CSCE 735 Fall 2022 – HW1

Name: Sai Namith Garapati
UIN: 832001176

Part 1. Shared-Memory Programming with Threads

1. **Execute the code for $n=10^8$ with p chosen to be 2^k, for k = 0, 1, ..., 13. Using the experimental data obtained from these experiments, answer the following questions. For plots, use a logarithmic scale for the x-axis.**

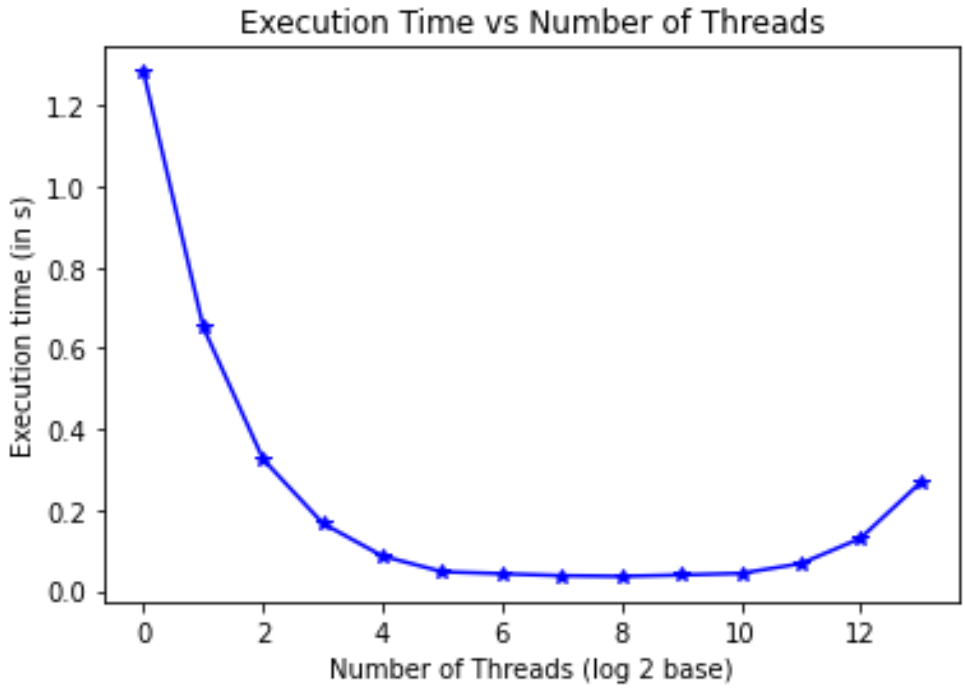
Solution:

Simulation Results:

```
Trials = 100000000, Threads =    1, pi =   3.1416149600, error = 7.10e-06, time (sec) =   1.2839
Trials = 100000000, Threads =    2, pi =   3.1417022800, error = 3.49e-05, time (sec) =   0.6488
Trials = 100000000, Threads =    4, pi =   3.1415910800, error = 5.01e-07, time (sec) =   0.3252
Trials = 100000000, Threads =    8, pi =   3.1415947600, error = 6.70e-07, time (sec) =   0.1653
Trials = 100000000, Threads =   16, pi =   3.1415291200, error = 2.02e-05, time (sec) =   0.0856
Trials = 100000000, Threads =   32, pi =   3.1415901200, error = 8.06e-07, time (sec) =   0.0466
Trials = 100000000, Threads =   64, pi =   3.1413512400, error = 7.68e-05, time (sec) =   0.0392
Trials = 100000000, Threads =  128, pi =   3.1429299200, error = 4.26e-04, time (sec) =   0.0350
Trials = 100000000, Threads =  256, pi =   3.1412995200, error = 9.33e-05, time (sec) =   0.0357
Trials = 100000000, Threads =  512, pi =   3.1468450800, error = 1.67e-03, time (sec) =   0.0364
Trials = 100000000, Threads = 1024, pi =   3.1514026000, error = 3.12e-03, time (sec) =   0.0436
Trials = 100000000, Threads = 2048, pi =   3.1453611600, error = 1.20e-03, time (sec) =   0.0675
Trials = 100000000, Threads = 4096, pi =   3.1494162400, error = 2.49e-03, time (sec) =   0.1279
Trials = 100000000, Threads = 8192, pi =   3.1377031200, error = 1.24e-03, time (sec) =   0.2668
```

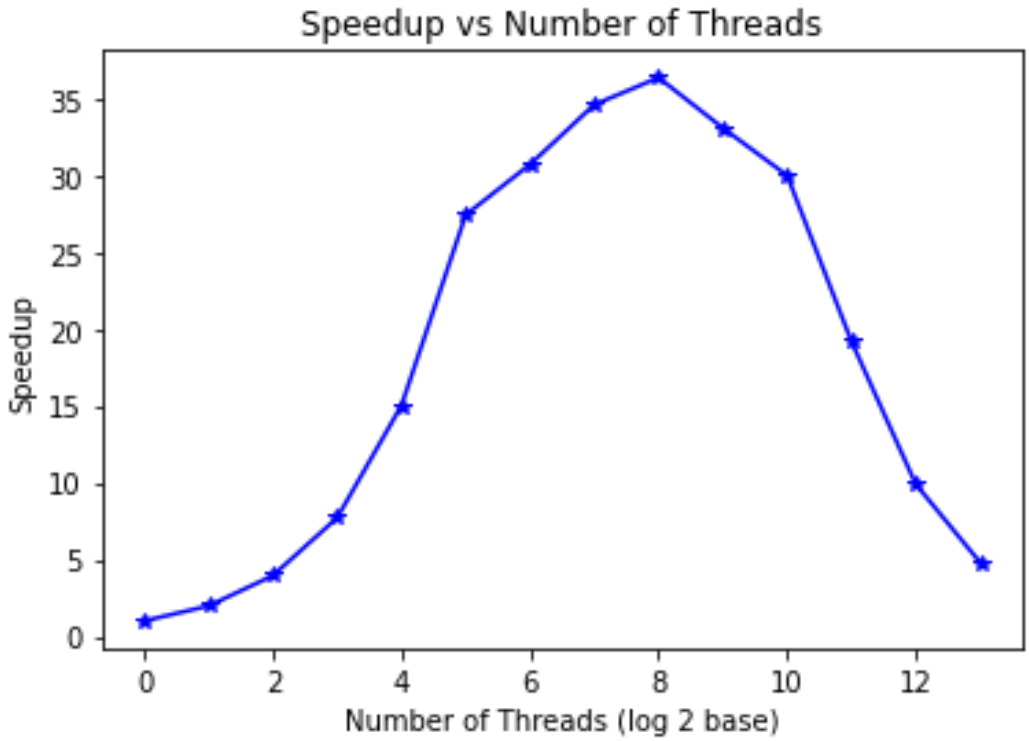
1.1. **(10 points) Plot execution time versus p to demonstrate how time varies with the number of threads.**

Solution: (next page)



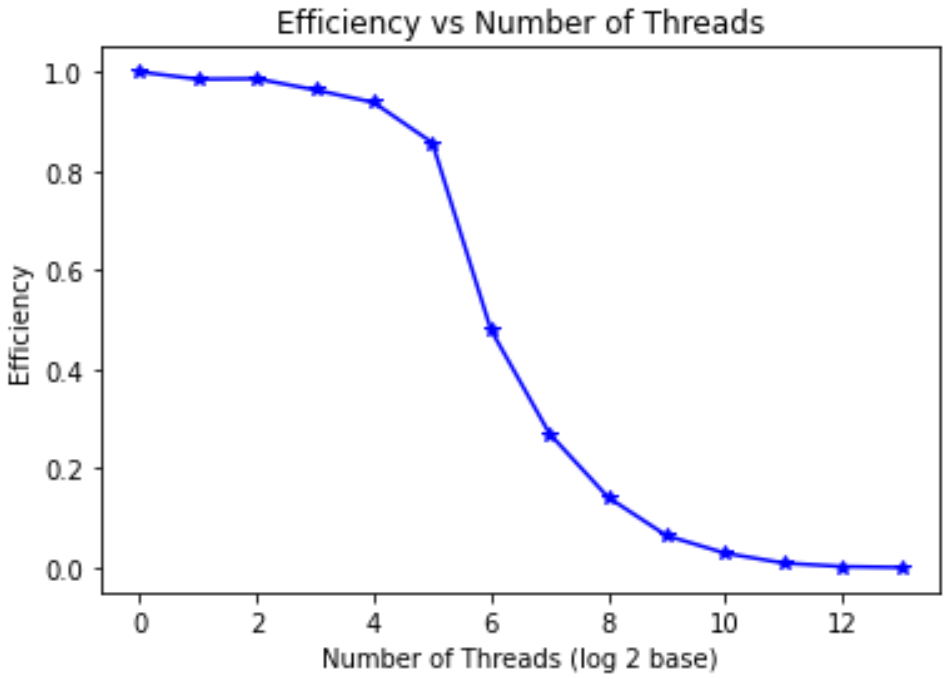
1.2. (10 points) Plot speedup versus p to demonstrate the change in speedup with p.

Solution:



1.3. (5 points) Using the definition: efficiency = speedup/p, plot efficiency versus p to demonstrate how efficiency changes as the number of threads are increased.

Solution:



1.4. (5 points) In your experiments, what value of p minimizes the parallel runtime?

Solution:

As observed from the results and plots, we can say that '**p = 128**' minimizes the parallel runtime.

2. Repeat the experiments with $n=10^{10}$ to obtain the execution time for $p=2^k$, for k = 0, 1, ..., 13.

Simulation Results:

```
Trials = 10000000000, Threads =    1, pi =   1.4236202260, error = 5.47e-01, time (sec) = 127.9119
Trials = 10000000000, Threads =    2, pi =   3.1416070092, error = 4.57e-06, time (sec) =  64.0837
Trials = 10000000000, Threads =    4, pi =   3.1416060276, error = 4.26e-06, time (sec) =  32.0492
Trials = 10000000000, Threads =    8, pi =   3.1416118228, error = 6.10e-06, time (sec) =  16.0359
Trials = 10000000000, Threads =   16, pi =   3.1416066896, error = 4.47e-06, time (sec) =   8.0279
Trials = 10000000000, Threads =   32, pi =   3.1416332068, error = 1.29e-05, time (sec) =   4.0209
Trials = 10000000000, Threads =   64, pi =   3.1416139092, error = 6.77e-06, time (sec) =   3.1285
Trials = 10000000000, Threads =  128, pi =   3.1415943424, error = 5.38e-07, time (sec) =   2.7322
Trials = 10000000000, Threads =  256, pi =   3.1416473120, error = 1.74e-05, time (sec) =   2.7004
Trials = 10000000000, Threads =  512, pi =   3.1415762012, error = 5.24e-06, time (sec) =   2.6967
Trials = 10000000000, Threads = 1024, pi =   3.1414319268, error = 5.12e-05, time (sec) =   2.6947
Trials = 10000000000, Threads = 2048, pi =   3.1412588724, error = 1.06e-04, time (sec) =   2.7038
Trials = 10000000000, Threads = 4096, pi =   3.1407169720, error = 2.79e-04, time (sec) =   2.7244
Trials = 10000000000, Threads = 8192, pi =   3.1412633928, error = 1.05e-04, time (sec) =   2.7719
```

2.1. (5 points) In this case, what value of p minimizes the parallel runtime?

Solution: As observed from the simulation results, we can observe that at '**P = 1024'**, we have the least parallel runtime.

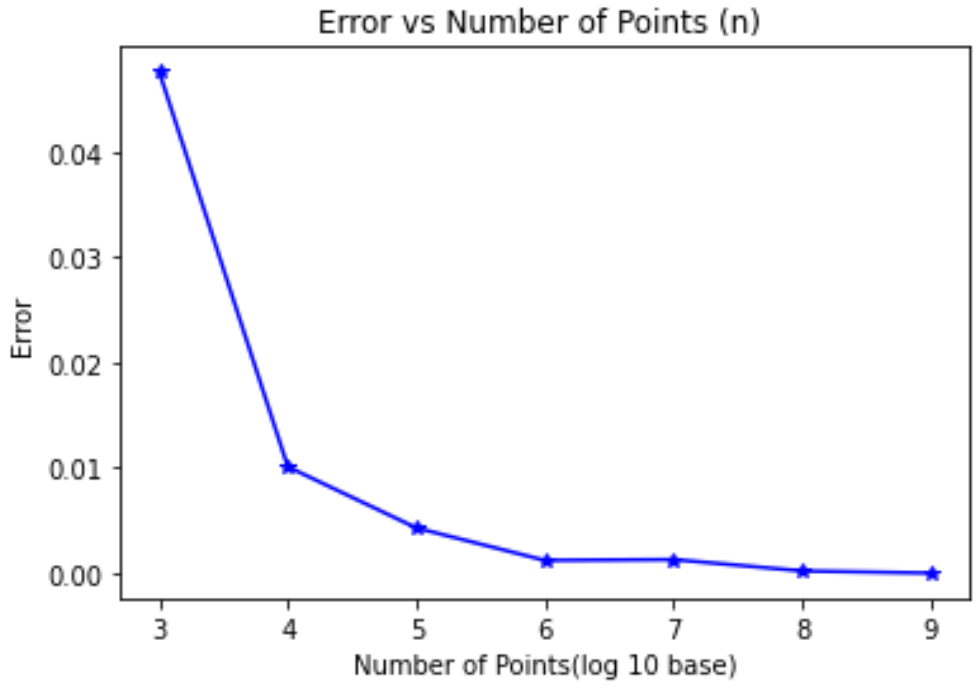
2.2. (5 points) Do you expect the runtime to increase as p is increased beyond a certain value? If so, why? And is this observed in your experiments?

Solution: From the experimental results in the homework, it can be observed that the runtime is decreased until p = 1024. Beyond **p = 1024**, we can see that the runtime increases. Beyond a certain value of p, runtime increases due to context switching. When there are a huge number of threads, the current state of each thread should be stored while switching to the next thread which results in a lot of overhead for switching. This results in an increase in runtime.

3. (5 points) Do you expect that there would be a difference in the number of threads needed to obtain the minimum execution time for two values of n? Is this observed in your experiments?

Solution: Yes, there would be a difference in the number of threads needed to obtain the minimum execution time for different values of n. This has been clearly observed in my experiments. For $n = 10^8$, the number of threads needed to obtain minimum time is 128 whereas for $n = 10^{10}$, the number of threads needed to obtain minimum execution time is 1024. Since the number of trails is increased, it would need more threads to process them parallelly, hence the difference is observed.

4. (5 points) Plot error versus n to illustrate the accuracy of the algorithm as a function of n. You may have to run experiments with different values of n; for example, n could be chosen to be 10^k, for k = 3, ..., 9. Use p = 48.



Part 2. Distributed-Memory Programming with MPI

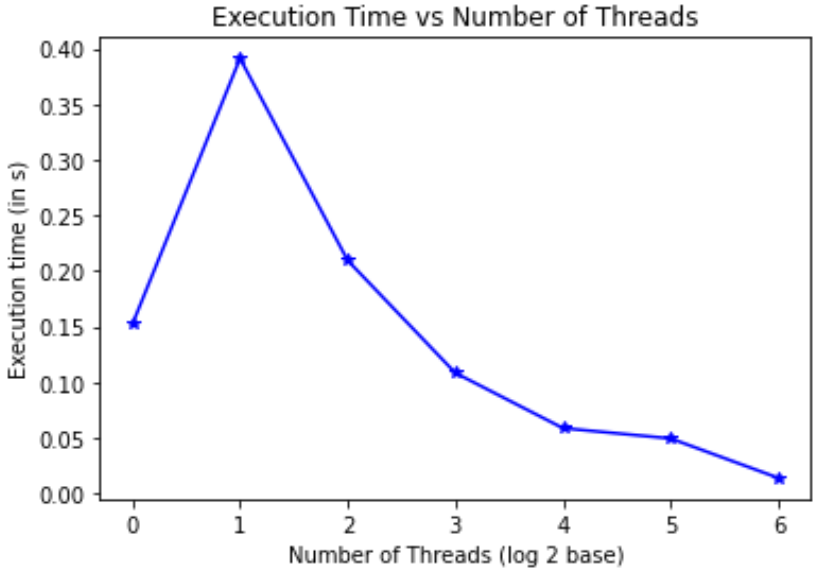
5. Execute the code for n=10^8 with p chosen to be 2^k, for k = 0, 1, ..., 6. Specify `ntasks-per- node=4` in the job file. Using the experimental data obtained from these experiments, answer the following questions. For plots, use a logarithmic scale for the x-axis.

Solution: Simulation Results:

```
Processes = 1
n = 100000000, p = 1, pi = 3.1415926535904264, relative error = 2.02e-13, time (sec) =   0.1519
Processes = 2
n = 100000000, p = 2, pi = 3.1415926535900223, relative error = 7.29e-14, time (sec) =   0.3892
Processes = 4
n = 100000000, p = 4, pi = 3.1415926535902168, relative error = 1.35e-13, time (sec) =   0.2131
Processes = 8
n = 100000000, p = 8, pi = 3.1415926535896137, relative error = 5.71e-14, time (sec) =   0.1106
Processes = 16
n = 100000000, p = 16, pi = 3.1415926535897754, relative error = 5.65e-15, time (sec) =   0.0590
Processes = 32
srun: Job 6271562 step creation temporarily disabled, retrying (Requested nodes are busy)
srun: Step created for job 6271562
n = 100000000, p = 32, pi = 3.1415926535897736, relative error = 6.22e-15, time (sec) =   0.0497
Processes = 64
n = 100000000, p = 64, pi = 3.1415926535897940, relative error = 2.83e-16, time (sec) =   0.0131
```

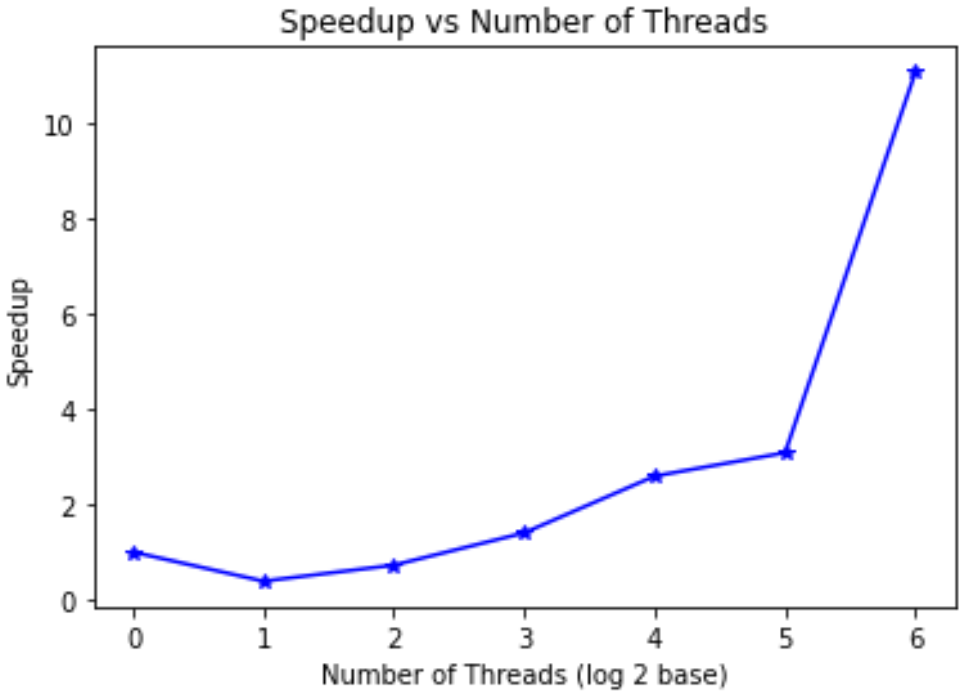
5.1. (10 points) Plot execution time versus p to demonstrate how time varies with the number of processes.

Solution:



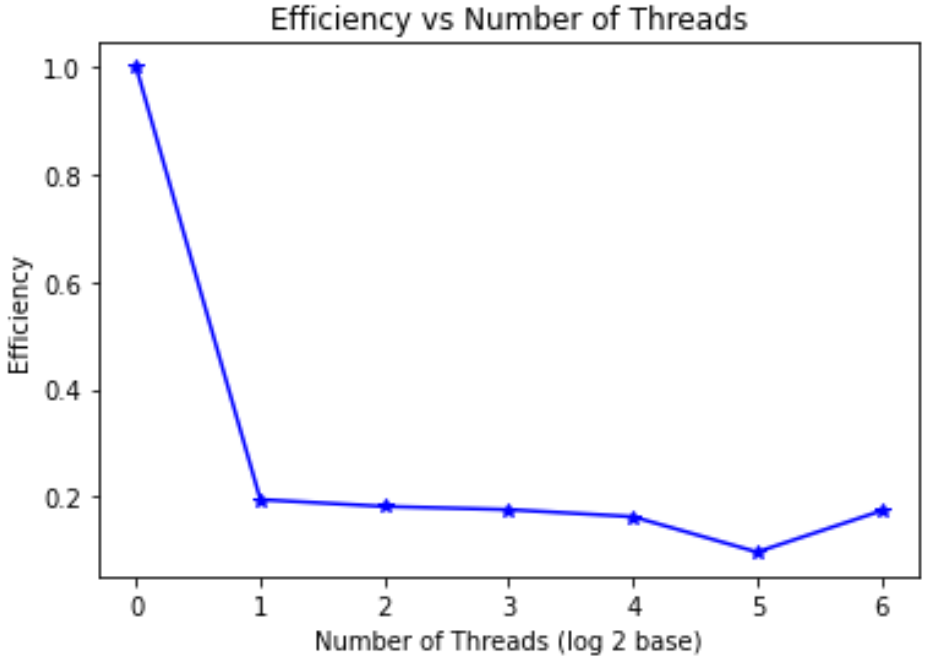
5.2. (10 points) Plot speedup versus p to demonstrate the change in speedup with p.

Solution:



5.3. (5 points) Using the definition: efficiency = speedup/p, plot efficiency versus p to demonstrate how efficiency changes as the number of processes are increased.

Solution:



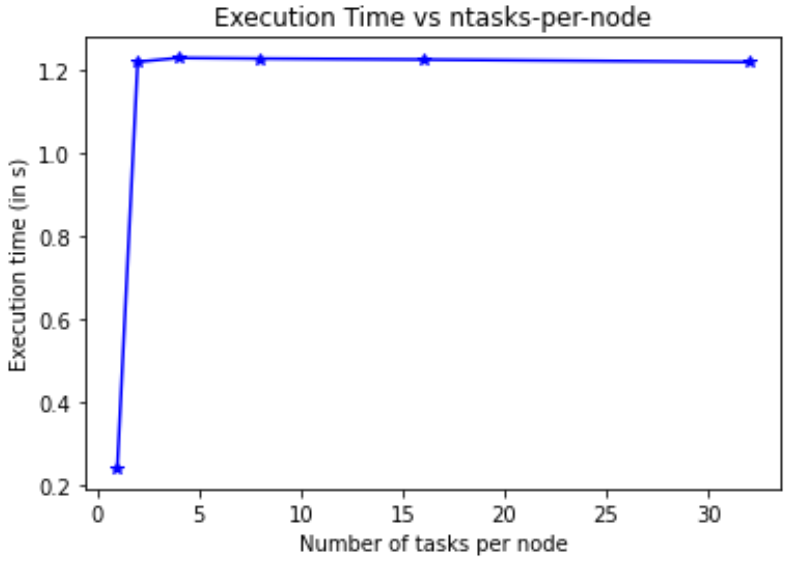
5.4. (5 points) What value of p minimizes the parallel runtime?

Solution: As observed from the results and plots, **P = 64** minimizes the parallel runtime.

6. (10 points) With n=10^{10} and p=64, determine the value of `ntasks-per-node` that minimizes the `total_time`. Plot time versus `ntasks-per-node` to illustrate your experimental results for this question.

Solution:

```
q2 - processes = 64
n = 10000000000, p = 64, pi = 3.1415926535897749, relative error =
5.80e-15, time (sec) =   0.2420
q2 - processes = 64
n = 10000000000, p = 64, pi = 3.1415926535897749, relative error =
5.80e-15, time (sec) =   1.2181
q2 - processes = 64
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error =
5.94e-15, time (sec) =   1.2275
q2 - processes = 64
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error =
5.94e-15, time (sec) =   1.2259
q2 - processes = 64
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error =
5.94e-15, time (sec) =   1.2236
q2 - processes = 64
n = 10000000000, p = 64, pi = 3.1415926535897749, relative error =
5.80e-15, time (sec) =   1.2175
```



From the results, it can be observed that the ntasks-per-node = 1minimizes the run time. Here when each processor gets a single task, it takes less runtime whereas the execution times were almost similar for the rest of ntask-per-node = 2, 4, 8, 16, 32.

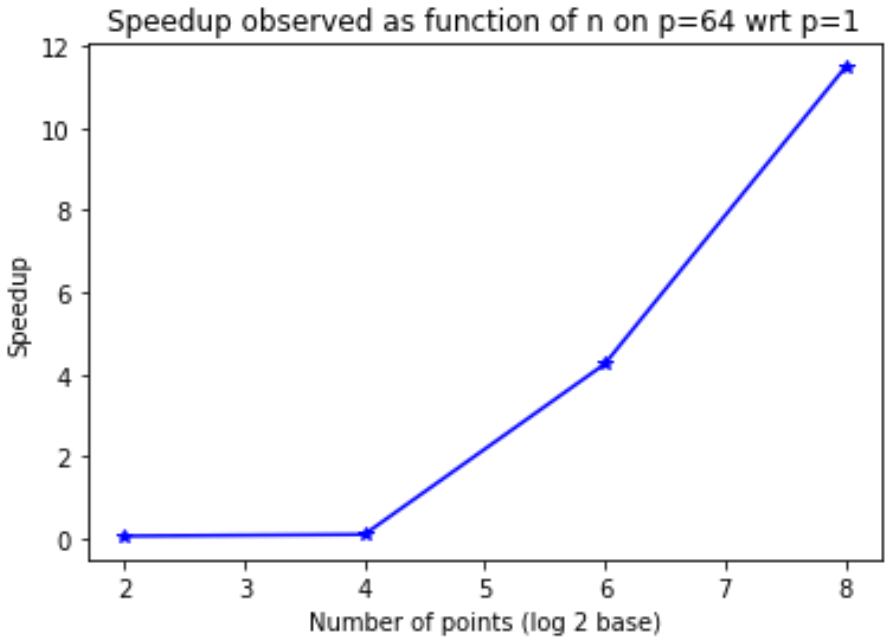
7. Execute the code with p=64 for n=10^2, 10^4, 10^6 and 10^8, with `ntasks-per-node=4`.

Solution: Simulation results:

```
Processes = 64
n = 100, p = 64, pi = 3.1416009869231249, relative error = 2.65e-06, time (sec) =   0.0023
Processes = 64
n = 10000, p = 64, pi = 3.1415926544231265, relative error = 2.65e-10, time (sec) =   0.0008
Processes = 64
n = 1000000, p = 64, pi = 3.1415926535898753, relative error = 2.62e-14, time (sec) =   0.0009
Processes = 64
n = 100000000, p = 64, pi = 3.1415926535897940, relative error = 2.83e-16, time (sec) =   0.0135
Processes = 1
n = 100, p = 1, pi = 3.1416009869231254, relative error = 2.65e-06, time (sec) =   0.0001
Processes = 1
n = 10000, p = 1, pi = 3.1415926544231341, relative error = 2.65e-10, time (sec) =   0.0001
Processes = 1
n = 1000000, p = 1, pi = 3.1415926535897643, relative error = 9.19e-15, time (sec) =   0.0016
Processes = 1
n = 100000000, p = 1, pi = 3.1415926535904264, relative error = 2.02e-13, time (sec) =   0.1521
```

7.1. (5 points) Plot the speedup observed as a function of n on p=64 w.r.t. p=1. You will need to obtain execution time on p=1 for n=10^2, 10^4, 10^6 and 10^8.

Solution:



7.2. (5 points) Plot the relative error versus n to illustrate the accuracy of the algorithm as a function of n.

Solution: