FAST ADAPTIVE SAMPLING TECHNIQUE FOR MULTI-DIMENSIONAL INTEGRAL ESTIMATION USING GPUS

Pradeep Rao
Infosys Limited
Bangalore, India
pradeep_rao@infosys.com

Foy Nathanaël
Mines Paristech
France
09foy@ensmp.fr

Sayantan Mitra
Infosys Limited
Bangalore, India
sayantan_mitra01@infosys.com

G. N. Srinivasa Prasanna
International Institute of Information Technology
Bangalore, India
gnsprasanna@iiitb.ac.in

Abstract:

Evaluating multi-dimensional integrals is a commonly encountered problem in many areas of science including Physics and Volume estimation of convex bodies. One of the widely used techniques for integral evaluation in large dimensions is the Monte Carlo method. Vanilla Monte Carlo methods of Integral Estimation use uniform sampling techniques. Variance of such uniform sampling reduces as 1/√Sample-size, which is too slow for most real life applications. In this study, we discuss about an adaptive sampling technique called VEGAS which reduces the variance at a much faster rate than uniform sampling. We present a new parallel implementation for VEGAS based on CUDA that can significantly reduce the computation time of multi-dimensional integrals. We show that our GPU based implementation of VEGAS achieves up to a 45x speed up over an equivalent CPU based implementation.

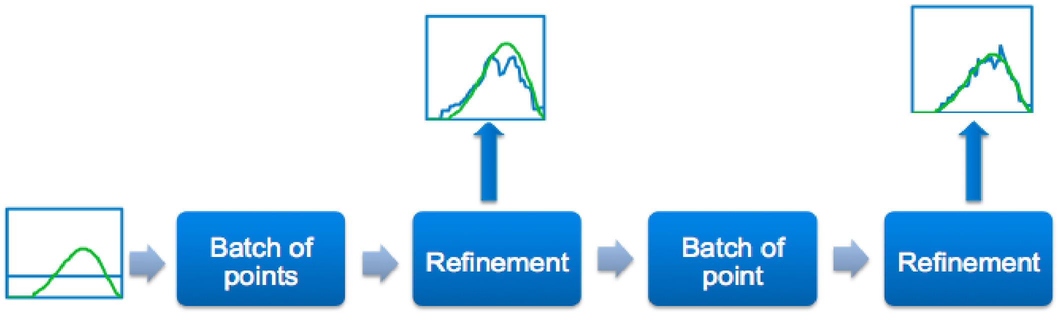
1. Introduction

Evaluating multi-dimensional integrals is a commonly encountered problem in many areas of science such as Physics and in Volume estimation of convex bodies. In particle Physics, examples include; calculation of scattering amplitudes using Feynman perturbation theory[6] and in calculating mean lifetime of muon particles[7].

One of the major problems in multidimensional integration estimation is the exponential growth of the samples with increasing dimension of the integration volume. One of the widely used techniques for integral evaluation is the Monte Carlo method. The main advantage with this method is that its convergence rate is independent of the dimensionality of the integral. Vanilla Monte Carlo methods of multi-dimensional integral estimation use uniform sampling technique to generate random points in the domain of integration. The variance of estimation has convergence rate proportional to 1/√Sample-size. This means, greater the number of random points generated, greater is the accuracy of the estimation. But an obvious downside to large sample sizes is the increase in computation time. Various techniques are used to reduce the variance at a faster rate such as by increasing the concentration of random points in the region where integrand is largest in magnitude. Two most popular methods for variance reduction are Importance Sampling and Stratified sampling. However these methods require detailed knowledge of the integrand's behavior prior to implementation. Most often, such knowledge is not available and we need an algorithm that adaptively learns about the function and arrives at the suitable sampling distribution iteratively. In this paper, we will discuss about one such adaptive sampling technique called VEGAS, which reduces the variance at a much faster rate than vanilla Monte Carlo method. We will then present a new parallel implementation of VEGAS on GPUs using CUDA which reduces the computation time of multidimensional integrals significantly.

2. VEGAS – Adaptive Sampling

G.P.Lapage[1] has developed an iterative and adaptive Monte Carlo algorithm, called VEGAS that can be used for general purpose multi-dimensional integration. This algorithm initially starts with uniform sampling in the domain of integration and uses the information generated about the integrand in this step to iteratively improve the probability density for subsequent steps. This helps in gradually reducing the empirical variance over several iterations. It uses a combination of Importance and Stratified sampling techniques (though stratified mode can be disabled if not needed).



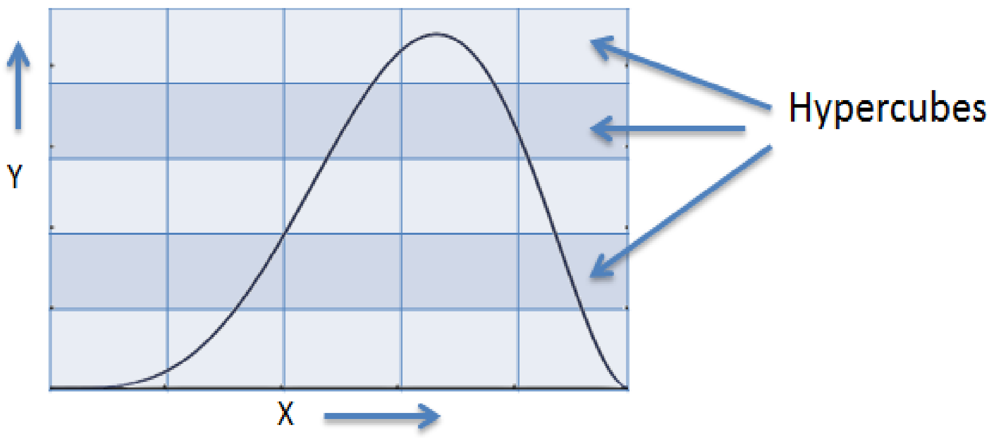
VEGAS Adaptive Sampling Workflow

2.1. Description of the density used in VEGAS

Theoretically, variance is minimized when the sample points generated, have maximum concentration in the region where integrand/function f(x) has the largest magnitude. This means, the new density p(x) should be

$$p(x) = \frac{|f(x)|}{\int dx\ |fx)|} \qquad (1)$$

In VEGAS, this approximation is done by dividing the integration volume into hypercubes using a rectangular grid.



Integration Volume divided into hypercubes

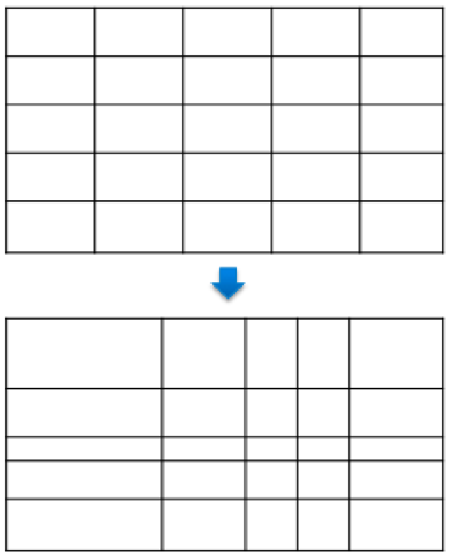
Random points are distributed so that the average number of points falling in any given hypercube is the same as in any other. In the first iteration, one starts with a uniform distribution. The probability distribution is then adjusted in the subsequent iteration according to the weights of the hypercubes or bins determined in the previous step. The weight r_i is determined by contribution of the hypercube to the total integral. It is calculated as below.

$$r_i = \left\{ \frac{1-\mathrm{w}_i}{log(\mathrm{w})_i} \right\} \alpha \qquad (2)$$

with

$$w_i \propto \sum_{\mathrm{x}_i \in bin} \frac{f(x)^2}{p(x)^2} \qquad (3)$$

The sizes of the hypercubes are adjusted after every iteration such that more hypercubes (and hence more sample points) are in the region where the integrand |f(x)| is largest.



Grid Refinement after Rebinning

In this fashion, an empirical variance reduction can be gradually achieved over several iterations.

3. CUDA implementation of VEGAS algorithm

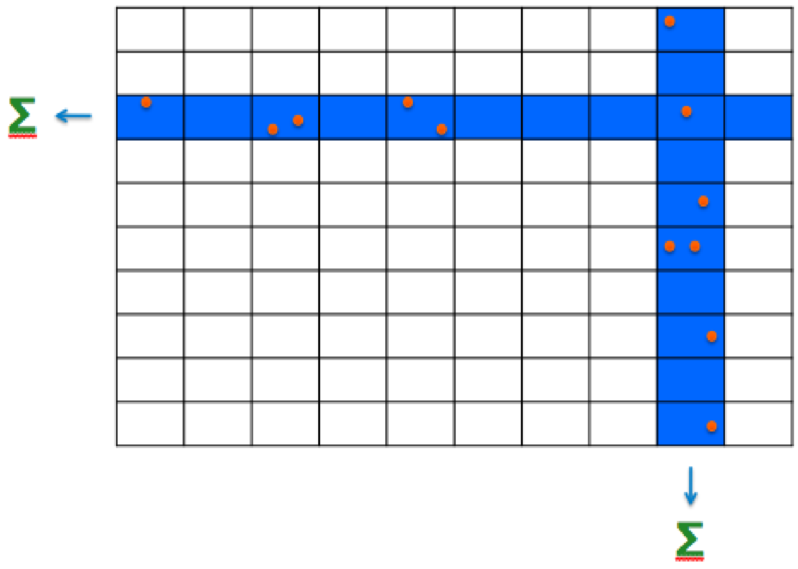
The original VEGAS algorithm is a sequential implementation. We have devised a GPU implementation of VEGAS using CUDA. We are yet to come across any published work that is related to porting the VEGAS algorithm to GPUs. Some of the earlier work for parallel implementation of VEGAS has been in a cluster environment such as the ones developed by Sinisa Veseli[8] and Richard Kreckel[9]. These are more applicable for clusters having compute nodes sharing the workload. They are based on MPI and have additional overhead of communication costs between the machines. Our approach is suitable for achieving massive parallelism on a single machine using GPUs though it can be extended to multi GPUs across machines.

We have followed a two pronged approach for parallelizing VEGAS algorithm. One is with an Importance Sampling only approach and the other as a combination of Importance and Stratified sampling. The reason for the segregation being the differences in the way random points are generated for these methods. In case of Importance Sampling, we sample more points in region where contribution to integrand is largest, whereas in case of Stratified sampling, more points are sampled in region where contribution to variance is largest. In both the cases, the accuracy of integral value is estimated for fixed confidence interval.

3.1. Parallel implementation with Stratified Sampling disabled

The first approach with the importance sampling only technique has been to simply dedicate one thread per random point. The methodology is as follows.

1. Set an initial sample size
2. Generate N CUDA threads, where N is the sample size
3. For every thread
   a. Generate a random point x_i.
   b. Identify the bin number in which the point falls
   c. Evaluate the function $f(x_i)$
   d. Calculate the cumulative sum of the integrals in every bin as shown in the diagram below. This is required to compute weight of the bin.
   A shared memory array variable is used to store the cumulative sum of the bins. Several threads may have to write to the same memory space when they belong to the same bin. For this purpose, we use atomic operations on the data in shared memory, to avoid concurrency problems.

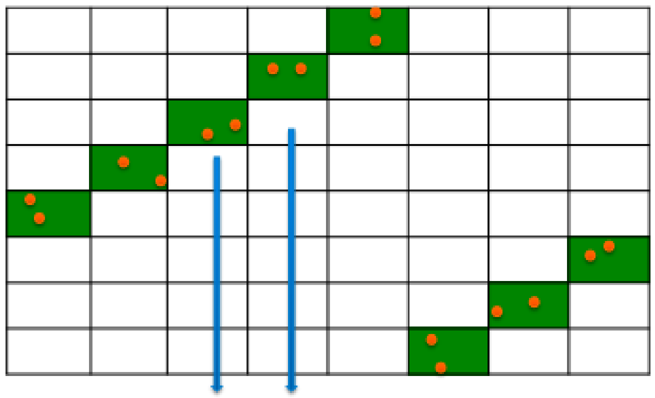


Sum reduction for every bin along each dimension

   e. The results for each bin are finally transferred back to the host thread
4. Compute the final weight of each bin as per equation (2).
5. Perform a re-bin operation and arrive at the new bin sizes based on the weights computed.
6. Repeat 1 -5 until the required confidence level is achieved.

3.2. Parallel implementation with Stratified Sampling enabled

The second approach has been to use stratified sampling. With this mode, the integration domain is divided in boxes and two points are generated for each box. In this case, we dedicate one thread to one box. The number of boxes generally exceeds the number of threads, so a loop guarantees that all boxes have been handled. We must go through all the boxes in a way that prevents two threads from writing to the same space memory. Therefore, the loop goes through the boxes crosswise as shown in the figure.



Vegas using CUDA with Stratified Sampling

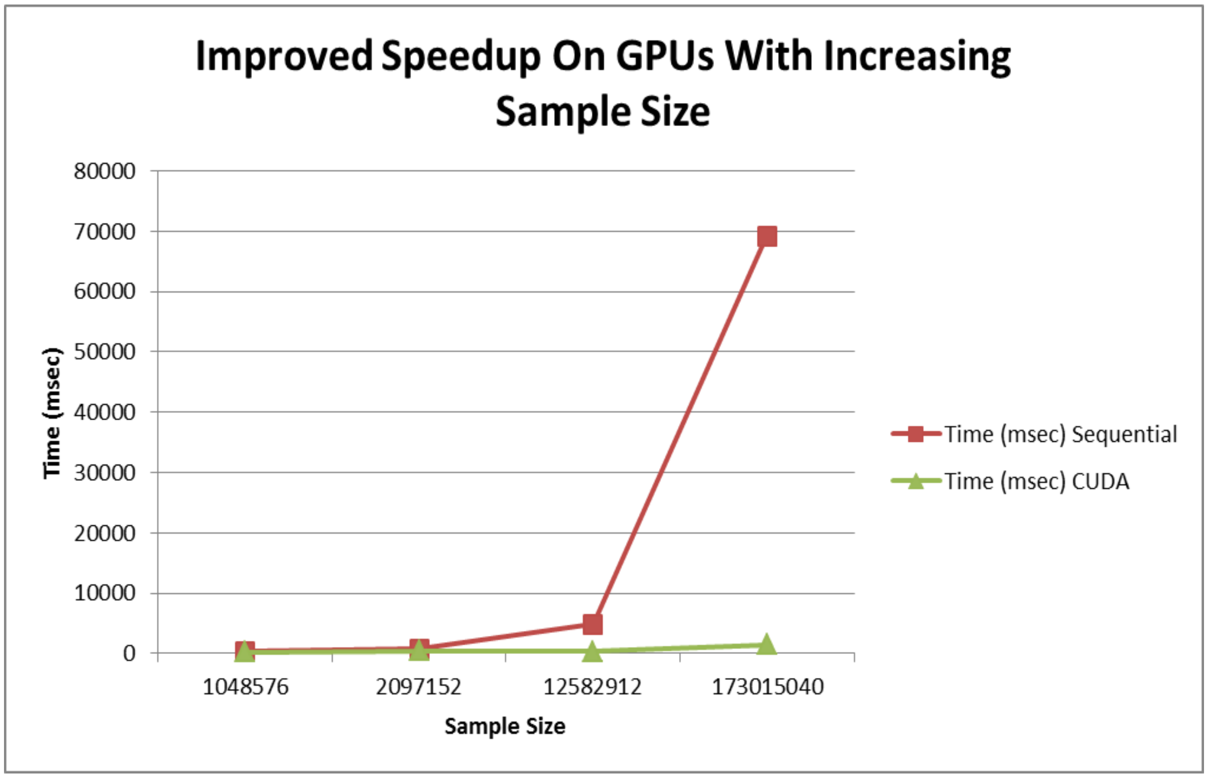
4. Results

We present our results on an NVIDIA GeForce GTX 480 Fermi GPU. The host is an Intel Xeon E5606 2.13GHz CPU running Linux Kernel 2.6.18. The host code has been compiled using gcc version 4.1.2 and the device code has been compiled using nvcc 4.0.

The results show a comparison in terms of computation time between a sequential version of the VEGAS code and the parallel implementation using different functions. Following test functions were used for comparison. The first function in the list is the one used in the original VEGAS paper by G.P.Lapage[1].

#	Integral	Sample Size	Time (msec)		SpeedUp
			Sequential	CUDA	
1	$\frac{100}{\pi}\int_0^1 dx_1 \int_{-1}^1 dx_2\, e^{-100(x_1{}^2+(x_2-1)^2}$	1048576	318	269	1.2
2	$\int_0^1 dx_1 \int_0^1 dx_2 \int_0^1 dx_3\, \sin(x_1\pi) * \pi * \sin(x_2\pi) * \pi * \sin(x_3\pi) * \pi$	2097152	790	485	1.6
3	$\int_0^{\pi} dx_1 \int_0^1 dx_2 \int_0^{\pi} dx_3\, \sin x_1{}^2 + x_2 * \sin x_3$	12582912	4898	375	13
4	$4 * \int_0^1 dx_1 \int_0^1 dx_2 \int_0^1 dx_3 \int_0^1 dx_4\, \frac{x_1 * x_3{}^2 * e^{x_1 * x_3}}{(x1 + x_2 + 2 * x_4)^2}$	173015040	69181	1519	45

Each multidimensional integral estimation was tested for a confidence interval of 95%. The above table shows the sample size at which convergence was achieved and the time it took for execution of the sequential implementation as well as the CUDA implementation. The corresponding speedup is also listed. The below graph depicts the same.



Comparison of Sequential and CUDA computation times

It can be deduced from the plot that for small sample sizes the sequential CPU code is comparable with that of CUDA version. But beyond a certain threshold point (~2million samples here), CUDA gives excellent speedups over the sequential counterpart. This means for higher dimensional integrals and ill-behaved integrals where large samples need to be generated, the CUDA implementation of VEGAS provides significant benefits.

5. Conclusions and Future Work

In this paper, we have presented a CUDA implementation of VEGAS adaptive sampling technique which is used for estimation of multidimensional integrals. Our results show the enormous speedups that can be attained by running a GPU implementation of VEGAS over an equivalent one on the CPU, for estimation of large and arbitrary integrals. This implementation will particular be suitable for accelerating the integral estimation when large sample sizes are required to realize the desired accuracy.

The current parallel implementation on CUDA is still far from being fully optimized. Lot of divergences need to be managed to improve speed. It would also not be surprising to get better timing by improving warp occupancy and eliminating possible bank conflicts. These are currently being worked on. We are also working on multi-core and MPI implementations of Vegas. These results will be updated and benchmarked with CUDA implementation, once experiments are complete.

Acknowledgements

We would like to thank the Department of Computer Science, Florida State University, USA, for having allowed us to use one of their machines equipped with a Fermi GPU for running our experiments.

References

[1] G.P. Lepage, A New Algorithm for Adaptive Multidimensional Integration, Journal of Computational Physics 27, 192-203, (1978)

[2] G.P. Lepage, VEGAS : An Adaptive Multi-dimensional Integration Program, Cornell preprint CLNS 80-447, March 1980

[3] Mark E. Irwin, Stratified Sampling, Harvard, Summer 2006, http://www.markirwin.net/stat110/Lecture/Section75.pdf

[4] NVIDIA CUDA, Programming Guide 4.0

[5] Mark Harris, Optimizing Parallel Reduction in CUDA

[6] Introduction to Feynman perturbation theory can be found in any textbook on quantum field theory or relativistic quantum mechanics. See, for example, F. Gross, Relativistic Quantum Mechanics and Field Theory, John Wiley and Sons, New York, 1993.

[7] Muon Decay, http://en.wikipedia.org/wiki/Muon#Muon_decay

[8] S. Veseli, Multidimensional integration in a heterogeneous network environment FERMILAB-PUB-97/271-T September 1997

[9] Richard Kreckel, pvegas: A Portable Parallel Multiple-Purpose MC Integrator, Comput. Phys. Commun, 1997.