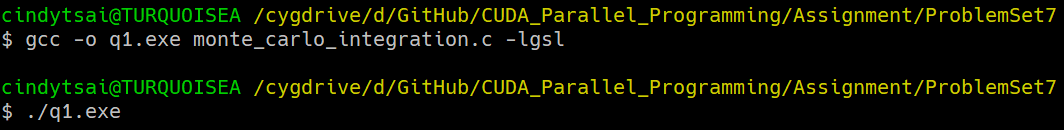
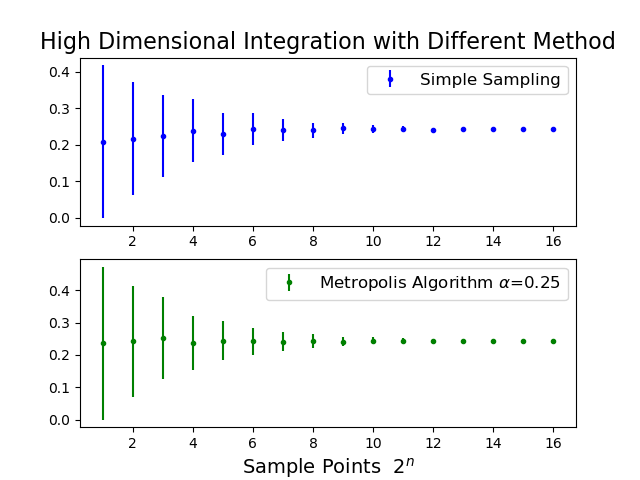
**CUDA Parallel Programming Problem Set 7**

R08244004 Shin-Rong Tsai

* **Monte Carlo Integration in 10 Dimension**

1. **Result**

Run and compile *monte\_carlo\_integration.c* with gsl library on my computer.

After finding the most proper I think, plot the results of Simple Sampling and Metropolis Algorithm with *plot\_result.py.* How I find the weight function parameter is in the discussion.

1. **Discussion**
2. Finding the weight function parameter

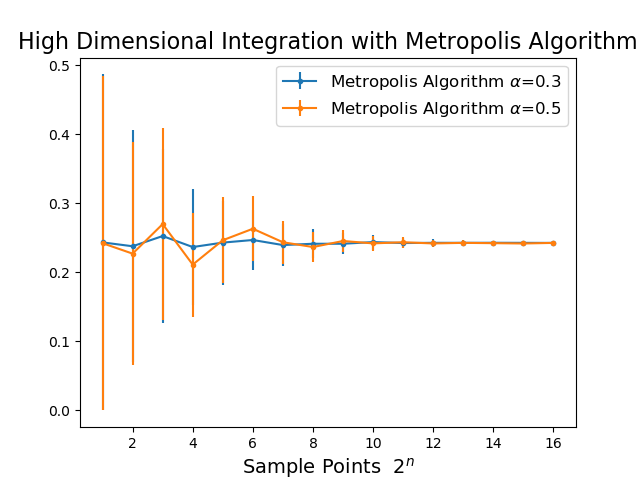
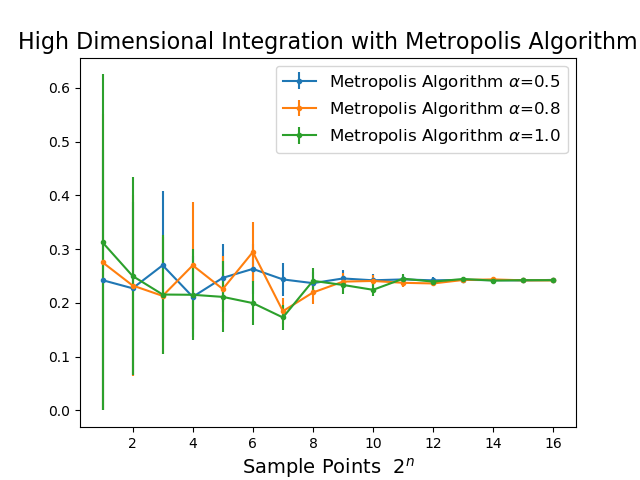
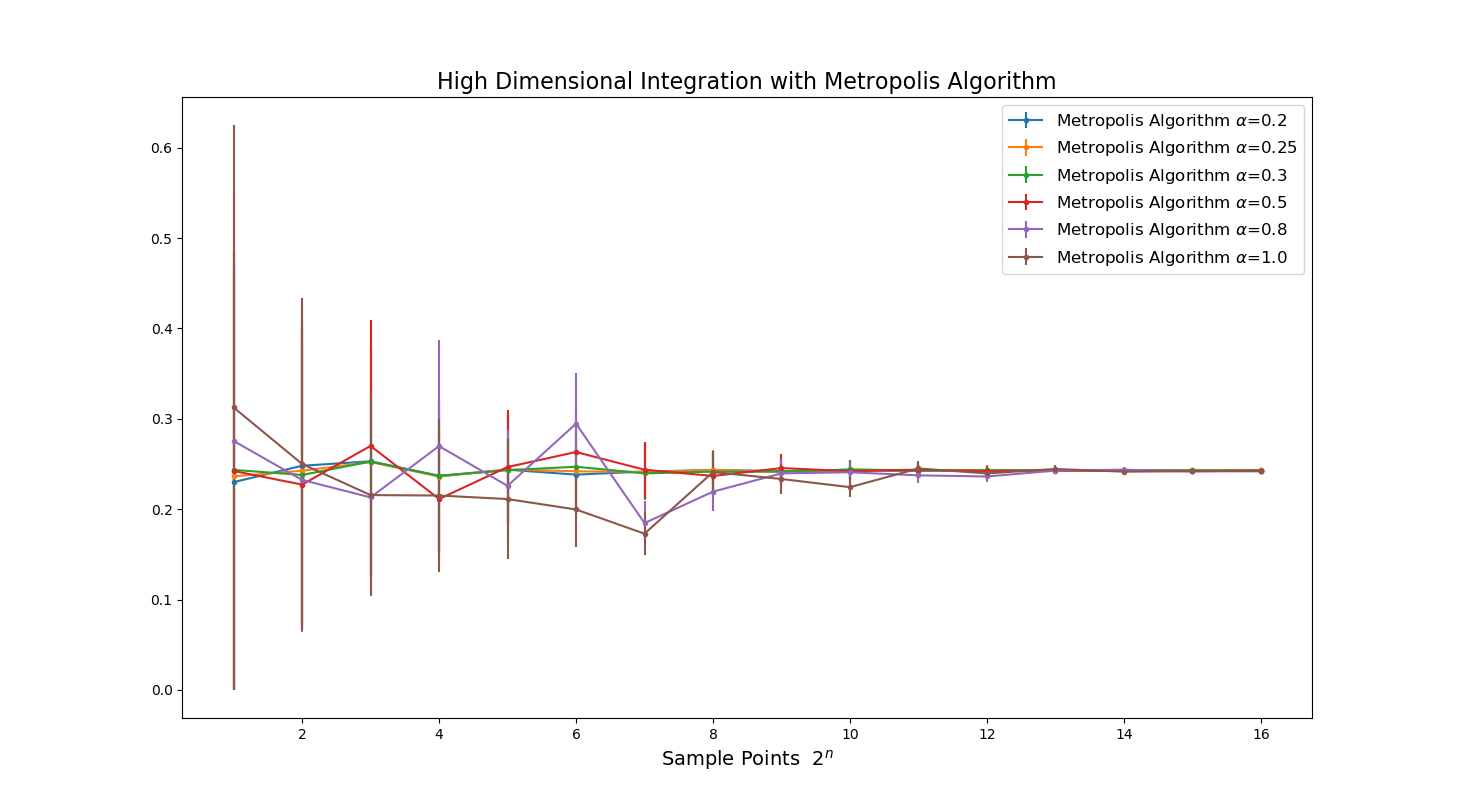
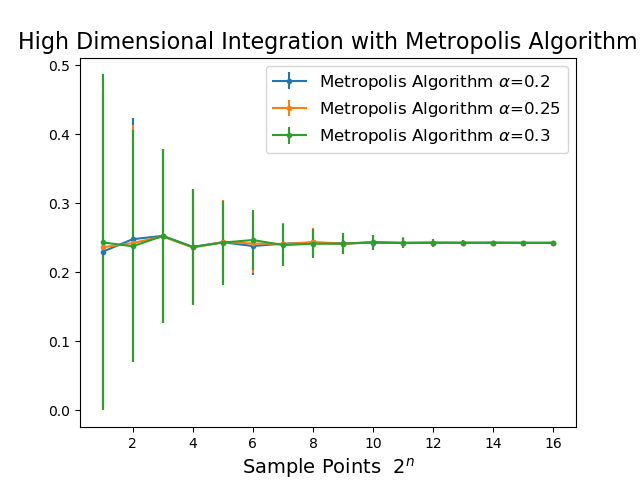
Since , every is symmetric, so weight function parameter of each dimension of *x* should be the same.

And weight function must satisfy

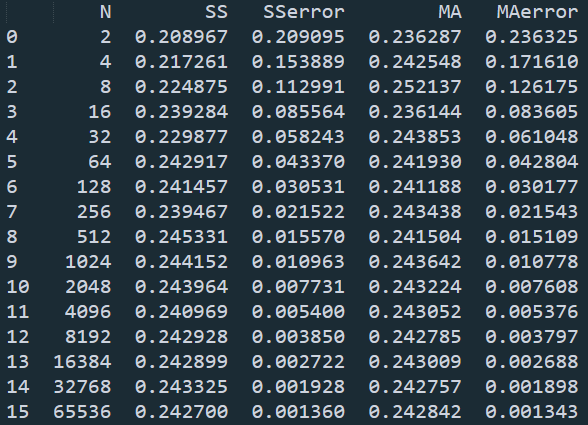
We only have to determine , and calculate *C* through

Because a better weight function would guess the distribution of more correctly, which means it does not vary that much when the number of sample points *N* are small and it converges faster as *N* becomes larger.

So through trying a series of , and plot them (in file *finding\_alpha.py*), we can clearly see that it is reasonable to choose .



1. Compare the result of Simple Sampling and Metropolis Algorithm

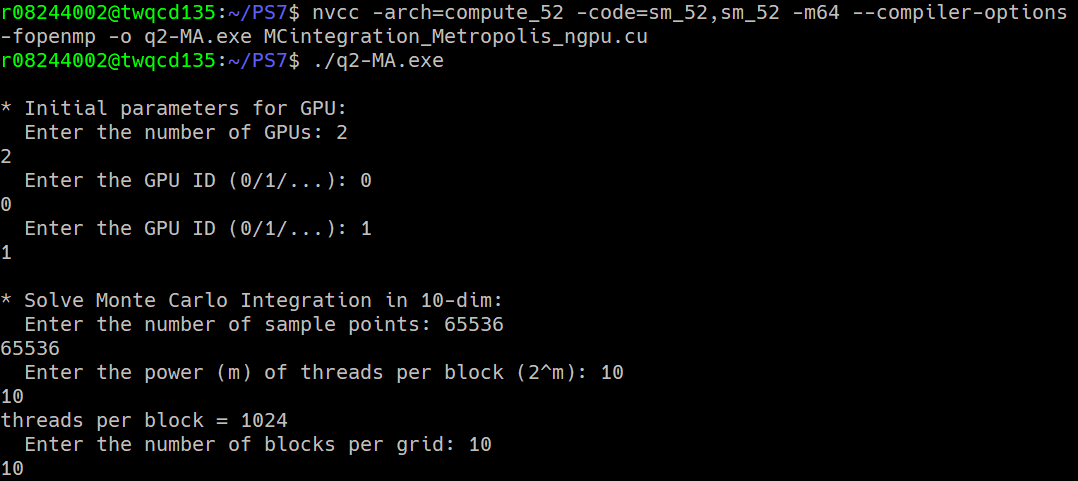
We can see that with a proper Metropolis Algorithm converge faster than Simple Sampling. And we can check the central limit theorem by dividing each of their error with , then we will approximately get the error of the next row below.

Through the result, we can say that this 10-dimension integral is approximately .

* **GPU accelerated Monte Carlo integration in 10-dimension**

1. **Result**

I write the GPU code for Simple Sampling and Metropolis Algorithm in files *MCintegration\_SimpleSampling\_ngpu.cu* and *MCintegration\_Metropolis\_ngpu.cu* respectively. The results of different number of sample points N for GPU with proper block size and grid size are basically the same as CPU.

The integration results are already shown in previous section, so I’ll discuss more on GPU’s performance.

1. **Discussion**
2. Implementation of getting random number for individual threads.

For simplicity, I send current time plus the OpenMP threads number times whatsoever to GPU, and then plus its own unique id as seed, to initialize cuda random number generator.

For this I assume that each seeds are not correlated, if they are, then this will affect the results of Monte Carlo Integration. From the results, it seems that they don’t, so I guess that’s OK.

1. Implementation of Simple Sampling and Metropolis Algorithm
2. Simple Sampling

Since the repeated part is generating , computing and , and finally, sum them up respectively. So the strategy I take:

1. Let all the threads compute their own and
2. Add to threads’ i.
3. Go to next round(i.) if current i is smaller than N.
4. Finally, do the parallel reduction for each block.
5. Pass the mean and sigma result back to CPU, and do some further computation.
6. Metropolis Algorithm

The structure is generally the same as Simple Sampling, but with dividing the weight function, and determine the acceptance.

So basically, we only have to change step i. to the metropolis algorithm, and the rest is the same.

1. The optimal block size and grid size for Simple Sampling(SS)

Speed up rate of SS-1GPU-65536:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | Simple Sampling 1GPU | | | | | | | | | |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|  | 1.69 | 4.14 | 7.68 | 14.11 | 22.86 | 32.33 | 36.62 | 44.00 | 35.29 | 22.89 |
|  | 15.86 | 25.42 | 34.46 | 41.89 | 43.80 | 37.89 | 24.38 | 11.06 | 5.02 |  |
|  | 17.67 | 16.94 | 15.31 | 13.20 | 9.33 | 4.05 |  |  |  |  |
|  | 2.33 | 1.55 |  |  |  |  |  |  |  |  |

Speed up rate of SS-2GPU-65536:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | Simple Sampling 2GPU | | | | | | | | | |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|  | 2.13 | 7.30 | 13.27 | 19.95 | 27.68 | 32.10 | 41.07 | 40.66 | 33.49 | 21.38 |
|  | 21.17 | 26.27 | 32.95 | 38.37 | 37.48 | 32.57 | 20.61 | 9.82 |  |  |
|  | 19.05 | 15.50 | 13.82 | 12.57 | 8.87 |  |  |  |  |  |
|  | 2.27 |  |  |  |  |  |  |  |  |  |

1. It seems that time used to run through sample points and are the same, so 2GPU did not accelerate here.
2. We know that the optimal block size and grid size are determined by the computational strength. The optimal block size and grid size for Simple Sampling is where (Threads Per Grid) falls between 1600 to 3200. This is the same in either 1GPU or 2GPU, since it is computational strength that matters.

Speed up rate of SS-1GPU-81920000:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | Simple Sampling 1GPU | | | | | | | | | |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|  | 2.58 | 5.14 | 10.20 | 20.16 | 39.29 | 77.40 | 144.86 | 262.05 | 419.92 | 539.65 |
|  | 24.16 | 45.99 | 97.80 | 169.56 | 388.32 | 558.24 | 667.04 | 597.00 | 622.63 | 540.56 |
|  | 46.78 | 93.50 | 152.28 | 344.78 | 578.59 | 659.92 | 582.54 | 410.43 | 20.07 | 6.81 |
|  | 53.45 | 98.21 | 181.25 | 328.02 | 76.74 | 13.64 | 5.12 | 2.26 | 0.90 | 0.33 |

Speed up rate of SS-2GPU-81920000:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | Simple Sampling 2GPU | | | | | | | | | |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|  | 5.13 | 10.18 | 20.17 | 39.15 | 75.52 | 137.37 | 269.15 | 479.54 | 790.93 | 934.48 |
|  | 47.13 | 97.64 | 163.94 | 384.59 | 585.71 | 961.03 | 1106.07 | 991.91 | 897.60 | 799.41 |
|  | 79.54 | 154.88 | 330.17 | 498.55 | 988.83 | 1007.39 | 864.79 | 351.20 | 20.16 | 6.79 |
|  | 106.09 | 173.98 | 326.14 | 455.17 | 75.65 | 13.61 | 5.10 | 2.25 | 0.90 | 0.33 |

1. With a proper block size and grid size, 2GPU accelerates for almost two times faster. The workloads for running through and are so different that it reflects on the speed up rate in 1GPU and 2GPU.
2. The optimal block size and grid size are the same in 1GPU and 2GPU. I marked them with yellow blocks.
3. The speed up rate tables in 1GPU and 2GPU are alike, since it is computational strength that matters.
4. The optimal block size and grid size for Metropolis Algorithm(MA)

Speed up rate of MA-1GPU-65536:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | Metropolis Algorithm 1GPU | | | | | | | | | |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|  | 6.41 | 15.00 | 28.12 | 50.33 | 79.55 | 111.01 | 131.73 | 144.33 | 123.32 | 100.35 |
|  | 57.58 | 88.52 | 116.22 | 141.50 | 143.03 | 134.06 | 98.05 | 43.37 | 21.65 |  |
|  | 64.11 | 64.87 | 61.69 | 54.73 | 38.45 | 16.74 |  |  |  |  |
|  | 9.39 | 6.11 |  |  |  |  |  |  |  |  |

Speed up rate of MA-2GPU-65536:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | Metropolis Algorithm 2GPU | | | | | | | | | |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|  | 7.93 | 27.21 | 46.44 | 77.59 | 105.96 | 127.16 | 139.25 | 139.07 | 106.65 | 87.45 |
|  | 82.59 | 101.18 | 114.58 | 138.54 | 117.58 | 128.33 | 80.95 | 40.25 |  |  |
|  | 64.25 | 62.56 | 55.17 | 52.25 | 35.01 |  |  |  |  |  |
|  | 9.48 |  |  |  |  |  |  |  |  |  |

1. Again, time used for computing and are almost the same, so 2GPU did not accelerate, even though Metropolis Algorithm is more complicated than Simple Sampling.
2. The optimal block size and grid size for 1 and 2GPU are slightly different, but their distribution are the same, see the yellow blocks.

Speed up rate of MA-1GPU-81920000:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | Metropolis Algorithm 1GPU | | | | | | | | | |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|  | 9.50 | 18.84 | 37.47 | 74.10 | 147.30 | 281.65 | 501.89 | 994.57 | 1646.14 | 2013.32 |
|  | 89.48 | 176.60 | 328.99 | 631.75 | 1205.47 | 2030.02 | 2474.78 | 2210.94 | 2460.58 | 2087.50 |
|  | 177.82 | 325.57 | 616.10 | 1059.47 | 1586.86 | 2334.31 | 2274.44 | 1823.69 | 92.02 | 31.08 |
|  | 190.73 | 352.30 | 650.45 | 1169.34 | 341.62 | 62.05 | 23.25 | 10.31 | 4.12 | 1.51 |

Speed up rate of MA-2GPU-81920000:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BlockSize  GridSize | Metropolis Algorithm 2GPU | | | | | | | | | |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |
|  | 20.59 | 40.59 | 80.93 | 159.39 | 303.86 | 562.70 | 1068.01 | 1979.43 | 3256.52 | 3926.42 |
|  | 189.83 | 347.28 | 670.38 | 1251.53 | 2406.64 | 3986.20 | 3101.18 | 3672.04 | 2963.49 | 3196.62 |
|  | 339.69 | 645.22 | 1236.40 | 2136.46 | 3968.33 | 4113.02 | 3838.92 | 2358.98 | 101.72 | 34.32 |
|  | 389.10 | 722.36 | 1317.53 | 1936.87 | 385.51 | 68.29 | 25.73 | 11.37 | 4.54 | 1.66 |

1. 2GPU accelerates for almost 2 times the speed up rate of 1GPU.
2. The optimal block size and grid size for 1GPU and 2GPU are in the yellow blocks. We can see that they are different, because that calculate and of Metropolis Algorithm is indeed a lot different than in Simple Sampling. So the optimal settings are different.
3. Errors and their time used compared with GPU

I compared the results from optimal block size and grid size only. Defined the error as

how much mean value calculate by GPU is different from CPU.

Errors:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Simple Sampling | | Metropolis Algorithm | |
| Sample Points N | 65536 | 81920000 | 65536 | 81920000 |
| 1GPU |  |  |  |  |
| 2GPU |  |  |  |  |

1. Cause Monte Carlo integration is determined by the random numbers at each run, so the more sample points, it should be more close to the result from CPU.
2. Numbers of GPU used basically won’t make the error smaller, which should be the case in Simple Sampling. And as the sample points becomes larger, the error is smaller.
3. For Metropolis Algorithm, the algorithm requires a series of random number such that they are not uniform. But the way I implement it makes each threads start their own series, which means by increasing sample points, the error won’t decrease that much as in Simple Sampling.

Time used (ms):

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Simple Sampling | | Metropolis Algorithm | |
| Sample Points N | 65536 | 81920000 | 65536 | 81920000 |
| 1GPU | 4.4 | 59.8 | 4.7 | 73.0 |
| 2GPU | 9.0 | 35.1 | 10.4 | 45.4 |
| CPU | 31.0 | 38153.2 | 136.6 | 173681.2 |

1. Metropolis Algorithm spends more time than Simple Sampling, as it spends some time to test for the acceptance.
2. Compared from the error, time used, and the implementation, I think it is better to use Simple Sampling when doing integration. Because determining the weight function is time consuming and the implementation of Metropolis Algorithm in GPU is not that ideal (in my case), and the worst of all is that as sample points increase, the error won’t drop that much as in Simple Sampling.