Name: I.Suventhan
Index No: 120649P

Assignment 1

Q1)

a) The pi-mystery.cu program is used to calculate the pi value. The code is the calculation of pi by the method of numeric integration.

tan(pi/4.0) = 1.0

arctan(1.0) = pi/4.0

So derivative of the arctan(x) is x'/(1 + x*x)

arctan(1.0) = 1.0/(1.0 + x*x)

By integrating this we can approximate Arctan(1.0) We'll perform a numerical integration to get an approximation of Pi/4, then multiply by 4.0 to get an approximation of Pi.

So this way pi-mystery.cu program calculate the pi value.

h)

# Trials	Single- or Double-Precision (SP/DP)		CPU Sequential [curand]	CPU Parallel [curand] (T = # Threads)			GPU			
				T = 2	T = 4	T = 8	Mystery	Myrand	Curand	Curand-thrust
2^{24}	SP	PI estimate	3.141484	3.141450	3.141713	3.141782	3.141920	3.144249	3.141898	3.141484
		Error	-0.000109	-0.000143	0.000120	0.000089	0.000327	0.002657	0.000305	-0.00010910 82
		Time	1.013342 s	13.676162 s	16.403097 s	78.784187 s	0.831115 s	0.840165 s	0.110122 s	0.840266 s
	DP	PI estimate	3.1414835453033447 27	3.1415967941284179 69	3.1416170597076416 02	3.1415832042694091 80	3.1415926535897 8445	3.1442494392395 01953	3.1418979167938 23242	3.1414835453033447266
		Error	-0.000109108286448389	0.000004140538624853	0.000024406117848486	-0.000009449320383936	0.000000000000 5329	0.002656785649708837	0.000305263204030126	-0.00010910828644838943546
		Time	0.953142 s	4.415300 s	12.799242 s	80.835434 s	0.839420 s	0.889233 s	0.109164 s	0.844356 s
2^{26}	SP	PI estimate	3.141926	3.141557	3.141424	3.141721	3.140936	3.139517	3.141390	3.141658

		Error	0.000333	-0.000036	-0.000168	0.000128	-0.000657	-0.002076	-0.000202	6.565262e-05
		Time	3.807110 s	20.844784 s	42.452942 s	327.071289 s	0.850269 s	0.837769 s	0.109963 s	0.840028 s
	DP	PI estimate	3.141925334930419922	3.141460657119750977	3.141400575637817383	3.141575574874877930	3.141592653589798889	3.139507114887237549	3.141393959522247314	3.141658306121826171 9
		Error	0.000332681340626806	-0.000131996470042139	-0.000192077951975733	-0.000017078714915186	0.000000000000005773	-0.002085538702555567	-0.000198694067545802	6.565253203305587703 7e-05
		Time	3.852591 s	66.837502 s	54.303696 s	349.687378 s	0.837182 s	0.845137 s	0.112228 s	0.850235 s
2^{28}	SP	PI estimate	3.141518	3.141582	3.141570	3.141549	3.140863	3.141507	3.141582	3.141579
		Error	-0.000075	-0.000010	-0.000023	-0.000044	-0.000730	-0.000085	-0.000011	-1.326393e-05
		Time	15.291717 s	120.398315 s	246.337570 s	1277.036255 s	0.854651 s	0.883519 s	0.123704 s	0.865743 s
	DP	PI estimate	3.141517639160156250	3.141556024551391602	3.141476869583129883	3.141643047332763672	3.141592653589822870	3.141498014330863953	3.141576066613197327	3.141579389572143554 7
		Error	-0.000075014429636866	-0.000036629038401514	-0.000115784006663233	0.000050393742970556	0.000000000000029754	-0.000094639258929163	-0.000016586976595789	-1.326401764956131046 3e-05
		Time	15.305373 s	137.652679 s	174.770157 s	1202.846924 s	0.857573 s	0.874993 s	0.876353 s	0.857896 s

Table shows the results of the pi calculation using different strategy. Table includes pi value and the error and time for the calculation for different number of trails. Results clearly shows that the execution time for the GPU version programs good compare than the sequential and CPU parallel version. CPU parallel version take more time than other two. This because of the task scheduling and final results calculations take some time. In GPU calculation pi-curand version is better than the other three versions. PI values calculated in each version is nearly same. In sequential and GPU parallel are given nearly same and minimum error for PI values compare than CPU parallel.

Q2)

Vector Dot product

N	Single-Or Double-Precision	CPU Sequential	CPU Parallel (T=# Threads)			GPU
			T=2	T=4	T=8	
10^7	SP	16.46	15.84	18.78	23.80	130.67
	DP	20.35	21.84	26.51	44.48	152.53
$5x10^7$	SP	81.29	73.30	53.84	66.40	112.63
	DP	98.33	94.32	98.45	112.06	393.66
10^8	SP	169.01	182.00	128.26	130.07	569.50
	DP	195.78	196.67	197.92	246.01	412.14

In the above table shows that the execution time results for the Sequential, CPU Parallel (OpenMP), GPU (CUDA) parallel computation of the vector dot product. Results are clearly shows that for all methods execution time increase with vector size. Also for each vector size implementation with double precision time greater than single precision. But CPU parallel and GPU parallel versions are taken more time than sequential version. Because in GPU parallel version vector array and final results are transfer between host to device and also memory allocation in device also take some time. Therefore GPU parallel version take more time. In CPU parallel version, time taken for execution increase with number of threads. This is because tasks allocation for each thread and the final results calculated in serial manner. CPU parallel code is simple version of the OpenMP, we can optimize the code in some ways to reduce the time.

Q3)
Matrix Multiplication

NxN	Single-Or Double-Precision	CPU Sequential	CPU Parallel (T=# Threads)			GPU	
			T=2	T=4	T=8	Basic	Enhanced
600x600	SP	430.16	255.75	149.54	83.57	3.45	3.00
	DP	1145.73	588.42	351.90	206.27	9.59	13.83
1200x1200	SP	3031.94	1548.48	881.05	475.20	22.99	21.54
	DP	8688.77	4322.22	2296.31	1211.51	49.43	84.17
1800x1800	SP	9864.39	4981.48	2595.91	1380.26	76.74	71.30
	DP	33203.86	18773.05	9854.78	5144.59	176.57	260.89

In above table shows the execution time for matrix multiplication of the Sequential, CPU parallel (OpenMP) and GPU parallel (CUDA) version. The results clearly shows that for all

version execution time increase with the matrix dimension. Also we can observe that the CPU parallel version better than the sequential version. For 2 threads CPU parallel version is half of the sequential version. In CPU parallel version execution time decrease with number of threads. CPU parallel optimized using 8 threads. In GPU parallel version much faster than sequential and CPU parallel version. But GPU basic and enhanced version there is no much difference.

References

[1] http://www.nvidia.com/content/gtc-2010/pdfs/2131_gtc2010.pdf
[2] https://computing.llnl.gov/tutorials/openMP/#Synchronization
[3] http://www.appentra.com/parallel-matrix-matrix-multiplication/
[4] http://www4.ncsu.edu/~srmagura/media/vector_slides.pdf
[5] https://wiki.scinet.utoronto.ca/wiki/images/8/8c/SCLecture11.pdf
[6] http://www.umiacs.umd.edu/~ramani/cmsc828e_gpusci/Lecture5.pdf