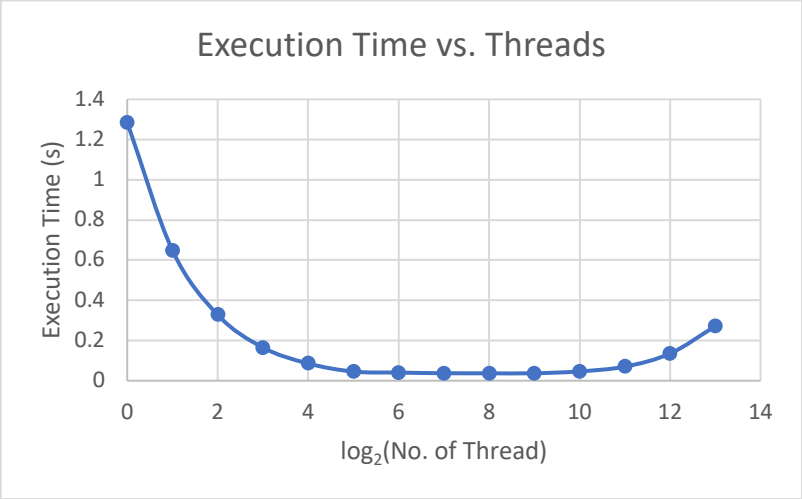
CSCE 735 Parallel Computing – HW1

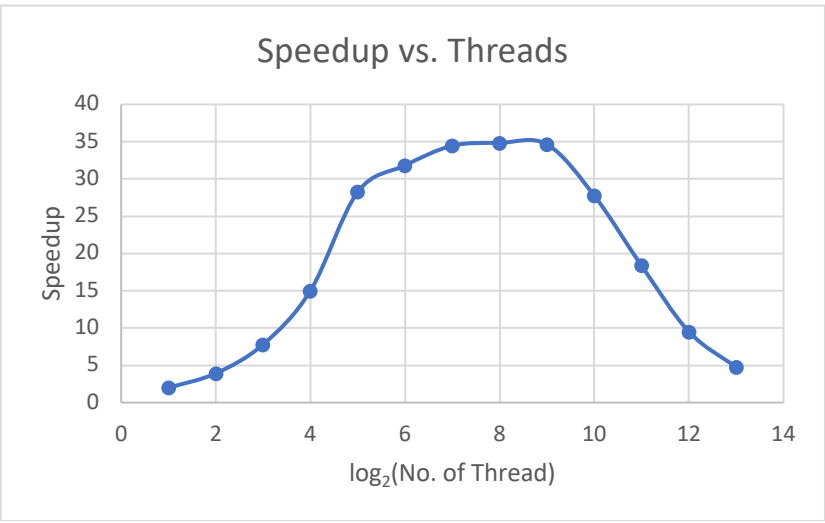
832001192 Chun-Sheng Wu

1.1 Plot execution time versus p to demonstrate how time varies with the number of threads.



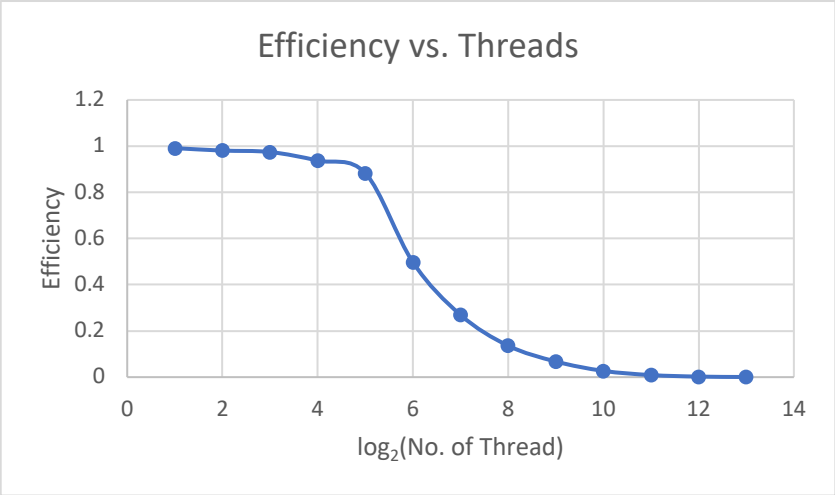
Above graph exhibits the behavior of execution time while number of thread is increasing. From the trend, we can observe that the execution time drops following raised threads, which means each thread is assigned tasks and worked in parallel. This trend becomes flat when approaching 256 threads and execution time increases subsequently with more threads involved in the execution. Theoretically, the execution time should be reduced, but it gradually costs more time out of the expectation. Guess that it is because this task is a light-loading one that do not require much resource to perform. Over-revoking in return causes additional time to manipulate the setting that consumes huge time.

1.2 Plot Speedup versus p to demonstrate the change in speedup with p.



Speedup is in similar pattern to the graph of execution time. They all reach the peak value with 256 threads and goes oppositely after that point.

1.3 Plot efficiency versus p to demonstrate how efficiency changes as p increased.



Efficiency reaches near 1 if speedup follows with more threads. Since efficiency is speedup dividing number of thread, efficiency drops significantly with increasing p. In this case can we infer that those threads below 16 obtains better performance than threads above it.

1.4 In your experiment, what value of p minimizes the parallel runtime?

Operating this task with 256 threads (p=256) fetches the minimum runtime in this experiment. However, the time reduction is not comparable with increase of hardware resource (threads). Although 256 threads can give us the least runtime, excessive threads do not have expected efficiency in tradeoff.

```
Trials = 100000000, Threads =    1, pi =   3.1416149600, error = 7.10e-06, time (sec) =   1.2855
Trials = 100000000, Threads =    2, pi =   3.1417022800, error = 3.49e-05, time (sec) =   0.6484
Trials = 100000000, Threads =    4, pi =   3.1415910800, error = 5.01e-07, time (sec) =   0.3277
Trials = 100000000, Threads =    8, pi =   3.1415947600, error = 6.70e-07, time (sec) =   0.1649
Trials = 100000000, Threads =   16, pi =   3.1415291200, error = 2.02e-05, time (sec) =   0.0857
Trials = 100000000, Threads =   32, pi =   3.1415901200, error = 8.06e-07, time (sec) =   0.0455
Trials = 100000000, Threads =   64, pi =   3.1413512400, error = 7.68e-05, time (sec) =   0.0404
Trials = 100000000, Threads =  128, pi =   3.1429299200, error = 4.26e-04, time (sec) =   0.0373
Trials = 100000000, Threads =  256, pi =   3.1412995200, error = 9.33e-05, time (sec) =   0.0369
Trials = 100000000, Threads =  512, pi =   3.1468450800, error = 1.67e-03, time (sec) =   0.0371
Trials = 100000000, Threads = 1024, pi =   3.1514026000, error = 3.12e-03, time (sec) =   0.0463
Trials = 100000000, Threads = 2048, pi =   3.1453611600, error = 1.20e-03, time (sec) =   0.0700
Trials = 100000000, Threads = 4096, pi =   3.1494162400, error = 2.49e-03, time (sec) =   0.1351
Trials = 100000000, Threads = 8192, pi =   3.1377031200, error = 1.24e-03, time (sec) =   0.2713
```
-> Execution status in 1.

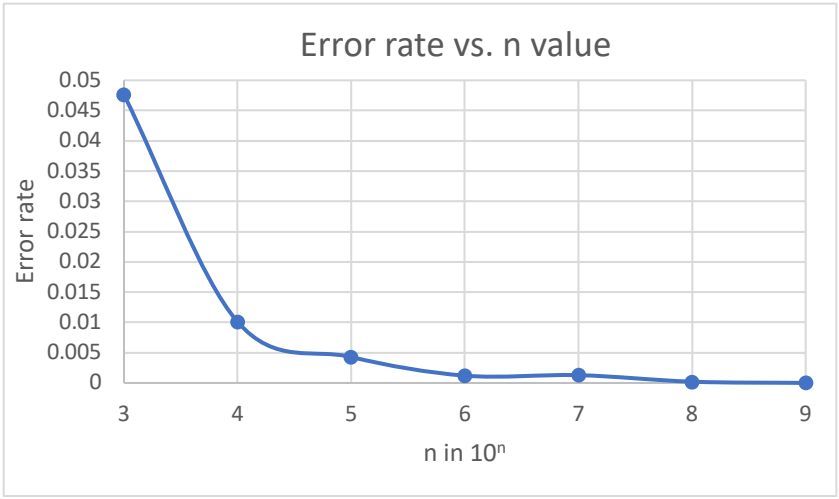
2.1 What value of p minimizes the parallel runtime?
Operating task with 1024 threads obtains the least runtime. The execution time slightly increases with more threads (2048, 4096, and 8192).

2.2 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?
Yes, it is convincing that the runtime will keep increasing as p is increased beyond a certain value. The execution time achieves its least time when performing with 1024 threads and raises with threads more than 1024. I guess that it is due to resource overcall. With the aspect to balance the execution and hardware, over-revoking in which side is expected to cause overhead. This situation is also shown in my experiment (graph below).

```
Trials = 10000000000, Threads =    1, pi =   1.4236202260, error = 5.47e-01, time (sec) = 127.8989
Trials = 10000000000, Threads =    2, pi =   3.1416070092, error = 4.57e-06, time (sec) =  64.0386
Trials = 10000000000, Threads =    4, pi =   3.1416060276, error = 4.26e-06, time (sec) =  32.0409
Trials = 10000000000, Threads =    8, pi =   3.1416118228, error = 6.10e-06, time (sec) =  16.0348
Trials = 10000000000, Threads =   16, pi =   3.1416066896, error = 4.47e-06, time (sec) =   8.0267
Trials = 10000000000, Threads =   32, pi =   3.1416332068, error = 1.29e-05, time (sec) =   4.0190
Trials = 10000000000, Threads =   64, pi =   3.1416139092, error = 6.77e-06, time (sec) =   3.1499
Trials = 10000000000, Threads =  128, pi =   3.1415943424, error = 5.38e-07, time (sec) =   2.7869
Trials = 10000000000, Threads =  256, pi =   3.1416473120, error = 1.74e-05, time (sec) =   2.7074
Trials = 10000000000, Threads =  512, pi =   3.1415762012, error = 5.24e-06, time (sec) =   2.6980
Trials = 10000000000, Threads = 1024, pi =   3.1414319268, error = 5.12e-05, time (sec) =   2.6931
Trials = 10000000000, Threads = 2048, pi =   3.1412588724, error = 1.06e-04, time (sec) =   2.7048
Trials = 10000000000, Threads = 4096, pi =   3.1407169720, error = 2.79e-04, time (sec) =   2.7225
Trials = 10000000000, Threads = 8192, pi =   3.1412633928, error = 1.05e-04, time (sec) =   2.7725
```
-> Execution status in 2.

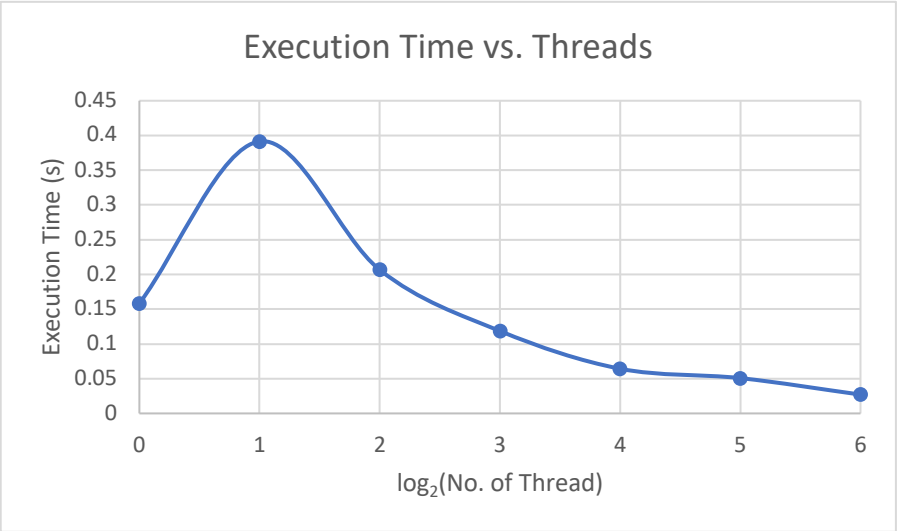
3.1 Do you expect that there would be a difference in the number of threads needed to obtain the minimum execution time for two values of n? Is this observed in your experiment?
Yes, I expect that the number of threads to acquire the minimum runtime for two different n. In the experiment, we have p = 256 to obtain the best execution time in 1., and p = 1024 in 2. It shows that different task loading (n) has corresponding threads (p) to have better exploitation and obtain the minimum execution time, which also exhibits in my experiment.

4.1 Plot error vs. n to illustrate accuracy of the algorithm.



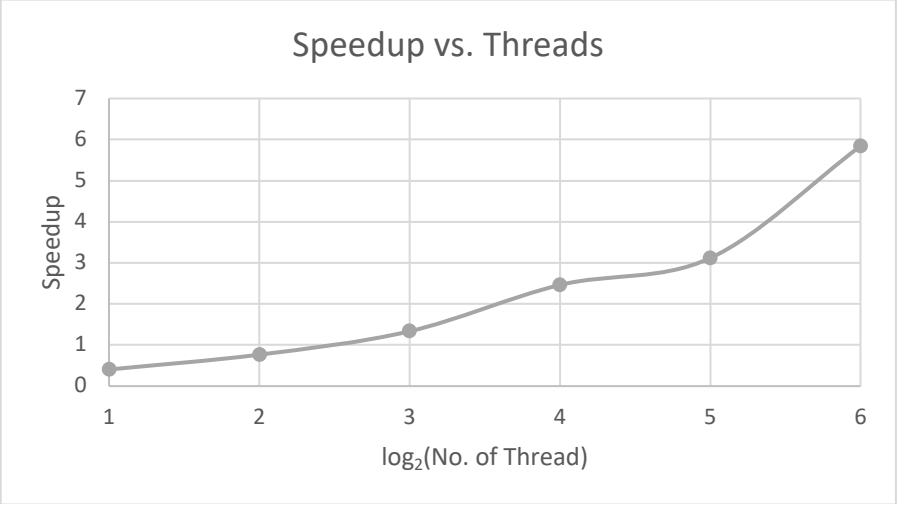
Error rate generally drops with increasing n. This implies that higher bits can perform more accurate values.

5.1 Plot execution time versus p to demonstrate how time varies with different p values.



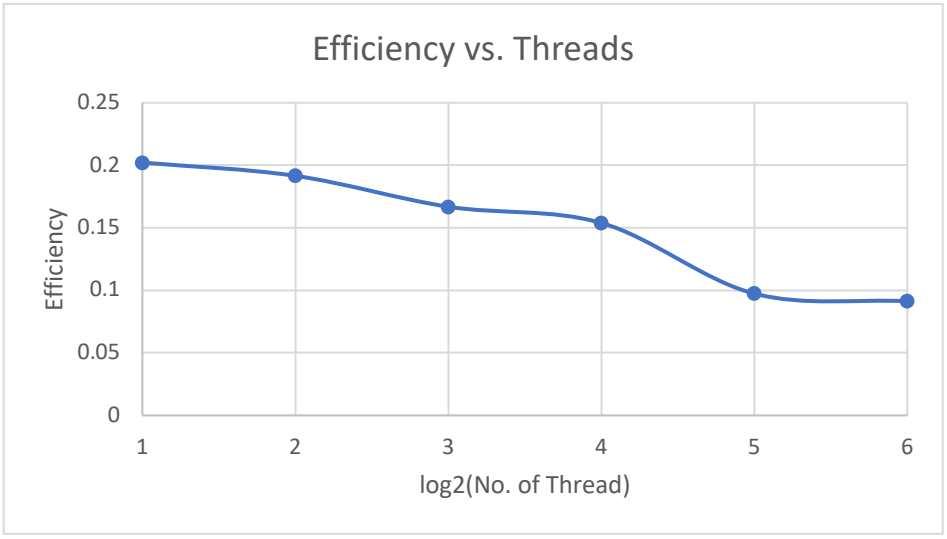
Execution time firstly increased when applying 2 threads to handle the task, and goes down with more threads involved. This trend is not like the task performing on the shared memory machine that runtime decreases at first and increases after reaching a optimal point (thread).

5.2 Plot speedup versus p to demonstrate the change in speedup with p.



The speedup value keeps going upward when we assign more and more threads handling this instruction. According to the trend pattern, probably speedup could raise with threads more than 64.

5.3 Plot efficiency versus p to demonstrate how efficiency changes as p increased.

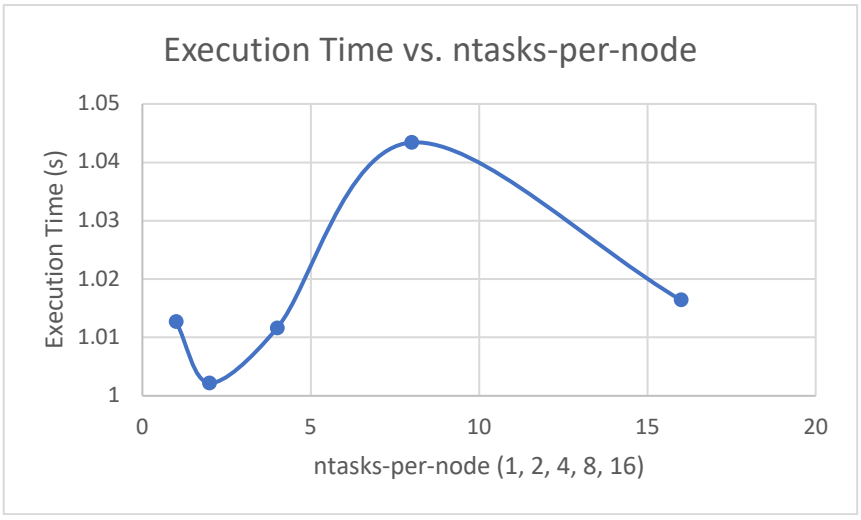


Contrast to speedup, efficiency goes down straight in the graph. But it will seemingly approach to a specific point (0.1 in this experiment) and becomes flat.

5.4 What value of p minimizes the parallel runtime?

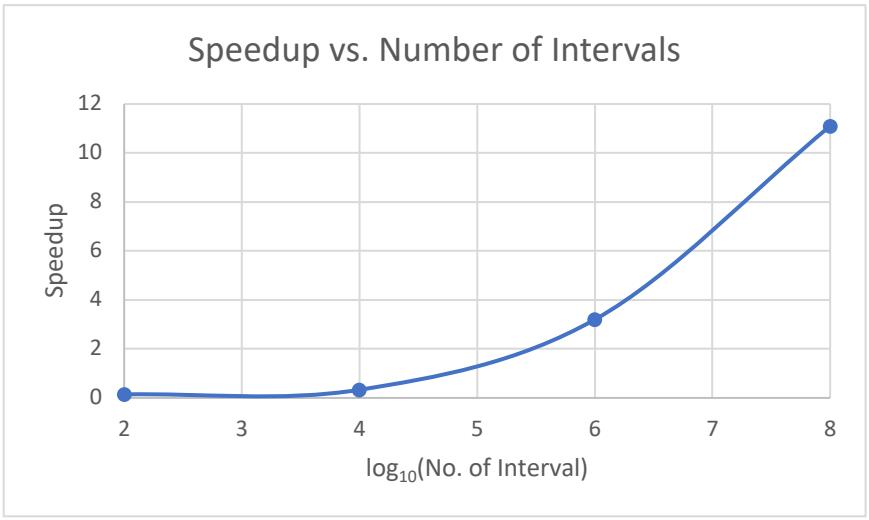
Operating the task with 64 threads (p=64) minimizes the parallel runtime in my experiment. Infer that it is due to distributed machine requires much more information communication while fetching data and calculating values than shared machine. More threads could facilitate the procedure of examining memory address across the machine, therefore reducing the execution time as they could bring back desired values in parallel.

6. Plot time versus ntasks-per-node to illustrate your experimental results for this question.



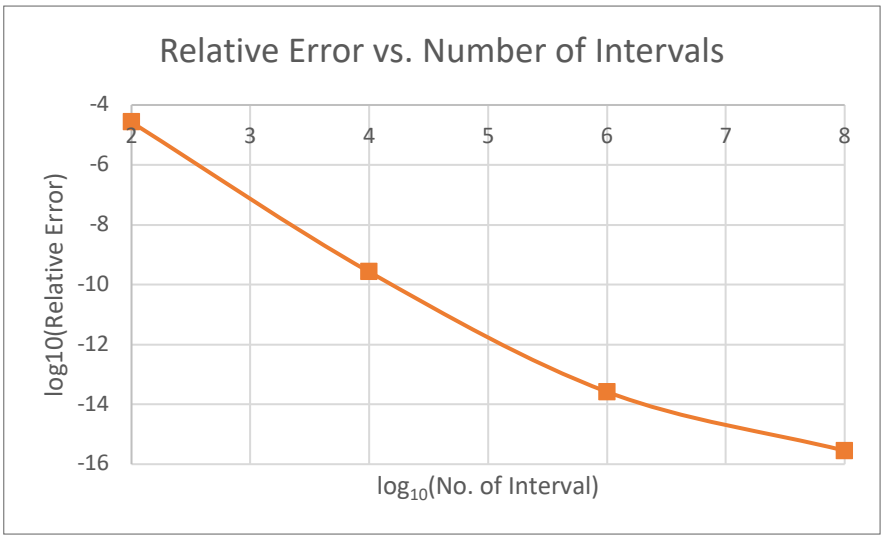
Since we request node for 16, I did the experiment with ntasks-per-node of 1, 2, 4, 8, and 16 to make the task could be assigned fairly. We could observe that it obtains the minimum runtime with 2 tasks per node. This might imply that distributing tasks and splitting it into pieces, which are balance for each node, are able to accelerate implementation.

7.1 Plot the speedup observed as a function of n on p=64 w.r.t. p=1.



The graph demonstrates that the more the computing loading is, the more speedup we obtain with fixed threads. From the speedup formula ($S_p = \frac{p}{sp+(1-s)}$), more complicated instruction could better exploit the assigning hardware resources, e.g., threads and processing units. That leads to higher speedup when we applied identical threads (p) in this experiment.

7.2 Plot the relative error versus n to illustrate the accuracy with various n.



Same to the error rate in shared machine, relative error reduces with more n is involved in the computation. The result could be closer to the ground truth value due to more digits rounding to attain possible answers.

```
Processes = 1, n=10**8
n = 100000000, p = 1, pi = 3.1415926535904264, relative error = 2.02e-13, time (sec) =   0.1583

Processes = 2, n=10**8
n = 100000000, p = 2, pi = 3.1415926535900223, relative error = 7.29e-14, time (sec) =   0.3915

Processes = 4, n=10**8
n = 100000000, p = 4, pi = 3.1415926535902168, relative error = 1.35e-13, time (sec) =   0.2066

Processes = 8, n=10**8
n = 100000000, p = 8, pi = 3.1415926535896137, relative error = 5.71e-14, time (sec) =   0.1186

Processes = 16, n=10**8
n = 100000000, p = 16, pi = 3.1415926535897754, relative error = 5.65e-15, time (sec) =   0.0643

Processes = 32, n=10**8
n = 100000000, p = 32, pi = 3.1415926535897736, relative error = 6.22e-15, time (sec) =   0.0508

Processes = 64, n=10**8
n = 100000000, p = 64, pi = 3.1415926535897940, relative error = 2.83e-16, time (sec) =   0.0271

Processes = 64, n=10**10
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error = 5.94e-15, time (sec) =   1.2177

Processes = 64, n=10**8
n = 100000000, p = 64, pi = 3.1415926535897940, relative error = 2.83e-16, time (sec) =   0.0137

Processes = 64, n=10**6
n = 1000000, p = 64, pi = 3.1415926535898753, relative error = 2.62e-14, time (sec) =   0.0005

Processes = 64, n=10**4
n = 10000, p = 64, pi = 3.1415926544231265, relative error = 2.65e-10, time (sec) =   0.0003

Processes = 64, n=10**2
n = 100, p = 64, pi = 3.1416009869231249, relative error = 2.65e-06, time (sec) =   0.0007

Processes = 1, n=10**8
n = 100000000, p = 1, pi = 3.1415926535904264, relative error = 2.02e-13, time (sec) =   0.1521

Processes = 1, n=10**6
n = 1000000, p = 1, pi = 3.1415926535897643, relative error = 9.19e-15, time (sec) =   0.0016

Processes = 1, n=10**4
n = 10000, p = 1, pi = 3.1415926544231341, relative error = 2.65e-10, time (sec) =   0.0001

Processes = 1, n=10**2
n = 100, p = 1, pi = 3.1416009869231254, relative error = 2.65e-06, time (sec) =   0.0001

Processes = 64, n=10**10, ntasks-per-node = 1
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error = 5.94e-15, time (sec) =   1.0127

Processes = 64, n=10**10, ntasks-per-node = 2
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error = 5.94e-15, time (sec) =   1.0022

Processes = 64, n=10**10, ntasks-per-node = 4
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error = 5.94e-15, time (sec) =   1.0116

Processes = 64, n=10**10, ntasks-per-node = 8
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error = 5.94e-15, time (sec) =   1.0434

Processes = 64, n=10**10, ntasks-per-node = 16
n = 10000000000, p = 64, pi = 3.1415926535897745, relative error = 5.94e-15, time (sec) =   1.0164
```

-> Execution Result on distributed machine