Assignment - 2

Question - 1

In this problem you will develop two different implementations of the computation of pi (p) numerically using pthreads and OpenMP. Then you will compare them in terms of scalability and accuracy. Undergraduates/PLUS-One students only need to complete parts b and c of this problem for full credit, though can complete part a (i.e., the pthreads implementation) to receive 20 points of quiz extra credit. In this problem, you will develop a program that computes the value of pi. You can refer to the following video that suggests a way to compute this using Monte Carlo simulation:

https://www.youtube.com/watch?v=M34TO71SKGk&ab_channel=PhysicsGirl

This is not the most efficient way to compute pi, though will provide you with a baseline. There are better ways to compute the value. Select a more efficient method (e.g., Leibniz's formula) and compare it to the Monte Carlo method in terms of convergence rate, assessing the accuracy of the value of pi as a function of runtime.

a. Evaluate the speedup that you achieve by using pthreads and multiple cores. You are free to use as many threads as you like. The program should take two input parameters, the number of threads and the number of "darts" thrown. Your program should print out the time required to compute pi and the final value of pi. Make sure to document the system you are running on and the number of hardware threads available.
b. Now develop the same program using OpenMP. Repeat all of the steps requested in part a.
c. Now compare the two implementations in terms of strong and weak scaling, where the number of Monte Carlo simulations (i.e., "darts" thrown) is used to assess weak scaling. Make sure you plot your results.

Answer:

1 (a):

System: COE Linux System
Model CPU: Intel ® Xeon ® CPU E5-2698 v4 @ 2.20GHz
Cores: 2 sockets, 20 core/socket, 2 thread/socket
Operating System: Rocky Linux Release 8.9 (Green Obsidian)
Number of hardware threads available: 80

Execution time for Leibniz formula and Monte Carlo method using pthreads:

```
[prayags@pi ~]$ g++ -o leibniz leibniz.cpp -pthread
[prayags@pi ~]$ ./leibniz 1 10000000
Leibniz Pi = 3.14159
Computation Time = 0.136071 seconds
[prayags@pi ~]$ ./leibniz 10 10000000
Leibniz Pi = 3.14159
Computation Time = 0.0143957 seconds
```

```
[prayags@pi ~]$ nano monte_carlo.cpp
[prayags@pi ~]$ g++ -o monte_carlo monte_carlo.cpp -pthread
[prayags@pi ~]$ ./monte_carlo 1 10000000
Monte Carlo Pi = 3.14086
Computation Time = 2.34566 seconds
[prayags@pi ~]$ ./monte_carlo 10 10000000
Monte Carlo Pi = 3.1415
Computation Time = 0.373184 seconds
```

Speedup = Execution time with 1 thread/Execution time with 10 threads

Leibniz Speedup in pthreads:

$Speedup_{Leibniz} = 0.136071/0.0143957 = 9.45$

Monte Carlo Speedup in pthreads:

$Speedup_{Monte\ Carlo} = 2.34566/0.373184 = 6.29$

1 (b):

Execution time for Leibniz formula and Monte Carlo method using OpenMP:

```
[prayags@pi ~]$ ./leibniz_openmp 1 10000000
Leibniz Pi = 3.14159
Computation Time = 0.0781223 seconds
[prayags@pi ~]$ ./leibniz_openmp 10 10000000
Leibniz Pi = 3.14159
Computation Time = 0.0100498 seconds
```

```
[prayags@pi ~]$ g++ -fopenmp -o monte_carlo_openmp monte_carlo_openmp.cpp
[prayags@pi ~]$ ./monte_carlo_openmp 1 10000000
Monte Carlo Pi = 3.14155
Computation Time = 2.3469 seconds

[prayags@pi ~]$ ./monte_carlo_openmp 10 10000000
Monte Carlo Pi = 3.14219
Computation Time = 0.373526 seconds
```

Speedup = Execution time with 1 thread/Execution time with 10 threads

Leibniz Speedup in OpenMP:

$Speedup_{Leibniz} = 0.0781223/0.0100498 = 7.77$

Monte Carlo Speedup in OpenMP:

$Speedup_{Monte\ Carlo} = 2.3469/0.373526 = 6.29$

These speedups are quite significant, showing that the parallelization of both algorithms has successfully utilized the multiple cores of your CPU. The results are nearly perfect, as the speedup would match the number of threads if there were no extra issues like managing threads and syncing them. The difference in speed between the two models may be due to how each system manages threads and handles extra work. OpenMP makes it easy to work with threads because it handles most of the setup and coordination on its own. This can improve how threads are spread out and reduces the extra code that developers have to write. This might explain why the Leibniz method performs a little better with OpenMP. On the other hand, pthreads, which require more manual control over thread creation and handling, offer developers fine-grained control over threading behavior. If handled well, this can improve performance, but it can also make things more complicated and less efficient.

1 (c):

In pthreads:

```
[prayags@pi ~]$ ./leibniz 1 10000000
Leibniz Pi = 3.14159
Computation Time = 0.13638 seconds
[prayags@pi ~]$ ./leibniz 5 10000000
Leibniz Pi = 3.14159
Computation Time = 0.027648 seconds
[prayags@pi ~]$ ./leibniz 10 10000000
Leibniz Pi = 3.14159
Computation Time = 0.0143399 seconds
[prayags@pi ~]$ ./leibniz 20 10000000
Leibniz Pi = 3.14159
Computation Time = 0.00806938 seconds
[prayags@pi ~]$ ./leibniz 40 10000000
Leibniz Pi = 3.14159
Computation Time = 0.00627383 seconds
```

```
[prayags@pi ~]$ ./monte_carlo 1 10000000
Monte Carlo Pi = 3.14113
Computation Time = 2.38775 seconds
[prayags@pi ~]$ ./monte_carlo 10 10000000
Monte Carlo Pi = 3.14198
Computation Time = 0.373381 seconds
[prayags@pi ~]$ ./monte_carlo 5 10000000
Monte Carlo Pi = 3.14179
Computation Time = 0.680672 seconds
[prayags@pi ~]$ ./monte_carlo 20 10000000
Monte Carlo Pi = 3.14165
Computation Time = 0.187769 seconds
[prayags@pi ~]$ ./monte_carlo 40 10000000
Monte Carlo Pi = 3.14165
Computation Time = 0.137028 seconds
```

Strong scaling efficiency here is calculated as

Efficiency= [Execution Time at 1 Thread × 100/ (Execution Time at N Threads × N)]

Threads	Number of darts	Execution Time in Monte Carlo method in seconds	Execution Time in Leibniz formula in seconds	Efficiency of Monte Carlo method	Efficiency of Leibniz formula
1	10000000	2.38775	0.13638	100%	100%
5	10000000	0.680672	0.027648	70.14%	99.05%
10	10000000	0.373381	0.0143399	63.95%	95.27%
20	10000000	0.187769	0.00806938	63.64%	84.37%
40	10000000	0.137028	0.00627383	43.73%	54.41%

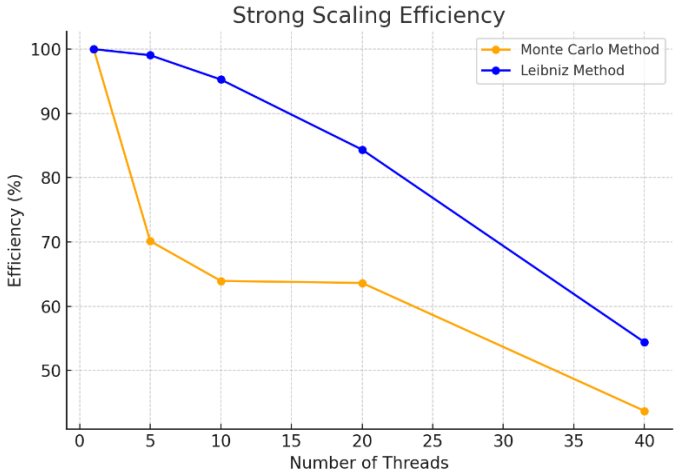
Weak scaling efficiency is calculated as:

Efficiency= (Execution Time at 1 Thread/Execution Time at N Threads)×100

Threads	Number of darts	Execution Time in Monte Carlo method in seconds	Execution Time in Leibniz method in seconds	Efficiency of Monte Carlo method	Efficiency of Leibniz formula
1	10000000	2.38775	0.137392	100%	100%
5	50000000	2.39883	0.138373	99.54%	99.29%
10	100000000	2.38724	0.137042	100.02%	100.26%
20	200000000	2.67005	0.137691	89.43%	99.78%
40	400000000	2.84473	0.145691	83.93%	94.30%

Here is the plotted graph:

For strong scaling:



For weak scaling:

In OpenMP:

Strong scaling efficiency here is calculated as

Efficiency= [Execution Time at 1 Thread × 100/ (Execution Time at N Threads × N)]

```
[prayags@rho ~]$ ./monte_carlo_openmp 1 10000000
Monte Carlo Pi = 3.14145
Computation Time = 1.23896 seconds
[prayags@rho ~]$ ./monte_carlo_openmp 5 10000000
Monte Carlo Pi = 3.14196
Computation Time = 0.271435 seconds
[prayags@rho ~]$ ./monte_carlo_openmp 10 10000000
Monte Carlo Pi = 3.14063
Computation Time = 0.150377 seconds
[prayags@rho ~]$ ./monte_carlo_openmp 20 10000000
Monte Carlo Pi = 3.14204
Computation Time = 0.0835055 seconds
[prayags@rho ~]$ ./monte_carlo 40 10000000
Monte Carlo Pi = 3.14148
Computation Time = 0.0434824 seconds

[prayags@rho ~]$ ./leibniz_openmp 1 10000000
Leibniz Pi = 3.14159
Computation Time = 0.0259669 seconds
[prayags@rho ~]$ ./leibniz_openmp 5 10000000
Leibniz Pi = 3.14159
Computation Time = 0.00592952 seconds
[prayags@rho ~]$ ./leibniz_openmp 10 10000000
Leibniz Pi = 3.14159
Computation Time = 0.00347226 seconds
[prayags@rho ~]$ ./leibniz_openmp 20 10000000
Leibniz Pi = 3.14159
Computation Time = 0.00244046 seconds
[prayags@rho ~]$ ./leibniz_openmp 40 10000000
Leibniz Pi = 3.14159
Computation Time = 0.00255688 seconds
```

Threads	Number of darts	Execution Time in Monte Carlo method in seconds	Execution Time in Leibniz formula in seconds	Efficiency of Monte Carlo method	Efficiency of Leibniz formula
1	10000000	1.23896	0.0259669	100%	100%
5	10000000	0.271435	0.00592952	91.35%	87.63%
10	10000000	0.150377	0.00347226	82.33%	74.86%
20	10000000	0.0835055	0.00244046	74.23%	53.24%
40	10000000	0.0434824	0.00255688	71.39%	50.80%

Weak scaling efficiency is calculated as:

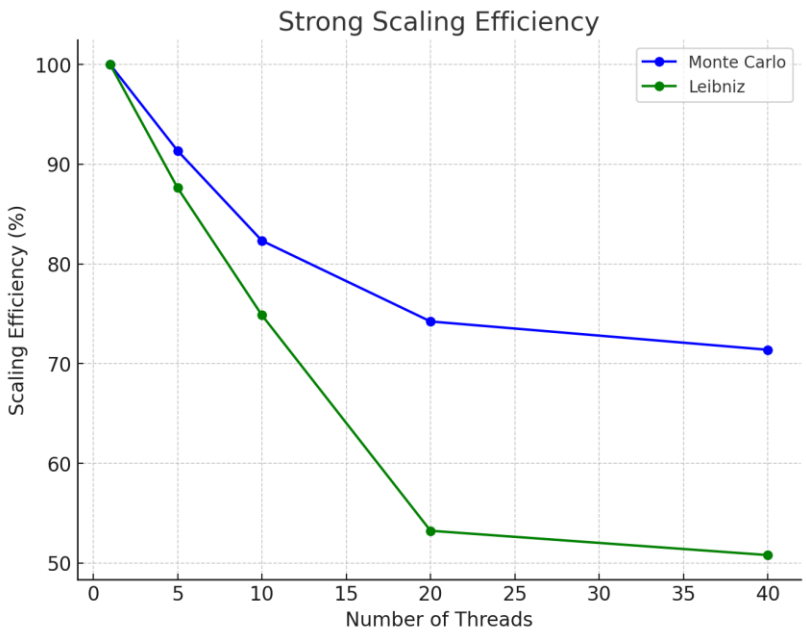
Efficiency= (Execution Time at 1 Thread/Execution Time at N Threads)×100

```
[prayags@rho ~]$ ./leibniz_openmp 5 50000000
Leibniz Pi = 3.14159
Computation Time = 0.0307917 seconds
[prayags@rho ~]$ ./leibniz_openmp 10 100000000
Leibniz Pi = 3.14159
Computation Time = 0.0319614 seconds
[prayags@rho ~]$ ./leibniz_openmp 20 200000000
Leibniz Pi = 3.14159
Computation Time = 0.0354383 seconds
[prayags@rho ~]$ ./leibniz_openmp 40 400000000
Leibniz Pi = 3.14159
Computation Time = 0.0495568 seconds
```

```
[prayags@rho ~]$ ./monte_carlo_openmp 5 50000000
Monte Carlo Pi = 3.14145
Computation Time = 1.40445 seconds
[prayags@rho ~]$ ./monte_carlo_openmp 10 100000000
Monte Carlo Pi = 3.14151
Computation Time = 1.44627 seconds
[prayags@rho ~]$ ./monte_carlo_openmp 20 200000000
Monte Carlo Pi = 3.14169
Computation Time = 1.65383 seconds
[prayags@rho ~]$ ./monte_carlo_openmp 40 400000000
Monte Carlo Pi = 3.14154
Computation Time = 1.75816 seconds
```

Threads	Number of darts	Execution Time in Monte Carlo method in seconds	Execution Time in Leibniz formula in seconds	Efficiency of Monte Carlo method	Efficiency of Leibniz formula
1	10000000	1.23896	0.0259669	100%	100%
5	50000000	1.40445	0.0307917	88.22%	84.32%
10	100000000	1.44627	0.0319614	85.66%	81.23%
20	200000000	1.65383	0.0354383	74.92%	73.24%
40	400000000	1.75816	0.0495568	70.46%	52.36%

Here is the plotted graph:
In strong scaling -



In weak scaling –