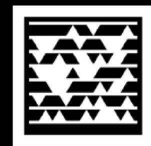

CATEGORY: **ALGORITHMS & NUMERICAL TECHNIQUES**

POSTER **AN09** | CONTACT NAME Kamesh Arumugam: akamesh@cs.odu.edu

GPU TECHNOLOGY CONFERENCE

An Efficient Deterministic Parallel Algorithm for Adaptive Multidimensional Numerical Integration on GPUs

Kamesh Arumugam,[1,2] Alexander Godunov,[3,2] Desh Ranjan,[1,2] Balša Terzić,[4,3,2] and Mohammad Zubair[1,2]

[1]Department of Computer Science, Old Dominion University, Norfolk, Virginia 23529

[2]Center for Accelerator Science, Old Dominion University, Norfolk, Virginia 23529

[3]Department of Physics, Old Dominion University, Norfolk, Virginia 23529

[4]Center for Advanced Studies of Accelerators, Jefferson Lab, Newport News, Virginia 23606

1. Abstract

Recent development in Graphics Processing Units (GPUs) has enabled a new possibility for highly efficient parallel computing in science and engineering. Their massively parallel architecture makes GPUs very effective for algorithms where processing of large blocks of data can be done in parallel. Multidimensional adaptive integration has a significant application in areas like computational fluid dynamics, quantum chemistry, plasma physics, molecular dynamics and signal processing. The computationally intensive nature of adaptive integration for higher dimension requires a high-performance implementation. In this study, we present an efficient parallel method for calculating multidimensional integrals using GPUs. Various CUDA optimization techniques are applied to maximize the utilization of the GPU. CUDA-based implementation outperforms the best known sequential methods and achieve up to 10X-100X speedup. It also shows good scalability with the increase in dimensionality.

2. Motivation

Today's scientists can gain insight into the challenges more easily by relying on computational methods. However, the growth of computational complexity requires better algorithmic efficiency and more computational power. Consequently, the ability to fully exploit parallelism is the only way to solve multiple complex problems in science and engineering using practically limitless computing power of multiple processors. In this context, modern powerful Graphic Processor Units (GPUs) open a new possibility for highly efficient parallel computing in science and engineering. Though we have access to cheap multiple cores, the software is still lagging behind the hardware in utilizing these cores on a processor. Applications need to be explicitly programmed to exploit multiple cores. In general, programming GPUs for general-purpose computing is difficult as it requires a change in programming philosophy and retraining. However, the design of efficient computational models should utilize advantages of both CPU and GPU architecture. Developing such algorithms whose performance is optimized on a hybrid CPU/GPU platform is necessary and important research topic in computer science.

The problem considered in this research is the approximation of the definite multidimensional integral

$$I = \int_{a_1}^{b_1}\int_{a_2}^{b_2}\dots\int_{a_n}^{b_n} f(x_1, x_2, \dots, x_n)\,dx_1\,dx_2\dots dx_n$$

to some accuracy ϵ. We are particularly interested in implementing globally adaptive algorithms on GPUs by leveraging the architectural difference between CPUs and GPUs, and using the resulting computational speedup to solve multidimensional integration with higher accuracy in acceptable times for most of the applications.

In recent years a number of computational models involve integration of various functions. Examples include quantum chemistry calculations necessitating evaluation of millions of two-electron integrals, solution of the Navier-Stokes equations using spectral element methods in 2-D and 3-D geometries requiring the ability to perform multiple integration for billions of points, lattice QCD simulations which have to evaluate highly dimensional integrals. In some of these computational problems computing values of integrated functions is a very time-consuming task. Therefore, one has to use highly efficient adaptive integration algorithms. Many such algorithms have been developed, and presented in widely used numerical libraries such as NAG, IMSL, QUADPACK and others. However, there are few only algorithms that have been developed for parallel computing. These considerations lead us to conclude that the most efficient algorithm for solving multidimensional integral should utilize the advantages of both CPU and GPU architecture, and develop algorithms whose performance is optimized on a hybrid CPU/GPU platform.

We base the multidimensional algorithms in this research on the adaptive routine CUHRE [1]. CUHRE is a deterministic algorithm. It uses the Genz–Malik cubature rules [2] in a globally adaptive subdivision scheme. It is the same as the original DCUHRE subroutine [1]. The algorithm is thus: Until the requested accuracy is reached, bisect the region with the largest error along the axis with the largest fourth difference. Sequential CUHRE implementation is available from the CUBA library [3, 4].

3. Adaptive Integration Methods

An approximation to the definite multidimensional integral,

$$I[f] = \int_{a_1}^{b_1}\int_{a_2}^{b_2}\dots\int_{a_n}^{b_n} f(\boldsymbol{x})d\boldsymbol{x}$$

is given by,

$$I[f] = \int_H f(\boldsymbol{x})d\boldsymbol{x} \approx \sum_{j=1}^{L} w_j f(\boldsymbol{x_j})$$

where H is the region of integration, $\boldsymbol{x_j}$ the evaluation points and w_j the corresponding weights, $j = 1, \dots, L$. Various traditional deterministic methods have been proposed in the past and are still being used to solve the problem at different dimension (mostly lower dimension). Some of the methods traditionally used for 1-D adaptive integration are Simpson's 3/5-points, Newton-Cotes 8-point, Gauss-Kronrod 7/15-points and Gauss-Kronrod 10/21-points. For integrands in 2-D and 3-D, Newton-Cotes 8-point, Gauss-Kronrod 7/15-points and Gauss-Kronrod 10/21-points are often used. However at higher dimension the execution time for these algorithms become unacceptable since the number of function evaluation grows exponentially with the dimension, necessitating the use of Monte Carlo techniques that have accuracy issues.

CUHRE [1] on the other hand is a deterministic algorithm which uses one of several cubature rules of polynomial degree in a globally adaptive subdivision scheme. CUHRE is the best known open source solution for solving multidimensional integration in reasonable amount of time. In moderate dimensions CUHRE is very competitive, particularly if the integrand is well approximated by polynomials. As the dimension increases, the number of points sampled by the cubature rules rises considerably, thereby reducing its usefulness. Also, with the increase in dimension, the execution time of CUHRE is unacceptable due to increase in the number of sampled points.

4. Parallel method on GPUs

GPU-based multidimensional integration is implemented in 2 phases.

PHASE1:

- Divide the whole integration region into n regions, where n is chosen based on:
  - Current GPU configuration, in order to have full occupancy of GPU;
  - Dimensionality of the problem.
- Assign every thread of GPU to a region and apply the integration rule locally.
  - Computes R_{err}, $\hat{I}$, N_{eval} and a subdivision axis k;
  - Every thread computes an estimate $\hat{I}$ for the integral I which for every component c fulfills $|\hat{I}_c - I_c| \le \max(\epsilon_{abs}, \epsilon_{rel}|I_c|)$;
  - A region is set to be active if $|\hat{I}_c - I_c| > \max(\epsilon_{abs}, \epsilon_{rel}|I_c|)$ i.e. the region has not satisfied the requested accuracy and has to be further subdivided. If $|\hat{I}_c - I_c| \le \max(\epsilon_{abs}, \epsilon_{rel}|I_c|)$ then the region is set inactive i.e. the computed value of integral $\hat{I}$ for the region satisfies the requested accuracy and the region is discarded from further computations.
- Group all the N' active regions and compute the intermediate results from I, R_{err}, N_{eval} for all the inactive regions using prefix sum from thrust library.
- Subdivide all the N' active regions along their respective subdivision axis by a factor k', which generates a new set of $N' * k'$ active regions for further processing. Parameter k' is chosen based on: fraction of input intervals those are active, current GPU configuration, in order to have full occupancy of GPU.

PHASE1 of the algorithm has the adaptive nature of eliminating all the regions where the integrand is well-behaved(satisfies the error criterion) and at the same time refining the resolution in the regions which require further application of integration rules due to complicated and sometimes very poorly behaved integrand. Using such hierarchical subdivision guarantees the maximum utilization of GPU and subsequently improves the performance of the integral computation. Number of iterations of PHASE1 is decided based on the fraction of input intervals that remain active after an iteration of PHASE1.

PHASE2:

- Set of active regions are assigned to every GPU thread.
- Every thread of GPU runs the GPU-optimized CUHRE integration rule to compute R_{err}, $\hat{I}$, N_{eval} for all the regions assigned to the thread.
- Compute the integral results from the intermediate values I, R_{err}, N_{eval} generated by every thread using prefix sum and scan operation from thrust library.

PHASE2 starts with all the active regions which were subdivided finer along the axis where the integrand has largest fourth difference. A portion of these new active regions can converge faster than others depending on the integrand behavior. In this case not all threads of GPU remain active at all times, which reduces the utilization of GPU. To avoid this scenario a set of regions are assigned to a single GPU threads to have a uniform load balance across the GPU. PHASE2 of the algorithm applies the integration rule (CUHRE) on every region until each satisfies the maximum error criterion or maximum function evaluation limit.

5. Performance Evaluation

Hardware Specification:

GPU: NVIDIA Tesla M2090

CPU: Intel(R) Xeon(R) CPU X5650@2.67Ghz

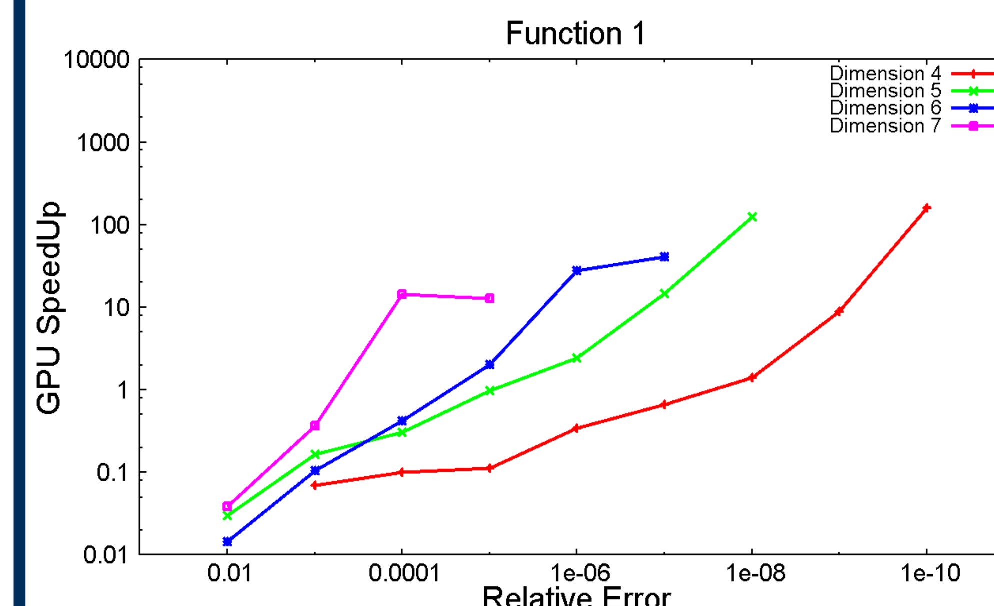

Figure 1 : Speed-up for Function 1

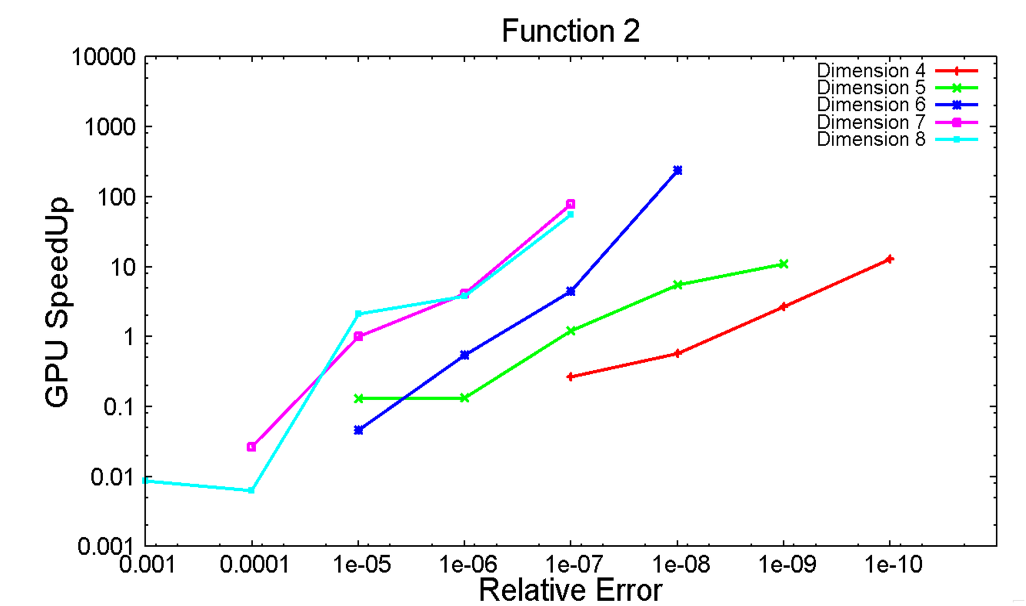

Figure 2 : Speed-up for Function 2

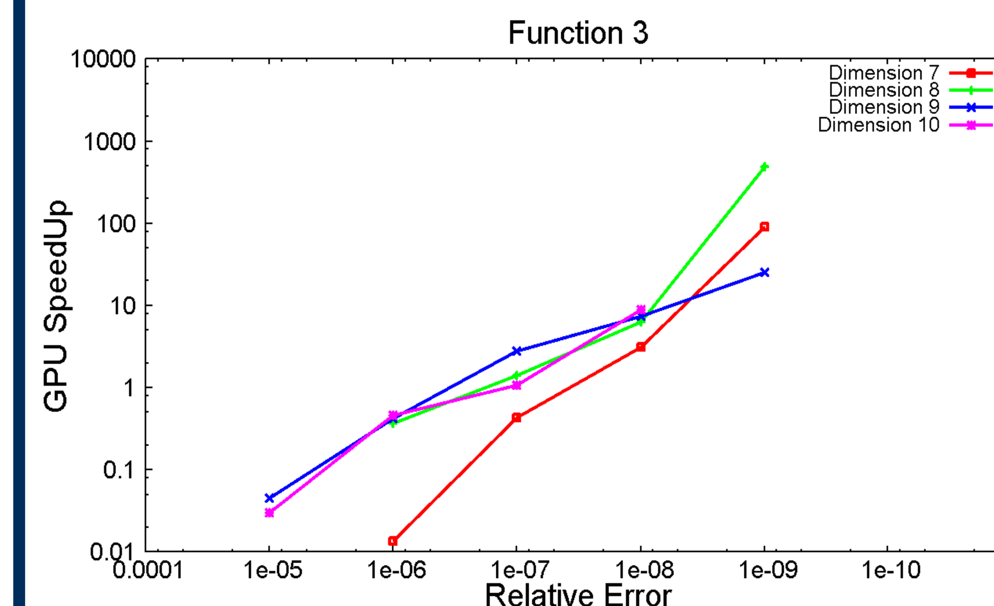

Figure 3 : Speed-up for Function 3

The integrals have the form

$$I[f] = \int_{a_1}^{b_1}\int_{a_2}^{b_2}\dots\int_{a_n}^{b_n} f(\boldsymbol{x})d\boldsymbol{x}$$

and for above results, the test integrands are:

1. $f_1(\boldsymbol{x}) = \frac{1.0}{\alpha + \left(Cos\left(\sum_{k=1}^{n} x_k^2\right)\right)^2}$, where α = 0.1,
2. $f_2(\boldsymbol{x}) = \frac{1}{2}\sum_{k=1}^{n}\frac{Cos(\alpha * x_i)}{\beta}$, where α = 10.0 and β = -0.0544021,
3. $f_3(\boldsymbol{x}) = Cos(2\pi\beta + \sum_{k=1}^{n}\alpha_i x_i)$, where β is picked randomly from [0, 1] and α_i are picked randomly from [0, 1] and then scaled according to

$$\sum_{k=1}^{n}\alpha_i = 15$$

where integration region is a n-dimensional hypercube and $\boldsymbol{x}$ is an n-vector. The algorithm input is n, **a**, **b**, **f**, an error tolerance *tol* and a limit L_{max}, on allowed number of evaluations of **f.** Some of the test functions were chosen from the technical report[7].

6. Conclusion

In this study, we presented a parallel implementation of an efficient deterministic algorithm for adaptive multidimensional numerical integration on a hybrid CPU/GPUs platform. Experiments showed good scalability for the parallel implementation on Tesla M2090 GPU with a speed-up of 10X-100X over the best known sequential method. The results showed that our proposed parallel algorithm is suitable even for higher dimensional integration. As the emergence of the CUDA programming model, GPU has become a promising platform for high performance computing. We believe the GPU-based parallel computing will provide compelling benefits for various problems in computational science. Current and future work includes extending our work with multiple GPU nodes along with optimizing our parallel implementation by reducing the GPU global memory access and increasing the utilization of GPU shared memory.

References

1. Berntsen, J.; Espelid, T. O.; and Genz, A. "An Adaptive Algorithm for the Approximate Calculation of Multiple Integrals." *ACM Transactions on Mathematical Software* Vol. 17, No. 4, December 1991, Pages 437-451.
2. A. Genz, A. Malik, SIAM J. Numer. Anal. 20 (1983) 580.
3. T. Hahn. "CUBA – a library for multidimensional numerical integration". *Computer Physics Communications 176* (2007) 712–713.
4. T. Hann. "CUBA – The CUBA library". *Nuclear Instruments and Methods in Physics Research,* A 559 (2006) 273–277.
5. NVIDIA, "NVIDIA's Next Generation CUDA Compute Architecture: Fermi", http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf.
6. Thrust, http://thrust.github.com/.
7. Berntsen, J.; Espelid, T. O.; and Genz, A. "A test of ADMINT". Reports in Informatics 31, Dept. Of Informatics, Univ. of Bergen, 1988.