Hw2 - Threads
Hirokatsu (Hiro) Suzuki

Problem

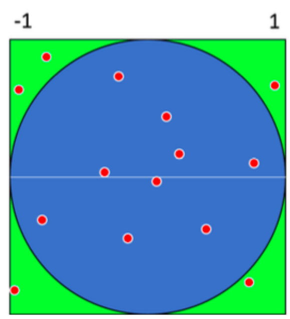
Part a.

Perform matrix multiplication using threads:

- m = 3000 (#rows = #cols)
- Data type = double precision (64 bits)
- # threads = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512}
- Assume *Ts* = Time with one thread, and measure speedup

Part b.

Compute π by "randomly" choosing points. π is four times the fraction that falls in the circle (imagine you're throwing darts to a target).



$Ac = \pi r^2$

$As = 2r*2r = 4r^2$

$P = Ac/As = \pi/4$

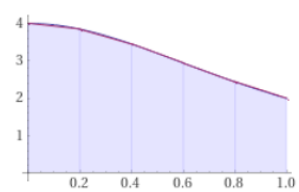
- You should use a thread-safe real uniform random generator
- Make a table for different values of *n* (# darts) and # threads

Part c.

Approximate the integral (area under the curve),

$$\int_0^1 \frac{4.0}{1+x^2}\,dx = \pi$$

using the *trapezoidal rule*.



n = 5

- Your results must show convergence (more trapezoids, better approximation)
- Measure the speedup, you should attain at least quasilinear speedup

Solution

Part a.

Threads	Runtime	Speed-up
1	411.293262	1
2	241.815628	1.70085476
4	123.393519	3.33318367
8	63.691683	6.45756624
16	34.639879	11.8734035
32	24.669605	16.6720652
64	25.010531	16.4448033
128	24.937965	16.4926554
256	24.483992	16.798456
512	24.270073	16.9465194

Table.1 Runtime and speed-up of 3000x3000 matrix multiplication with threads. Runtime based on an average of 5 runs.

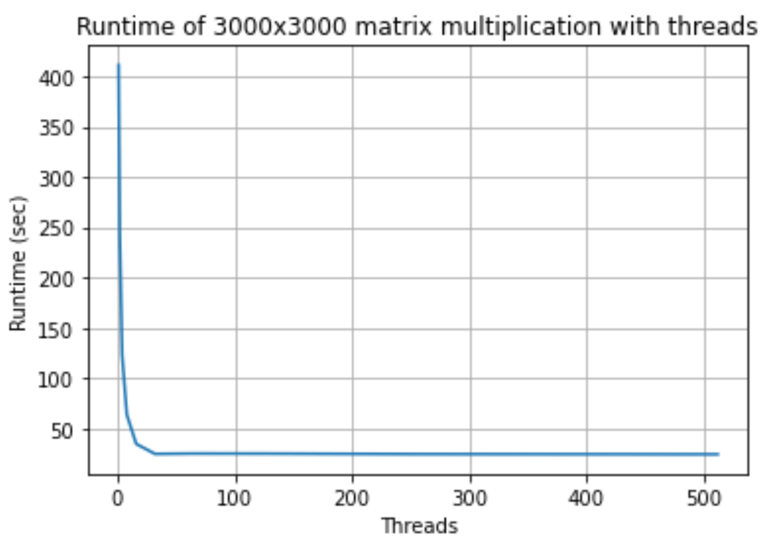


Figure.1 Plot of runtime vs. threads.

Since the dgx.sdsu.edu server was down and was not available for use, I accessed the notos.sdsu.edu server to compute the matrix multiplication.

In this section, I used ijk-form of matrix multiplication. With only 1 thread, the runtime was 411 sec to complete the process. The runtime decreased as I double the threads. Starting with 32 threads, the runtime did not speed-up as much.

Part b.

Threads	n=100	n=10000	n=1000000	n=100000000
1	2.92	3.1444	3.14166	3.14159
2	2.92	3.1444	3.14188	3.14182
4	2.92	3.1528	3.14191	3.14227
8	2.96	3.1444	3.14183	3.14173
16	2.84	3.1464	3.1511	3.14215
32	2.96	3.1432	3.14204	3.14124
64	2.92	3.1468	3.148	3.14265
128	2.92	3.1452	3.15563	3.1561

Table.2 Approximating Pi using different number of darts vs. threads.

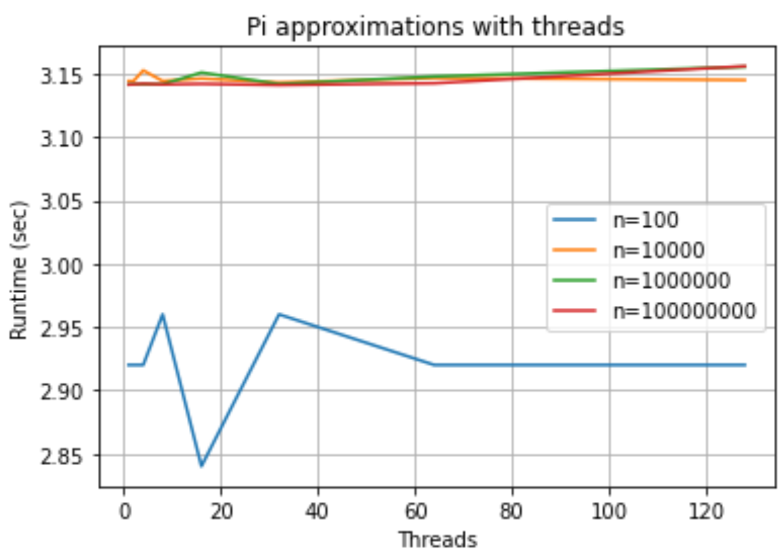


Figure.2 Plot of pi approximation vs. threads.

Since the dgx.sdsu.edu server was down and was not available for use, I accessed the notos.sdsu.edu server to compute the pi approximation.

In this section, I used up to 100000000 darts to approximate the pi value. With only hundreds of darts, the approximation of pi was disastrous. As I use larger number of darts, I obtained better approximated pi value. The number of threads used did not affect the approximation tremendously.

Part c.

Threads	n=5	n=50	n=100	n=10000	n=1000000	n=100000000
1	0.00002	0.000022	0.000024	0.000387	0.033193	1.764528
2	0.000144	0.00015	0.000145	0.000334	0.017811	1.000826
4	0.001854	0.006329	0.005103	0.003002	0.017492	0.620482
8	0.005683	0.006504	0.012313	0.011719	0.007335	0.332895
16	0.019603	0.026682	0.020663	0.009411	0.017173	0.188873
32	0.026087	0.0167	0.01833	0.028187	0.030302	0.176999
64	0.002598	0.00279	0.002748	0.002628	0.006399	0.152044
128	0.00526	0.005361	0.005587	0.005217	0.018007	0.133838

Table.3 Runtime using different number of trapezoids vs. threads. Runtime based on an average of 5 runs.

Threads	n=5	n=50	n=100	n=10000	n=1000000	n=100000000
1	3.134926	3.141526	3.141576	3.141594	3.141349	0.671089
2	3.134926	3.141526	3.141576	3.141594	3.142063	1.342177
4	3.134926	3.141526	3.141576	3.141594	3.14145	2.684355
8	3.134926	3.141526	3.141576	3.141594	3.141496	3.146091
16	3.134926	3.141526	3.141576	3.141594	3.14161	3.164652
32	3.134926	3.141526	3.141576	3.141594	3.141644	3.147811
64	3.134926	3.141526	3.141576	3.141594	3.141587	3.145799
128	3.134926	3.141526	3.141576	3.141594	3.141592	3.143363

Table.4 Integral evaluation using different number of trapezoids vs. threads.

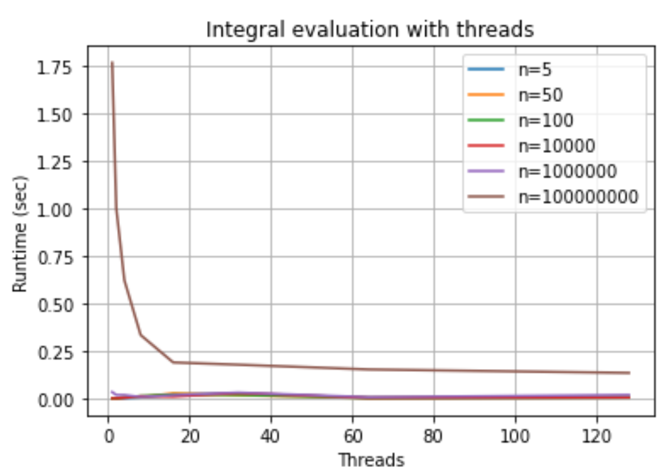


Figure.3 Plot of integral approximation vs. threads.

Since the dgx.sdsu.edu server was down and was not available for use, I accessed the notos.sdsu.edu server to compute the integral evaluation.

The approximation of integral started somewhat accurate result. As I increase the number of trapezoids, the integral outputs value closer to pi. I was not able to observe a quasilinear speedup until I used 100000000 trapezoids which resulted in poor evaluation of integral with less threads.