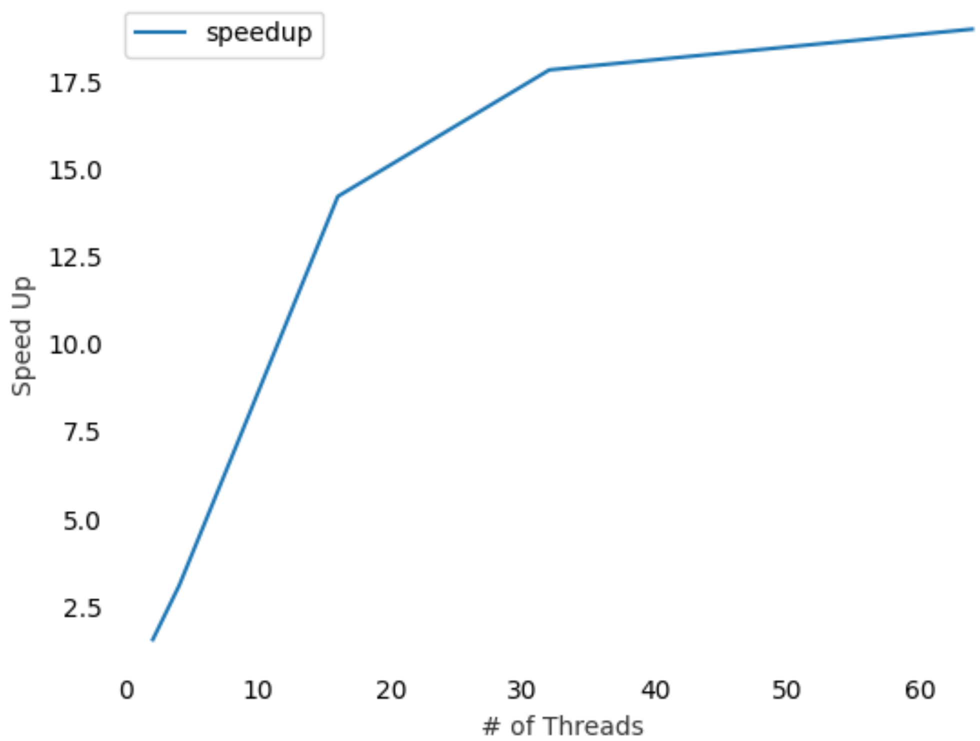
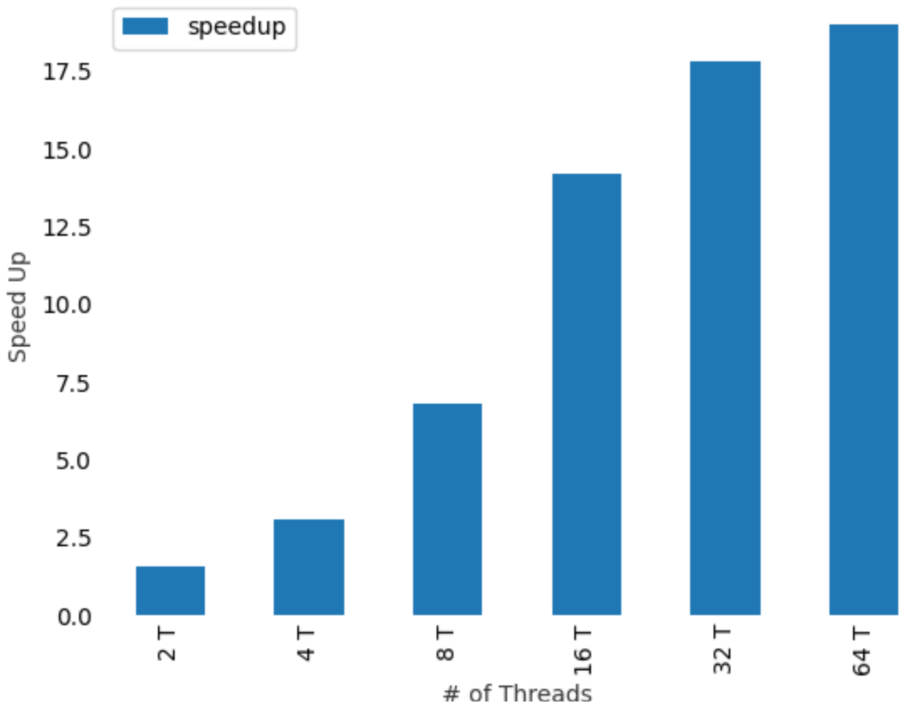


Figure 6: Bar plot of speedup vs number of threads for MPI Program.



Justification: This is a classic speedup curve and we can clearly see that the speed initially increases and then tens to approach an asymptotic constant. This happens because the entire work in a program can be divided into the two parts; The sequential part (like generation of random array) and the parallelizable part(finding the element). As the number of cores increases, the time taken to execute the parallel part decreases while the time taken to execute the sequential part stays the same. So ideally (assuming no synchronization overheads and other overheads of parallel programming), as the number of cores double, the time taken to execute the parallel portion of the code halves. But the time taken to execute the sequential portion of the code stays the same. So, theoretically, as the number of cores increase and become infinite, we will approach the asymptotic constant which is the sequential portion. (**Note: this is only theoretical, practically we will see parallelization overheads will become significant).

This entire curve is in accordance with **Amdahl's Law.****Note: we use Amdahl's law and not Gustafson's Law as this is **strong scaling** as we are not changing the problem size.

Amdahl's Law:

$$Speedup = \frac{1}{f_s + \frac{f_p}{P}}$$

$f_s = sequential\ work$

$f_p = paralell\ work$

$p = \#\ of\ processors$

Thus, we get the above curve which is in line with this law.

**Note: Even though we know that, practically, there will be overheads associated with parallelism and these overheads will become significant with high number of cores as in the case of OpenMP programs, it seems like in this case (MPI), at least till 64 threads, they seem to stay in check and we get performance that is in line with Amdahl's Law.

Screen Shots:

OpenMP Execution 10 times:

```
0.161305
0.162573
0.161906
0.163817
0.162621
0.164072
0.162861
0.161702
0.163119
0.162535
muttaqiahmad@turing:~/Assignment_Problem_1$
```

MPI Execution 50 times

```
total time is =0.026219
total time is =0.191035
total time is =0.104029
total time is =0.188006
total time is =0.180232
total time is =0.189612
total time is =0.192071
total time is =0.189332
total time is =0.190104
total time is =0.171760
total time is =0.192557
total time is =0.195592
total time is =0.188975
total time is =0.186739
total time is =0.190238
total time is =0.181303
total time is =0.178539
total time is =0.191930
total time is =0.190334
total time is =0.185902
total time is =0.186472
total time is =0.171391
total time is =0.184496
total time is =0.181761
total time is =0.005501
total time is =0.111797
total time is =0.190277
total time is =0.200917
total time is =0.194487
total time is =0.190786
total time is =0.189869
total time is =0.189726
total time is =0.004677
total time is =0.175233
total time is =0.188816
total time is =0.189653
total time is =0.184330
total time is =0.183938
total time is =0.057332
total time is =0.169764
total time is =0.184997
total time is =0.185748
total time is =0.190493
total time is =0.187544
total time is =0.187899
total time is =0.165277
total time is =0.193292
total time is =0.186028
total time is =0.051600
total time is =0.184322
muttaqiahmad@turing:~/Assignment_MPI$
```

Note: The values above MPI values may be different from those in the raw results as I hadn't captured any screenshots while making the report. The screenshots were captured later to fulfill the requirements of this assignment.

RAW Results: This section contains the time values!
STATIC VALUES OPEN-MP:

- Sequential program (with all OpenMP commands commented):
    1. 0.161305
    2. 0.162573
    3. 0.161906
    4. 0.163817
    5. 0.162621
    6. 0.164072
    7. 0.162861
    8. 0.161702
    9. 0.163119
    10. 0.162535
- 1 thread static: avg =0.16657729999999998
    1. 0.166334
    2. 0.166272
    3. 0.166209
    4. 0.168736
    5. 0.168982
    6. 0.16705
    7. 0.169292
    8. 0.168469
    9. 0.16859
    10. 0.155839
- 2 thread static: avg =0.11474340000000001
    1. 0.122143
    2. 0.11166
    3. 0.111934
    4. 0.115973
    5. 0.107722
    6. 0.114651
    7. 0.114728
    8. 0.117687
    9. 0.10943
    10. 0.121506
- 4 thread static: avg=0.04849139
    1. 0.056385
    2. 0.0505615
    3. 0.0505394
    4. 0.0480742
    5. 0.0460362
    6. 0.0471504
    7. 0.0476071
    8. 0.044509
    9. 0.0473687
    10. 0.0466824

- 8 thread static: avg=0.04199683
    1. 0.0387208
    2. 0.0464555
    3. 0.0417397
    4. 0.031519
    5. 0.0485641
    6. 0.0340741
    7. 0.0418834
    8. 0.0353986
    9. 0.0700856
    10. 0.0315275

- 16 thread static: avg=0.14521569999999998
    1. 0.134312
    2. 0.152184
    3. 0.148096
    4. 0.14992
    5. 0.136438
    6. 0.152864
    7. 0.145588
    8. 0.140065
    9. 0.145543
    10. 0.147147
- 32 thread static: avg=0.2555496
    1. 0.260556
    2. 0.273162
    3. 0.249569
    4. 0.265561
    5. 0.248866
    6. 0.241428
    7. 0.25023
    8. 0.249921
    9. 0.25222
    10. 0.263983

DYNAMIC VALUES OPEN-MP:

- 1 thread dynamic
    1. 0.574962
    2. 0.44104
    3. 0.445474
    4. 0.442693
    5. 0.449992
    6. 0.437777
    7. 0.444221
    8. 0.578157
    9. 0.579931
    10. 0.580025

- 2 thread dynamic
  1. 1.33465
  2. 1.3116
  3. 1.28554
  4. 1.28954
  5. 1.22316
  6. 1.14572
  7. 1.21935
  8. 1.19379
  9. 1.23482
  10. 1.28209
- 4 thread dynamic
  1. 1.75425
  2. 1.82273
  3. 1.70508
  4. 1.79929
  5. 1.75018
  6. 1.85969
  7. 1.73333
  8. 1.70742
  9. 1.79199
  10. 1.66841
- 8 thread dynamic
  1. 1.97088
  2. 1.71536
  3. 1.65543
  4. 1.71482
  5. 1.73453
  6. 1.7017
  7. 1.8178
  8. 1.94309
  9. 1.94098
  10. 1.917
- 16 thread dynamic
  1. 1.779987
  2. 1.94294
  3. 1.90832
  4. 2.06697
  5. 2.02656
  6. 2.02389
  7. 1.96821
  8. 2.1087
  9. 2.11159
  10. 2.02128
- 32 thread dynamic
  1. 2.14421
  2. 2.16596

3. 2.17596
4. 2.01175
5. 2.02446
6. 2.11971
7. 2.03874
8. 2.11786
9. 2.01813
10. 2.03861

MPI-Assignment

Times recorded:

Sequential code:(1 processor)

1. total time is =0.027499
2. total time is =0.187619
3. total time is =0.099823
4. total time is =0.186348
5. total time is =0.190305
6. total time is =0.189825
7. total time is =0.189206
8. total time is =0.189511
9. total time is =0.189740
10. total time is =0.167472
11. total time is =0.186152
12. total time is =0.186227
13. total time is =0.186623
14. total time is =0.169931
15. total time is =0.184345
16. total time is =0.185687
17. total time is =0.186579
18. total time is =0.186338
19. total time is =0.193494
20. total time is =0.186325
21. total time is =0.189772
22. total time is =0.175033
23. total time is =0.183467
24. total time is =0.165575
25. total time is =0.005022
26. total time is =0.127482
27. total time is =0.192176
28. total time is =0.186860
29. total time is =0.187405
30. total time is =0.187058
31. total time is =0.189200
32. total time is =0.192264
33. total time is =0.005081

34. total time is =0.184976
35. total time is =0.185869
36. total time is =0.189237
37. total time is =0.191303
38. total time is =0.191595
39. total time is =0.057455
40. total time is =0.154387
41. total time is =0.184600
42. total time is =0.190787
43. total time is =0.182114
44. total time is =0.186986
45. total time is =0.182968
46. total time is =0.191390
47. total time is =0.185766
48. total time is =0.187304
49. total time is =0.046901
50. total time is =0.189165

2 processors;

1. total time is =0.028818
2. total time is =0.135448
3. total time is =0.085514
4. total time is =0.132125
5. total time is =0.112843
6. total time is =0.110086
7. total time is =0.083744
8. total time is =0.118911
9. total time is =0.105722
10. total time is =0.109066
11. total time is =0.128057
12. total time is =0.106135
13. total time is =0.133784
14. total time is =0.129672
15. total time is =0.133380
16. total time is =0.128728
17. total time is =0.125855
18. total time is =0.112666
19. total time is =0.112996
20. total time is =0.105006
21. total time is =0.143037
22. total time is =0.113130
23. total time is =0.112571
24. total time is =0.111596
25. total time is =0.004703
26. total time is =0.140665
27. total time is =0.123463
28. total time is =0.137912

29. total time is =0.107940
30. total time is =0.106952
31. total time is =0.108095
32. total time is =0.105926
33. total time is =0.006084
34. total time is =0.093727
35. total time is =0.109792
36. total time is =0.108423
37. total time is =0.130903
38. total time is =0.114810
39. total time is =0.059937
40. total time is =0.105439
41. total time is =0.111373
42. total time is =0.085859
43. total time is =0.111830
44. total time is =0.109994
45. total time is =0.112676
46. total time is =0.086113
47. total time is =0.140357
48. total time is =0.141691
49. total time is =0.051387
50. total time is =0.108470

4processors:

1. total time is =0.029183
2. total time is =0.062134
3. total time is =0.064701
4. total time is =0.043061
5. total time is =0.043245
6. total time is =0.066866
7. total time is =0.061065
8. total time is =0.050018
9. total time is =0.043332
10. total time is =0.062932
11. total time is =0.046007
12. total time is =0.059987
13. total time is =0.064740
14. total time is =0.055490
15. total time is =0.056710
16. total time is =0.048777
17. total time is =0.062603
18. total time is =0.059135
19. total time is =0.050509
20. total time is =0.063573
21. total time is =0.064145
22. total time is =0.049899
23. total time is =0.061787

24. total time is =0.059474
25. total time is =0.062773
26. total time is =0.060246
27. total time is =0.066041
28. total time is =0.045375
29. total time is =0.043226
30. total time is =0.063893
31. total time is =0.057607
32. total time is =0.046793
33. total time is =0.006296
34. total time is =0.042788
35. total time is =0.061536
36. total time is =0.043042
37. total time is =0.070112
38. total time is =0.068810
39. total time is =0.042502
40. total time is =0.051107
41. total time is =0.060936
42. total time is =0.064896
43. total time is =0.043432
44. total time is =0.043007
45. total time is =0.060279
46. total time is =0.071746
47. total time is =0.044970
48. total time is =0.051152
49. total time is =0.037139
50. total time is =0.042938

8processors:

1. total time is =0.026773
2. total time is =0.027788
3. total time is =0.022299
4. total time is =0.024772
5. total time is =0.033978
6. total time is =0.022285
7. total time is =0.022429
8. total time is =0.022947
9. total time is =0.022325
10. total time is =0.022474
11. total time is =0.022282
12. total time is =0.022404
13. total time is =0.022585
14. total time is =0.023013
15. total time is =0.023510
16. total time is =0.022709
17. total time is =0.022430
18. total time is =0.035752

19. total time is =0.022401
20. total time is =0.021448
21. total time is =0.032363
22. total time is =0.022696
23. total time is =0.022417
24. total time is =0.024774
25. total time is =0.024513
26. total time is =0.022353
27. total time is =0.022326
28. total time is =0.038423
29. total time is =0.026625
30. total time is =0.022372
31. total time is =0.023289
32. total time is =0.022729
33. total time is =0.022737
34. total time is =0.027069
35. total time is =0.038902
36. total time is =0.023726
37. total time is =0.022305
38. total time is =0.026625
39. total time is =0.023819
40. total time is =0.022858
41. total time is =0.022946
42. total time is =0.022402
43. total time is =0.022445
44. total time is =0.022453
45. total time is =0.023258
46. total time is =0.022847
47. total time is =0.022385
48. total time is =0.022965
49. total time is =0.022394
50. total time is =0.022542

16 processors:

1. total time is =0.012717
2. total time is =0.012620
3. total time is =0.012839
4. total time is =0.011204
5. total time is =0.011634
6. total time is =0.011568
7. total time is =0.011589
8. total time is =0.011423
9. total time is =0.011313
10. total time is =0.011485
11. total time is =0.011137
12. total time is =0.011762
13. total time is =0.011369
14. total time is =0.011145

15. total time is =0.011587
16. total time is =0.011652
17. total time is =0.011595
18. total time is =0.011455
19. total time is =0.011577
20. total time is =0.011385
21. total time is =0.011430
22. total time is =0.011113
23. total time is =0.011341
24. total time is =0.011490
25. total time is =0.003478
26. total time is =0.011241
27. total time is =0.011307
28. total time is =0.011531
29. total time is =0.011288
30. total time is =0.011225
31. total time is =0.011597
32. total time is =0.011653
33. total time is =0.003404
34. total time is =0.011481
35. total time is =0.012706
36. total time is =0.011305
37. total time is =0.011471
38. total time is =0.011451
39. total time is =0.011623
40. total time is =0.024416
41. total time is =0.011434
42. total time is =0.011375
43. total time is =0.027851
44. total time is =0.011536
45. total time is =0.011230
46. total time is =0.011909
47. total time is =0.011334
48. total time is =0.011301
49. total time is =0.006994
50. total time is =0.012023

32 processors:

1. total time is =0.008262
2. total time is =0.007981
3. total time is =0.025752
4. total time is =0.007609
5. total time is =0.007361
6. total time is =0.031605
7. total time is =0.023141
8. total time is =0.007633
9. total time is =0.017145

10. total time is =0.007657
11. total time is =0.007547
12. total time is =0.016695
13. total time is =0.017402
14. total time is =0.007463
15. total time is =0.007625
16. total time is =0.007312
17. total time is =0.007552
18. total time is =0.023602
19. total time is =0.007686
20. total time is =0.007653
21. total time is =0.007473
22. total time is =0.023506
23. total time is =0.009021
24. total time is =0.007673
25. total time is =0.004586
26. total time is =0.007369
27. total time is =0.007882
28. total time is =0.007621
29. total time is =0.007608
30. total time is =0.007342
31. total time is =0.006732
32. total time is =0.020358
33. total time is =0.004502
34. total time is =0.007456
35. total time is =0.007689
36. total time is =0.000982
37. total time is =0.001770
38. total time is =0.007556
39. total time is =0.002903
40. total time is =0.007639
41. total time is =0.007480
42. total time is =0.001520
43. total time is =0.001515
44. total time is =0.007541
45. total time is =0.007474
46. total time is =0.007405
47. total time is =0.007670
48. total time is =0.007428
49. total time is =0.000303
50. total time is =0.007071

64 threads:

1. total time is =0.005511
2. total time is =0.005282
3. total time is =0.005082
4. total time is =0.005063
5. total time is =0.030224

6. total time is =0.040095
7. total time is =0.005115
8. total time is =0.005393
9. total time is =0.009182
10. total time is =0.005130
11. total time is =0.005050
12. total time is =0.041926
13. total time is =0.004936
14. total time is =0.002805
15. total time is =0.004925
16. total time is =0.005243
17. total time is =0.033146
18. total time is =0.045137
19. total time is =0.005142
20. total time is =0.005116
21. total time is =0.005145
22. total time is =0.004908
23. total time is =0.001199
24. total time is =0.005126
25. total time is =0.001385
26. total time is =0.005105
27. total time is =0.005147
28. total time is =0.005306
29. total time is =0.035698
30. total time is =0.005190
31. total time is =0.009245
32. total time is =0.002963
33. total time is =0.005026
34. total time is =0.001235
35. total time is =0.005179
36. total time is =0.001038
37. total time is =0.004151
38. total time is =0.005015
39. total time is =0.002112
40. total time is =0.007134
41. total time is =0.009084
42. total time is =0.005095
43. total time is =0.008691
44. total time is =0.005140
45. total time is =0.005185
46. total time is =0.005090
47. total time is =0.004967
48. total time is =0.002549
49. total time is =0.001236
50. total time is =0.005249