Proceedings of the 5th International Conference on Nonlinear Dynamics
ND-KhPI2016
September 27-30, 2016, Kharkov, Ukraine

Analyzing Parallel Computation of the Functions Created with R-operations in CUDA

Roman A. Uvarov[1*]

Abstract
Brief overview of the recent general tasks for parallel computation on graphics processing units is represented. Adequacy of the parallel computation approach for single analytic function constructed with R-operations is carried out. Brief overview of ray tracing technique and its connection with constructive apparatus of R-functions is considered. Sample comparison calculations to show the benefits of CUDA are provided.

Keywords
Parallel computation, CUDA, ray tracing, theory of R-functions

[1] Scientific Fellow at Podgorny Institute for Mechanical Engineering Problems of NAS of Ukraine,
2/10 Dm. Pozharsky str., Kharkiv, Ukraine
*** Corresponding author:** rqa0001@gmail.com

Introduction

In recent years, much has been made of the computing industry's widespread shift to parallel computing. Nearly all consumer computers since the year 2010 until today will ship with multicore central processors. From the introduction of dual-core, low-end netbook machines to 8- and 16-core workstation computers, no longer will parallel computing be relegated to exotic supercomputers or mainframes. Moreover, electronic devices such as mobile phones and portable music players have begun to incorporate parallel computing capabilities in an effort to provide functionality well beyond those of their predecessors.

More and more, software developers will need to cope with a variety of parallel computing platforms and technologies in order to provide novel and rich experiences for an increasingly sophisticated base of users. Command prompts are out; multithreaded graphical interfaces are in. Cellular phones that only make calls are out; phones that can simultaneously play music, browse the Web, and provide GPS services are in.

For 30 years, one of the important methods for the improving the performance of consumer computing devices has been to increase the speed at which the processor's clock operated. In 2005, faced with an increasingly competitive marketplace and few alternatives, leading CPU manufacturers began offering processors with two computing cores instead of one. Over the following years, they followed this development with the release of three-, four-, six-, and eight-core central processor units. Sometimes referred to as the multicore revolution, this trend has marked a huge shift in the evolution of the consumer computing market. In comparison to the central processor's traditional data processing pipeline, performing general-purpose computations on a graphics processing unit (GPU) is a relatively new concept.

1. Appliance of CUDA to parallel computation of the functions created with R-operations

In general, the following base set of R-operations [1] is used for constructing the single functions ω_i representing a geometrical object of any complexity:

$$x \wedge_\alpha y = \frac{1}{1+\alpha}\left(x + y - \sqrt{x^2 + y^2 - 2\alpha xy}\right),$$

$$x \vee_\alpha y = \frac{1}{1+\alpha}\left(x + y + \sqrt{x^2 + y^2 - 2\alpha xy}\right),$$

$$\bar{x} = -x,$$

where $\alpha = \alpha(x, y),\ -1 < \alpha \le 1$.

The calculation of a single analytic function ω_i, describing the geometrical object and consisting of a number of support functions, in each point of specified area needs the parallelization. It is obvious that the efforts for computing the function of a complex geometrical object will increase with the number of the support functions if such a process is sequential. The idea of parallel computing in this case is based on the fact that the problem of computing the function of a complex geometrical object is divided into a set of tasks for computing the support functions, which can be solved simultaneously.

The ***aim of this work*** is to analyze the effectiveness of CUDA usage for parallel computation of the functions created with R-operations.

Therefore, compute unified device architecture (CUDA) technology in the Nvidia's GPUs of 8th series and higher was used to implement this idea. Along with Stream for AMD / ATI's GPUs, this technology is intended to allow the programmer use a modification of C language for executing the algorithms on the respective GPUs. Most GPUs from Nvidia and AMD are currently still oriented on calculating the numbers of single precision (32-bit) rather than double precision (64 bits) common to CPUs. At the same time, in contrast to the CPU, the GPU architecture has a parallel capacity, which aims to perform many concurrent threads slowly, rather than a single thread very quickly. Also SLI technology is supported by GPUs, it allows to connect multiple graphics processors to work in parallel.

The following hardware and software tools were involved for the work:

- CPU – Intel Core i7 with 16GB RAM;
- GPU – Nvidia GeForce 680 GTX with 2GB RAM;
- Microsoft Visual Studio 2013 Express Edition (C++);
- CUDA Toolkit 5.0;
- CUDA SDK.

Workflow of the program with CUDA [2] has the following specifics. At first, the input data are copied from the main memory to the memory of the GPU. Then CPU instructs the GPU to perform the task. After that GPU executes the calculations in parallel on each of its cores. When the result is ready, the result is copied back from the GPU memory to the main memory.

A simplest program sample to calculate the R-conjunction of two real arguments using CUDA on modified C language is represented at the Fig. 1.

It completely reflects the workflow in CUDA. Specific qualifiers are __device__, saying that the operation will be carried out only on the GPU, and __global__, indicating that the result of the operation is stored in the main memory of the GPU. Although the texture and constant memory of the GPU, which can also store the result, may be specified, these types of memory are different in capacity and access time. Calling these functions is accompanied by the parameters in the triple angle brackets, which indicate the number of blocks and threads for parallel computing. Functions with prefixes cuda are responsible for the operations relating to managing the data in memory.

```
#include <iostream>

__device__ float R_ANDem(float a, float b ) {
    float alpha = 0.5;
    return 1.0/(1+alpha)*(a+b-sqrt((a*a+b*b-
           2*alpha*a*b)*1.0));
}
__global__ void R_AND( float a, float b, float *c ) {
*c = R_ANDem( a, b );
}
int main( void ) {
float c;
int *dev_c;

HANDLE_ERROR(
cudaMalloc( (void**)&dev_c, sizeof(int) ) );

R_AND <<<1,1>>> ( 2, 7, dev_c );

HANDLE_ERROR(
cudaMemcpy( &c, dev_c, sizeof(float),
cudaMemcpyDeviceToHost ) );
printf( "Result = %5.2f\n", c );
cudaFree( dev_c );

return 0;
}
```

Figure 1. A simplest program on modified C language in CUDA

For visual representation of the results, which are three-dimensional graphics of functions, OpenGL technology and ray tracing were used as well. In brief, ray tracing is one way of producing a two-dimensional image of a scene consisting of three-dimensional objects. Fig. 2 illustrates the idea of this representation. We choose a spot in our scene to place an imaginary camera, which will produce a two-dimensional image. Each pixel of the resulting image casts out the ray into the scene, and intensity of the ray of light that hits that spot sensor is returned.

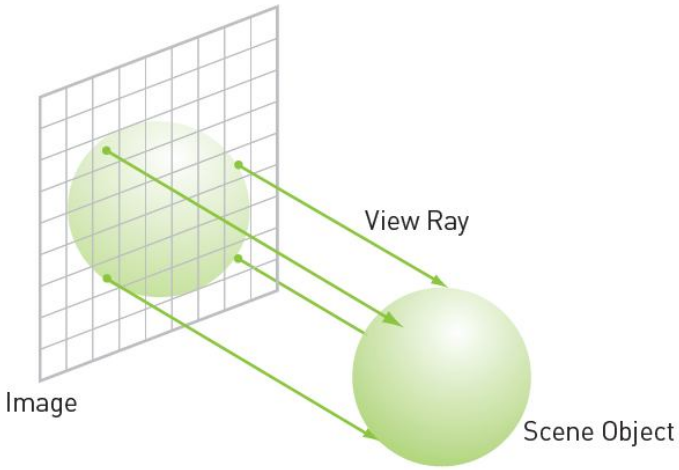

Figure 2. A simple ray tracing scheme

Finding the intersection points of view ray and the scene object is most expensive effort in this technique. In the classical case, an approach to produce a sphere may be considered as an example, when the intersection point of ray and sphere should be found. The sphere function becomes equal to the line (ray) function, quadratic equation is solved, and one of the roots, nearest in distance to the ray vertex, is the solution. This approach is not universal and the solution of the equation in the case of a complex area can lead to serious difficulties.

In the work, various cases for constructing a geometric object using ray tracing were considered. These are finding z from sphere function and defining the visible faces in the case of a rectangle. But the most effective result was obtained with an appliance of a constructive apparatus of the theory of R-functions, when intersection points of the ray with a single geometric object, defined with support functions and R-operations, were searched for. This approach is universal for geometric objects, i.e. there is no need to solve different types of equations to find the roots.

By using this approach a production of the sphere with a spherical dent and the box with a spherical dent rotated by 10 degrees were obtained successfully (Fig. 3).

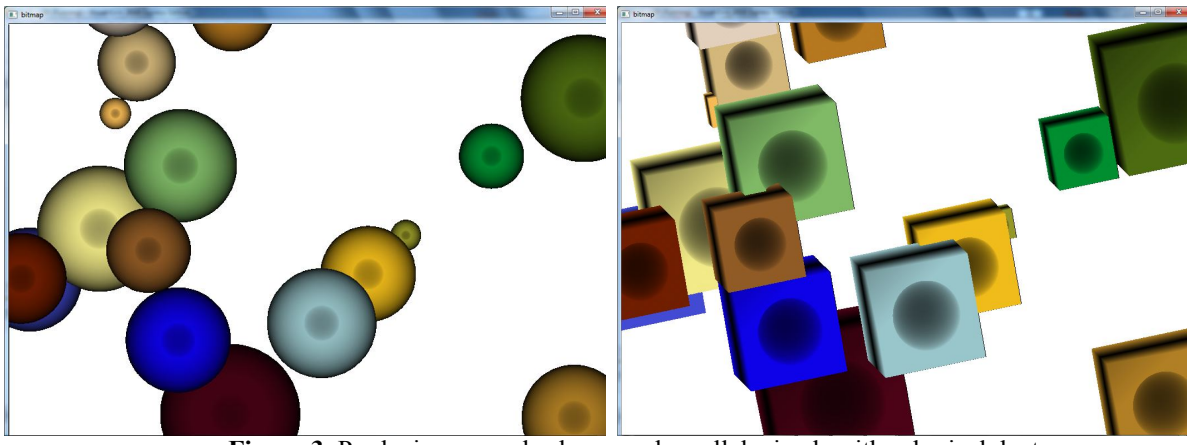

Figure 3. Producing several spheres and parallelepipeds with spherical dent

For comparison purposes, calculations of various functions with R-operations and without them were carried out in the two-dimensional domain and in three-dimensional domain. Calculations were performed using both the CPU and GPU. The results are represented in Table 1.

Table 1. Total time of calculation of the functions in each point of the three-dimensional domain of 4000 x 4000 x 40 points in size, in seconds

N	Function	CPU time, sec	GPU time, sec
1	Unbounded cylinder	2.44	0.01
2	R-conjunction of scalar values (alpha = 0.5)	11.89	0.01
3	R-conjunction of two cylinders	14.65	0.01
4	Prism with a convex quadrangle as a base	12.32	0.01
5	R-disjunction of two spheres	13.9	0.01
6	Parallelepiped	14.5	0.01

Conclusions

As a conclusion it may be said that in general, Nvidia's CUDA technology is oriented on solving the dynamical problems of graphical and computational kinds. In this work its appliance to a stage of computing the single analytic function constructed based on the constructive apparatus of the theory of R-functions was analyzed. Obtained results allow highlight its benefits and limitations. In general, the use of CUDA technology has significantly reduced the time spent for computing the complex functions.

References

[1] Rvachev V.L. *Theory of R-functions and its several applications.* Kiev: Nauk. dumka; 1982.
[2] Sanders J., Kandrot E. *CUDA by example: an introduction to general-purpose GPU programming.* Boston: Addison-Wesley; 2010.