Using GPU matrix vector multiplication for computing Walsh spectra

Dusan Bikov, Aleksandra Stojanova
Faculty of Computer Science
"Goce Delcev" University -UGD
Stip, Macedonia
{dusan.bikov, aleksandra.stojanova}@ugd.edu.mk

***Abstract* — This paper discusses different approaches for computing the Walsh spectra on graphics processor unite (GPU) using CUDA C. Here we present the results from several experiments that evaluate the performance of NVIDIA processors, implementing on two GPU with different performances. The conclusions from experiments made indicate what speed-ups can be expected, when instead of standard CPUs, accelerators in the form of presented GPUs are used, for considered computational kernels.**

Keywords — Walsh transforms, Hadamard matrix, CUDA, GPU

I. INTRODUCTION

The use of modern graphics processing units (GPUs) has become attractive for scientific computing which is due to its massive parallel processing capability. The GPU's advanced capabilities were originally used primarily for 3D game and graphics rendering but today's modern GPUs are more than very efficient devices used for rendering the graphics and accelerate the creation of images. Their highly parallel structure makes them more effective than general-purpose CPU for algorithmic tasks because 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. Capabilities of GPUs are used to accelerate computational workloads in financial modeling, scientific research, high computations, oil and gas exploration [3] [4]. One of the most important application domains is the linear algebra [5] [6].

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

A. Overview of this paper

In Section II we give a brief introduction in GPU Computing model with CUDA. In Section III we present the mathematical background for a Walsh. In Section IV, we summarize the results and give some conclusions.

II. 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 [7].

A. CPU versus GPU

A simple way to understand the difference between a 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.

B. The CUDA programming model and hardware interface

Modern NVIDIA GPUs are powerful computing platform developed for general purpose computing using CUDA (Compute Unified Device Architecture) [8]. It allows programmers to interact directly with the GPU and run programs on them, thus effectively utilizing the advantages of parallelization. Depends of 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.

III. WALSH TRANSFORM

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. Any Boolean function *f* of *n* variables is uniquely determined by its truth table, denoted by $TT(f)$, which is a 2^n-dimensional vector whose coordinates are the function values of f after the lexicographic ordering of the inputs. We denote the Truth Table considered as a vector-column by [f].

Associate with the Boolean function f is the function $(-1)^f = 1-2f$ whose function values belong to the set $\{-1;1\}$. The corresponding vector that contains the function values of $(-1)^f$ is called polarity truth table (PTT) of the function f. In the following, the $2^n \times 1$ column matrix $[(-1)^f]$ represents the transpose of $PTT(f)$. We use PTT to obtain the Walsh spectrum [17] of a Boolean function.

Definition 1. The Walsh transform f^W of the Boolean function f is an integer valued function, defined by

$$f^W(a) = \sum_{x \in F_2^n} (-1)^{f(x) \oplus \langle a,x \rangle} \qquad (1)$$

The values of f^W are called Walsh coefficients of the Boolean function f. To understand the Walsh coefficients, we calculate

$$f(x) \oplus \langle a,x \rangle = f(x) \oplus f_a(x) \qquad (2)$$

where $f_a(x) = a_1x_1 \oplus a_2x_2 \ldots \oplus a_nx_n$.

Hence,

$$f^W(a) = \sum_{x \in F_2^n} (-1)^{f(x) \oplus f_a(x)} = \sum_{x \in F_2^n, f(x) = f_a(x)} 1 + \sum_{x \in F_2^n, f(x) \neq f_a(x)} (1-)$$

$$= \#\{x \in F_2^n, f(x) = f_a(x)\} - \#\{x \in F_2^n, f(x) \neq f_a(x)\}$$

$$= 2^n - 2d_H(f, f_a) \qquad (3)$$

For any Boolean function f and any vector $a \in F_2^n$ we have $-2^n \le f^W(a) \le 2^n$. The function $f_a(x) = \langle a,x \rangle$ and $\overline{f}_a(x) = \langle a,x \rangle \oplus 1$ have the maximum and minimum Walsh coefficients, namely $f_a^W = 2^n$ and $\overline{f}_a^W = -2^n$. Moreover, if $\overline{f}(x) = f(x) \oplus 1$ then

$$\overline{f}^W(a) = \sum_{x \in F_2^n} (-1)^{f(x) \oplus \langle a,x \rangle} = \sum_{x \in F_2^n} (-1)^{1 \oplus f(x) \oplus \langle a,x \rangle} =$$

$$= -\sum_{x \in F_2^n} (-1)^{f(x) \oplus \langle a,x \rangle} = -f^W(a) \qquad (4)$$

Consider the vectors of F_2^n ordered lexicographically. Then we can order the Walsh coefficients $f_a^W(a)$ and consider them as coordinates of a vector. This vector is called Walsh spectrum of the Boolean function and denoted by $[W_f]$ (consider as a column). Therefore,

$$[W_f]^T = (f^W(\bar{0}), f^W(\bar{1}), \ldots, f^W(\overline{2^n-1})) \qquad (5)$$

The Walsh spectrum of a Boolean function measures its distance to the linear and affine functions [17]. For better understanding of this connection, it is necessary to understand the $TT(f)$ of the linear Boolean functions.

Let $S_n = ([f_{\bar{0}}], [f_{\bar{1}}], \ldots, [f_{\overline{2^n-1}}])$ be the matrix whose columns are the $TT(f)$ of all linear Boolean functions ordered lexicographically.

$$S_n = ([\langle \bar{0}, x \rangle], [\langle \bar{1}, x \rangle], \ldots, [\langle \overline{2^n-1}, x \rangle]) =$$

$$= \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & \langle \bar{1}, \bar{1} \rangle & \cdots & \langle \overline{2^n-1}, \bar{1} \rangle \\ \vdots & \vdots & \vdots & \vdots \\ 0 & \langle \bar{1}, \overline{2^n-1} \rangle & \cdots & \langle \overline{2^n-1}, \overline{2^n-1} \rangle \end{pmatrix} \qquad (6)$$

Obviously, $S_n(i,j) = \langle \bar{i}, \bar{j} \rangle = \langle \bar{j}, \bar{i} \rangle \; S_n(i,j)$, for $0 \le i, j \le 2^n - 1$, which proves that this matrix is symmetric and therefore its rows are $TT(f)$ of all linear functions. Making a connection with Coding Theory, we see that actually this matrix consists of all code words in the $[2^n, n, 2^{n-1}]$ binary simplex code with added a zero coordinate in the beginning of each codeword.

If we take $\langle \bar{\upsilon}, x \rangle$ instead of x^υ for $\upsilon = 0,1,\ldots,2^n-1$ and replace the column $[x^\upsilon]$ by $[\langle \bar{\upsilon}, x \rangle]$, the matrix A_n goes to S_n. Where the matrix A_n is a binary matrix of size $2^n \times 2^n$ with determinant 1 [17]. Therefore, we can expect that after some transformations we can have a more effective algorithm.

$$f^W(a) = \sum_{x \in F_2^n} (-1)^{f(x) + \langle a,x \rangle} = \sum_{x \in F_2^n} (-1)^{\langle a,x \rangle} (-1)^{f(x)} \qquad (7)$$

According to equation (7) we obtain following equation for Walsh spectrum:

$$[W_f]=\begin{pmatrix} f^W(\bar{0}) \\ f^W(\bar{1}) \\ \vdots \\ f^W(\overline{2^n-1}) \end{pmatrix}=\begin{pmatrix} \sum_{x\in F_2^n}(-1)^{f(x)} \\ \sum_{x\in F_2^n}(-1)^{\langle \bar{1},x\rangle}(-1)^{f(x)} \\ \vdots \\ \sum_{x\in F_2^n}(-1)^{\langle \overline{2^n-1},x\rangle}(-1)^{f(x)} \end{pmatrix}=$$

$$=\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & (-1)^{\langle \bar{1},\bar{1}\rangle} & \cdots & (-1)^{\langle \overline{2^n-1},\bar{1}\rangle} \\ \vdots & \vdots & \vdots & \vdots \\ 0 & (-1)^{\langle \bar{1},\overline{2^n-1}\rangle} & \cdots & (-1)^{\langle \overline{2^n-1},\overline{2^n-1}\rangle} \end{pmatrix}\begin{pmatrix} (-1)^{f(\bar{0})} \\ (-1)^{f(\bar{1})} \\ \vdots \\ (-1)^{f(\overline{2^n-1})} \end{pmatrix}=$$

$$=H_n\left[(-1)^f\right] \quad (8)$$

Where $H_n=((-1)^{\langle \bar{i},\bar{j}\rangle})i,j$ is a $2^n\times 2^n$ matrix.

It is obvious that the matrix H_n can be obtained from S_n by replacing zeros with 1's and ones with -1's. Taking in mind that the first coordinate of $\bar{i}$ is 0 if $i<2^{n-1}$ and 1 otherwise, we have

$$\langle \bar{i},\bar{j}\rangle=\begin{cases} \langle \bar{i},\overline{j-2^{n-1}}\rangle, i<2^{n-1}, j\geq 2^{n-1} \\ \langle \overline{i-2^{n-1}},\overline{j-2^{n-1}}\rangle, i\geq 2^{n-1}, j<2^{n-1} \\ 1\oplus\langle \overline{i-2^{n-1}},\overline{j-2^{n-1}}\rangle, i\geq 2^{n-1}, j\geq 2^{n-1} \end{cases} \quad (9)$$

From equations (9) and (8), it follows that

$$H_n=\begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & H_{n-1} \end{pmatrix} \quad (10)$$

and the matrix H_n can be defined recursively as

$$H_0=(1),\ H_1=\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},\ H_n=\begin{pmatrix} H_{n-1} & H_{n-1} \\ H_{n-1} & H_{n-1} \end{pmatrix}=$$

$$=H_1\otimes H_{n-1}, n\geq 2 \quad (11)$$

The matrix H_n is a Hadamard matrix of Sylvester type called also Sylvester matrix or Walsh matrix [10]. Using that H_n is symmetric and $H_nH_n^T=2^nI_{2^n}$ we obtain that

$$H_n^{-1}=\frac{1}{2^n}H_n \quad (12).$$

A good survey on Hadamard matrices can be downloaded from Encyclopedia of Design Theory [10].

IV. EXPERIMENTAL EVALUATION

The experiments that we run are the following:

a) $[W_f]=H_n[(-1)^f]$ on CPU versus GPU

b) $[W_f]=H_n[(-1)^f]$ different version of GPU program implementation with global memory, shared memory

c) $[W_f]=H_n[(-1)^f]$ on GPU use cuBLAS library [7]

While the first one is used to check the validity of the performance of equation (8) implementation, the second one is often an underrepresented experiment that sheds light on the issue of the quality of the speed up reported and third we use special library for Basic Linear Algebra Subroutines.

The experiments were performed using an two different computer configuration the first was with Intel i3-3110M [11] with 2.4 GHz and 4 GB of RAM and NVIDIA GeForce GT 740M [12], cards with a total of 384 cores running at 0.9 GHz and a 28.8 GB/sec memory bandwidth and Intel Pentium®4 [13] with 3.0 GHz and 2 GB of RAM and NVIDIA GeForce GTS 450 [14], cards with a total of 192 cores running at 1.56 GHz and a 57.7 GB/sec memory bandwidth. The CUDA [8] kernel were developed using MS Visual Studio 2010.

We can use several metrics for evaluating the performance. Here, we are focused only on execution time for computing the Walsh spectrum. From the previous section we can say that Walsh spectrum can calculate like matrix vector multiplication, where matrix is a Hadamard matrix, and vector is array of ±1 and from our experiment is a random array.

A graphic representation of results from $[W_f]$ calculations are shown in figure 1 (1. CPU WT ($[W_f]$, i3-3110M), 2. CPU WT ($[W_f]$, Pentium®4). Results are as expected, we obtained faster calculation and better performance for $[W_f]$ with CPU i3-3110M (because of the resolution of the picture, the difference for small sizes is difficult to be noticed). We used Visual Studio 2010, Active solution configuration Release and Active solution platform Win32.

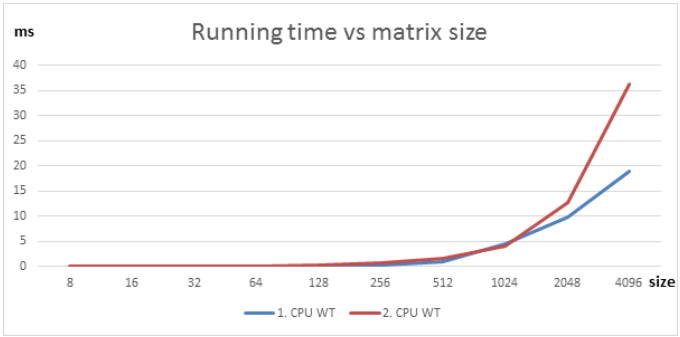

Fig. 1. Time for calculating $[W_f]$ on CPU for different sizes

Results from calculations of Walsh spectra using GPU are shown in Figure 2. In this figure are shown calculations with different GPU program implementation: 1. GPU MV (GT 740M), 2. GPU MV (GTS 450) (this is simple Matrix vector multiplication [6]), 1. GPU MV-SM (GT 740M), 2. GPU MV-SM (GTS 450) (here we have matrix vector multiplication with shared memory - shown as Algorithm 1), and 1. cuBLAS (GT 740M), 2. cuBLAS (GTS 450) (here we use GPU-accelerated version of the complete standard BLAS - Basic Linear Algebra Subroutines library [15][16]). In this case we also used Visual Studio 2010, Active solution configuration Release, Active solution platform Win32.

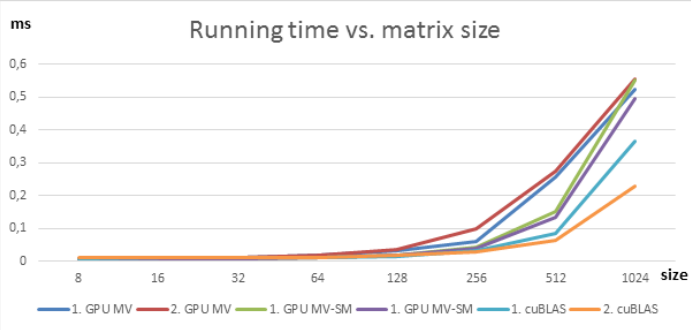

Fig. 2. Time for calculating $[W_i]$ on GPU for different sizes

Algorithm 1. CUDA matrix-vector product algorithm with shared memory

```
__global__ void  mv_kernel(float * A_G, float *
x_G, float * y_G, int n)
{
//@@ Insert code to implement matrix vector here
   __shared__ float ds_M[TILE_WIDTH][TILE_WIDTH];
   __shared__ float Xds[TILE_WIDTH];
   int bx = blockIdx.x, by = blockIdx.y,
   tx = threadIdx.x, ty = threadIdx.y,
   Row = by * TILE_WIDTH + ty,
   Col = bx * TILE_WIDTH + tx;
   float Pvalue = 0;

   for (int m = 0; m < (n-1)/TILE_WIDTH+1; ++m)
        {
        if (Row < n && m*TILE_WIDTH+tx < n)
        ds_M[ty][tx] = A_G[Row*n +m*TILE_WIDTH+tx];
        else
        ds_M[ty][tx] = 0;

        if(m*TILE_WIDTH + tx < n)
        Xds[tx] = x_G[m*TILE_WIDTH + tx];
         else
        Xds[tx] = 0;
   __syncthreads();
         for (int k = 0; k < TILE_WIDTH; ++k)
        Pvalue += ds_M[ty][k] * Xds[k];
   __syncthreads();
         }
         if (Row < n && Col < 1)
         y_G[Row*1+Col] = Pvalue;
}
```

The comparison between the fastest implementation from the experimented sequential program is shown in Figure 3: 1. CPU WT (i3-3110M), and run-time of the parallel algorithm 2. cuBLASGPU (GTS 450). For this comparison we use the same Visual Studio and configurations.

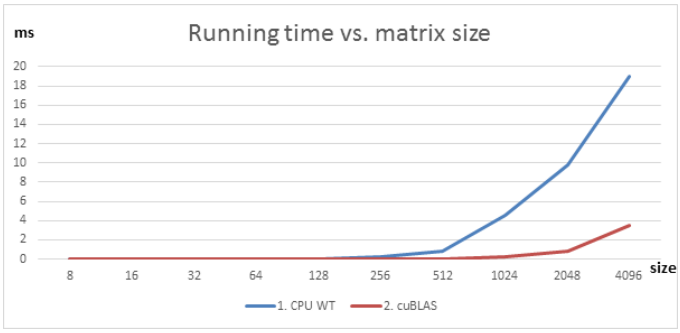

Fig. 3. Comparison of times between CPU with fast time and GPU with fast time

Speed up is defined by:

$$S_p = \frac{T_{(1,n)}}{T_{p(n)}} \qquad (13)$$

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

TABLE I. CPU VS. GPU SPEED UP FOR DIFFERENT SIZE

Size	1.CPU WT	2. cuBLAS	Speed up
8	0,000855	0,00947	/
16	0,001283	0,01053	/
32	0,005559	0,00989	/
64	0,017105	0,00995	1,7191
128	0,057730	0,01651	3,4967
256	0,211674	0,02966	7,1367
512	0,879621	0,06458	13,621
1024	4,496460	0,22896	19,639
2048	9,838760	0,86227	11,410
4096	18,98730	3,47539	5,4634

Table 1 shows the speed ups for different sizes of the matrix. An important conclusion is that the GPU processing makes sense only for large size problems. It is important the prices of buffer creation and transfers to be acceptable, so in this particular case when GPU is used, the experiment shows that the size of the input has to be ≥ 64 which means that for matrix vector multiplication the dimension of the vector (and the size of the matrix is ≥ 64). CPU is faster for small problems and can work faster than couple threads, which is the reason for this limitation. For larger sizes more threads are

used and therefore the computation is faster than in the case of sequential programming. However, there are boundaries which depend on several things (the problem, the algorithm, GPUs, the libraries, the model, etc.).

In our experiment with CUDA C program without using library cuBLAS we have fixed the number of threads per block (32 threads per block) and different number of blocks. The usage of different configurations of grid will be in the focus of our next research and other similar experiments.

Here, in our experiment we used sequential algorithm for generating of Hadamard matrix. We can implement parallel algorithm for generation of Hadamard matrix. That also is part of our planed future research.

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 algorithms proposed for matrix-vector product (Walsh transform), in CUDA, still can be improved. By choosing proper matrix sizes and appropriate methods one can get increased efficiency and improved performances. This is a part of our planed 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] A. Mahajan, M.Chanana, D. Sharma, K.Sachdeva "General purpose computing on Graphical Processing Unit: extending parallel processing", International Journal of Advanced Research in IT and Engineering, ISSN: 2278-6244, Vol. 2 ,No. 11, November 2013

[4] J. Zhong, B. He "Kernelet: High-Throughput GPU Kernel Executions with Dynamic Slicing and Scheduling", IEEE Transactions on Parallel and Distributed Systems, Volume 25 Issue 6, June 2014

[5] 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.

[6] 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.

[7] NVIDIA, OpenCL Programming Guide for the CUDA Architecture, 2011.

[8] CUDA homepage: http://www.nvidia.com/object/cuda_home_new.html.

[9] CUDA Programming Guide: http://docs.nvidia.com/cuda/#axzz3HNpg3SNW.

[10] P. J. Cameron, Hadamard matrices, Encyclopaedia of Design Theory. 2006. Available from: http://designtheory.org/library/encyc/.

[11] i3-3110M specification [Online]. Available: http://ark.intel.com/products/65700/Intel-Core-i3-3110M-Processor-3M-Cache-2_40-GHz.

[12] NVIDIA GeForce GT 740M specification [Online]. Available: http://www.geforce.com/hardware/notebook-gpus/geforce-gt-740m/specifications.

[13] Pentium®4 specification [Online]. Available: http://ark.intel.com/products/27508/Intel-Pentium-4-Processor-supporting-HT-Technology-3_00E-GHz-1M-Cache-800-MHz-FSB.

[14] NVIDIA GeForce GTS 450 specification [Online]. Available: http://www.geforce.com/hardware/desktop-gpus/geforce-gts-450/specifications.

[15] A. Chrzeszczyk, Kielce, J. Chrzeszczyk, "Matrix computations on the GPU CUBLAS and MAGMA by example", book, August, 2013.

[16] cuBLAS library [Online]. Available: https://developer.nvidia.com/cublas.

[17] Carlet C. (2010), 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.