CSCE 735 Fall 2023
Name: Ashutosh Chauhan
UIN: 232009024

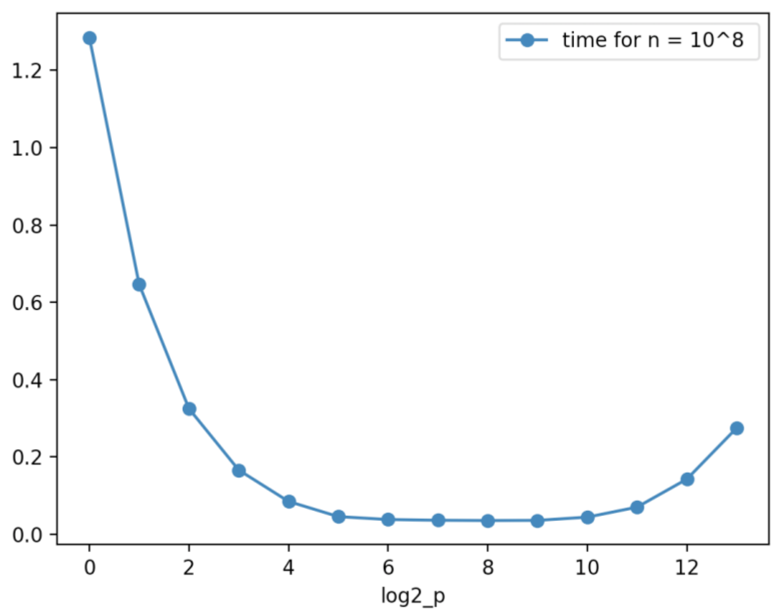
HW 1: Parallel Programming on a Multicore Multiprocessor

Part 1. Shared-Memory Programming with Threads

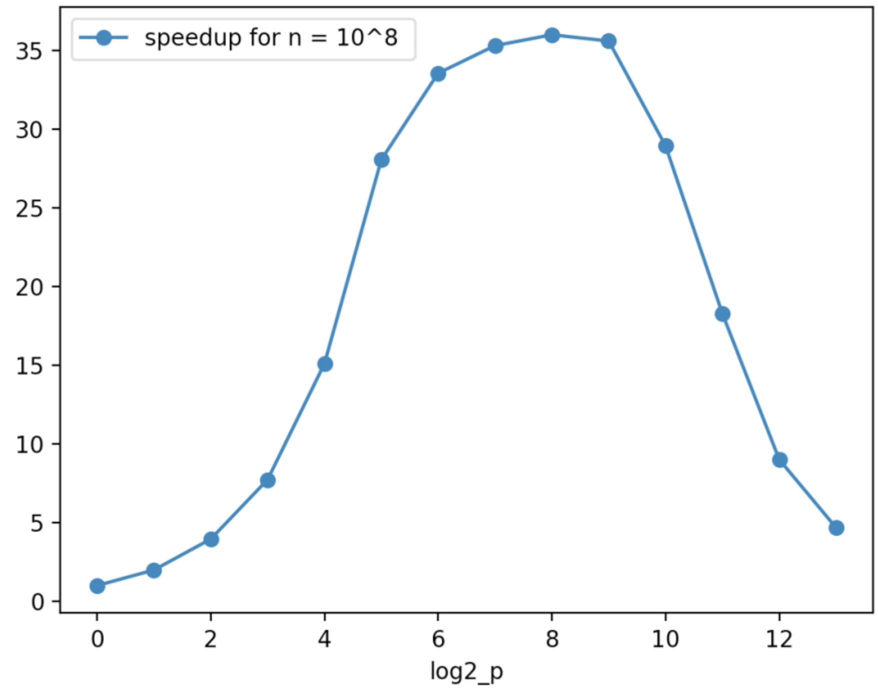
1. Execute the code for n=10^8 with p chosen to be 2^k, for k = 0, 1, …, 13. Using the experimental data obtained from these experiments, answer the following questions. For plots, use a logarithmic scale for the x-axis.

	Trials	p	pi	error	time	speedup	efficiency	log2_p
0	100000000	1	3.141615	7.100000e-06	1.2845	1.000000	1.000000	0.0
1	100000000	2	3.141702	3.490000e-05	0.6462	1.987775	0.993887	1.0
2	100000000	4	3.141591	5.010000e-07	0.3249	3.953524	0.988381	2.0
3	100000000	8	3.141595	6.700000e-07	0.1662	7.728640	0.966080	3.0
4	100000000	16	3.141529	2.020000e-05	0.0851	15.094007	0.943375	4.0
5	100000000	32	3.141590	8.060000e-07	0.0458	28.045852	0.876433	5.0
6	100000000	64	3.141351	7.680000e-05	0.0383	33.537859	0.524029	6.0
7	100000000	128	3.142930	4.260000e-04	0.0364	35.288462	0.275691	7.0
8	100000000	256	3.141300	9.330000e-05	0.0357	35.980392	0.140548	8.0
9	100000000	512	3.146845	1.670000e-03	0.0361	35.581717	0.069496	9.0
10	100000000	1024	3.151403	3.120000e-03	0.0444	28.930180	0.028252	10.0
11	100000000	2048	3.145361	1.200000e-03	0.0703	18.271693	0.008922	11.0
12	100000000	4096	3.149416	2.490000e-03	0.1427	9.001402	0.002198	12.0
13	100000000	8192	3.137703	1.240000e-03	0.2748	4.674309	0.000571	13.0

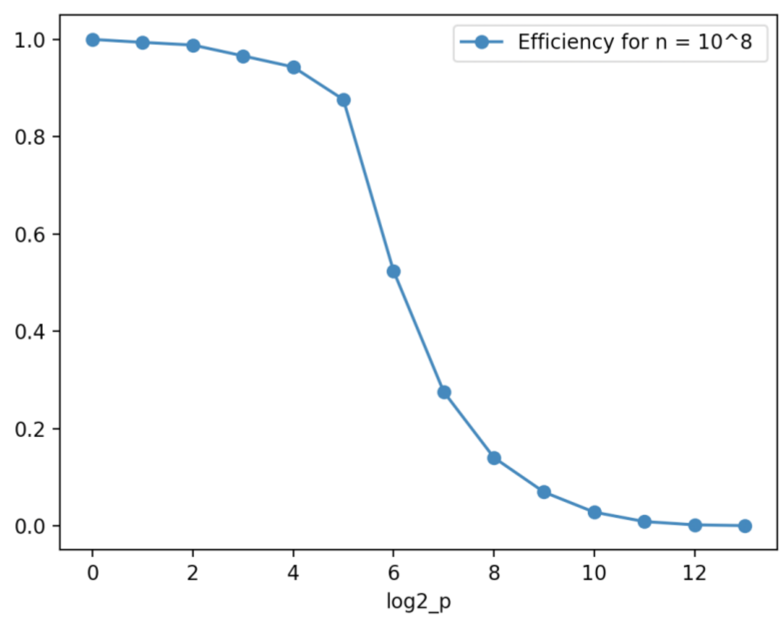
1.1. (10 points) Plot execution time versus p to demonstrate how time varies with the number of threads.



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



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.



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

	Trials	p	pi	error	time	speedup	efficiency	log2_p
0	100000000	1	3.141615	7.100000e-06	1.2845	1.000000	1.000000	0.0
1	100000000	2	3.141702	3.490000e-05	0.6462	1.987775	0.993887	1.0
2	100000000	4	3.141591	5.010000e-07	0.3249	3.953524	0.988381	2.0
3	100000000	8	3.141595	6.700000e-07	0.1662	7.728640	0.966080	3.0
4	100000000	16	3.141529	2.020000e-05	0.0851	15.094007	0.943375	4.0
5	100000000	32	3.141590	8.060000e-07	0.0458	28.045852	0.876433	5.0
6	100000000	64	3.141351	7.680000e-05	0.0383	33.537859	0.524029	6.0
7	100000000	128	3.142930	4.260000e-04	0.0364	35.288462	0.275691	7.0
8	100000000	256	3.141300	9.330000e-05	0.0357	35.980392	0.140548	8.0
9	100000000	512	3.146845	1.670000e-03	0.0361	35.581717	0.069496	9.0
10	100000000	1024	3.151403	3.120000e-03	0.0444	28.930180	0.028252	10.0
11	100000000	2048	3.145361	1.200000e-03	0.0703	18.271693	0.008922	11.0
12	100000000	4096	3.149416	2.490000e-03	0.1427	9.001402	0.002198	12.0
13	100000000	8192	3.137703	1.240000e-03	0.2748	4.674309	0.000571	13.0

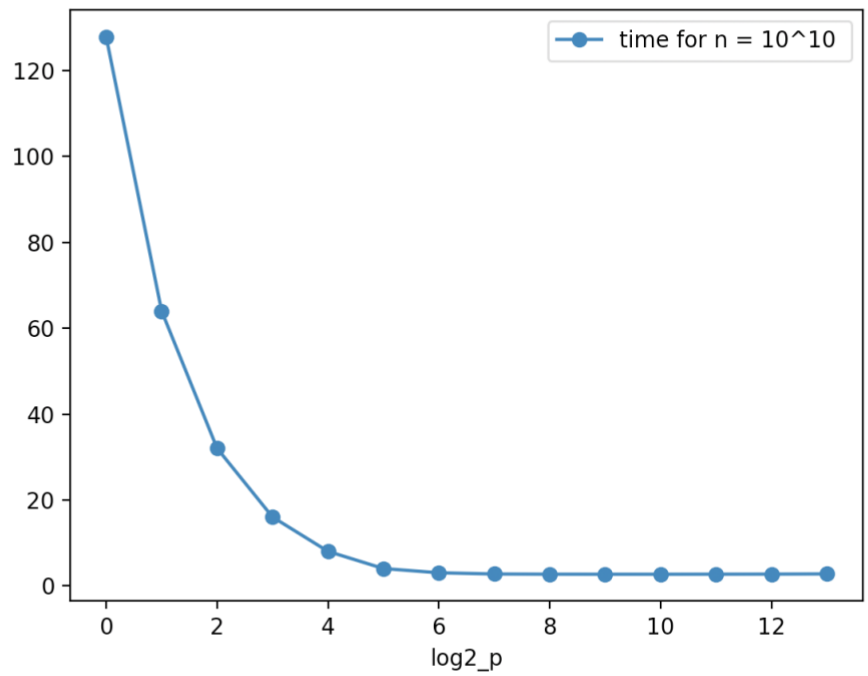
We can observe minimum parallel runtime at p = 256 when n = 100,000,000.

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

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

We can observe a minimum parallel runtime at p = 1024 when n = 10,000,000,000.

	Trials	p	pi	error	time	speedup	efficiency	log2_p
14	10000000000	1	1.423620	5.470000e-01	127.8794	1.000000	1.000000	0.0
15	10000000000	2	3.141607	4.570000e-06	64.0254	1.997323	0.998661	1.0
16	10000000000	4	3.141606	4.260000e-06	32.0342	3.991965	0.997991	2.0
17	10000000000	8	3.141612	6.100000e-06	16.0278	7.978600	0.997325	3.0
18	10000000000	16	3.141607	4.470000e-06	8.0252	15.934731	0.995921	4.0
19	10000000000	32	3.141633	1.290000e-05	4.0208	31.804467	0.993890	5.0
20	10000000000	64	3.141614	6.770000e-06	3.0316	42.182148	0.659096	6.0
21	10000000000	128	3.141594	5.380000e-07	2.7506	46.491456	0.363215	7.0
22	10000000000	256	3.141647	1.740000e-05	2.7065	47.248993	0.184566	8.0
23	10000000000	512	3.141576	5.240000e-06	2.6940	47.468226	0.092711	9.0
24	10000000000	1024	3.141432	5.120000e-05	2.6935	47.477037	0.046364	10.0
25	10000000000	2048	3.141259	1.060000e-04	2.7069	47.242011	0.023067	11.0
26	10000000000	4096	3.140717	2.790000e-04	2.7237	46.950619	0.011463	12.0
27	10000000000	8192	3.141263	1.050000e-04	2.7718	46.135868	0.005632	13.0



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.

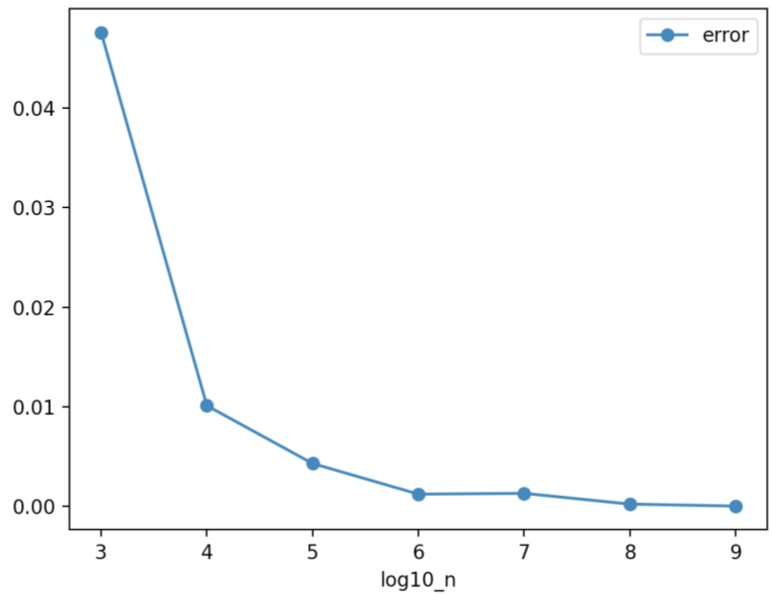
Yes, the runtime will increase as p is increased beyond a certain value. This is because every thread requires some system resources like stack, memory etc. for context switching. As we increase the number of threads the overhead associated with managing and switching between threads also increases. This outweighs the benefits that we get because of multiple threads after a certain value. Hence we see this behavior.
Yes, same behavior is observed in my experiment also.

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.

Yes, from the data we can observe that for n=10^8 the minimum time was observed at p = 256 whereas for n = 10^10 the minimum time was observed at p = 1024. For more number of trials the minimum time is observed at higher value of p. This is because with increase in number of trials the workload per thread increases due to which it takes more time.
In above case when n = 10^8 and p = 256, the workload at every process was around 10^8/256. But at n = 10^10 and p = 256 the load at every process will be around 10^10/256. Due to this it will take more time. We observed a minimum time at n = 10^10 when we further increase the threads.
Yes, this can be clearly observed in my experiment.

4. (5 points) Plot error versus n to illustrate 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.

Trials	p	pi	error	time	speedup	efficiency	log2_p	log10_n
1000	48	2.992000	0.047600	0.0018	71044.111111	1480.085648	5.584963	3.0
10000	48	3.173200	0.010100	0.0017	75223.176471	1567.149510	5.584963	4.0
100000	48	3.155080	0.004290	0.0016	79924.625000	1665.096354	5.584963	5.0
1000000	48	3.145396	0.001210	0.0022	58127.000000	1210.979167	5.584963	6.0
10000000	48	3.145637	0.001290	0.0079	16187.265823	337.234705	5.584963	7.0
100000000	48	3.140940	0.000208	0.0317	4034.050473	84.042718	5.584963	8.0
1000000000	48	3.141618	0.000008	0.2836	450.914669	9.394056	5.584963	9.0

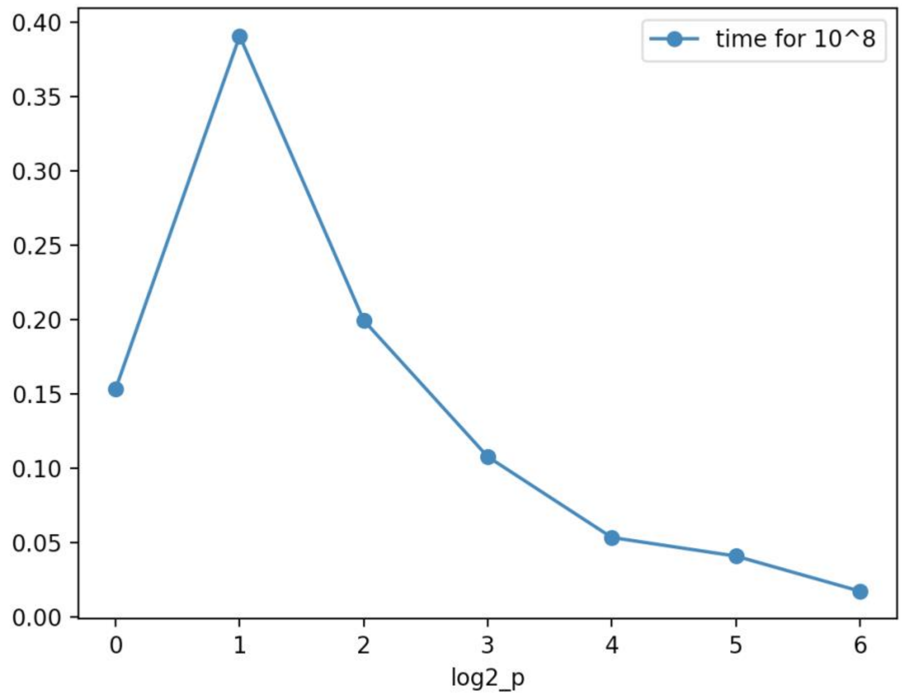


Part 2. Distributed-Memory Programming with MPI

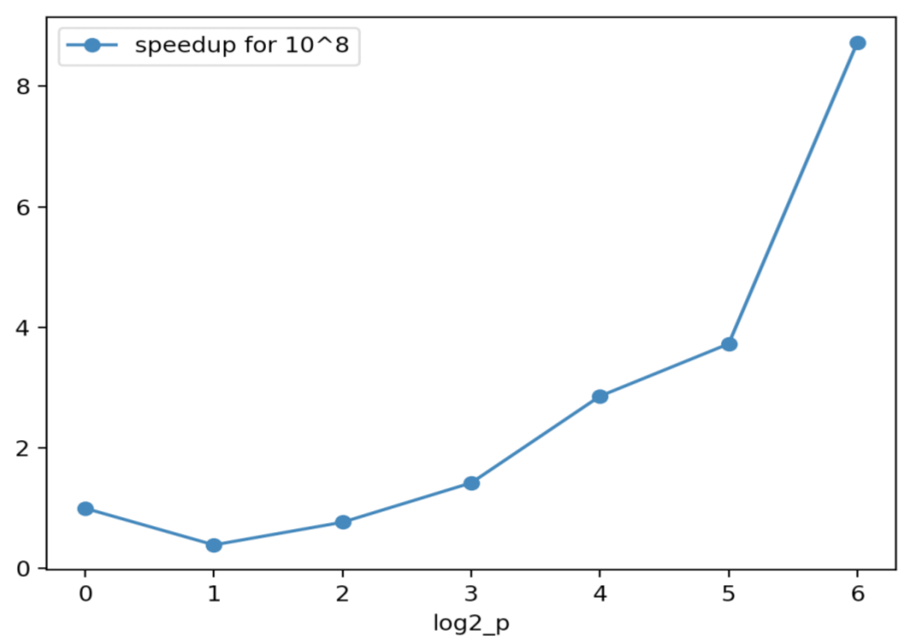
5. Execute the code for n=10_8 with p chosen to be 2_k, for k = 0, 1, …,_ _6. Specify **`ntasks-per-node=4`** in the job file. Using the experimental data obtained from these experiments, answer the following questions. For plots, use a logarithmic scale for the x-axis.

n	p	pi	error	time	speedup	efficiency	log2_p
100000000	1	3.141593	2.020000e-13	0.1536	1.000000	1.000000	0.0
100000000	2	3.141593	7.290000e-14	0.3909	0.392939	0.196470	1.0
100000000	4	3.141593	1.350000e-13	0.1997	0.769154	0.192288	2.0
100000000	8	3.141593	5.710000e-14	0.1080	1.422222	0.177778	3.0
100000000	16	3.141593	5.650000e-15	0.0537	2.860335	0.178771	4.0
100000000	32	3.141593	6.220000e-15	0.0412	3.728155	0.116505	5.0
100000000	64	3.141593	2.830000e-16	0.0176	8.727273	0.136364	6.0

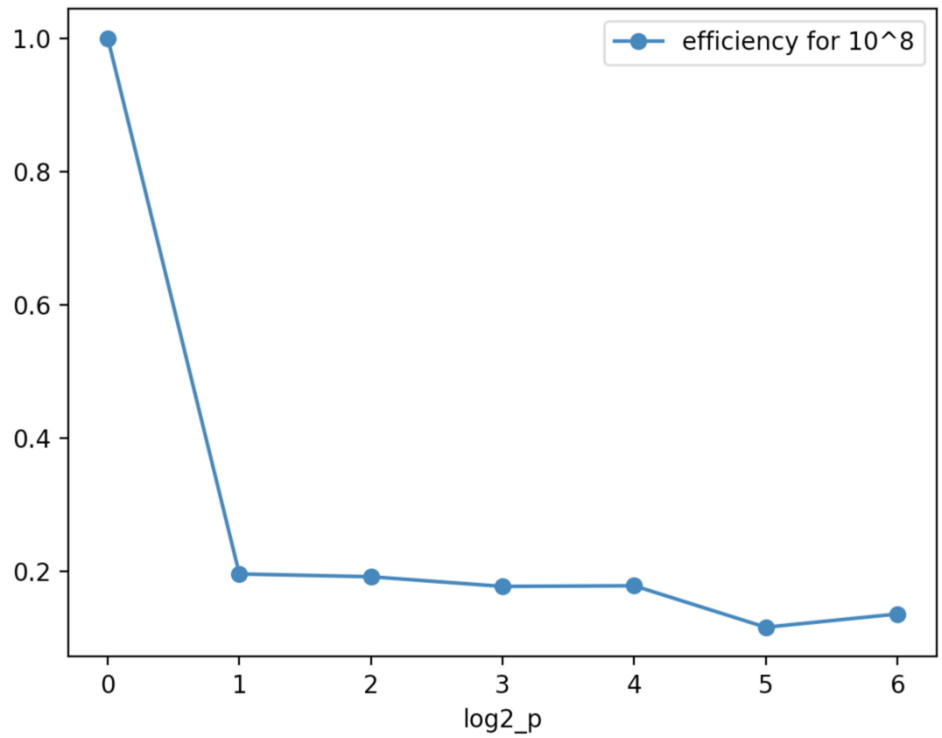
5.1. (10 points) Plot execution time versus p to demonstrate how time varies with the number of processes.



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



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



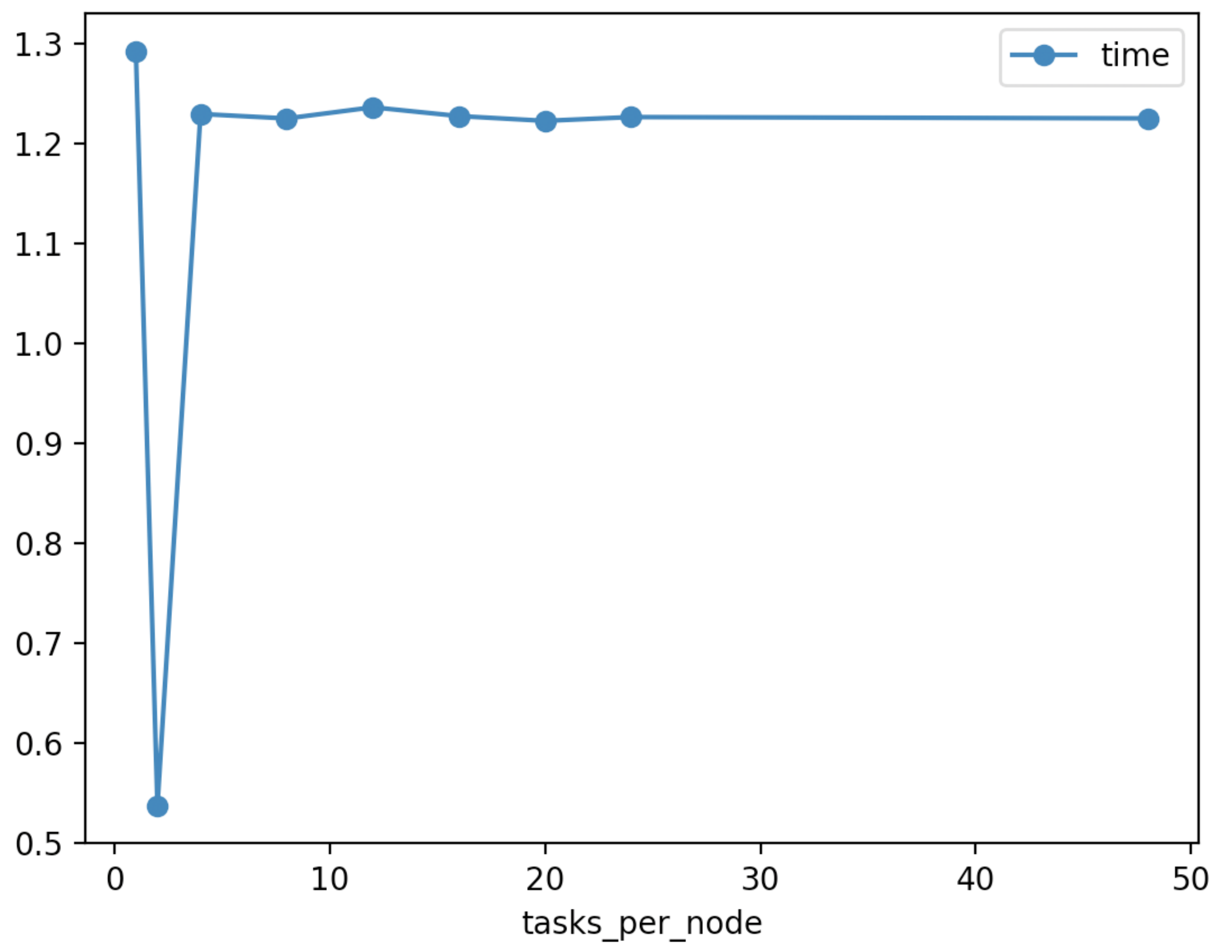
5.4. (5 points) What value of p minimizes the parallel runtime?

n	p	pi	error	time	speedup	efficiency	log2_p
100000000	1	3.141593	2.020000e-13	0.1536	1.000000	1.000000	0.0
100000000	2	3.141593	7.290000e-14	0.3909	0.392939	0.196470	1.0
100000000	4	3.141593	1.350000e-13	0.1997	0.769154	0.192288	2.0
100000000	8	3.141593	5.710000e-14	0.1080	1.422222	0.177778	3.0
100000000	16	3.141593	5.650000e-15	0.0537	2.860335	0.178771	4.0
100000000	32	3.141593	6.220000e-15	0.0412	3.728155	0.116505	5.0
100000000	64	3.141593	2.830000e-16	0.0176	8.727273	0.136364	6.0

We can observe a minimum parallel runtime at p = 64 with 16 nodes and ntask-per-node = 4 when n = 10,000,000,000.

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.

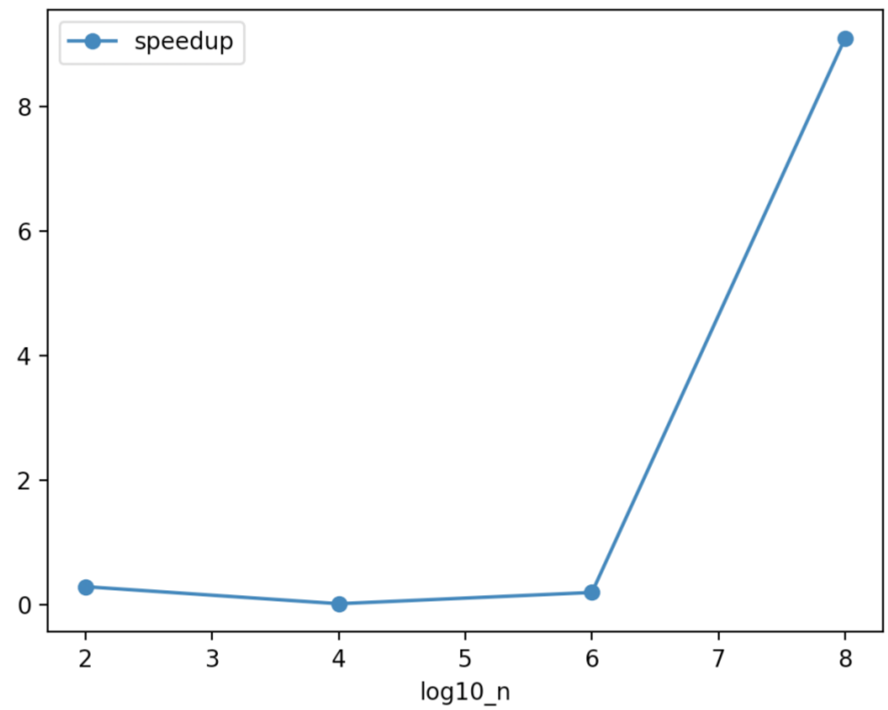
tasks_per_node	time
1	1.2923
2	0.5368
4	1.2295
8	1.2250
12	1.2362
16	1.2274
20	1.2226
24	1.2264
48	1.2251



At ntasks-per-node = 2 we get minimum total time.

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

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.



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