Simple Monte Carlo Integration by ArrayFire Programming Model

Morteza Namvar

September 21, 2021

1 Introduction

In this report, the vector programming is investigated by using an open-source library named ”ArrayFire”. ArrayFire has been released in 2007 particularly for parallel computing. Thanks to using ArrayFire, it is possible to develop code without considering the parallel side and also using a unique code for executing it on both CPU and GPU. To do the investigation the Monte-Carlo integration approach is used in this report.

Suppose that we pick N random points, uniformly distributed in a multidimensional volume V. Call them $x_1, ..., x_N$. Then the basic theorem of Monte Carlo integration estimates the integral of a function f over the multidimensional volume,

$$\int f \, dV = V\langle f\rangle \pm V\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}} \tag{1}$$

Here the angle brackets denote taking the arithmetic mean over the N sample points,

$$\langle f\rangle = \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{2}$$

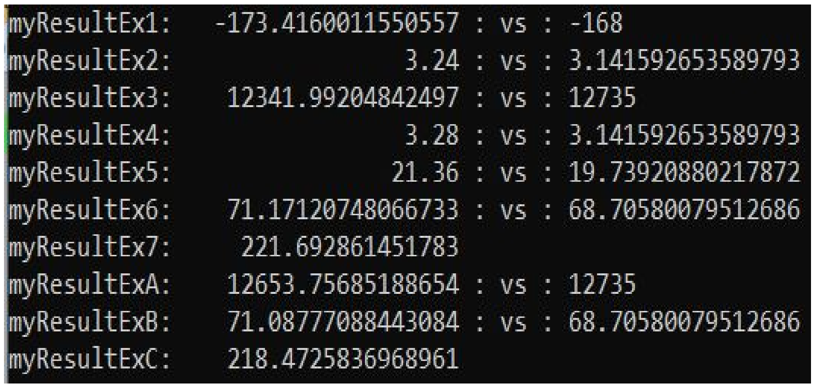
$$\langle f^2\rangle = \frac{1}{N}\sum_{i=1}^{N} f^2(x_i) \tag{3}$$

The ”plus-or-minus” term in Eq. 1 is a one standard deviation error estimate for the integral, not a rigorous bound; further, there is no guarantee that the error is distributed as a Gaussian, so the error term should be taken only as a rough indication of probable error.

Suppose that you want to integrate a function g over a region W that is not easy to sample randomly. For example, W might have a very complicated shape. No problem. Just find a region V that includes W and that can easily be sampled, and then define f to be equal to g for points in W and equal to zero for points outside of W (but still inside the sample V). You want to try to make V enclose W as closely as possible because the zero values of f will increase the error estimate term of Eq. 1.

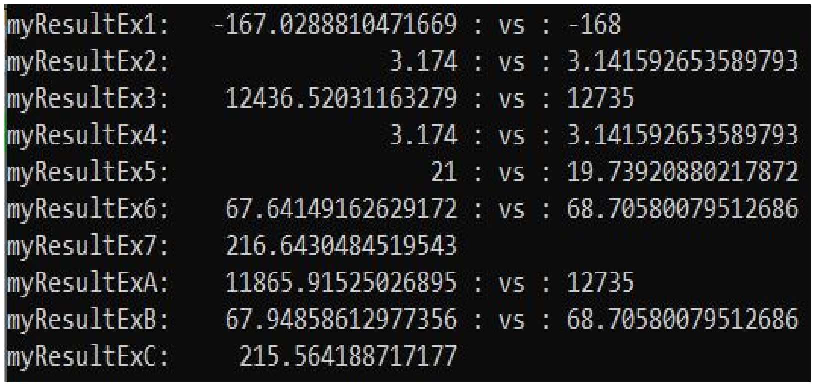
2 The values computed by the program

To verify the implementation of the code, the results obtained for different batch sizes are illustrated in Fig. 1. As depicted in this figure, the minimum points required to have an acceptable result is about 10e4, as shown in Fig. 1c. Hence, more increase in the number of nodes to calculate the integral doesn’t result in more accuracy, as illustrated in Fig. 1d.



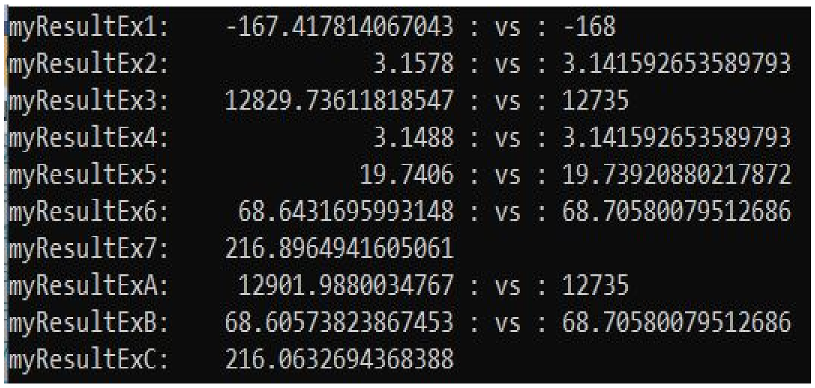
```
myResultEx1:   -173.4160011550557 : vs : -168
myResultEx2:                 3.24 : vs : 3.141592653589793
myResultEx3:    12341.99204842497 : vs : 12735
myResultEx4:                 3.28 : vs : 3.141592653589793
myResultEx5:                21.36 : vs : 19.73920880217872
myResultEx6:    71.17120748066733 : vs : 68.70580079512686
myResultEx7:     221.692861451783
myResultExA:    12653.75685188654 : vs : 12735
myResultExB:    71.08777088443084 : vs : 68.70580079512686
myResultExC:    218.4725836968961
```

(a) nBatch = 100.



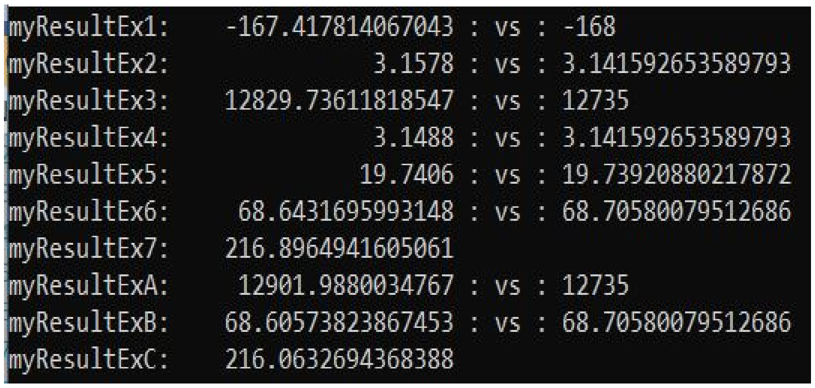
```
myResultEx1:   -167.0288810471669 : vs : -168
myResultEx2:                3.174 : vs : 3.141592653589793
myResultEx3:    12436.52031163279 : vs : 12735
myResultEx4:                3.174 : vs : 3.141592653589793
myResultEx5:                   21 : vs : 19.73920880217872
myResultEx6:    67.64149162629172 : vs : 68.70580079512686
myResultEx7:    216.6430484519543
myResultExA:    11865.91525026895 : vs : 12735
myResultExB:    67.94858612977356 : vs : 68.70580079512686
myResultExC:     215.564188717177
```

(b) nBatch = 1000.



```
myResultEx1:    -167.417814067043 : vs : -168
myResultEx2:               3.1578 : vs : 3.141592653589793
myResultEx3:    12829.73611818547 : vs : 12735
myResultEx4:               3.1488 : vs : 3.141592653589793
myResultEx5:              19.7406 : vs : 19.73920880217872
myResultEx6:     68.6431695993148 : vs : 68.70580079512686
myResultEx7:    216.8964941605061
myResultExA:     12901.9880034767 : vs : 12735
myResultExB:    68.60573823867453 : vs : 68.70580079512686
myResultExC:    216.0632694368388
```

(c) nBatch = 10000.



```
myResultEx1:    -167.417814067043 : vs : -168
myResultEx2:               3.1578 : vs : 3.141592653589793
myResultEx3:    12829.73611818547 : vs : 12735
myResultEx4:               3.1488 : vs : 3.141592653589793
myResultEx5:              19.7406 : vs : 19.73920880217872
myResultEx6:     68.6431695993148 : vs : 68.70580079512686
myResultEx7:    216.8964941605061
myResultExA:     12901.9880034767 : vs : 12735
myResultExB:    68.60573823867453 : vs : 68.70580079512686
myResultExC:    216.0632694368388
```

(d) nBatch = 100000.

Figure 1: The result of Monte-Carlo integral computation different sizes on points. The quantity "nBatch" is the number of points to evaluate the implementation.

In the depicted results the Ex. 1 is a basic calculation for a 2D integral. The exact value for the below integral is -168, while the numerical estimate by Monte-Carlo is about -167.41, which is an acceptable result.

$$\int_{-4}^{5}\int_{-1}^{3} x^2 - y^2 \, dxdy$$

Based on the results illustrated in Fig. 1c the π number estimation is done in Ex. 2 is 3.1578 using 2D Monte-Carlo algorithm. However, using a 3D Monte-Carlo results in a more accurate result.

In Ex. 3, the below integral has been computed using Mont-Carlo 3D scheme. To do this, the "monteCarloIntegral3" is used. This function is only for 3D integral but, in Ex. A more general function named "monteCarloIntegralN" was applied, which could be used for any dimensions.

$$\int_{-7}^{8}\int_{-4}^{5}\int_{-1}^{3} x^2 - y^2 + z^3 \, dxdydz$$

To verify the implementation of the function "monteCarloIntegralN" the results of Ex. 6 and Ex.7, which are 4D integrals, compared to the results using the "monteCarloIntegralN" function. As could be seen from the results, the implementation of these functions has been verified as the results are given.

3 The "monteCarloIntegralN" function

In this section, the execution time of the "monteCarloIntegralN" function is investigated. All of the simulations were executed with double-precision floating-point numbers. The computer used for the simulations was equipped with an Intel(R) Core(TM)i7-7700 @2.80GHz processor and a 16GB of main memory and the graphic card was GTX-1080 with 6 GB of memory.

To evaluate the performance of "monteCarloIntegralN" function, its execution time is compared to the "monteCarloIntegral3" and "monteCarloIntegral4" functions. To do that, the number of points to calculate the integrals is selected to 100,000. And to estimate the performance, an average of the execution time of each iteration is used. As can be seen from Table 1, the execution time of "monteCarloIntegralN" is more than "monteCarloIntegral3" and "monteCarloIntegral4" for both the CPU and CUDA backend. However, this downgrade for the OpenCL backend is much as CUDA and CPU backend. The more interesting advantage of using ArrayFire is a considerable speed-up obtained from GPU as a computing device. As can be seen from Table 1, using CUDA and OpenCL backend decrease the execution time by about an order of magnitude.

Table 1: Comparison of execution time for CPU, CUDA, and OpenCL backend. Time is in second.

Exercise	**CPU**	**CUDA**	**OpenCL**
monteCarloIntegral3	0.1112	0.0012	0.0018
onteCarloIntegralN	0.1307	0.0151	0.0014
monteCarloIntegral4	0.1415	0.0051	0.0043
onteCarloIntegralN	0.1755	0.0165	0.0025

After investigating the execution time, memory usage, as the other side of any computational code, should be studied. To do that, the memory usage of the ”monteCarloIntegral3” function is compared to the ”monteCarloIntegralN” which are used in Ex. 7 and Ex. C respectively. In this simulation, nBatch=10e6 has been used. As could be seen in Table 2. the memory consumption of ”monteCarloIntegralN” is about three times as ”monteCarloIntegral4”. One probability of the increment in the memory usage could be the usage of the ”tile” function in the ArrayFire.

Table 2: Comparison of memory usage for CPU backend.

Function name	**Memory usage in MB for CPU backend**
monteCarloIntegral4	64
onteCarloIntegralN	171

4 Advantages and drawbacks of array programming

The most interesting aspect of ArrayFire is the possibility to have a unique code to be executed both on CPU and GPU. In addition, the parallel part of programming is completely hidden as all the computations are done by using the ”array” object of the ArrayFire library. At the same time using this programming paradigm could result in more memory usage mainly when the ”tile” function is used for vectorization.