

Vol. 10, No. 1, 2015.

ISSN 1840-1503

TECHNICS TECHNOLOGIES EDUCATION MANAGEMENT

t t e m

JOURNAL OF SOCIETY FOR DEVELOPMENT OF TEACHING AND BUSINESS PROCESSES IN NEW NET ENVIRONMENT IN B&H



ISSN 1840-1503



9 771840 150002

EDITORIAL BOARD

Editor	<i>Dzafer Kudumovic</i>
Secretary	<i>Nadja Sabanovic</i>
Technical editor	<i>Eldin Huremovic</i>
Cover design	<i>Almir Rizvanovic</i>
Lector	<i>Mirnes Avdic</i>
Lector	<i>Adisa Spahic</i>
Members	<i>Klaudio Pap (Croatia)</i> <i>Nikola Mrvac (Croatia)</i> <i>Slobodan Kralj (Croatia)</i> <i>Davor Zvizdic (Croatia)</i> <i>Janez Dijaci (Slovenia)</i> <i>Ivan Polajnar (Slovenia)</i> <i>Tadeja Zupancic (Slovenia)</i> <i>Jelena Ivanovic Sekularac (Serbia)</i> <i>Nebojsa Vidanovic (Serbia)</i> <i>Hasan Hanic (Serbia)</i> <i>Amir Pasic (Bosnia and Herzegovina)</i> <i>Vesna Maric-Aleksic (Bosnia and Herzegovina)</i> <i>Avdo Voloder (Bosnia and Herzegovina)</i> <i>Samir Causevic (Bosnia and Herzegovina)</i>

Address of the Editorial Board
 Sarajevo,
Hamdije Kresevljakovica 7A
 phone/fax 00387 33 640 407
ttem_bih@yahoo.com,
http://www.ttem.ba

Published by *DRUNPP, Sarajevo*
 Volume 10 *Number 1, 2015*
 ISSN *1840-1503*
 e-ISSN *1986-809X*

Table of Contents

A Study on the Authority Perception of School Administrators	3
<i>Fatih Toremen, Yunus Bozgeyik, Necati Cobanoglu</i>	
Soft Lithography: Part 1 - Anti-fouling and self-cleaning topographies	9
<i>Anka Trajkovska Petkoska, Anita Trajkovska-Broach</i>	
The application of BIM technology and its reliability in the analysis of structures	18
<i>Tatjana Baros</i>	
Hindrances to the use of E-learning system by architecture students	28
<i>Bhzad Sidawi</i>	
Can Lean Six Sigma implementation experiences from Slovenia help Bosnia?	42
<i>Dusan Gosnik, Bojan Mevlja, Klemen Kavcic</i>	
Proportion of the captain's house dating from 1842. in Lepetani- the Bay of Kotor	52
<i>Dusan Tomanovic, Julija Aleksic, Mirko Grbic</i>	
Evaluation of the parallel computing performances based on the nVIDIA GeForce GPU	60
<i>Mitko Bogdanoski, Jelena Gjorgjev</i>	
System for Quality and Risk Management - Managing on-site Courses Using Online Technology: A Study on Al Ain University of Science and Technology.....	71
<i>Loay Alnaji</i>	
Expertise of nonmaterial damage due to violation of bodily integrity	78
<i>Haso Sefo, Nedzad Korajlic, Mustafa Sefo, Muris Mujanovic</i>	
Contribution to structural analysis of bailey bridges according to contemporary regulations in Bosnia and Herzegovina	89
<i>Besim Demirovic, Zijad Pozegic</i>	
Postural Quality Analyzes for Children Recorded in Kinematic 2D and 3D Contemplas Method	99
<i>Sinisa Kovac, Safet Kapo, Haris Alic, Gordana Manic</i>	

Table of Contents

Investigation of original stone fragments used for the construction of the Aladza Mosque in Foca 107
Nedjo Djuric, Kristina Saric, Dijana Djuric

Strikes and social problems as a consequence of the applied model of the privatization of the public companies in the Republic of Serbia 112
Mehmed Avdagic, Slobodan Devic, Desnica Radivojevic, Dzenan Golic, Maja Radic

Body Composition and Balance Measured by Tanita and BBS - The Importance for Competitive Success of Karate Athletes 119
Safet Kapo, Nedim Covic, Nusret Smajlovic, Husnija Kajmovic, Anida Kapo, Armin Topalovic

Employee Attrition in ITeS Call Centers in Selected Clusters of North India: Need to have a Relook 124
Gagandeep Kaur, Neeraj Pandey, Ravi Kiran, Shailendra Kumar

Instructions for the authors 136

Evaluation of the parallel computing performances based on the nVIDIA GeForce GPU

Mitko Bogdanoski¹, Jelena Gjorgjev²

¹ Military Academy “General Mihailo Apostolski”, Goce Delcev University, Skopje, R. Macedonia,

² European University - Republic of Macedonia, Skopje, R. Macedonia.

Abstract

Parallel programming is a form of computation in which the calculations are carried out simultaneously, operating on the principle where large problems can be divided into smaller, which are then solved in parallel.

Most common this programming is used in high performance computing, but due to the physical constraints which prevent frequency scaling the interest is even higher. As computers consumption has become a problem in the recent years, the parallel programming has grown into the dominant paradigm in computer architecture, mainly in the form of multicore processors.

The paper shows the process of designing parallel programs and it includes some results obtained by testing the parallel programming performance of different nVIDIA GeForce graphics cards. The purpose of the test is to compare performance of several types of GPUs for various applications. The results of the test could not specifically say which GPU is best due to the different features of the cards, but they can be used to determine which card offers better performances for different parts of the tests.

Key words: Parallel computing, GPU, CUDA, GeForce, nVIDIA

1. Introduction

Computer software is traditionally written for serial calculations. To solve a problem an algorithm implemented as a serial stream of instructions is created. These instructions are executed in the computer’s Central Processing Unit (CPU). Only one instruction can be executed at one time, and after instruction is completed, it is followed by the execution of the next instruction. [1]

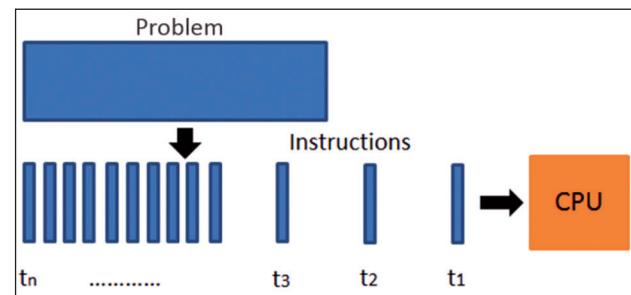


Figure 1. Solving problem by serial computations and CPU

Contrary to this the parallel programming in order to solve this problem uses multiple processing elements simultaneously. This is achieved by dividing the problem into independent parts so that each processing element can execute its part of the algorithm simultaneously with the others. [2]

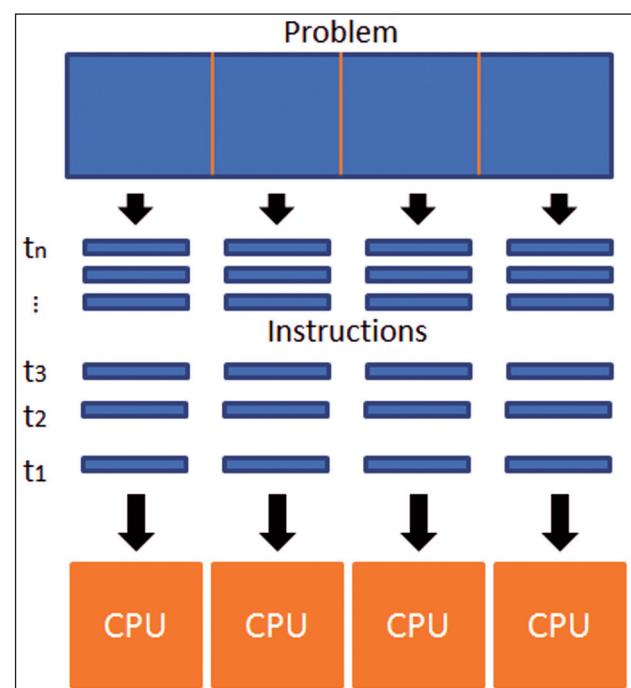


Figure 2. Solving the problem by parallel computing

The processing elements can be different and they can include resources such as a computer

with multiple processors, several networked computers, specialized hardware, or any combination of the foregoing.

The solving problem should allow dividing into several parts which will enable performing multiple program instructions at any time and speeding up the processing time using multiple computer's resources rather than using a single computing resource.

Some of the reasons for the increased usage of the parallel computing are the following:

- saving time and money,
- solving big problems,
- ensuring competitiveness,
- using non-local resources,
- limitations of the serial programming.

The rest of the paper is organized as follows. The next section gives some basics about designing and building parallel programs. Moreover, section 3 explains the limitations and expenses related to the parallel computing. The next section gives a brief description of the experiment. Section 5 explains used techniques and programs during the experiment. The obtained results for all tested applications, as well as a comparison of these results are given in section 6. Finally, section 7 concludes our work.

2. Designing and Building Parallel Programs

Designing and building parallel programs is a manual process where the programmer is responsible for parallelism identification and implementation. Very often manual development of parallel code is time-consuming, and it is a complex, iterative process prone to errors. The tools that assist the programmer in converting serial programs into parallel programs exist for a while. The most commonly used tool for automated parallelizing of the serial program is parallelizing compiler or pre-processor.

The parallelizing compiler works in two modes, fully automatic and program managed.

2.1. Understanding the problem and program

The first step in the parallel software development is to understand the problem which should be solved in parallel. Before spending time try-

ing to develop a parallel solution to the problem, it should be determined whether the problem can ever be parallelized.

The following example shows a problem that can be solved in parallel.

Calculate the potential energy for each of several thousand independent conformations of a molecule. When done, find the minimum energy

This problem can be solved in parallel so that each of the molecular conformation is independently determined. The calculation of the minimum energy conformation is also a problem that can be parallelized.

2.2. Partitioning

One of the first steps in designing a parallel program is to break the problem into discrete pieces that can then be distributed in multiple tasks. This is known as decomposition or partitioning.

There are two basic ways of partitioning of the calculating work between the parallel tasks: domain partitioning and functional partitioning. [3]

* Domain partitioning

In this type of partitioning the data related to the problem are decomposed. Each parallel task then work on part of the data.

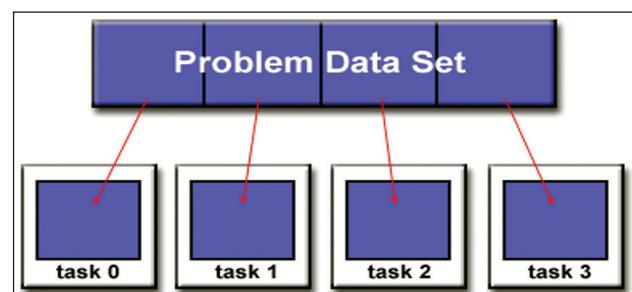


Figure 3. Domain partitioning

There are different ways of data partitioning (Figure 4) [1]

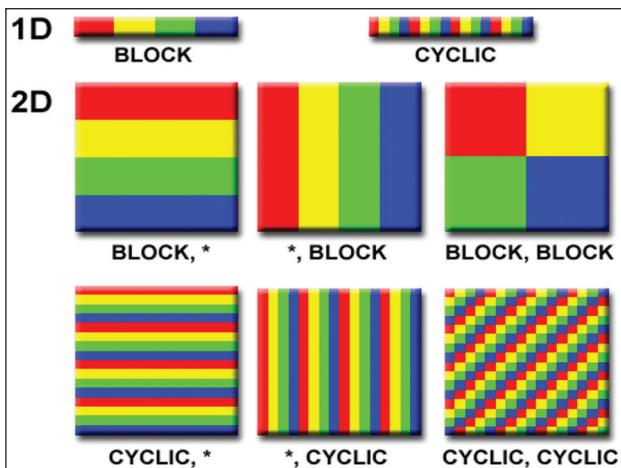


Figure 4. Data partitioning

* Functional partitioning

The focus in this approach is on computation and the problem is decomposed according to the work that must be performed. Then each task executes part of the entire work (Fig. 5).

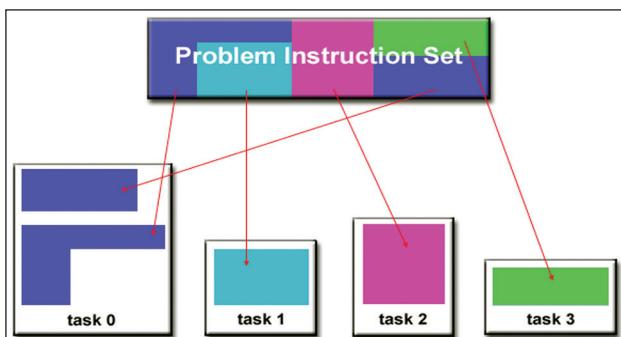


Figure 5. Functional partitioning

2.3. Data dependencies

The data dependency is one of the key issues of real time programming, and it includes sequential and parallel processing. [4] The data dependencies are the result of multiple use of the same location in the warehouse of various tasks.

The dependencies are very important for parallel programming, because they are one of the main inhibitors of parallelism. The analyses of data dependence are needed only for automatic detection of parallelism, but they are also essential for many other important compiler transformations, such as improving the memory location and load balancing. [5]

Although during parallel programs designing process it is important to identify all data dependencies, the dependencies in the loop are especial-

ly important because the loops are the most common target of the parallelism.

To deal with data dependencies we need:

- Distributed memory architectures - the necessary data communicate in synchronized points.
- Shared memory architecture - synchronized read / write operations between tasks.

2.4. Load balancing

Load balancing refers to the practice of distributing work among tasks so that all tasks are kept busy all the time. It can be considered as minimization of the assignment idle time. Load balancing is important for parallel programs because of its efficiency.

For example, if all assignments are subject to barrier synchronization, the smallest task will determine the overall performance.

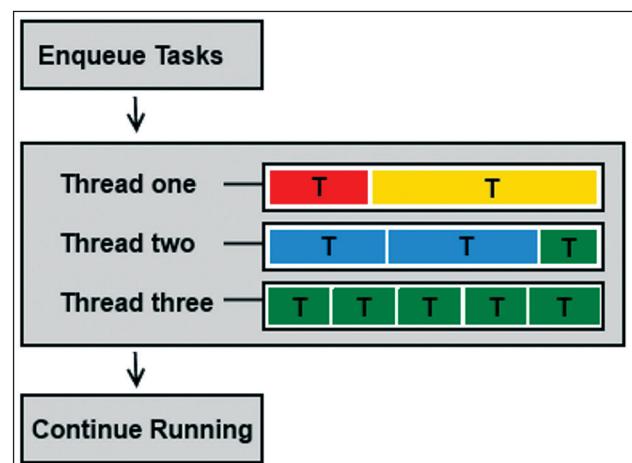


Figure 6. Tasks balancing

In general the load balancing techniques fall into two categories: centralized load balancing and distributed load balancing. Centralized scheme usually has a main node that is responsible for load handling. As the cluster size increases, for a short time the node becomes a bottleneck and causes significant performance degradation. To solve this scalability problem, the workload can be sent to multiple nodes in the cluster, and these emerging the idea of distributed dynamic load balancing. [6]

The load balancing can be achieved in the following way:

- Equal work partitioning for each received task:
 - o For array / matrix operations where each task performs similar work, the data set is evenly distributed among the tasks.

- For loop iterations where the work done in each iteration is similar, the iterations are equally distributed among the tasks.
- If a heterogeneous mix of machines with varying performance characteristics is being used, performance analysis tools need to be used in order to detect any load imbalances.
- Use of dynamic work assignment:
 - Certain classes of problems result in unequal load imbalances, even if the data is evenly distributed among the tasks:
 - * Sparse arrays - some tasks will have actual data to work with, while others will usually have a “zeros”.
 - * Adaptive grid methods - some tasks will have to rewrite their network.
 - * N-body simulations - some parts may migrate to / from the original task domain to another task domain, where parts of some tasks require more work than parts of other tasks.
 - When the workload that each task will execute is variable, or is unable to predict, then it would be useful to use a planner - *thread pool* approach. As each thread completes its work, it goes in the row in order to get a new job.
 - It may be necessary to design an algorithm which can detect and handle load imbalances, as it will appear in the code.

2.5. Granularity

The granularity of parallel programming is defined as a ratio of the time required for basic communication operation and the time required for basic computer operation. [7] In short, granularity is a qualitative measure of the calculations and communication ratio.

- *Fine-grain Parallelism* means that individual tasks are relatively small in terms of code and execution time. Data is often transmitted in a small amount of computer calculations. It makes load balancing easier.
- *Coarse-grain Parallelism* is reversed process of the previously mentioned. Here the data are transmitted less frequently after

large amounts of computer calculations. This parallelism implies a greater opportunity for increased performance, and it is more difficult to establish a load balancing.

The most efficient granularity depends on the algorithm and the hardware environment in which this algorithm is executed. In most cases the overall costs associated with communication and synchronization are in large proportion to the execution speed so it is advantageous to have coarse granularity.

3. Limitations and Expenses of the Parallel Programming

The Amdahl's law states that the potential program speedup is defined by a fraction of the code (P) which can be parallelized:

$$\text{speedup} = \frac{1}{1 - P}$$

If none part of the code can be parallelized, P=0 and speedup=1 (no speedup). If all of the code is parallelized, then P=1 and the speedup is infinite (in theory).

If 50% of the code can be parallelized, then maximum speedup =2, which means that the code will run twice as fast. Introducing the number of parallel processors, the ratio can be modeled by:

$$\text{speedup} = \frac{1}{\frac{P}{N} + S}$$

Where P = parallel fraction, n = number of processors and S = serial fraction.

It is obvious that there are limits to the parallelism scalability. For example:

N	speedup		
	P = .50	P = .90	P = .99
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1000	1.99	9.91	90.99
10000	1.99	9.91	99.02
100000	1.99	9.99	99.90

Problems that increase the percentage of parallel time with their size are more scalable than problems with a fixed percentage of parallel time.

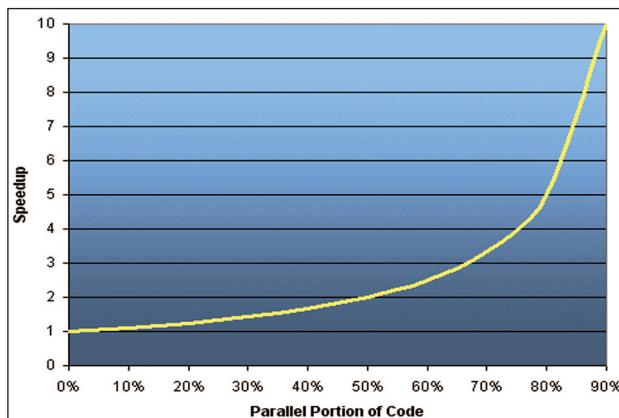


Figure 7. Parallel part of the code and acceleration (speedup)

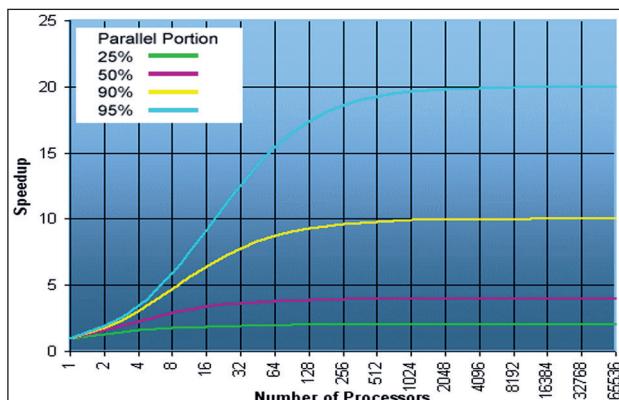


Figure 8. Number of processors and acceleration (speedup)

3.1. Complexity

In general, the parallel applications are more complex than the corresponding serial applications. They will not only have more instructions to be executed simultaneously, but also there will be a data flowing between them.

The costs of complexity are measured in programming time in every aspect of the software development cycle:

- Design,
- Coding,
- Debugging,
- Setup,
- Maintenance.

4. Experiment Description

The purpose of testing and experiments was to check the performance of several different nVIDIA GeForce GPUs (Graphical Processor Units), in order to analyze the processing speed, the startup time, the length of processing time, etc. Some of the GPUs were part of the PC, and some part of the portable computers (laptops). Moreover, some of the machines worked on 32-bit and some of them on 64 –bit operating system, so we were able to check whether these differences also play a role in getting different results during the tests. We must also mention that 4 of the computers worked on Windows 7 and one on Windows 8 Server operating system. We tested the same tests (Box Filter, Bilateral Filter and Mandelbrot) on all 5 GPUs that we used during the experiment:

- GeForce GTX 260 – PC – 64 bit Windows 7
- GeForce G105M – Laptop – 64 bit Windows 7
- GeForce GT 635M – Laptop – 64 bit Windows 7
- GeForce GTX 480 – Laptop – 64bit Windows 8 Server
- GeForce 9400M – Laptop – 32bit Windows 7.

During the experiment we obtained different results, so that certain GPU in a particular part of the test gives better results than the others.

4.1. GPU features

GPUs on which the testing was performed are of the same type, but they all have different features and offer a variety of options. Some of these features are the CUDA cores, graphics and memory clock, memory bandwidth, memory interface, supported technologies etc.

Table 1 shows the important features of the used GPUs, as well as operating system and type of computer.

5. Used techniques and programs

For the testing purposes CUDA platform and nVIDIA Visual Profiler program were used. CUDA platform is built on nVIDIA GPU processors, while nVIDIA Visual Profiler is a program in which parallel applications were made and tested on the GPUs.

Table 1. GPUs features: GeForce GTX 260, GeForce G105M, GeForce GT 635M, GeForce GTX 480, GeForce 9400M

	GeForce GTX 260	GeForce G105M	GeForce GT 635M	GeForce GTX 480	GeForce 9400M
GPU Engine Specs					
CUDA Cores:	96	8	Up to 144		
Gigaflops:	396	38			54
Graphics Clock (MHz):		1600 MHz	Up to 675 MHz		1100 MHz
Texture Fill Rate (billion/sec):			Up to 16.2	18.7	3.6
Processor Cores:					16
Memory Specs					
Memory Clock:	Up to 2000MHz	500(DDR2)/ 700(GDDR3) MHz	DDR3	1200 MHz	DDR3 1066/ DDR2 800
Standard Memory Config:		Up to 512 MB			
Memory Interface Width:	128-bit	64-bit	Up to 192bit	256-bit	128-bit
Memory Bandwidth (GB/sec):		8 (DDR2)/11 (GDDR3)	Up to 43.2	76.8	21
Feature Support					
OpenGL:	2.1	2.1	4.1	3.2	3.3
Supported Technologies:	CUDA, PhysX	CUDA	CUDA, OpenCL, DirectCompute, 3D Vision, DirectX 11, Optimus, PhysX	SLI, CUDA, 3D Vision, DirectX 11, PhysX	SLI, CUDA, PhysX
Operating System and Computer Type	Windows 7, 64 bit PC	Windows 7, 64 bit, Laptop	Windows 7, 64 bit, Laptop	Windows 8, 64 bit, Laptop	Windows 7, 32 bit, Laptop

5.1. CUDA (Compute Unified Device Architecture)

CUDA is a parallel programming platform created by nVIDIA and implemented in their processors (GPU). [8]

To understand CUDA, we must first know what GPGPU (general-purpose computing on graphics processing units) is. Simply put, it is the technique in which the GPU is employed to handle and perform computations that were previously handled only by the CPU. However, the GPU doesn't have the same flexibility and calculation precision as the CPU that's built for general purpose usage, and this is where CUDA comes in.

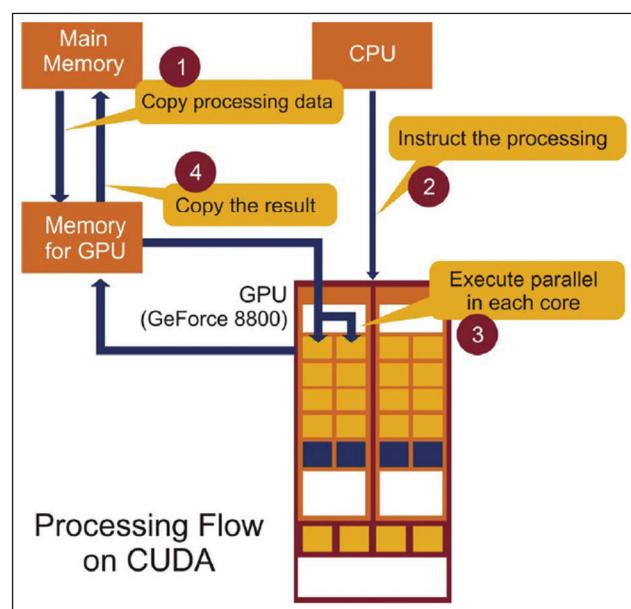


Figure 9. A flowchart showing how CUDA allows the GPU to work in tandem with the CPU

CUDA allows developers access to the virtual instruction and parallel computing elements memory in CUDA GPU. The GPUs have parallel architecture that allows execution of multiple segments going slower rather the fast, and subsequently performing of the segments.

CUDA program executes serial code on the CPU, which then calls a series of cores running on the GPU. [9]

CUDA provides low and high level API (Application Programming Interface). It works with all nVIDIA GPU G8x series onwards, including GeForce, Quadro and Tesla and it is compatible with most of the operating systems. There are several advantages that distinguish this platform compared to others:

- The code can read information from arbitrary addresses in the memory,
- There is a large shared memory (up to 48KB of multiprocessors) that can be used by all,
- Faster download from and to GPU,
- Full support for integer and bitwise, including text searches. [10]

5.2. nVIDIA Visual

The program that we used to perform GPU tests is nVIDIA Visual Profiler. This tool is a crossover platform that allows developers to get vital feedback for optimizing CUDA C / C ++ applications.

The tool displays a timeline for the CPU and GPU activities while running the application and it includes automated analysis to identify optimizing opportunities. [11]

6. Use of abbreviations

This section shows the obtained results for all tested applications, as well as a comparison of these results.

6.1. Bilateral Filter

Bilateral Filter is a nonlinear smoothing filter that is implemented with CUDA and OpenGL rendering. It can be used for image recovery. [12]

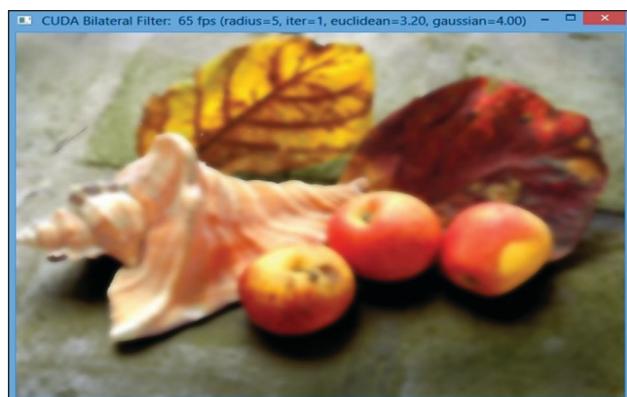


Figure 10. Bilateral Filter application

Bilateral Filter has been tested on all five graphics cards. Table 2 gives the results for the startup time, the test duration, the GPU throughput and the number of instructions required to perform the test.

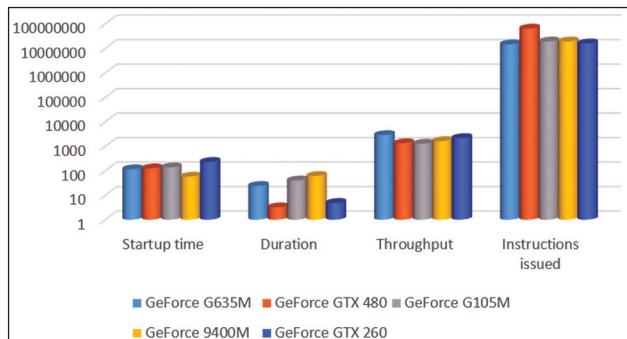


Figure 11. Results obtained by Bilateral Filter testing

Figure 11 shows the graph of the results of Bilateral Filter testing that has been done. From the graph we see that there is a big difference in the performance of these activities for all GPUs. From the comparison of the startup time, it can be noted that the GeForce 9400M (single computer

Table 2. Results from the Bilateral Filter testing.

	Startup time	Duration	Throughput	Instructions issued
GeForce G635M	115,12 ms	24,005 ms	2856,96 MB/s	14490459
GeForce GTX 480	125,218 ms	3,211 ms	1320,96 MB/s	63804424
GeForce G105M	136,604 ms	40,028 ms	1269,76 MB/s	18950424
GeForce 9400M	57,475 ms	60,899 ms	1617,92 MB/s	18916815
GeForce GTX 260	229,229 ms	4,864 ms	2160,64 MB/s	15792211

with 32-bits OS) has a far better startup time than other GPUs that have similar startup time. Furthermore, if we consider the time required to perform the test, it can be noticed that two of the five considered GPUs have significantly less execution time, which means that the GeForce GTX 480 and GeForce GTX 260 have a better execution time compared to the other graphics cards.

In terms of throughput, it is evident that it is approximately the same for almost all GPUs except GeForce G635M which has higher throughput than others. If we compare the values of the GPUs for required instructions, it can be noticed that there is a huge difference between the computer that has Windows 8 operating system, and other computers with Windows 7 operating system. GeForce GTX 480 (Windows 8) needs almost six times more instructions compared to all other graphics that need nearly the same number of instructions.

6.2. Mandelbrot

This is the second test that we performed in order to compare the GPUs, and obtained results are shown in Table 3. The principal activities which were of our interest are the same as in Bilateral Filter testing (startup time, duration, throughput and number of required instructions for execution).

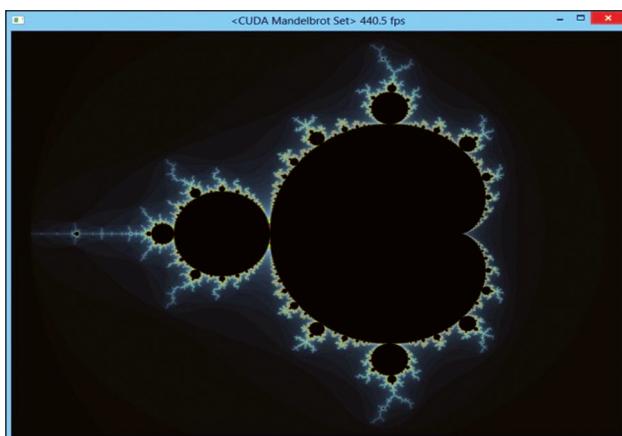


Figure 12. Mandelbrot application

Table 3. Results from the Mandelbrot testing

	Startup time	Duration	Throughput	Instructions issued
GeForce G635M	496,506 ms	4,448 ms	4689,9 KB/s	4621752
GeForce GTX 480	388,905 ms	0,638 ms	4403,2 KB/s	20873774
GeForce G105M	458,15 ms	15,692 ms	700,88 KB/s	10901919
GeForce 9400M	1968 ms	17,169 ms	659,84 KB/s	10145804
GeForce GTX260	303,448 ms	1,393 ms	943,49 KB/s	560690

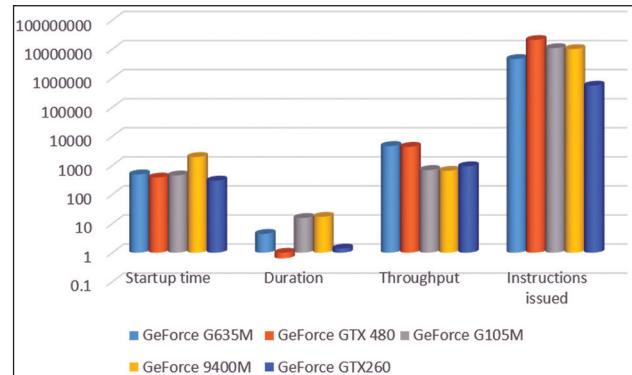


Figure 13. Results obtained by Mandelbrot testing

The figure 13 shows the graph of the values obtained by Mandelbrot test. From the graph we can see that the startup time of the GeForce 9400M differs compared to the other GPU, but now this time it is slower.

Regarding the execution duration, the graph shows a huge difference in the results.

GeForce GTX 480 performs the test in less than 1ms, and it has the shortest execution time. Very similar results show the GeForce GTX 260, but the other three GPUs need a significantly longer time to execute the test. Throughput is approximately same for the first two cards, and it is about five times higher compared to the next three GPUs. The instruction issued are higher for GeForce GTX 480, and half of this instructions are enough for GeForce G105M, while the GeForce GTX260 requires much less instructions.

6.3. Boxfilter

The last test is BoxFilter testing. As in previous experiments, here we had compared the startup time, execution duration, the throughput and instructions required for execution. Table 4 shows the results of this testing, and it is followed by a description of the resulting graph (Figure 15).



Figure 14. Boxfilter application

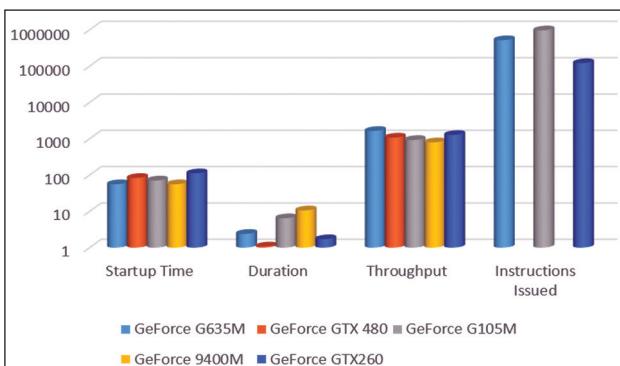


Figure 15. Results obtained by BoxFilter testing

For this test, GeForce GTX260 has the longest startup, while the GeForce G635M and the GeForce 9400M have almost the same shortest startup time. The situation with the execution duration is again the same as with the previous two tests, confirm-

ing that the GeForce GTX480 has the shortest test duration, and again, GeForce GTX260 is close to it, while the other three cards need significantly more time to execute the test. The throughput is similar in all cards, and if we compare all results it can be seen that GeForce G635M has highest throughput, while GeForce 9400M shows worst results. For issued instructions we get data only for three of the five GPUs, and according to the obtained results the GeForce G105M needs most instructions.

6.4. Device Query

Device query lists the features of the CUDA devices in the system. We have tested four features: the total amount of memory bytes per block, the total amount of registers available per block, the maximum number of threads per multiprocessors and the maximum number of threads per block. The obtained results are given in Table 5. The Figure 16 shows a comparison of the obtained results for the different GPUs.

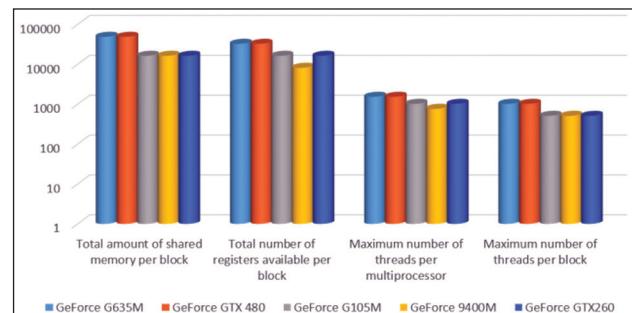


Figure 16. Results obtained from Device query

Table 4. Results from the BoxFilter testing

	Startup Time	Duration	Throughput	Instructions Issued
GeForce G635M	55,828 ms	2,397 ms	1638,4 KB/s	516778
GeForce GTX 480	82,955 ms	1,079 ms	1075,2 KB/s	
GeForce G105M	70,698 ms	6,467 ms	921,65 KB/s	963511
GeForce 9400M	55,694 ms	10,605 ms	793,07 KB/s	
GeForce GTX260	111,635 ms	1,709 ms	1269,76 KB/s	120473

Table 5. Results from the Device query

	Total amount of shared memory bytes	Total amount of registers available per block	Maximum number of threads per multiprocessor	Maximum number of threads per block
GeForce G635M	49152 B	32768 B	1536	1024
GeForce GTX 480	49152 B	32768 B	1536	1024
GeForce G105M	16384 B	16384 B	1024	512
GeForce 9400M	16384 B	8192 B	768	512
GeForce GTX260	16384 B	16384 B	1024	512

From the results it can be noticed that the GeForce G635M and the GeForce GTX 480 have approximately the same results for all tested features, and the other three GPUs are similar to each other, but with slightly weaker features compared to the GeForce G635M and the GeForce GTX 480.

6.5. Bandwidth Test

We also made a comparison of the information bandwidth for the GPUs, for the cases of host to device, device to host and device to device communication. The obtained results are shown in Table 6 and Figures 17.

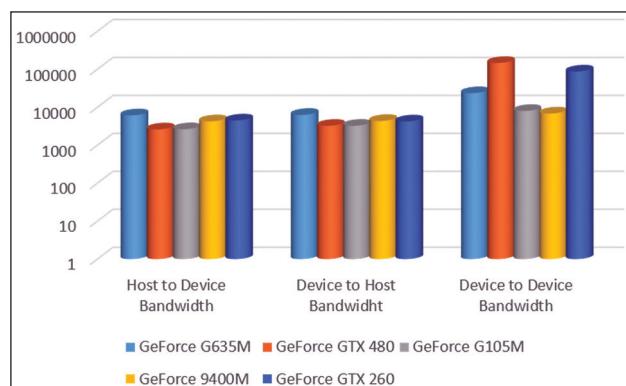


Figure 17. Results obtained by Bandwidth test

The results conclude that the highest bandwidth for host to device and device to host connection gives GeForce 635M, while during the device to device information exchange GeForce GTX 480 has a significantly higher bandwidth.

In some of the cases, some of the obviously weaker GPUs give better results than the GPUs with better characteristics. The main reason for this is that they skip some parts of the tests that are not compatible due to the lack of some features of these graphics, as for example less CUDA cores or less bits on the memory interface.

Table 6. Information Bandwidth

	Host to Device Bandwidth	Device to Host Bandwidth	Device to Device Bandwidth
GeForce G635M	6128 MB/s	6300,4 MB/s	23489,1 MB/s
GeForce GTX 480	2612,4 MB/s	3243,5 MB/s	146736,2 MB/s
GeForce G105M	2635,2 MB/s	3250,9 MB/s	7983 MB/s
GeForce 9400M	4257,3 MB/s	4354,8 MB/s	6842,3 MB/s
GeForce GTX 260	4494,9 MB/s	4239,2 MB/s	87432,8 MB/s

7. Conclusion

The parallel programming requires systems with good performances. From the obtained results during the experiments, it can be noted that the type of the GPU and the type of the operating system play a very important role in the parallel computing.

According to the tests on different GPUs for multiple applications created with parallel programming, we came to the results in which we cannot give the correct answer to the question which is the best GPU. This is due to the different graphics power, different number of CUDA cores, and different operating systems. However, we can see that every GPU is better than the others in different segments and in different parts of the test, which depends on the:

- power,
- number of cores,
- poor performances (what was the reason why all tests were not supported)
- bits of the memory interface
- memory bandwidth, etc..

Although the parallel programming requires expensive machines, however the number of users of this type of computing continues to increase due to performance for better and easier large problems solving that cannot or are hard to be solved by classic serial programming.

References

1. Barney B. *Introduction to parallel computing.* (Online), Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/parallel_comp/.
2. Faxen KF, Albertsson L, Popov K, Janson S. *Embla-Data dependence profiling for parallel programming.* International Conference on Complex, Intelligent and Software Intensive Systems, 2008. CISIS 2008.
3. Foster I. *Designing and Building parallel programs.* (Online), <http://www.mcs.anl.gov/~itf/dbpp/>.
4. Hossain MA, Kabir U, Tokhi MO. *Impact of data dependencies in real-time high performance computing.* Microprocessors and Microsystems, accepted 17 April 2002, 2002; 26: 253-261.
5. Peterson PM, Padua DA. *Static and dynamic evaluation of data dependence analysis techniques.* IEEE transactions on parallel and distributed systems, November 1996; 7(11).
6. Kwiatkowski J. *Evaluation of parallel programs by measurement of its granularity.* Computer Science Department, Wroclaw University of Technology, Poland.
7. Qin X, Jiang H, Manzanares A, Ruan X, Yin S. *Communication-Aware load balancing for parallel applications on clusters.* IEEE Transactions on computers, January 2010; 59(1).
8. CUDA Parallel Computing Platform, nVidia, http://www.nvidia.com/object/cuda_home_new.html
9. Dehne F, Yogaratnam K. *Exploring the limits of GPUs with parallel graph algorithms.* School of Computer Science Carleton University, Ottawa, Canada, February 24, 2010.
10. Sanders J, Kandrot E. *CUDA by Example: An Introduction to General-Purpose GPU Programming.*
11. nVIDIA, *Profiler User's Guide, DU-05982-001_v5.5,* http://docs.nvidia.com/cuda/pdf/CUDA_Profiler_Users_Guide.pdf, July 2013.
12. CUDA Toolkit Documentation, <http://docs.nvidia.com/cuda/cuda-samples/index.html#bilateral-filter>

*Corresponding Author
Mitko Bogdanoski,
Military Academy "General Mihailo Apostolski",
Goce Delcev University,
Skopje,
R. Macedonia,
E-mail: mitko.bogdanoski@ugd.edu.mk*