GPU implementation of VEGAS algorithm for adaptive multidimensional integration

Riccardo Galbiati

Abstract

VEGAS is a powerful algorithm for multidimensional integration. Its efficiency lies in being an iterative and adaptive Monte Carlo scheme.
The structure of the algorithm is perfect for a parallel GPU implementation, which allows us to obtain a great speedup in the execution time of the computation.
This algorithm is presented and the efficiency of its parallel implementation shown.

1 Introduction

With the term Monte Carlo methods we refer to a broad set of algorithms which hinge on the use of random sampling to obtain a numerical result.
Throughout this article, we will refer to Monte Carlo integration, i.e. the use of these techniques in order to obtain a numerical estimation of a definite integral in a multidimensional domain. The main difference with other approximation techniques (for example quadrature methods) is the fact that Monte Carlo evaluates the function at random points (while quadrature methods normally use a regular-spaced grid).

Nevertheless, the algorithm we consider here has another great feature: it can adapt to the integration domain we are considering in order to contrast the curse of dimensionality. This adaptability will be achieved through two different techniques: the importance sampling and the stratified sampling.

The whole implementation is realized in a parallel way, so to harness the parallel power of a GPU in a CUDA environment and obtain a great enhancement of our performances.

We will start by briefly explaining the problem we are considering and showing what a Monte Carlo integration is. We will show our adaptive algorithm in a single dimension and then generalize the problem. At the end, we will consider the differences in term of performances for the different realizations.

2 Monte Carlo Integration

We will consider - without any loss of generality - a mono dimensional problem restricted to $[0, 1]$ domain. It will be then extended to multidimensional functions.
Our problem is simple: we want to find the value of the defined integral of some function $f(x)$, i.e.:

$$I = \int_0^1 f(x)dx. \tag{1}$$

In order for our problem to be interesting, we obviously imagine that we are not able to obtain an analytical expression for the primitive of $f(x)$. For the sake of generality, we can think that the function is not even continuous on our domain.
Our purpose is thus to obtain an estimation for eq.(1).

The simplest approach we can think of is represented by the quadrature methods. The idea is that we can equally divide our domain in n intervals (each one with a step size $h = \frac{1}{n}$). For each interval, we somehow estimate the value of the area covered by the function (i.e. we perform a quadrature) and then we add up all the contributions and obtain an estimation of I. Since we have different methods (rectangle rule, trapezoidal rule, Simpson's rule) to approximate the area, we can only write a general expression:

$$\hat{I} = \sum_{i=1}^{n} w_i f(x_i) \tag{2}$$

where x_i are all the points where we evaluated our function and w_i are the weights of each contribution. Each quadrature method relies on a different approximation of the area, so each has an error affecting the overall estimation. This error, in general, decreases as the number of the steps n is increased. If we are considering a 1D function, the error reduces rapidly as n increases; bigger problems arise when we are considering higher dimensions.
In fact, we can extend the approximation we considered to a s-dimensional function:

$$\hat{I} = \sum_{i_1=1}^{n} \sum_{i_2=1}^{n} ... \sum_{i_s=1}^{n} w_{i_1} w_{i_2} ... w_{i_s} f(x_{i_1}, x_{i_2} ... x_{i_s}). \tag{3}$$

We use quadrature rules with n samples in each dimension, thus requiring n^s samples in total.
Under these circumstances, the main drawback of quadrature methods arises.
If we consider a 1D function, the error bound is $O(n^{-r})$ with $r = 2$ or 4 depending on the quadrature method. However, if we consider a s-dimensional function and use n^s samples, the error bound is $O(n^{-r/s})$; so the larger the number of dimensions, the less negligible the error. Sometimes we refer to this and other problems with the expression 'curse of dimensionality'. Basically, the error grows larger than our ability to increase samples number.
Because of this problem, quadrature methods are not efficient when dealing with higher dimensions. Monte Carlo integration, on the other hand, has the property that the error bound is always $O(n^{-1/2})$, regardless of dimensionality. That is the reason why we use it for multidimensional integration.

In order to understand Monte Carlo integration, we first consider a random variable F_n built from the random variable X; F_n is defined as:

$$F_n = \frac{1}{n} \sum_{i=1}^{n} \frac{f(X_i)}{p(X_i)} \tag{4}$$

where $p(X_i)$ is an arbitrary distribution on the interval $[0, 1]$. Intuitively, the meaning of F_n is that if we draw more samples in some area of our domain, their weights must be scaled down; on the other hand, if we draw a few examples in some other area, their weights must be scaled up. This adjustment accounts for the area they represent together.

We now want to find what the expected value of the new random variable F_n is:

$$\begin{aligned} E[F_n] &= E\left[\frac{1}{n} \sum_{i=1}^{n} \frac{f(X_i)}{p(X_i)}\right] \\ &= \frac{1}{n} \sum_{i=1}^{n} E\left[\frac{f(X_i)}{p(X_i)}\right] \\ &= \frac{1}{n} \sum_{i=1}^{n} \int_0^1 \frac{f(x)}{p(x)} p(x) dx \\ &= I. \end{aligned}$$

This is the result we want, because it implies:

$$F_n \to I \quad as \quad n \to \infty. \tag{5}$$

Furthermore, the variance of F_n is given by:

$$\begin{aligned}
V[F_n] &= V\left[\frac{1}{n}\sum_{i=1}^{n}\frac{f(X_i)}{p(X_i)}\right] \\
&= \frac{1}{n^2}\sum_{i=1}^{n} V\left[\frac{f(X_i)}{p(X_i)}\right] \\
&= \frac{1}{n^2} n V\left[\frac{f(X)}{p(X)}\right] \\
&= \frac{E\left[\left(\frac{f(x)}{p(x)}\right)^2\right] - E^2\left[\left(\frac{f(x)}{p(x)}\right)\right]}{n}.
\end{aligned}$$

Starting from this the fact that the error decreases as $O(n^{-1/2})$ when we increase the number of samples can be easily shown.

We introduce the term $F_n^{(1)}$:

$$F_n^{(1)} = \frac{1}{n}\sum_{i=1}^{n}\frac{f(X_i)}{p(X_i)} \tag{6}$$

and the approximated value (for n large) for the variance:

$$\sigma^2 \simeq \frac{F_n^{(2)} - \left(F_n^{(1)}\right)^2}{n-1}, \tag{7}$$

where $F_n^{(2)}$ is:

$$F_n^{(2)} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{f(X_i)}{p(X_i)}\right)^2 \tag{8}$$

3 An adaptive algorithm

Given this theoretical structure, we now want to develop a technique which allows us to reduce the variance σ^2 for a fixed number n of samples. We will chiefly focus on two methods, which are among the most used: the importance sampling and the stratified sampling.

3.1 Importance sampling

What is the best distribution $p(x)$ to use in the expression (4) for our sampling? The best choice is simply to sample according to the function itself. The optimal choice can be shown to be:

$$p(x) = \frac{|f(x)|}{\int_0^1 |f(x)|}, \tag{9}$$

i.e. function evaluations are mainly localized where the function itself is largest in magnitude. This is the optimal choice because it leads to $V[F_n] = 0$: a zero variance estimator.

Obviously, this is hard to implement in an actual algorithm because it needs a detailed knowledge of the integrand before the integration itself is performed. Nevertheless, we will show an adaptive approach which will heavily rely on this idea.

3.2 Stratified sampling

Another idea to reduce the variance is to divide the integration domain in M subvolumes and perform a Monte Carlo integration in each of them. We can then vary the relative sizes and positions of the subvolumes in order to obtain an identical contribution to the variance from each one of them (σ^2/M).
This technique thus leads to a large amount of function evaluations in the areas where the potential error is maximum, namely where the function varies rapidly.

We will implement both of these techniques in order to understand which one is better suited for the needs of multidimensional integration up to 9 dimensions.
In order to overcome the aforementioned problems relative to an a priori knowledge of the behavior of the integrand, we will consider a mock starting condition and then adapt it with a precise rule.

4 What is VEGAS?

4.1 VEGAS in one dimension

As stated before, we are considering a 1D integral as in eq.(1).
Our algorithm will consist of two moments: a first part in which we will adjust the grid of subvolumes according to a certain rule and a second part when we will perform a straightforward Monte Carlo integration with the new grid. The differences between importance and stratified sampling will emerge only in the first part.

In order to avoid the main drawback of the aforementioned sampling techniques (i.e. the fact that a detailed a priori knowledge of the integrand is needed) we start by considering a mock uniform probability density $p(x) = 1$; the domain is divided in M identical subintervals. We will take advantage of the evaluations of our function to refine the probability density and obtain an empirical variance reduction; this goal will be reached by the resizing of the subvolumes (or, in the 1D case, the subintervals) we originally divided our domain into.
In order for us to have a simple mathematical picture, we will consider the probability density to be a step function with M steps. For each one of these intervals the probability of a random number to be chosen from them will be the same, and it will be then constrained by the condition:

$$p(x) = \frac{1}{M\Delta x_i}, \tag{10}$$

where Δx_i is the width of the $i-th$ interval. For the initial condition, where $\Delta x_i = \frac{1}{M}\ \forall i$, this evidently means that $p(x) = 1$ as stated before.

This is the core of the whole adaptive algorithm idea: the probability distribution will be adapted simply by adjusting the increment sizes Δx_i and then modifying $p(x)$ according to eq.(10). The intervals will be shrunk where the integrand has an erratic behavior and will be expanded where it is almost constant. The rule according to which we will adjust the grid is exactly one of the aforementioned importance sampling or stratified sampling. In both cases, the number of steps M will be held constant throughout the process of adjustment because of storage space reasons. If we consider the importance sampling, the rule we follow when we adapt the $i-th$ interval in the $k-th$ cycle is:

$$\Delta x_i^{(k)} = \left[\log\left(1 + \Delta x_i^{(k-1)} \frac{\frac{1}{M}}{\frac{\bar{f}_i \Delta x_i}{\sum_j \bar{f}_j \Delta x_j}}\right)\right]^{\alpha} \tag{11}$$

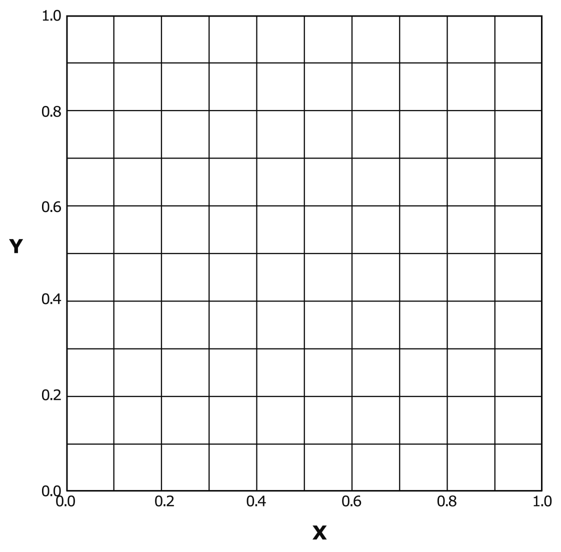

Figure 1: Starting point for the adjustment of the 2D grid of a strongly peaked function.

where the parameter α determines the rate of convergence and is typically set between 0.2 and 2, and:

$$\overline{f_i} = \sum_{x \,\in\, x_i - \Delta x_i}^{x_i} |f(x)| \,. \tag{12}$$

In this way, each interval is resized according to is contribution to the total integral. Please note that after this process we need to remap all the intervals in the original domain (in this case $[0, 1]$). The new grid can then be refined again with a new cycle, and so on until the optimal grid has been obtained. This behavior will be reached when the contribution to the total area $\frac{\overline{f_i}\Delta x_i}{\sum_j \overline{f_j}\Delta x_j}$ is almost the same for each interval: that is exactly the meaning of the term $\frac{1}{M}$ at the numerator of eq.(11). If we consider the stratified sampling, the rule for adapting the grid is really similar. Given the rule (7) for calculating the variance in each interval, the only difference lies in the fact that we resize the intervals on the base of the contribution given to the total variance, i.e.:

$$\Delta x_i^{(k)} = \left[\log \left(1 + \Delta x_i^{(k-1)} \frac{\frac{1}{M}}{\frac{\sigma_i^2}{\sigma_{tot}}} \right) \right]^{\alpha} . \tag{13}$$

In both cases we typically need a cycle of K iterations to obtain our optimal grid; after it, we can just resort to a Monte Carlo integration where the uncertainty is proportional to $O(n^{-1/2})$.

4.2 VEGAS in s dimensions

The algorithm just presented can be generalized in order to deal with a s-dimensional function. If we consider the 2D case, we see that the trick is simply to consider the simplest probability density we can think of, i.e.:

$$p(x, y) = p_x(x)p_y(y)$$

and this allows us to handle each dimension individually.
We can thus apply the algorithms (11) and (13) along each axis but for example $\overline{f_i}$ will be defined

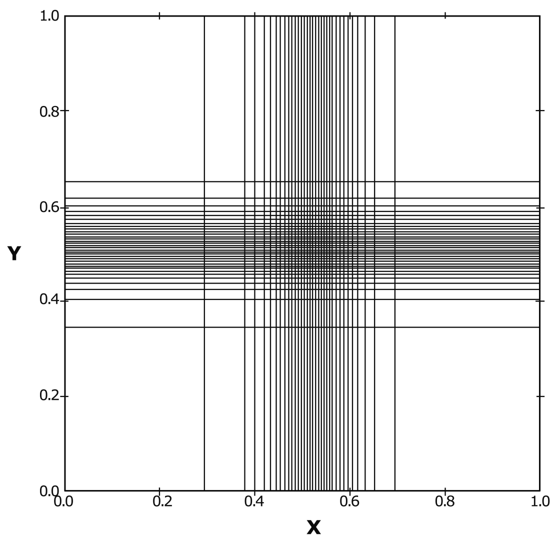

Figure 2: Optimal configuration for the 2D grid of a strongly peaked function.

by:

$$\overline{f_i} = \sum_{y} \sum_{x\,\in\,x_i-\Delta x_i}^{x_i} |f(x,y)|$$

for the x axis and an analogous expression for y. Similarly, for stratified sampling we consider the contribution to the total variance along each axis for a fixed interval.

In fig.1 and fig.2 we can see the process by which we obtain the optimal grid for a 2D integrand with a strong peak in the center of the domain.

5 Algorithm implementation on a GPU

The algorithm we just presented is particularly suitable for an implementation on a GPU because of the highly parallel nature of the operations we are going to perform (generation of a large number of random samples, independent evaluation of the function in each subvolume).
In order to test our program, we chose two different integrands: the first one shows an oscillatory behavior:

$$I_{osc} = \int_0^1 dx_1 \int_0^1 dx_2 \int_0^1 dx_3 \;\; \pi sin(\pi x_1)\, \pi sin(\pi x_2)\, \pi sin(\pi x_3), \tag{14}$$

while the second one has a single peak in the center of the integration domain:

$$I_{Gauss} = \left(\frac{1}{a\sqrt{\pi}}\right)^s \int_0^1 d^s x \;\exp\left(-\sum_{i=1}^{s} \frac{(x_i-0.5)^2}{a^2}\right) \tag{15}$$

where $a = 0.1$ and $s = 5, 7, 9$.
We compared the results for a CPU and a GPU implementation and reported the results in the table 1 and 2.
In fig.3 and fig.4 we represented visually the magnitude of the speedup obtained with the GPU implementation.

	Oscillatory3D	Gaussian5D	Gaussian7D	Gaussian9D
Dimensions	3	5	7	9
N. evaluations / iteration	500’000	30’000’000	100’000’000	100’000’000
Exact result	8	1	1	1
Computed value	7.999	1.004	0.998	0.995
Error σ	0.001	0.005	0.003	0.006
CPU execution time (secs)	8	44	172	1412
GPU execution time (secs)	0.2	1.8	3.2	11.3
Speedup	40	24	54	125

Table 1: Importance sampling performances comparison: CPU vs GPU

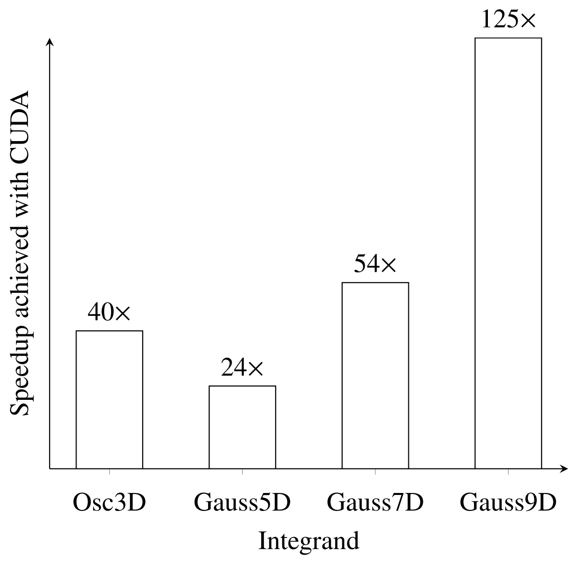

Figure 3: Speedup achieved on the GPU for the importance sampling.

6 Conclusion

We just showed that an adaptive Monte Carlo integration is much effective, especially in high dimensions where a simple generalization of the quadrature methods we use in 1D is not feasible. Even for an integrand as difficult as we can think of, we are able to obtain efficiently the 3 or 4 significant digits we usually need in science.

	Oscillatory	Gaussian5D	Gaussian7D	Gaussian9D
Dimensions	3	5	7	9
N. evaluations / iteration	500’000	30’000’000	100’000’000	100’000’000
Exact result	8	1	1	1
Computed value	8.0008	0.997	0.996	0.993
Error σ	0.001	0.005	0.004	0.008
CPU execution time (secs)	8	46	180	1452
GPU execution time (secs)	0.2	1.7	3.3	12.3
Speedup	40	27	54	118

Table 2: Stratified sampling performances comparison: CPU vs GPU

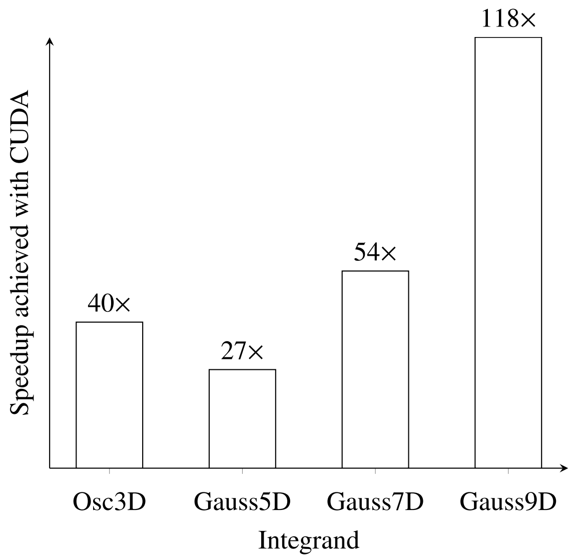

Figure 4: Speedup achieved on the GPU for the stratified sampling.