World Applied Sciences Journal 23 (5): 644-649, 2013
ISSN 1818-4952

DOI: 10.5829/idosi.wasj.2013.23.05.13098

Abilities of Modern Graphics Adapters for Optimizing Parallel Computing

Alexander Arsenevich Zolotarev and Oleg Igorevich Agibalov
Southern Federal University, Rostov-on-Don, Russia

Submitted: May 31, 2013; **Accepted:** Jun 26, 2013; **Published:** Jun 30, 2013

Abstract: The subject of the research is the abilities for optimization with using modern graphics units. The purpose of the paper is to test basic abilities for optimization with using modern graphics units. Different technologies of GPU programming were discussed in the paper. The results of performing the simplest basic and complex functional operations are presented. Advantages and disadvantages of modern GPUs are described. The questions of optimization become the key in determining and implementing modern technical and economical processes. Usually creating and developing new scientific technologies generates the range of optimization tasks. And the successful solution of such tasks is able to significantly increase the effectiveness of these technologies. All the modern optimization methods and algorithms are required to be adapted for the specifics of implementing processes in spite of the high level of formalization and unification. For instance, the most important task in IT (Information Technology) industry is development high-performance technologies, based on designing parallel algorithms and creating parallel program structures. Even today the most widely used computing device is CPU (Central Processing Unit), although the faster tools exist and they have in many times higher performance comparatively with CPU. One of such perspective high-performance technologies is implemented based on graphics processors GPU (Graphics Processing Unit). All the modern graphics cards are the potent specialized devices that are contained in practically each modern PC. The leading companies on a GPU market are NVidia and AMD (Advanced Micro Devices) Corporations with their GeForce and Radeon lines respectively. At the last time Intel Corporation develops CPUs that contain both CPU cores and GPU module. Today it is implemented in the architectures Sandy Bridge, Sandy Bridge-E and Ivy Bridge [1]. The performance capabilities of GPU impress but it is necessary to be able to use them effectively. Only the optimization way allows making a breakthrough which will make it able to bring high-performance technologies and technologies of processing data a new level.

Key words: Parallel computing · CPU · GPU · Optimization · Numerical experiment · OpenCL · DirectCompute · GPGPU · Nvidia · AMD · Intel · Thread

INTRODUCTION

Technologies and functionality of modern graphics processors are developed constantly. Thus, recently they processed only the computer graphics: two-dimensional and three-dimensional. But it was changed since the leading manufacturers have been developed technologies that allow total using of graphics units for general purpose computing-GPGPU technologies (General-purpose Graphics Processing Unit). Today there are many such technologies. The leading one is NVidia CUDA (Compute Unified Device Architecture) [2]. As for the CUDA the most significant here is technical support not only by NVidia but also by its dozen of partner companies. Another interesting technology is OpenCL [3]. It is cross platform and cross device tool which overcomes the main disadvantage of CUDA-NVidia CUDA works only with NVidia GPUs. Thus it is possible to use OpenCL with graphics modules of Intel and AMD corporations too. Microsoft also developed their own framework for GPU computing which is called DirectCompute [4]. Being a new module of DirectX API it works only on Windows OS and this fact doesn't allow it to compete with OpenCL and CUDA. But it is important that Microsoft designed so-called C++ AMP for Visual Studio based on Direct Compute [5].

It is interesting that all the described technologies have the similar programming convey. It is caused by the similar GPU hardware architectures from various manufacturers. Each GPU consists of dozens of so-called

Corresponding Author: Zolotarev, Southern Federal University, 105/42 Bolshaya Sadovaya Str., Rostov-on-Don, 344006, Russia

multiprocessors. Each multiprocessor provides a set of thread groups (blocks) and any blocks consist of many threads in three-dimensional space. Thus each thread can be characterized by 3D index in *x*, *y* and *z* axes. But the most important thing is that all the threads are independent. Each thread works separately and in parallel with all the others. But in this case there is the problem of using memory. How to provide the ability for threads to communicate between each other?.

There are several memory types inside GPU that allow solving this problem. Each type can be accessed for only specific modules of GPU. For example, each thread has its own local memory. No other threads are able to operate it. There is also the memory which is common for the whole group of threads. And each thread from this group can access it. This memory is used for communicating between threads in a single block. But this block can't operate with local memory from other groups. For this the common memory of multiprocessor exists. And this memory is accessible for any thread groups of multiprocessor. The flexibility of architecture and maximal access speed are implemented due to such complex architecture [6].

Applied Description of GPGPU Technologies: All the GPGPU technologies may be divided into several groups depend on level of using and of applications. The first group is represented by such developments as CUDA, OpenCL, DirectCompute, FireStream. They are the APIs that allow programming heterogeneous systems. To use them it is necessary to include special libraries in a program. Then the two types of programs may be written: for CPU and GPU. Such tools are very flexible and are characterized by many different parameters that provide the ability for broad optimization. Leading technology in this group is NVidia CUDA. It is the most widely spread among all the presented technologies. It is widely supported not only by NVidia but also by dozen of its partner companies. So we may use CUDA both with professional high-level languages like C++ and with more scientific languages like Fortran, for example, through PGI CUDA Fortran Compiler [7]. The use of Fortran allows programmer to put aside the details of low-level programming. It is necessary to say that there are two languages that are mostly used for scientific purposes: Fortran and Python. As for the CUDA it has the single disadvantage – it works with only NVidia's accelerators. Although we are able to use OpenCL with AMD devices. And it is important that OpenCL provides abilities for cross platform programming and programming for GPUs from various vendors. There are also technologies of using OpenCL with Fortran, but they are non-official which means that they are developed independently from Khronos Group. Nevertheless openCL is the best choice when you need to create a cross platform application. But in the most cases you need to use C++ with OpenCL. Also OpenCL is a part of both CUDA SDK and FireStream SDK. Generally these technologies are similar and very powerful. They allow developing programs for several devices and provide the highest level of performance. Moreover OpenCL, CUDA, DirectCompute and FireStream are freeware, excluding specialized releases, for instance for CUDA Fortran. DirectCompute is the most unattractive here. It was developed as a part of DirectX and many its elements are aimed to universalization. That's why the data transfer speed is too small sometimes. But it is more interested that the Microsoft developed C++ AMP based on application programming interface DirectCompute.

C++ AMP is a heterogeneous tool of programming various systems. Its idea is in using directives. And they are putted in a code in the Visual Studio. These directives tell compiler that it needs to analyze the code and distribute its execution between CPU and GPU. The main advantage of this way is that you don't need to be good at GPGPU. All that you should do is to put directives in a correct place. But this technology works only in Visual Studio and requires Windows OS. And this is the main disadvantage of C++ AMP. The key instruction is restrict (amp). It requires compiler to analyze the code and execute it in C++ AMP accelerator [8].

The keyword restrict means the necessity to analyze if the function relevant for to be executed on GPU [9]. The sample code is presented below.

```
voidmyFunc() restrict (amp)
{//Do something}
```

Thus the C++ AMP represents the second group of GPGPU technologies. The third group contains such developments as OpenACC and OpenHMPP (HMPP for Hybrid Multicore Parallel Programming) [10]. OpenACC and OpenHMPP are the cross platform sets of directives that are supported by many companies. The principle is similar to C++ AMP. You may put directive into your code and thus it is possible not just to parallelize the code but also control the transfers between GPU memory and RAM. These tools are supported by such companies as PGI and CAPS. As the NVidia partners they create technologies for its accelerators. And this is the single disadvantage of OpenACC and OpenHMPP. Due to NVidia the technologies of OpenACC and OpenHMPP work not only with C++ but also with C and Fortran

languages. As for the OpenHMPP it is the extended version of OpenACC with additional abilities and directives. The sample OpenACC code is presented below [11].

```
#include <openacc.h>
#include <stdio.h>
#include <stdlib.h>
void main() {
int n = 100;
float a[n][n];
float b[n][n];
float c[n][n];
float elements [n];
for(int i = 0; i< n; i++)
for (int j=0; j<n; j++){
a[i][j] = i+j;
b[i][j] = 100 + 2 * i;
}
#pragma acc kernels loop independent
for(int i = 0; i< n; i++)
for (int j=0; j < n; j++){
for (int k=0; k<n; k++)
c[i][j]=+a[i][k]*b[k][j];
}
free(a); free(b); free(c);
} // main
```

Finally there are tools for any cases. Developers are free in choosing between dozens technologies with different levels of abstraction. The leading NVidia Corporation provides anything to developers for fastest and effective developments. And due to OpenACC and OpenHMPP they even don't need to research GPGPU architecture.

GPGPU are widely used already today. Many companies apply these technologies for a various computations in different fields. One of them is Murex. For twenty five years Murex has focused exclusively on developing software platforms and technologies for the capital markets. The Murex trading platform enables trading, risk management and trade execution across multiple asset classes for over 36,000 users at over 200 institutions in 65 countries. Using NVIDIA Tesla GPUs, Murex has achieved speedups of between 60 and 400 times using MACS in a grid environment, making it possible to manage books of complex exotic products in near real time, instead of only computing analytics once or twice a day. As a result MACS clients can enjoy more accurate and timely risk management [12].

It is also very interesting that MathWorks supports GPUs architectures in their MatLab. The innovative technical development by MathWorks leverages NVIDIA's feature-rich CUDA computing toolkit, helping to allow MathWorks to bring the benefits of GPU computing to the MATLAB community. MATLAB users can now easily enjoy the benefits of GPU computing from within MATLAB, without C/C++ or FORTRAN programming [13]. And MathWorks is not alone. Here is the Table 1 which provides information on several program packages and companies that built in GPGPUs in their products [14].

Results of GPGPU Working: As it was pointed, optimization abilities of modern GPU should be studied to reach best results. This means testing various operations both basic and complex functional. It is also important to understand how GPU works with different basic types of variables and how the time is increased with bigger capacity of input arrays.

GPGPUs Results in Working with Basic Operations: The basic operations are mathematical operators such as addition, subtraction, multiplication and division.

All the values were calculated both for GPU and CPU. That's why it is possible to compare these computing units. The results for arithmetical operations are presented in a table below (Table 2).

All the computations were performed for arrays with 20 000 000 elements. The GPUs superiority is obvious. Moreover the GPU uses similar time intervals for all the operations. At the same time CPU needs more time for division than for addition, subtraction and multiplication. Therefore it is proved that GPU effectiveness is still higher for division operation, in spite of all others operations require equal time.

Processing Time Dependence from Data Array Capacity: It is obvious that the time of computing depends on the capacity of data arrays. And the nature of this dependency should be studied. Is it possible to tell about linear dependence, or not? And the series of tests were performed for solving this task (Fig. 1).

Horizontal axis, which is titled V, reflects the capacity of data $V*10^{+6}$ elements in array. Vertical axis is the time of computing T in milliseconds. Thus the dependency is linear. Moreover it is linear both for CPU and GPU. And the angle between horizontal axis and the line is equal for both devices (Fig. 1).

Table 1: Companies and their products that use GPGPUs

ISV/Application	Supported Features	Expected Speedup*	Release Status
AccelerEyes Jacket	CUDA kernels with Matlab, distributing standalone apps with Matlab	20x-40x	Released
MathWorks Matlab	CUDA kernels with Matlab, distributing standalone apps with Matlab	3x-5x	Released
Mathematica Wolfram	Development environment for CUDA and OpenCL kernels	3x-5x	Released

Table 2: Computing basic arithmetic operations

Operator	TavCPU, ms	TavGPU, ms
Addition	3332,5	57,2
Subtraction	3332,8	58,5
Multiplication	3381,6	60,1
Division	5242,1	60,4

Table 3: Dependence from type of input data

Measurement	Int	Float
Average for CPU, ms.	2826,7	3336,4
Average for GPU, ms.	50,3	49,9

Table 4: Characteristics of the CPU and GPU on the complex operations

Operator	Average working time for CPU, ms.	Average working time for GPU, ms.	Deviation of CPU	Deviation of GPU
Sine	49850,5	205,5	17,6	3,0
Cosine	49859,0	207,2	24,1	4,8
Tangent	64534,6	190,6	11,9	56,2
Arcsine	15867,9	210,4	24,9	2,0
Arccosine	15155,3	211,7	25,9	1,8
Arctangent	23384,8	211,5	36,3	1,34
Sqrt	8900,3	206,5	8,4	1,7
Log	26436,3	210,1	48,7	1,9
Pow(3.5)	99117,6	170,8	45,8	73,8
Exp	21884,2	207,0	106,6	3,4

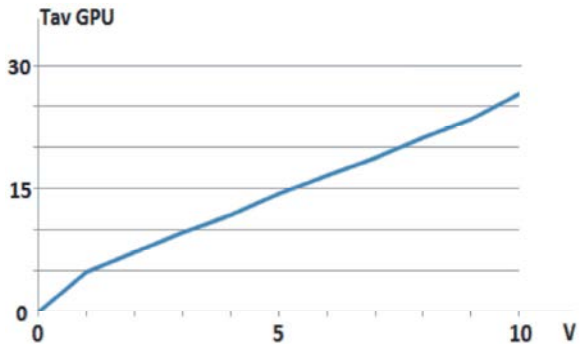

Fig. 1: Dependence of time from data capacity

Results of Data Types: Another important thing in optimization process is the ability of working with different types of variables, for example integer and float. It is necessary to understand how strong the type of data influence at time of operation computing. And the results were gotten during the testing and are presented in table below (Table 3).

And they are really interesting. For example, it became clear that there is no difference for GPU between computing integer or float values. Although GPUs were primary optimized for working with real types of data. Then it could be possible to expect higher speed for float arrays. But the real results show that the times are similar for both types of data.

Results with Complex Operations: However the arithmetic operations do not provide the full view on how GPU works. In most cases they play the role of relations between complex operations that are used in operating canonical functional structures. The results of the experiments with such complex operations as sine, cosine, tangent, pow, abs and others are presented below in Table 4.

According these results the GPU speed is in 250 times higher than CPU speed. Deviation here is standard deviation, which can be computed with using next equation.

$$Dev = \sqrt{\frac{\sum (x - \bar{x})^2}{n - 1}}$$

In this equation $\bar{x}$ is a sample mean and n is the size of sample.

Table 5: The results of CPU and GPU on the tangent operation

Size	Function	*i*	Time of GPU, Tig	Time of CPU, Tic
20 000 000	for(int i=0;i<100;i++)	1	207	64549
	{	2	214	64543
	out[num] = tan(in[num]);	3	209	64544
	}	4	209	64524
		5	208	64540
		6	201	64550
		7	210	64524
		8	205	64523
		9	31	64518
		10	212	64531

Table 6: Characteristics of tangent operation

Min Time of CPU	Max Time of CPU	Average CPU	Min Time of GPU	Max Time of GPU	Average GPU	Dispersion CPU	Dispersion GPU
1	2	3	4	5	6	7	8
64518	64550	64534,6	31	214	190,6	11,9	56,2

However there are several operations that have really large dispersion and should be studied better. One of such operations is tangent, whose results are presented below in Table 5.

Tig here is GPUs working time, Tic is CPU working. All the values are presented in milliseconds.

Statistical characterization for tangent is presented below in Table 6.

Evidently, there is a value which is equal to 31 in the table. And it differs (in several times) from all other values. And the reason of such deviation is unclear. There are many background processes in PC. Both GPUs and CPUs constantly perform dozens system operations that require resources.

The similar situation is repeated for the pow operation too. The degree is not integer. And this is not accidentally. The float degree is the most general case for scientific computations.

It is interesting that in most cases the GPUs require the similar time for various type of operations: sine, cosine, tangent, sqrt and others.

It is clear from all the previous data tables that CPUs need time in 100-1000 times more than GPUs. Thus the quotient of times for tangent operation is equal to this: (Aver GPU)/(Aver CPU) ~ 10^{-3}.

Advantages and Disadvantages of GPUs: The modern GPUs provide in many hundred times higher performance than CPU. But there are no perfect technologies and GPGPUs have a set of disadvantages too.

The advantages are clear. They were presented through the result, gotten in this paper. As for the disadvantages, they are hidden in non-obvious contexts.

First of all, graphics card is a perfectly parallel architecture which computes data, encapsulated in arrays, independently from each other. And this limits the tasks that are able for solving with using GPUs. For instance, the task should be prepared for computing on GPU. Developer should understand that the clock rate of GPU core is lower than CPU's clock rate. Therefore the serial operations require more time when working with GPUs. And the main advantage of GPU is in is parallelism. Thus, you should be sure that your tusk may be prepared for parallel computing. And it should be packed in array with independent elements inside.

Another important problem is various secondary operations needed by GPUs architecture. They were already described in GPUs programming conveys. All the data are initialized by CPU. Then they are transferred into GPU memory and are computed there. After this GPU transfers data back to the RAM. And the high-capacity data arrays require really many time. But the GPUs are developed for processing huge arrays. Therefore sometimes it is necessary to analyze if it is better to use GPU or CPU? Then you first need to analyze this point and based on the model of optimal scheduling of tasks.

Moreover developer should take into account all the differences between CPUs and GPUs architectures. And this means that there will be an intermediate step in developing and performing task, which is based on developing new hardware and programming methods. And the differences are multiple. For example, all the virtual threads exist in three-dimensional space. And there is a special function or a variable for thread identification in GPGPU API. So programmer first needs to determine a thread index and then transmit it as index in a target array. If this is not done, all the computations will be performed by all the GPU cores in parallel. All the operations that are programmed for one core will be executed by any processors of GPU. And the thread identification is need for avoiding this problem [15].

CONCLUSION

Based on the above, it clear that GPGPU have many advantages comparatively with CPU. They are in many times faster and this is the main advantage in the modern informative community. However it has disadvantages too, but they do not detract its advantages. They just impose several restrictions and require programmers to be more attentive in programming process and in the process of developing task.

And they have all the necessary to do it. Because now often you even don's need to be good at the details of GPUs architecture and their basic technologies. It is possible due to such developments as OpenACC, OpenHMPP and C++ AMP. All the work is done by compilers and human just should type several additional string of code.

It simplifies the developing process, especially because the range of technologies is really wide. These technologies are designed first of all by the leading IT corporations of the industry. They have enough resources for creating applications of any level of complexity. They usually produce GPUs and consequently know about the whole range of their features as no one else.

REFERENCES

1. Ivy Bridge architecture extends industrial software. Date View 10.06. 2013 http://embedded.communities.intel.com/community/en/software/blog/2012/06/28/roving-reporter-ivy-bridge-architecture-extends-industrial-software.
2. What is CUDA? Date View 10.06.2013 http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html.
3. Open CL The Open Standard for Heterogeneous Parallel Programming. Date View 10.06.2013 http://developer.download.nvidia.com/presentations/2009/GDC/OpenCL_Overview_GDC_Mar09.pdf.
4. Compute Shader Overview. Date View 10.06.2013 http://msdn.microsoft.com/en-us/library/windows/desktop/ff476331(v=vs.85).aspx.
5. C++ AMP Open Specification. Date View 10.06.2013 http://blogs.msdn.com/b/somasegar/archive/2012/02/03/c-amp-open-specification.aspx.
6. CUDA: New architecture for GPGPU computing. Date View 10.06.2013 http://www.nvidia.ru/content/cudazone/download/ru/CUDA_for_games.pdf.
7. PGI CUDA Fortran Compiler. Date View 10.06.2013 http://www.pgroup.com/resources/cudafortran.htm.
8. Restrict (C++ AMP). Date View 10.06.2013 http://msdn.microsoft.com/en-us/library/hh388953.aspx.
9. C++ AMP. Date View 10.06.2013 http://en.wikipedia.org/wiki/C++_AMP.
10. Open HMPP Directives. Date View 10.06.2013 http://www.caps-entreprise.com/openhmpp-directives/.
11. Introduction to Open ACC. Date View 10.06.2013 http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/S0622-GTC2012-PGI-OpenACC-Compilers.pdf.
12. Murex Analytics on Tesla GPUs. Date View 15.06.2013 http://www.nvidia.co.uk/object/tesla-murex-analytics-uk.html.
13. MATLAB Acceleration on Tesla and Quadro GPUs. Date View 15.06.2013 http://www.nvidia.co.uk/object/tesla-matlab-accelerations-uk.html.
14. Numeral Analytics. Date View 15.06.2013 http://www.nvidia.co.uk/object/numerical-packages-uk.html.
15. Agibalov, O. and G. Muratova, 2013. Problems of using GPGPU. In the proceeding of the 2013 scientific-practical conference Modern information technologies: trends and perspectives of development, pp: 25-26.