ВТУ „СВ. СВ. КИРИЛ И МЕТОДИЙ”

ФАКУЛТЕТ „МАТЕМАТИКА И ИНФОРМАТИКА”

ЮБИЛЕЙНА НАУЧНА КОНФЕРЕНЦИЯ

25 ГОДИНИ

ФАКУЛТЕТ „МАТЕМАТИКА И ИНФОРМАТИКА”

27 – 28 ноември 2015

ВЕЛИКО ТЪРНОВО

2015

ВТУ „СВ. СВ. КИРИЛ И МЕТОДИЙ”
ЮБИЛЕЙНА НАУЧНА КОНФЕРЕНЦИЯ
25 ГОДИНИ
ФАКУЛТЕТ „МАТЕМАТИКА И ИНФОРМАТИКА”
27 – 28 ноември 2015
ТРЕТИ УЧЕБЕН КОРПУС НА ВТУ „СВ. СВ. КИРИЛ И МЕТОДИЙ”
АУДИТОРИЯ „ДЖОН АТАНАСОВ”

ПРОГРАМЕН КОМИТЕТ:

Председател:

проф. д-р Тихомир Стефанов Трифонов

проф. дмн Стефка Христова Буюклиева
проф. д-р Маргарита Кръстева Тодорова
проф. д-р Мирослав Николов Гълъбов
проф. д-р Емилия Стойчева Тодорова
проф. д-р Милко Дамянов Такев
доц. д-р Милен Йорданов Христов
доц. д-р Златко Георгиев Върбанов
доц. д-р Александър Маринов Ковачев

ОРГАНИЗАЦИОНЕН КОМИТЕТ:

Председател:

проф. д-р Даринка Ненчева Гълъбова

проф. д-р Виолета Маринова Маринова
проф. д-р Маргарита Генова Върбанова
проф. д-р Елена Иванова Георгиева
доц. д-р Галя Василева Накова
доц. д-р Николай Пенчев Николов
доц. дн Олег Димитров Асенов
доц. д-р Ивайло Петров Дончев
доц. д-р Марияна Иванова Николова
доц. д-р Емилиян Георгиев Петков
доц. д-р Живка Василева Радева
доц. д-р Тодор Йорданов Тодоров
доц. д-р Доника Веселинова Вълчева
гл. ас. д-р Мирослав Димитров Петров

СЪДЪРЖАНИЕ

ИНФОРМАТИКА И КОМПЮТЪРНИ НАУКИ

МАТЕМАТИКА И ПРИЛОЖНА МАТЕМАТИКА

ОБРАЗОВАНИЕТО ПО МАТЕМАТИКА И ИНФОРМАТИКА. БИБЛИОТЕЧНО-ИНФОРМАЦИОННИ ДЕЙНОСТИ. КНИГОИЗДАВАНЕ.

Walsh Transform Algorithm and its Parallel Implementation with CUDA on GPUs

Dusan Bikov and Iliya Bouyukliev

Abstract: *This paper discusses different approaches for computing the Walsh spectra on graphics processor unit (GPU) using CUDA C. The aim here is to present an efficient algorithm for calculation of the Walsh spectrum. The main task is an algorithm for sequential calculation of Walsh spectrum and its parallel implementation in CUDA C. Moreover, we will present our experimental results.*

Key words: *Walsh transforms, CUDA C, GPU, Fast Walsh Transform.*

INTRODUCTION

The use of modern graphics processing units (GPUs) has become attractive for scientific computing which is due to its massive parallel processing capability. Modern GPUs are more than very efficient device use for rendering the graphics and accelerate the creation of images, their highly parallel structure makes them more effective than general-purpose CPU for algorithm where processing of large blocks of data is done in parallel [1] [2]. Compared with multi-core CPUs, new generation GPUs can have much higher computation power and memory bandwidth. Therefore they are attractive in many application areas. One of the most important application domains is the linear algebra [3] [4].

Boolean functions are basic objects in discrete mathematics. A boolean function f of n variables is a mapping from F_2^n into F_2, where $F_2 = \{0,1\}$ is a field with 2 elements. Truth Table is 2^n- dimensional vector which has the function values of f for all vectors from F_2^n as coordinates. We can consider the vectors in F_2^n as binary presentations of the integers in the interval $[0,...,2^n - 1]$. This consideration is very useful when we try to describe and explain some transformations of boolean functions and related algorithms.

Very important cryptographic property of a boolean function f is its non-linearity which is related to the distances from f to the linear functions. The aim here is to present an efficient algorithm for the calculation of the Walsh spectrum [5]. Practically, the considered algorithm can be presented as a matrix vector multiplication. In our case the considered matrices have not only recursive structure but this structure is quite specific and enables a very effective (*butterfly*) multiplication.

The purpose of this paper is to assess the performance of the recent, inexpensive and widely used NVIDIA GPUs in performing Walsh Transforms. Our approach for the calculation of the Walsh spectrum is a Fast Walsh Transform and the mathematical background for this approach will be described below in this paper. Here we experiment with different models for calculation and present the results of these experiments. Also we are focused on reducing the time for calculation of Walsh spectra for different sizes of the considered elements.

GPU COMPUTING MODEL WITH CUDA

GPUs are designed for efficient execution of thousands of threads in parallel on as many processors as possible at each moment. The computation processes are divided into many simple tasks that can be performed at the same time. This intensive multi-threading allows execution of various tasks on the GPU processors while data is fetched from or stored to the GPU global memory. It also ensures the scalability of the GPU computing model, since processors are abstracted as threads, and support parallel programming model [6].

A simple way to understand the difference between CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing while a GPU has a massively parallel architecture consisting of thousands of smaller, more efficient cores

designed for handling multiple tasks simultaneously. This ability of a GPU with hundred and more cores to process thousands of threads can significantly accelerate the software over a CPU.

Modern NVIDIA GPUs are powerful computing platform developed for general purpose computing using CUDA (Compute Unified Device Architecture) [7]. It allows programmers to interact directly with the GPU and run programs on them, thus effectively utilizing the advantages of parallelization. Depending on architecture CUDA cores can be organized into SMs (streaming multiprocessor), each having a set of registers, constants and texture caches, and on-chip shared memory as fast as local registers (one cycle latency). At any given cycle, each core executes the same instruction on different data (SIMD), and communication between multiprocessors is performed through global memory. As a programming interface, CUDA C is not a new language, it is a set of C language library functions with GPU specific commands, options and operations [9], and the CUDA-specific nvcc compiler generates the executable for the NVIDIA GPU from a source code.

LINEAR BOOLEAN FUNCTIONS AND WALSH SPECTRUM

Let $f(x) = u_1x_1 \oplus u_2x_2 \oplus \ldots \oplus u_nx_n$ be a linear boolean function of n variables. We use the notation $u_1x_1 \oplus u_2x_2 \oplus \ldots \oplus u_nx_n = x^{(\oplus u)}$. The binary n-dimensional vector u uniquely defines $f(x)$ and therefore we denote it by $f^{(\oplus u)}(x)$. The Truth Table of $f^{(\oplus u)}(x)$ has the form

$$\begin{pmatrix} f^{(\oplus u)}(\bar{0}) \\ f^{(\oplus u)}(\bar{1}) \\ \vdots \\ f^{(\oplus u)}(\overline{2^n-1}) \end{pmatrix} = \begin{pmatrix} \bar{0}^{(\oplus u)} \\ \bar{1}^{(\oplus u)} \\ \vdots \\ \overline{(2^n-1)}^{(\oplus u)} \end{pmatrix} = (S_{mat}^{(n)})^{(\oplus u)}$$

The values of the linear functions for $\bar{0},\bar{1},\ldots,\overline{2^n-1}$ form the following matrix:

$$H_n^+ = \begin{pmatrix} \bar{0}^{\oplus\bar{0}} & \bar{0}^{\oplus\bar{1}} & \cdots & \bar{0}^{\oplus\overline{2^n-1}} \\ \bar{1}^{\oplus\bar{0}} & \bar{1}^{\oplus\bar{1}} & \cdots & \bar{1}^{\overline{2^n-1}} \\ & & \cdots & \\ \overline{(2^n-1)}^{\oplus\bar{0}} & \overline{(2^n-1)}^{\oplus\bar{1}} & \cdots & \overline{(2^n-1)}^{\oplus\overline{2^n-1}} \end{pmatrix}$$

Hence

$$H_n^+ = \left((S_{mat}^{(n)})^{\oplus\bar{0}}, \quad (S_{mat}^{(n)})^{\oplus\bar{1}} \quad ,\ldots, \quad (S_{mat}^{(n)})^{\oplus\overline{2^n-1}} \right)$$

$$= \left(\begin{pmatrix} 0S_{mat}^{(n-1)} \\ 1S_{mat}^{(n-1)} \end{pmatrix}^{\oplus\bar{0}} \cdots \begin{pmatrix} 0S_{mat}^{(n-1)} \\ 1S_{mat}^{(n-1)} \end{pmatrix}^{\oplus\overline{2^{n-1}-1}} \begin{pmatrix} 0S_{mat}^{(n-1)} \\ 1S_{mat}^{(n-1)} \end{pmatrix}^{\oplus\overline{2^n-1}} \cdots \begin{pmatrix} 0S_{mat}^{(n-1)} \\ 1S_{mat}^{(n-1)} \end{pmatrix}^{\oplus\overline{2^n-1}} \right)$$

For the matrix H_n^+ we have

$$\left(\left(0S_{mat}^{(n-1)}\right)^{\oplus\bar{0}} \ldots \left(0S_{mat}^{(n-1)}\right)^{\oplus\overline{2^{n-1}-1}} \right) = \left(\left(1S_{mat}^{(n-1)}\right)^{\oplus\bar{0}} \ldots \left(1S_{mat}^{(n-1)}\right)^{\oplus\overline{2^{n-1}-1}} \right) = H_{n-1}^+,$$

$$\left(\left(1S_{mat}^{(n-1)}\right)^{\oplus\overline{2^{n-1}}} \ldots \left(1S_{mat}^{(n-1)}\right)^{\oplus\overline{2^{n-1}}} \right) = \overline{H_{n-1}^+},$$

where the matrix $\overline{H_{n-1}^+}$ is obtained from H_{n-1}^+ after replacing 0 by 1 and 1 by -1. It follows that

$$H_{n-1}^{+}=\begin{pmatrix} H_{n-1}^{+} & H_{n-1}^{+} \\ H_{n-1}^{+} & \overline{H_{n-1}^{+}} \end{pmatrix}, H_1^{+}=\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, H_2^{+}=\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}.$$

It is easy to see that H_n is a symmetric matrix. Its rows (and columns) form n dimensional linear space. In coding theory this space (without zero coordinate) is known as a simplex code. This space together with its coset with representative (11 . . . 1), form the first order Reed-Muller code.

Let $a=(a_1,a_2,...,a_m)$ be a binary vector. The polarity representation $a^{(p)}$ of a is obtained from a after replacing 0 by 1 and 1 by −1. Consider the scalar product $s=a^{(p)}\cdot b^{(p)}$ over the integers. Let s^- (respectively s^+) be the number of the coordinates, for which $a_j^{(p)}b_j^{(p)}=-1$ (respectively $a_j^{(p)}b_j^{(p)}=+1$. Then $s^-=d(a,b)$ is the number of coordinates with different value for a and b. And s^+ is the number of the coordinates with equal values for a and b. We have that $s=s^+-s^-$ and $m=s^++s^-$ or $s^-=(m+s)/2, s^+=(m+s)/2$.

Let us denote by $PTT(f)$ and H the polarity representations of $TT(f)$ and the matrix H^+. The vector $W_f=H\cdot(PPT(f))^t=(f^w(\bar{0}), f^w(\bar{1}),..., f^w(\overline{2^n-1}), W_f=(W_0,...,W_{2^n-1})$, is called Walsh spectrum, and the function $f^w(\overline{a})$ defines the Walsh transform. The value W_i determines the distance between the Truth Table of f and the Truth Table of the linear function $x^{\oplus i}$, which is equal to $(2^n-W_i)/2$, and also the distance between $TT(f)$ and the Truth Table of the affine function $1+x^{\oplus i}$ which equal to $(2^n-W_i)/2$.

Matrix vector multiplication $H\cdot(PPT(f))^t$ can be given by a butterfly diagram and a corresponding algorithm, namely Diagram 2 and Algorithm 2 [8]. This algorithm passes all elements of the matrix $S_{mat}^{(n)}$ in n steps column by column starting from the last one. Depending on the value in the i-th row and $(n-j+1)$-th column of the matrix $S_{mat}^{(n)}$ the algorithm calculates a new values for $W_f[i]$ and $W_f[i+2^i]$. This algorithm entirely depends on the binary representation of the nonnegative integers smaller than 2^n.

Fast Walsh Transform can be implement parallel, by using base concept on Algorithm 2 [8] but with acceptable modification to be suitable for parallel implementation. For our parallel implementation we use CUDA C and we make several versions where we use various optimization techniques, model and different memory to get better performance and efficiency.

EXPERIMENTAL EVALUATION

In the previous section we mentioned that we make several parallel implementation versions of Fast Walsh Transform, and every version has improvements in execution time, compared with the prevision version or it is an experiment which helps us to understand better the problem. Each parallel version is implemented in CUDA C.

The first version is based on Algorithm 2 [8] but with suitable modification in order to implement it in parallel. Here we have a problem with synchronization on threads from different blocks [8], and our solution is the calling the kernel function from the main function n times (number of steps). Another problem is the memory usage and entirely we use global memory but it is the slowest memory in GPU. All other versions are modifications of the first one.

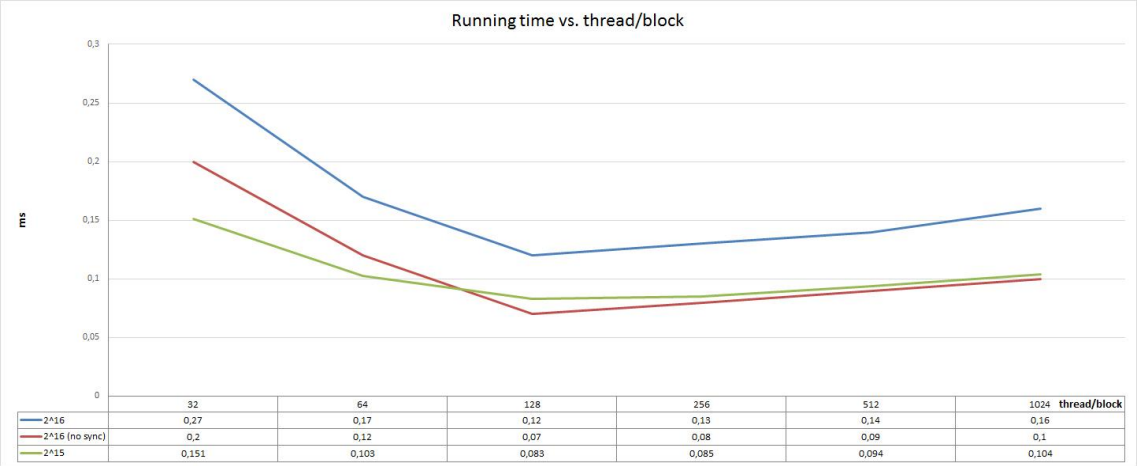
The second version is a modification of the first version, and in kernel function statement if – else is replaced with algebraic expression to avoid warp divergence.

The third version again is a modification on the first one, and here we increase the work per thread. In this way we get slower and slower execution time.

The fourth version is a version where we use shared memory combined with the first version. On first stage we use shared memory for calculations until certain steps, depending on the shared memory limitation, and from then on the calculations are done with the global memory.

The fifth version is a version where we use shared memory combined with the memory pattern. Memory pattern is for tracking the intermediate results from the steps of the calculation and comes from the shared memory limitation. Here we obtain better performance compared with the other versions.

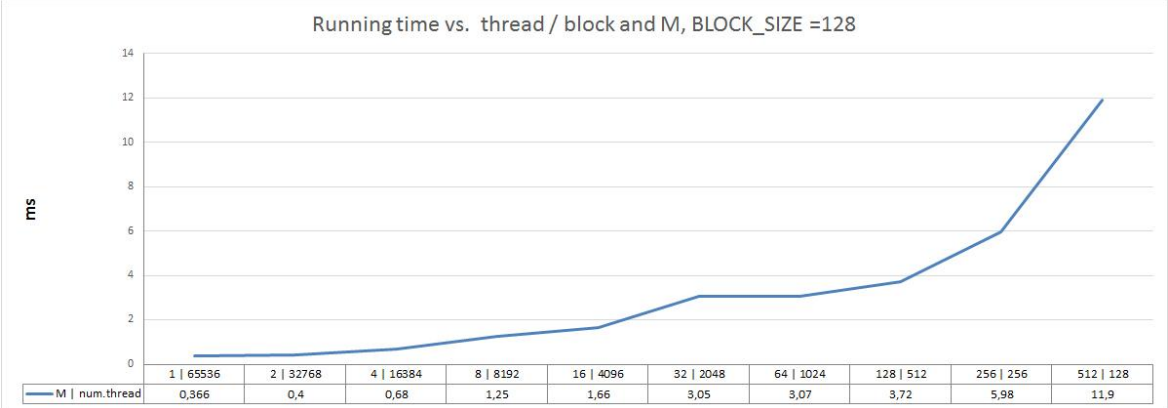
The experiments were performed using the following computer configuration: Intel i3-3110M [9] with 2.4 GHz and 4 GB of RAM and NVIDIA GeForce GT 740M [10], cards with a total of 384 cores running at 0.9 GHz and a 28.8 GB/sec memory bandwidth. The CUDA [7] kernel were developed using MS Visual Studio 2010, Active solution configuration Release, Active solution platform Win32.



thread/block	32	64	128	256	512	1024
2^16	0,27	0,17	0,12	0,13	0,14	0,16
2^16 (no sync)	0,2	0,12	0,07	0,08	0,09	0,1
2^15	0,151	0,103	0,083	0,085	0,094	0,104

Figure 1. Relationship between time and number of threads/block

Figure 1 shows the execution time for calculating the Walsh spectrum for the different number of threads per block. The first and the second versions compute Walsh spectrums on 2^{16} elements, but the second one does not use synchronization (in some cases the program without synchronization does not give a right answer but there is another problem – the synchronization slows down the execution time). The third version computes Walsh spectrum on 2^{15} element. From this we can evaluate that we have fast execution time when using 128 threads per block and also can see the price of synchronization. Price of synchronization is approximately one step of calculation.



M | num.thread	1 | 65536	2 | 32768	4 | 16384	8 | 8192	16 | 4096	32 | 2048	64 | 1024	128 | 512	256 | 256	512 | 128
	0,366	0,4	0,68	1,25	1,66	3,05	3,07	3,72	5,98	11,9

Figure 2. Work per tread vs. execution time

Result from increasing work per thread versus execution time is shown in figure 2. Here we calculate Walsh spectrum on 2^{16} elements, and from right to left it shows the increasing work per tread or we have less threads but still the same 2^{16} elements for computation.

The comparison between the CPU Algorithm 2 [8], first, fourth and fifth version is shown on figure 3.

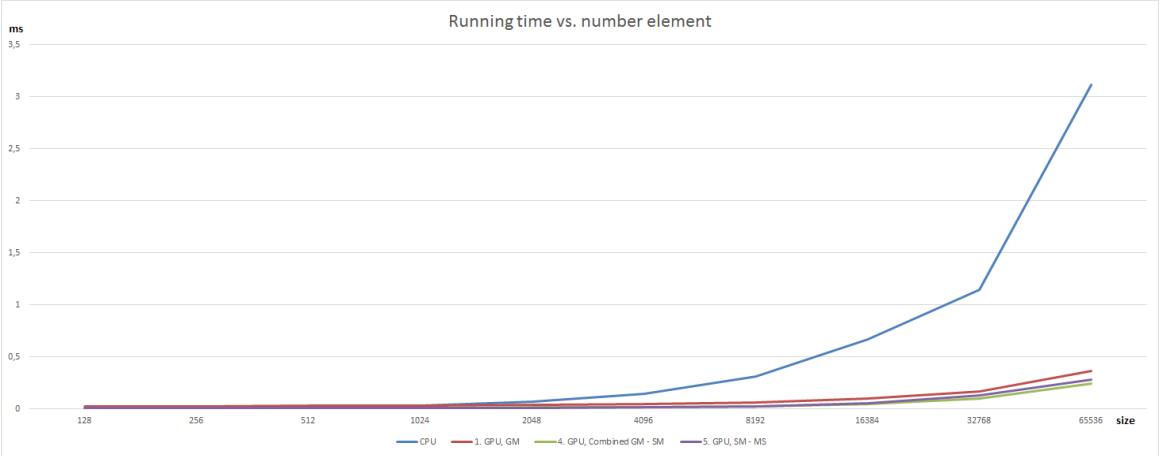


Figure 3. Time for calculating [Wf] CPU vs. different GPU implementation

The numbers next to the line show the version of parallel implementation and the first blue line is the CPU Algorithm 2 [1] implementation in C++. Because of the resolution of the figure it is difficult to notice and to make statements. Obviously in one point GPU implementation becomes faster, but the details are shown in table 1.

Speed up is defined by:

$$S_p = \frac{T_{(1,n)}}{T_{p(n)}}$$

where $T_{(1,n)}$ is the run-time of the fastest known sequential algorithm and $T_{p(n)}$ is the run-time of the parallel algorithm, and n is the size of the input.

TABLE I. CPU VS. GPU IMPLEMENTATION, SPEED UP FOR DIFFERENT NUMBER ELEMENT

Number element	CPU FWT	1. GPU, GM	Speed up CPU vs. 1.GPU, GM	GPU, 4.GM - SM	Speed up CPU vs. 4. GPU, GM-SM	5. GPU, SM	Speed up CPU vs. 5.GPU, SM
128	0,003	0,024	/	0,0066	/	0,0066	/
256	0,007	0,026	/	0,0066	1.060	0,0066	1.060
512	0,015	0,028	/	0,0069	2.272	0,0069	2.272
1024	0,033	0,034	/	0,0071	4.647	0,0071	4.647
2048	0,068	0,039	2	0,013	5.230	0,0124	5.483
4096	0,145	0,048	3.02	0,019	7.631	0,0147	9.863
8192	0,308	0,062	4.967	0,026	11.846	0,023	13.391
16384	0,665	0,096	6.927	0,048	13.854	0,052	12.788
32768	1,1487	0,165	6.961	0,1	11.487	0,13	8.836
65536	3,116	0,366	8.513	0,24	12.983	0,28	11.128
131072	6,87	1,561	4.401	0,85	8.082	0,595	11.546
262144	14,818	3,571	4.149	2,056	7.207	1,207	12.276

Table 1 shows different implementations of FWT for different number of elements and speed ups which appear in the GPU implementation. An important conclusion is that the GPU processing makes sense only for large size problems. It is important that the prices of buffer creation and transfers have to be acceptable, so in this particular case when GPU is used, for fourth and fifth version on implementation this experiment shows that the number of elements has to be at least 256. CPU is faster for small problems and can work faster than couple threads, which

is the reason for this limitation. For more elements more threads are used and therefore the computation is faster than in the case of sequential programming. However, there are boundaries (limitations) which depend on several things (the problem, the algorithm, GPUs, the libraries, the model, etc.).

Another interesting observation about the fourth and fifth version is intersection on time executions. In one point the memory pattern has higher price (spends more time on memory movement) than shared memory computations.

CONCLUSION

In this paper we proposed a performance model for computing Walsh transform with wide use NVIDIA GPU by using popular models in the parallel algorithm community. In this paper we presented the effect of considering a CPU versus GPU speed up contrasted by the use of GPU versus GPU speed up measures. Wide used modern GPU has become attractive for scientific computing. This is one of the many examples and here we can see the benefits of using it. Note that here we use low class of GPU.

This experiment shows that parallel version of implementation proposed, in CUDA C, still can be improved. One of the problems here is the synchronization and it affects the performance, the number of threads per block has influence in the time execution and etc. By choosing proper optimization techniques and appropriate methods an increased efficiency can be obtained and the performance can be improved. This will be a part of our future research.

REFERENCES

1. D. Gajic, R. Stankovic, GPU Accelerated Computation of Fast Spectral Transforms", Facta universitatis (Nis) Electronics and Energetics 2011 Volume 24, Issue 3, Pages: 483-499, 2011
2. D. A. Jamshidi, M. Samadi, S. Mahlke, "D2MA: accelerating coarse-grained data transfer for GPUs " PACT '14 Proceedings of the 23rd international conference on Parallel architectures and compilation Pages 431-442, ISBN: 978-1-4503-2809-8, 2014.
3. J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. "GPU Computing". Proceedings of the IEEE, 96(5):879–899, May 2008.
4. P. Maciol and Krzysztof Banas. "Testing tesla architecture for scientific computing: The performance of matrix-vector product." In Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on, pp. 285-291. IEEE, 2008.
5. Carlet C. Boolean Functions for Cryptography and Error Correcting Codes. In: Crama C, Hammer PL, (Eds.), Boolean Models and Methods in Mathematics, Computer Science, and Engineering, Cambridge University Press, 257–397. 2010.
6. NVIDIA, OpenCL Programming Guide for the CUDA Architecture, 2011.
7. CUDA homepage: http://www.nvidia.com/object/cuda_home_new.html.
8. Iliya Bouyukliev, Dusan Bikov. Applications of the binary representation of integers in algorithms for boolean functions. SMB, 2015.
9. i3-3110M specification [Online]. Available: http://ark.intel.com/products/65700/Intel-Core-i3-3110M-Processor-3M-Cache-2_40-GHz.
10. NVIDIA GeForce GT 740M specification [Online]. Available: http://www.geforce.com/hardware/notebook-gpus/geforce-gt-740m/specifications.
11. CUDA C Programming Guide: https://docs.nvidia.com/cuda/cuda-c-programming-guide/

ABOUT THE AUTHORS

Dusan Bikov, Department of Information Technologies, Faculty of Mathematics and Informatics, Veliko Turnovo University, e-mail: dule.juve@gmail.com

Iliya Bouyukliev, Institute of Mathematics and Informatics, Bulgarian Academy of Sciences, e-mail: iliyab@math.bas.bg

Contact details:

Dusan Bikov, Department of Information Technologies, Faculty of Mathematics and Informatics, Veliko Turnovo University, e-mail: dule.juve@gmail.com