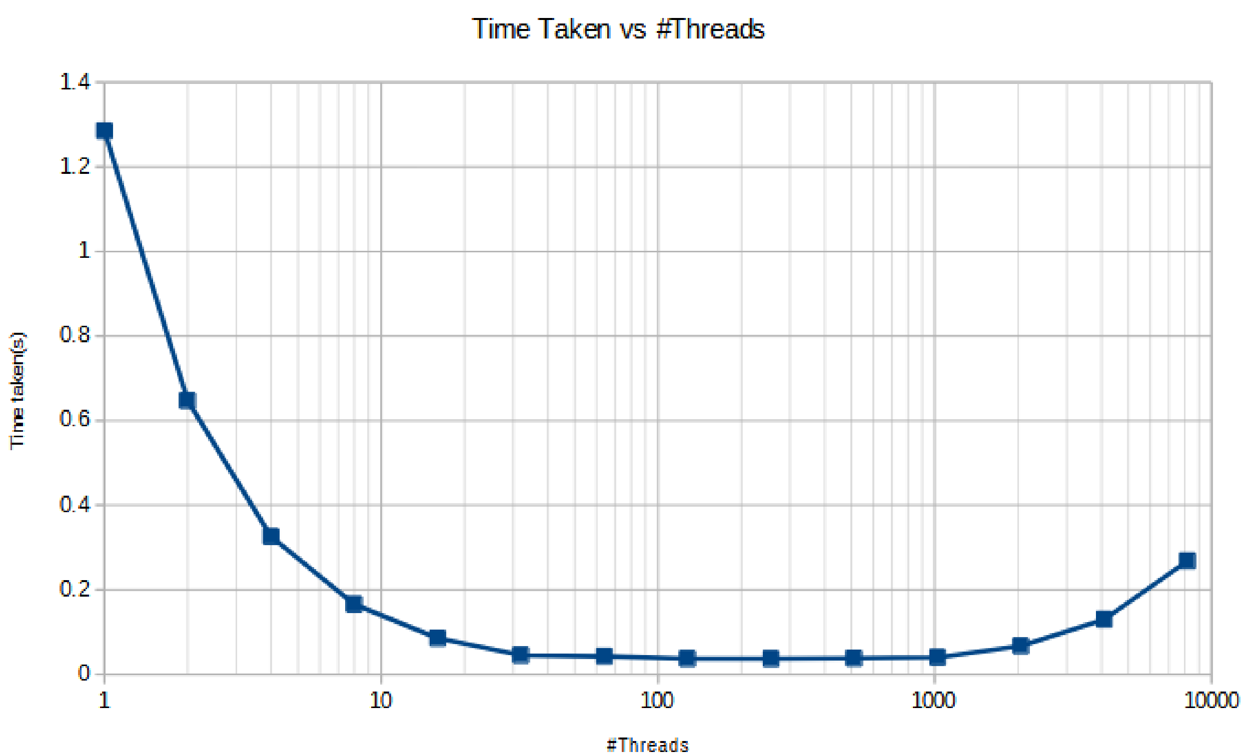
Part 1. Shared-Memory Programming with Threads

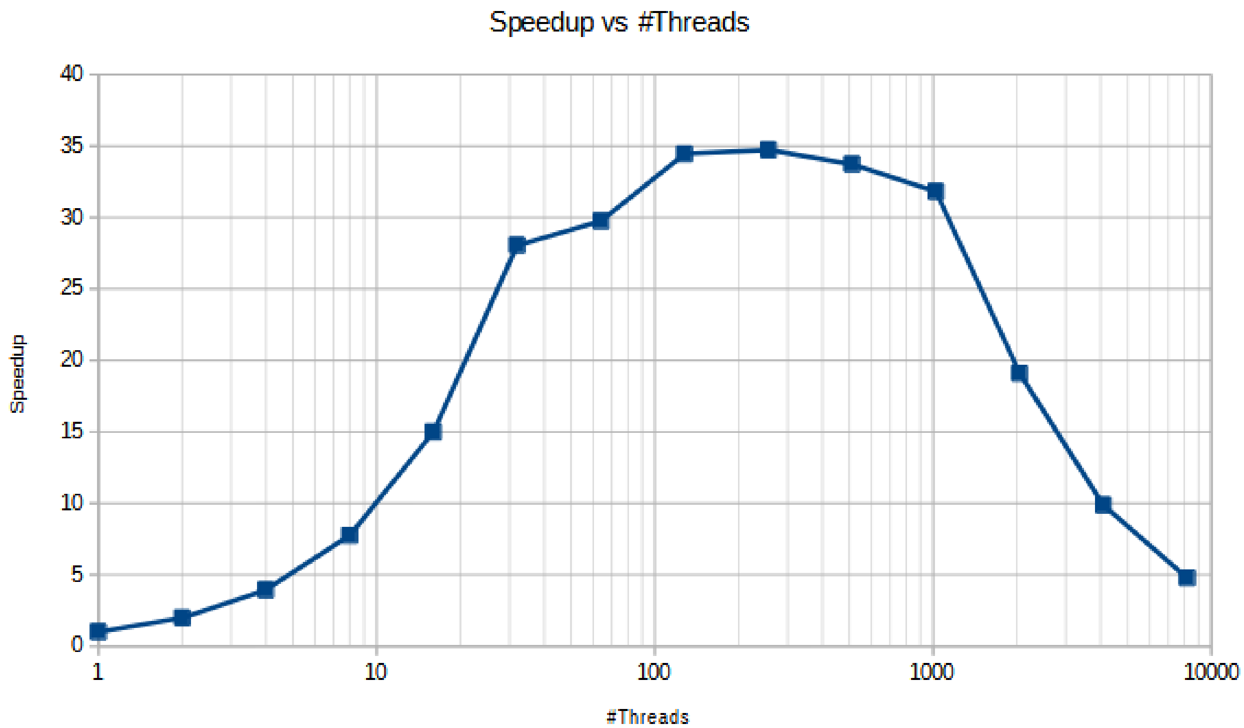
1. Execute the code for n=10^8 with p chosen to be 2^k, for k = 0, 1, …, 13. Using the experimental data obtained from these experiments, answer the following questions. For plots, use a logarithmic scale for the x-axis.

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

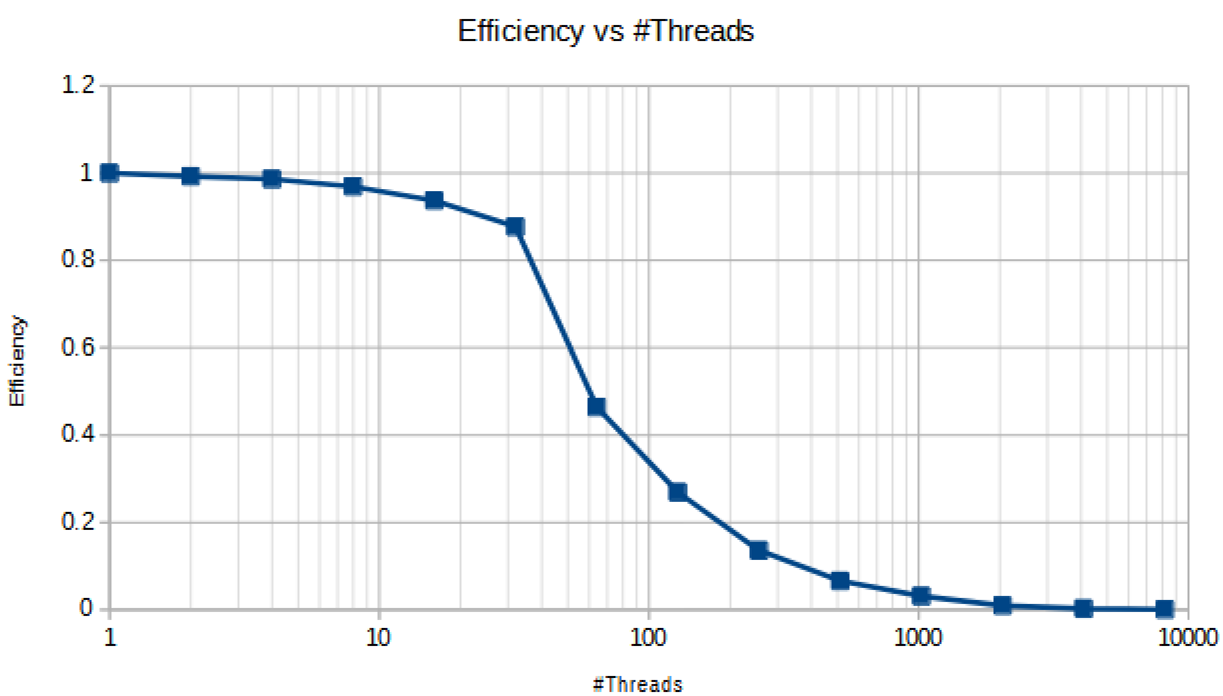
Trials	100000000	Threads	1	pi	3.14161496	error	7.10E-06	time (sec)	1.285	Speedup	1	Efficiency	1
Trials	100000000	Threads	2	pi	3.14170228	error	3.49E-05	time (sec)	0.6473	Speedup	1.98516916	Efficiency	0.99258458
Trials	100000000	Threads	4	pi	3.14159108	error	5.01E-07	time (sec)	0.3258	Speedup	3.94413751	Efficiency	0.98603438
Trials	100000000	Threads	8	pi	3.14159476	error	6.70E-07	time (sec)	0.1657	Speedup	7.75497888	Efficiency	0.96937236
Trials	100000000	Threads	16	pi	3.14152912	error	2.02E-05	time (sec)	0.0857	Speedup	14.9941657	Efficiency	0.93713536
Trials	100000000	Threads	32	pi	3.14159012	error	8.06E-07	time (sec)	0.0458	Speedup	28.0567686	Efficiency	0.87677402
Trials	100000000	Threads	64	pi	3.14135124	error	7.68E-05	time (sec)	0.0432	Speedup	29.7453704	Efficiency	0.46477141
Trials	100000000	Threads	128	pi	3.14292992	error	4.26E-04	time (sec)	0.0373	Speedup	34.4504021	Efficiency	0.26914377
Trials	100000000	Threads	256	pi	3.14129952	error	9.33E-05	time (sec)	0.037	Speedup	34.7297297	Efficiency	0.13566301
Trials	100000000	Threads	512	pi	3.14684508	error	1.67E-03	time (sec)	0.0381	Speedup	33.7270341	Efficiency	0.06587311
Trials	100000000	Threads	1024	pi	3.1514026	error	3.12E-03	time (sec)	0.0404	Speedup	31.8069307	Efficiency	0.03106146
Trials	100000000	Threads	2048	pi	3.14536116	error	1.20E-03	time (sec)	0.0673	Speedup	19.0936107	Efficiency	0.00932305
Trials	100000000	Threads	4096	pi	3.14941624	error	2.49E-03	time (sec)	0.1302	Speedup	9.86943164	Efficiency	0.00240953
Trials	100000000	Threads	8192	pi	3.13770312	error	1.24E-03	time (sec)	0.2684	Speedup	4.7876304	Efficiency	0.00058443



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?

Soln: p=256 minimizes the parallel runtime the most. (t=0.037 sec)

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?**

Soln:

Trials	1E+10	Threads	1	pi	1.42362023	error	5.47E-01	time (sec)	127.9104	Speedup	1	Efficiency	1
Trials	1E+10	Threads	2	pi	3.14160701	error	4.57E-06	time (sec)	64.0838	Speedup	1.99598651	Efficiency	0.99799325
Trials	1E+10	Threads	4	pi	3.14160603	error	4.26E-06	time (sec)	32.0484	Speedup	3.99116337	Efficiency	0.99779084
Trials	1E+10	Threads	8	pi	3.14161182	error	6.10E-06	time (sec)	16.0326	Speedup	7.97814453	Efficiency	0.99726807
Trials	1E+10	Threads	16	pi	3.14160669	error	4.47E-06	time (sec)	8.0246	Speedup	15.9397852	Efficiency	0.99623657
Trials	1E+10	Threads	32	pi	3.14163321	error	1.29E-05	time (sec)	4.0214	Speedup	31.8074302	Efficiency	0.9939822
Trials	1E+10	Threads	64	pi	3.14161391	error	6.77E-06	time (sec)	3.2252	Speedup	39.65968	Efficiency	0.6196825
Trials	1E+10	Threads	128	pi	3.14159434	error	5.38E-07	time (sec)	2.7733	Speedup	46.1220928	Efficiency	0.36032885
Trials	1E+10	Threads	256	pi	3.14164731	error	1.74E-05	time (sec)	2.7057	Speedup	47.2744207	Efficiency	0.18466571
Trials	1E+10	Threads	512	pi	3.1415762	error	5.24E-06	time (sec)	2.6903	Speedup	47.5450322	Efficiency	0.09286139
Trials	1E+10	Threads	1024	pi	3.14143193	error	5.12E-05	time (sec)	2.6931	Speedup	47.4955999	Efficiency	0.04638242
Trials	1E+10	Threads	2048	pi	3.14125887	error	1.06E-04	time (sec)	2.7042	Speedup	47.3006434	Efficiency	0.02309602
Trials	1E+10	Threads	4096	pi	3.14071697	error	2.79E-04	time (sec)	2.7331	Speedup	46.800483	Efficiency	0.0114259
Trials	1E+10	Threads	8192	pi	3.14126339	error	1.05E-04	time (sec)	2.7718	Speedup	46.1470525	Efficiency	0.00563319

For n=10^10, p=512 threads had the lowest time consumed.(t = 2.6903 seconds)

   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**

Soln: Yes, I believe that after a certain number of threads, the runtime would increase. This can also be seen in the both the experiments (where n was 10^8 where the least time taken was when p = 256 threads and when n=10^10, where least time was achieved with p=512 threads)

This behavior can be attributed to the following reasons:

- The ultimate parallelized runtimes may depend on the actual amount of resources, or processing units, that are available and whether or not they are in use. Instead of the number of launched threads, this quantity(resource availability) has a greater impact on the overall runtime. Additionally, runtimes may suffer due to scheduling delays and overheads if more threads are created than there are processing units available.
- Context switching is used when there are more threads than processing units available. To clarify, many threads would be assigned to the same processing units. As a result, the processing unit would need to come up with a strategy for scheduling multiple threads' execution on itself and periodically switching between them. If the time required to switch contexts is longer than the time required to execute that thread, an increase in thread count might be expected to cause the runtime to degrade.
- When the number of threads increases and shared resources are being accessed, access must be synchronized, meaning some threads may have to wait until another thread has finished its read- or write-operation.

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?**

Soln: Yes, there would be a difference in the minimum runtime needed for the two cases of n. This is also observed in the experiments. For n=10^8, the minimum runtime occurred when p=256, while for the case of n=10^10, when p=512, we received the minimum time.

Each processing unit's share of the work is important; for instance, when n = 1024 and p = 256, we may infer that each thread will be handling 4 units of work. And let's assume that the processor can complete the job of one thread in the time assigned to it by the CPU. However, as n is expanded, say to n = 4096, each thread is forced to handle 16 units of work. As each thread doesn't finish execution inside the given CPU time in this case, the processor may need to switch out and switch in threads. As a consequence, runtimes may be sped up by increasing the number of threads such that each thread deals with fewer work units than can be executed in the allocated CPU time. This would minimize the frequency of context transitions.

$s_p = p/(1 + p^2/n)$

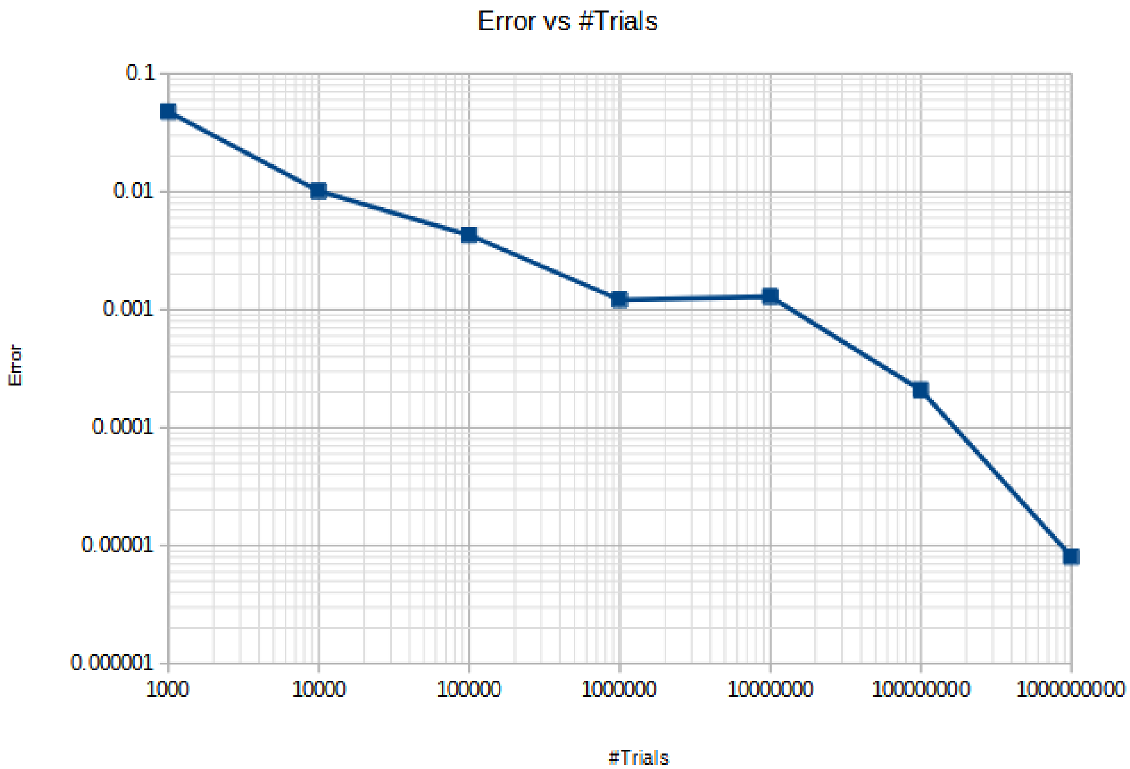
When n is raised, speed up tends to maximize at a later value of p for the number of processes p and the number of trials n. Additionally, I noticed that performing the same tests at various times produced inconsistent results, meaning that the least number of threads required to obtain the shortest runtimes differed for each run's input values. This should imply that some external factors, such as the present workload of the processing units, job scheduling techniques, etc., may also influence the number of threads required to achieve the minimal runtime.

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.**

Soln:

Trials	1000	Threads	48	pi	2.992	error	0.0476	time (sec)	0.0017
Trials	10000	Threads	48	pi	3.1732	error	0.0101	time (sec)	0.0016
Trials	100000	Threads	48	pi	3.15508	error	0.00429	time (sec)	0.0016
Trials	1000000	Threads	48	pi	3.145396	error	0.00121	time (sec)	0.0027
Trials	10000000	Threads	48	pi	3.1456372	error	0.00129	time (sec)	0.0084
Trials	100000000	Threads	48	pi	3.14094004	error	0.000208	time (sec)	0.0399
Trials	1000000000	Threads	48	pi	3.14161791	error	0.00000804	time (sec)	0.2745

Note: Both axis are in logarithmic scale for better visualization

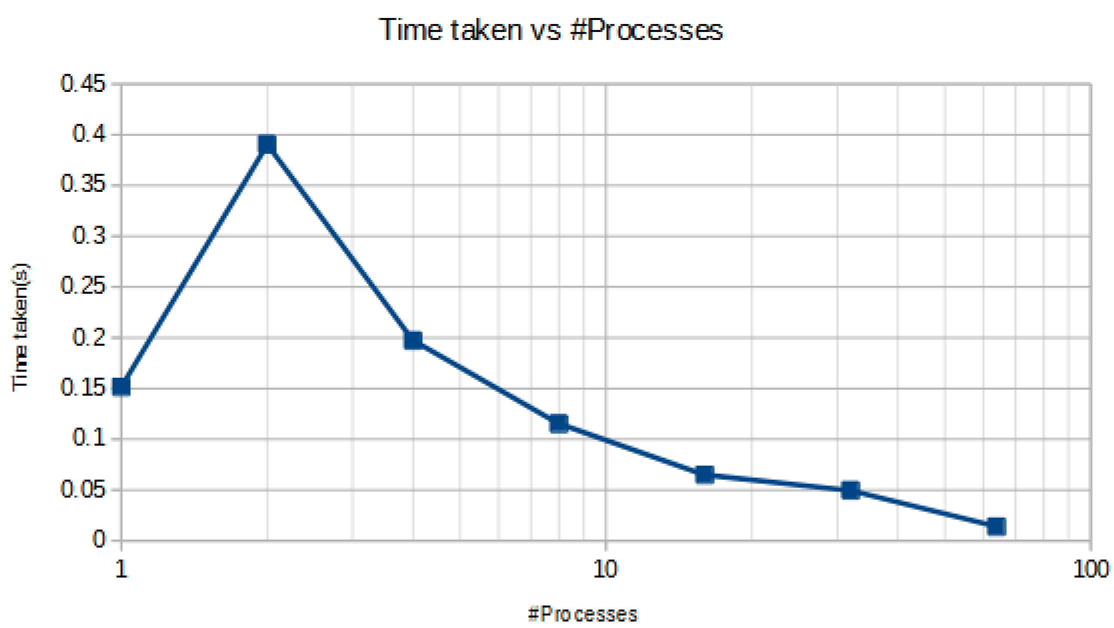


Part 2. Distributed-Memory Programming with MPI

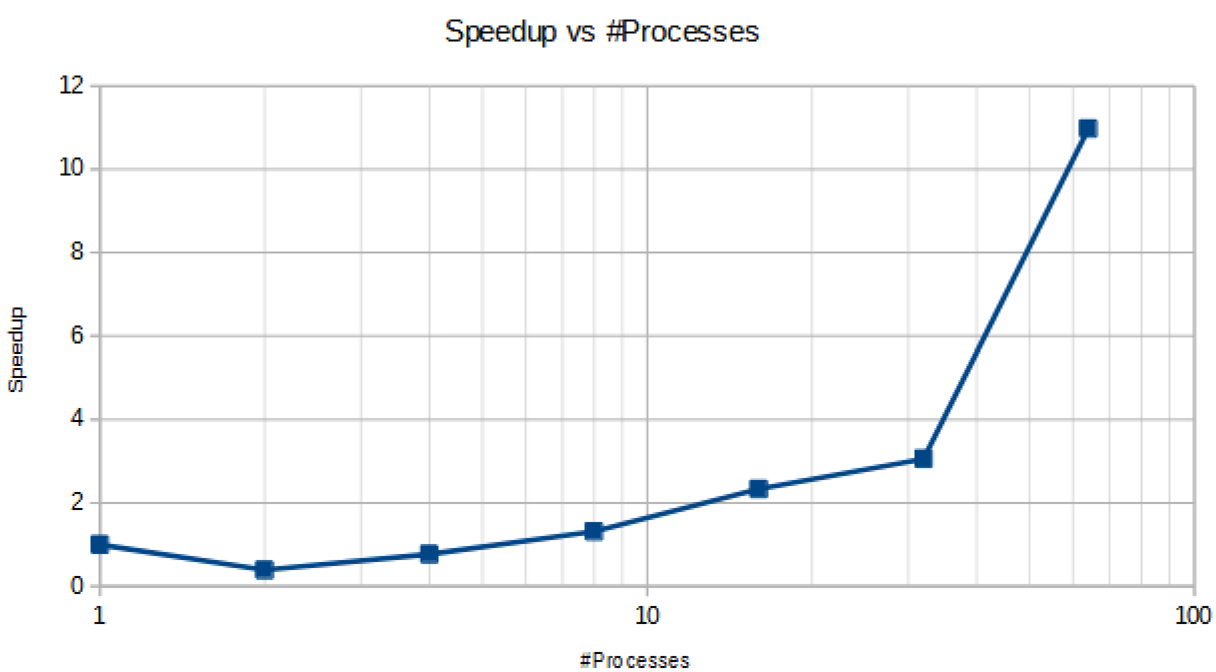
5. **Execute the code for n=10^8 with p chosen to be 2^k, for k = 0, 1, …, 6. Specify ntasks-per-node=4 in the job file. Using the experimental data obtained from these experiments, answer the following questions. For plots, use a logarithmic scale for the x-axis.**

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

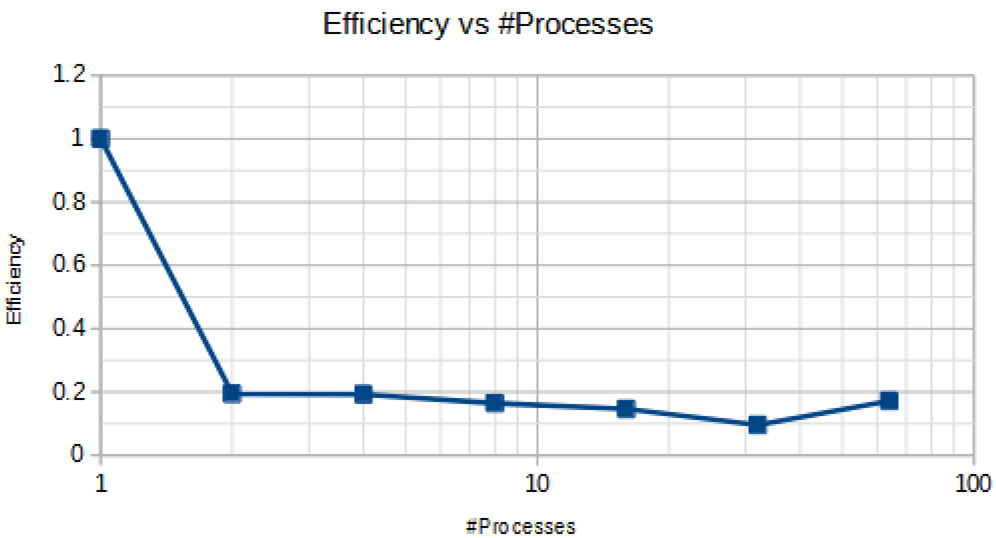
n	100000000	p	1	pi	3.14159265	relative error		2.02E-13	time (sec)	0.1515
n	100000000	p	2	pi	3.14159265	relative error		7.29E-14	time (sec)	0.3904
n	100000000	p	4	pi	3.14159265	relative error		1.35E-13	time (sec)	0.1971
n	100000000	p	8	pi	3.14159265	relative error		5.71E-14	time (sec)	0.1152
n	100000000	p	16	pi	3.14159265	relative error		5.65E-15	time (sec)	0.0649
n	100000000	p	32	pi	3.14159265	relative error		6.22E-15	time (sec)	0.0495
n	100000000	p	64	pi	3.14159265	relative error		2.83E-16	time (sec)	0.0138



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.**



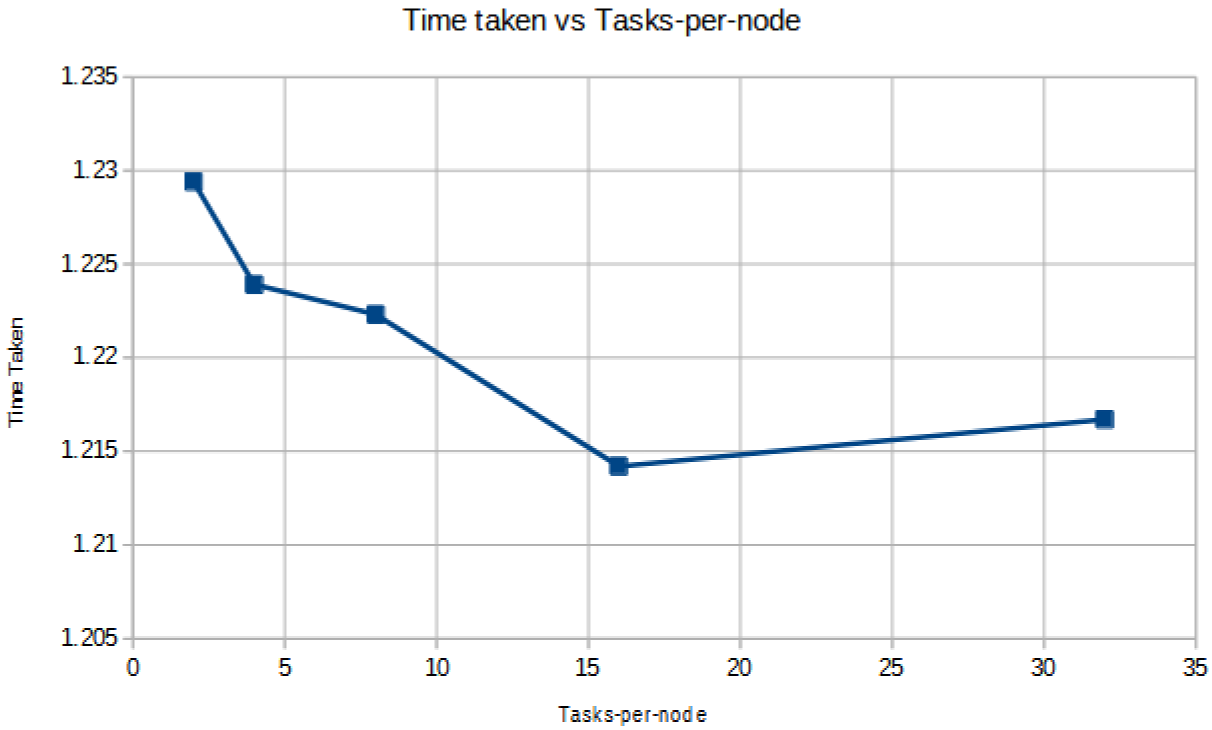
5.4. (5 points) What value of p minimizes the parallel runtime?

n	100000000	p	1	pi	3.14159265	relative error	2.02E-13	time (sec)	0.1515
n	100000000	p	2	pi	3.14159265	relative error	7.29E-14	time (sec)	0.3904
n	100000000	p	4	pi	3.14159265	relative error	1.35E-13	time (sec)	0.1971
n	100000000	p	8	pi	3.14159265	relative error	5.71E-14	time (sec)	0.1152
n	100000000	p	16	pi	3.14159265	relative error	5.65E-15	time (sec)	0.0649
n	100000000	p	32	pi	3.14159265	relative error	6.22E-15	time (sec)	0.0495
n	100000000	p	64	pi	3.14159265	relative error	2.83E-16	time (sec)	0.0138

Soln: 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.

n	10000000000	p	64	pi	3.14159265359	relative error	5.80E-15	time (sec)	1.2294	Ntask-per-node	2
n	10000000000	p	64	pi	3.14159265359	relative error	5.94E-15	time (sec)	1.2239	Ntask-per-node	4
n	10000000000	p	64	pi	3.14159265359	relative error	5.94E-15	time (sec)	1.2223	Ntask-per-node	8
n	10000000000	p	64	pi	3.14159265359	relative error	5.94E-15	time (sec)	1.2142	Ntask-per-node	16
n	10000000000	p	64	pi	3.14159265359	relative error	5.94E-15	time (sec)	1.2167	Ntask-per-node	32

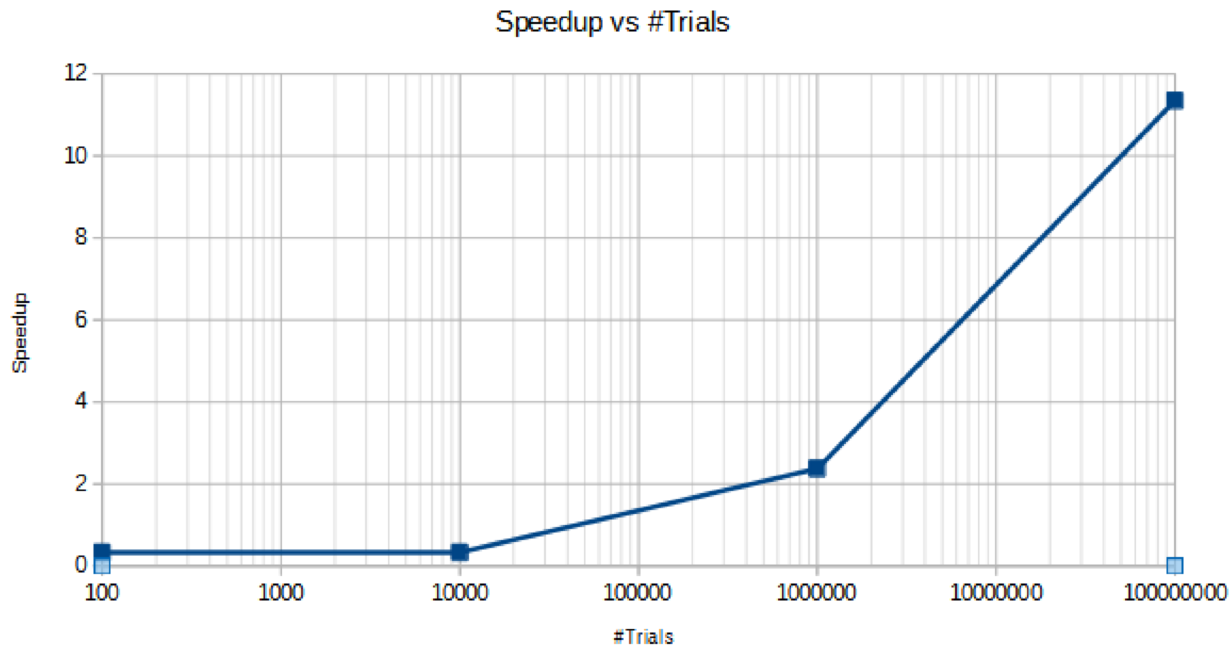


N-task-per-node that minimizes runtime is 16 tasks per node

7. Execute the code with p=64 for n=102, 104, 106 and 108, 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.

n	100000000	p	64	pi	3.14159265	relative error	2.83E-16	time (sec)	0.0134
n	1000000	p	64	pi	3.14159265	relative error	2.62E-14	time (sec)	0.0008
n	10000	p	64	pi	3.14159265	relative error	2.65E-10	time (sec)	0.0003
n	100	p	64	pi	3.14160099	relative error	2.65E-06	time (sec)	0.0003
n	100000000	p	1	pi	3.14159265	relative error	2.02E-13	time (sec)	0.152
n	1000000	p	1	pi	3.14159265	relative error	9.19E-15	time (sec)	0.0019
n	10000	p	1	pi	3.14159265	relative error	2.65E-10	time (sec)	0.0001
n	100	p	1	pi	3.14160099	relative error	2.65E-06	time (sec)	0.0001



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

Note: Both axis are in logarithmic scale for better visualization