FACTA UNIVERSITATIS
Series: **Automatic Control and Robotics** Vol. 10, N^o 1, 2011, pp. 59 - 70

COMPUTATION OF DYADIC CONVOLUTION ON GPU FOR EFFICIENT MODELING OF DYADIC LTI SYSTEMS [*]

UDC 62-231.2 519.216 004.272.2:004.42 004.438GPU

Dušan B. Gajić, Radomir S. Stanković

Department of Computer Science, Faculty of Electronic Engineering,
University of Niš, Aleksandra Medvedeva 14, 18 000 Niš, Serbia
E-mail: dule.gajic@gmail.com, radomir.stankovic@gmail.com

Abstract. *Convolution systems are an important class of systems often used in many practical applications in different areas of science and engineering. In the case of systems with binary encoded input and output signals, finite dyadic groups are usually assumed as the underlying algebraic structure. Modeling and handling of such systems requires frequent computation of the dyadic convolution. This paper discusses a method for efficient computation of the dyadic convolution on Graphics Processing Units (GPUs) with the related algorithm implemented in Open Computing Language (OpenCL). We consider several important issues concerning the efficient mapping of the algorithm to the GPU architecture. Performance of the proposed implementation is compared with the referent C/C++ implementation processed on the Central Processing Unit (CPU). Experimental results confirm that significant speedups are achieved by the application of the proposed GPU calculation method.*

Key words: *Linear translation invariant (LTI) systems, Dyadic convolution, Fast Walsh transform (FWT), GPU parallel programming, OpenCL*

1. INTRODUCTION

Linearity, causality, and translation invariance are features often assumed for mathematical models of many systems useful in practice. These features directly lead to convolution systems, since any causal linear translation invariant (LTI) system can be represented by a convolution system. At the same time, any convolution system is linear, causal, and translation invariant.

Depending on the class of signals to be processed, systems can be time-invariant, shift-invariant, or rotation-invariant. For these systems, groups of real numbers and complex numbers, respectively, are usually assumed as domains for the definition of input and

Received July 06, 2011

[*] **Acknowledgement**. *The research reported in this paper is partly supported by the Ministry of Education and Science of the Republic of Serbia, project OI 174026 (2011-2014).*

output signals. There are, however, areas where systems on groups different from these groups are more suitable as mathematical models.

In particular, dyadic systems, i.e., systems modeled on dyadic groups (defined bellow), are often met in communications, computing, and cryptography. Dyadic systems were introduced by Weisner in [36], and their theory was further contributed by Harmuth [16, 17], based on the background mathematical theory by Polyak and Shreider [29]. The dyadic systems have been intensively studied by Pichler in a series of publications [22, 23, 24, 25, 26, 27, 28], and by several other authors [5, 7]. Recently, logic networks, which can be viewed as facets of digital systems modeled on dyadic groups, were studied from the system theory point of view [34].

In study and practical applications of dyadic systems, computation of the dyadic convolution is frequently required. Efficiency of these computations often determines the applicability of such mathematical models in engineering practice.

With that motivation, in this paper we present an efficient technique for an accelerated calculation of the dyadic convolution, through a parallel implementation of the fast algorithm derived from the convolution theorem and processed on the Graphics Processing Unit (GPU). Due to the convolution theorem, computation of the dyadic convolution converts into performing two direct and an inverse Walsh transform, which can be done by the corresponding FFT-like algorithms, i.e., the Fast Walsh Transform – FWT [4, 8, 18]. The proposed implementation is developed using the Open Computing Language (OpenCL) and processed on a GPU. Experimental results and comparisons with the classical implementation confirm that the proposed method leads to a significant computational speedup.

This paper is based on a shorter preliminary version presented at the conference ICEST 2011 [11], and it is organized as follows. After an introduction to the necessary background theory in Section 2, Section 3 presents a brief review of related work. Section 4 is devoted to the description of the mapping of the fast algorithm for computing the dyadic convolution to the GPU architecture and the design of the corresponding OpenCL implementation. In Section 5, we present the experimental environment used to evaluate the method, and report experimental results. The closing Section 6 offers some conclusions drawn from the presented research.

2. BACKGROUND THEORY

2.1. Dyadic Convolution

Convolution is a mathematical operation that expresses relationships between values of two signals (modeled by functions f and g) in points at a fixed distance. The convolution $C = f * g$is a function that resembles either the function f or the function g, modified by the other.

When the finite dyadic group is used as the algebraic structure on which the convolution is defined, we use the term dyadic convolution [4, 18, 30, 32].

The finite dyadic group of order n is defined as $C_2^n = \times_{i-1}^{n} C_2$, where $C_2 = (\{0,1\}, \oplus)$, and $\oplus$ stands for the addition modulo 2, and $\times$ is the direct (Cartesian) product.

For two functions $f, g : C_2^n \to \mathbb{R}$, where $\mathbb{R}$ is the field of rational numbers, the dyadic convolution is defined as

$$C_{f*g}(\tau) = \sum_{x=0}^{2^n-1} f(x)\cdot g(x \oplus \tau), \quad \tau = 0,1,...,2^{n-1}. \tag{1}$$

In binary notation, $x = (x_1, x_2,\ldots, x_n)$, and $\tau = (\tau_1, \tau_2,\ldots,\tau_n)$, where x_i, $\tau_i \in \{0,1\}$.

A direct calculation of the dyadic convolution from (1) has the exponential complexity in the number of inputs n and is unfeasible in practice for large signals. Therefore, algorithms for the fast computation of convolution are derived by using the convolution theorem [18] on the corresponding algebraic structure.

2.2. Walsh Transform

The Walsh transform [18] is defined by the Walsh matrix

$$\mathbf{W}(n) = \bigotimes_{i=1}^{n} \mathbf{W}(1), \tag{2}$$

where $\otimes$ denotes the Kronecker product and

$$\mathbf{W}(1) = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{3}$$

is the basic Walsh matrix. From (2), $\mathbf{W}(n)$ is generated by the Kronecker product and, therefore, the Walsh functions (rows of $\mathbf{W}(n)$) appear in the Hadamard order, see for instance [17].

Since $\mathbf{W}(n)$ is a self-inverse matrix up to the scalar 2^{-n}, the inverse Walsh transform is defined as

$$\mathbf{W}^{-1}(n) = 2^{-n}\mathbf{W}(n). \tag{4}$$

It follows that both the direct and the inverse Walsh transforms can be computed by using the same algorithm.

The Walsh spectrum $\mathbf{S}_{f,h} = [S_{f,h}(0), S_{f,h}(1), \ldots, S_{f,h}(2^n\text{-}1)]^T$ of a function f: $C_2^n \to \mathbb{R}$, specified by the function vector $\mathbf{F} = [f(0), f(1), ..., f(2^n\text{-}1)]^T$, is defined as

$$\mathbf{S}_{f,h} = \mathbf{W}(n)\mathbf{F}. \tag{5}$$

The spectral coefficients appear in Hadamard ordering, which is indicated by the index h in $\mathbf{S}_{f,h}$. The function f is reconstructed from the Walsh spectrum as

$$\mathbf{F} = 2^{-n}\mathbf{W}^{-1}(n)\mathbf{S}_{f,h}, \tag{6}$$

and (5) and (6) form the Walsh transform pair.

Example 1. If $f : C_2^2 \to \mathbb{R}$ is specified by the function vector $\mathbf{F} = [1, 0, 1, 1]^T$, then the corresponding Walsh spectrum $\mathbf{S}_{f,h} = [S_{f,h}(0), S_{f,h}(1), S_{f,h}(2), S_{f,h}(3)]^T$ can be calculated directly by using its definition (5)

$$\mathbf{S}_{f,h}=\left[\bigotimes_{i=1}^{2}\mathbf{W}(1)\right]\mathbf{F}=\mathbf{W}(2)\mathbf{F}=\begin{bmatrix}1 & 1 & 1 & 1\\ 1 & -1 & 1 & -1\\ 1 & 1 & -1 & -1\\ 1 & -1 & -1 & 1\end{bmatrix}\cdot\begin{bmatrix}1\\0\\1\\1\end{bmatrix}=\begin{bmatrix}3\\1\\-1\\1\end{bmatrix}. \tag{7}$$

2.3. Fast Walsh Transform

The computation of the Walsh transform based on its definition (5) is inefficient, since it expresses the $O(N^2)$ time complexity, where $N = 2^n$ is the size of the input vector. Fortunately, a more efficient algorithm, the Fast Walsh transform (FWT) with the time complexity of $O(N\log N)$ exist, see for instance [4, 18].
The Fast Walsh transform can be defined by using the following factorization [18]

$$\mathbf{W}(n)=\prod_{i=1}^{n}\mathbf{C}_{w_i}(n)_i\ , \tag{8}$$

where

$$\mathbf{C}_{w_i}(n)=\bigotimes_{j=1}^{n}\mathbf{C}_{w_i}^{j}(1),\quad \mathbf{C}_{w_i}^{j}(1)=\begin{cases}\mathbf{W}(1), i=j,\\ \mathbf{I}(1), i\neq j.\end{cases} \tag{9}$$

The matrix $\mathbf{C}_{w_i}(n)$ defines the partial Walsh transform and corresponds to the i-th step of the FWT. This particular factorization leads to an FFT-like algorithm of the Cooley-Tukey type [12, 18].

Example 2. For the function $f:\ C_2^2\to\mathbb{R}$ in Example 1, the Walsh spectrum can be calculated with 4 additions and 4 subtractions by applying the FWT algorithm, instead of 16 multiplications and 12 additions in the direct computation using the definition. This algorithm is derived from the following factorization of the matrix $\mathbf{W}(2)$

$$\mathbf{S}_{f,h}=\mathbf{W}(2)\mathbf{F}=(\mathbf{C}_1\mathbf{C}_2)\mathbf{F}, \tag{10}$$

where

$$\mathbf{C}_1=\mathbf{W}(1)\otimes\mathbf{I}(1)=\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\otimes\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}=\begin{bmatrix}1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & -1 & 0\\ 0 & 1 & 0 & -1\end{bmatrix}, \tag{11}$$

$$\mathbf{C}_2=\mathbf{I}(1)\otimes\mathbf{W}(1)=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\otimes\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}=\begin{bmatrix}1 & 1 & 0 & 0\\ 1 & -1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 0 & 0 & 1 & -1\end{bmatrix}. \tag{12}$$

Computations in $\mathbf{C}_1$ and $\mathbf{C}_2$ can be performed by the algorithms whose flow-graphs are shown in Fig. 1 (*a*) and (*b*). In this figure, the solid and dashed lines denote addition and subtraction, respectively.

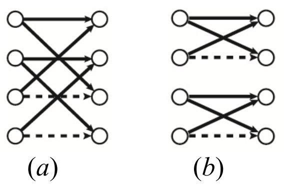
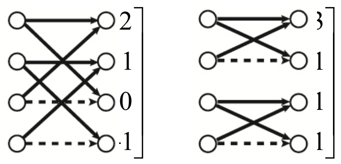


Fig. 1 Flow-graphs of the FWT for $n = 2$,
(*a*) First step derived from $\mathbf{C}_1$, (b) Second step derived from $\mathbf{C}_2$.

Therefore, computing the Walsh spectrum goes as follows

$$\mathbf{F} = \begin{bmatrix} f(0) \\ f(1) \\ f(2) \\ f(3) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$



$$= \begin{bmatrix} S_{f,h}(0) \\ S_{f,h}(1) \\ S_{f,h}(2) \\ S_{f,h}(3) \end{bmatrix} = \mathbf{S}_{f,h}. \qquad (13)$$

2.4. Convolution Theorem

In classical Fourier analysis, the convolution theorem states that the Fourier transform of the convolution function $C = f * g$ is the componentwise product of the Fourier transforms of f and g, thus $S_{f*g} = S_f \cdot S_g$, see for instance [4, 8, 18]. In other words, a rather complex operation in the original domain converts into a simple operation in the spectral domain.

For functions on the finite dyadic group C_2^n, the computation of the dyadic convolution through the application of the convolution theorem is done as follows

$$\mathbf{C}_{f*g} = 2^{-n}\mathbf{W}^{-1}(n)((\mathbf{W}(n)\mathbf{F})(\mathbf{W}(n)\mathbf{G})). \qquad (14)$$

Therefore, an efficient algorithm for the computation of the dyadic convolution can be developed in terms of the FWT, as illustrated in Fig. 2 for $n = 2$, $N = 4$.

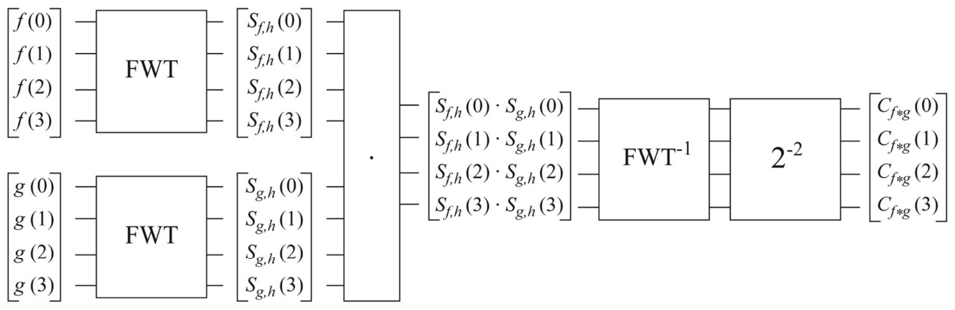


Fig. 2 Computation of the dyadic convolution through the application of the convolution theorem for $n = 2$, $N = 4$

Example 3. For two functions f, g : $C_2^2 \rightarrow \mathbb{R}$, given by their respective function vectors $\mathbf{F} = [1, 0, 1, 1]^T$ and $\mathbf{G} = [0, 1, 0, 1]^T$, performing the FWT on each of them produces their Walsh spectra $\mathbf{S}_{f,h} = [3, 1, -1, 1]^T$ and $\mathbf{S}_{g,h} = [2, -2, 0, 0]^T$, respectively. The result of the componentwise multiplication of these two spectra is $\mathbf{S}_{f.g,h} = [6, -2, 0, 0]^T$. After the inverse FWT is performed on $\mathbf{S}_{f.g,h}$, followed by multiplication with the scaling factor 2^{-2}, we get the dyadic convolution coefficients $\mathbf{C}_{f*g} = [1, 2, 1, 2]^T$.

3. Background Work

The fast algorithm for the dyadic convolution is based on the application of the Walsh transform which is the Fourier transform on finite dyadic groups. The implementation of various Fast Fourier Transforms (FFTs) on different technological platforms is a widely considered topic, see for instance [4, 8, 11, 15] and references therein. The calculation of dyadic convolution on classical Central Processing Units (CPUs) through the application of the convolution theorem, both on vectors and decision diagrams, is presented in [30]. Reference [4] presents an application of the dyadic convolution for the fast multiplication of hyper-complex numbers.

In recent years, the technique of performing General Purpose computations on the GPU (GPGPU) has proven to be a suitable approach in solving many computationally-intensive tasks [2, 3, 12, 15, 19]. In particular, the GPU-accelerated calculation of FFT algorithms using CUDA is described in [15, 20]. Reference [12] presents an OpenCL implementation of the FWT that uses the GPU hardware and leads to significant speedups over traditional CPU processing.

However, in our best knowledge, there are no publications except the preliminary version of this work [11] that discuss neither CUDA nor OpenCL GPU implementations of the fast algorithm for the computation of dyadic convolution based on the convolution theorem. This fact, together with the intended application of the dyadic convolution for the modeling of dyadic LTI systems, was the motivation for the research reported in this paper.

4. Mapping of the Algorithm and Implementation Details

The convolution theorem, expressed in (14), leads to the following fast algorithm for the calculation of the dyadic convolution, illustrated by the example in Fig. 2:

Step 1. Perform the FWT on f and g and compute their Walsh spectra $\mathbf{S}_f$ and $\mathbf{S}_g$.

Step 2. Perform the componentwise multiplication of $\mathbf{S}_f$ and $\mathbf{S}_g$.

Step 3. Perform the inverse FWT over $\mathbf{S}_f \cdot \mathbf{S}_g$ to obtain $\mathbf{C}_{f*g}$.

```
Algorithm 1 FAST CALCULATION OF DYADIC CONVOLUTION $C = f * g$
1  Allocate buffers buffer1 and buffer2 in the global memory of the GPU device.
2  Transfer input vectors $f$ and $g$ from the host CPU memory to GPU buffers buffer1 and
   buffer2, respectively.
3  Perform the Walsh transform on vectors stored in buffer1 and buffer2 using the following
   in-place OpenCL implementation of the Cooley-Tukey algorithm for the FWT:
   a. For each step of the FWT, from $step \leftarrow 0$ to $step \leftarrow (\log_2 N) - 1$, call the OpenCL kernel
      for the FWT with input parameters being the appropriate buffer in the GPU's global
      memory and the value of the current step $2^{step}$. The kernel is executed by $N/2$ threads in
      parallel on the GPU. Each thread reads two elements, determined by (15) and (16),
      from the buffer, performs the operations defined by the Walsh matrix $\mathbf{W}(1)$ and stores
      back the results in the same locations.
4  After computing the FWT of both vectors, execute the OpenCL kernel for the componen-
   twise multiplication of the two Walsh spectra with $N$ threads executed in parallel. The re-
   sulting vector is stored in buffer1.
5  Perform the inverse FWT on buffer1 using the same kernel as for the FWT.
6  Scale the contents of buffer1 with the factor $2^{-n}$ using the OpenCL kernel with $N$ threads
   executed in parallel.
7  Transfer the contents of the GPU buffer buffer1, which holds the resulting dyadic convolu-
   tion coefficients, back to the host CPU memory.
```

Since both multiplication and componentwise multiplication of vectors can be done fast on modern CPUs and GPUs, the key issue in developing an efficient implementation of the above algorithm is performing the FWT and the inverse FWT, required in steps 3 and 5 of Algorithm 1, in an efficient manner. Therefore, we developed a kernel containing an OpenCL in-place implementation of the Cooley-Tukey algorithm for the FWT [12, 18]. As in all FFT-like algorithms, steps of the algorithm are executed sequentially and parallelism is used only within the steps. In each step, $N/2$ threads are executed in parallel. This large number of threads permits to overcome the data access latency to the GPU global memory [2, 3].

Each thread reads two elements from the GPU buffer with indices *op1* and *op2* calculated as

$$op1 \leftarrow thread\_id \bmod step + 2 \times step \times (thread\_id/step), \tag{15}$$

$$op2 \leftarrow op1 + step. \tag{16}$$

Parameters *thread_id* and *step* are the global identifier of the thread and the identifier of the current step of the algorithm, respectively. All threads execute the elementary butterfly operation defined by the basic Walsh transform matrix $\mathbf{W}(1)$ and store the results back in the same locations in the GPU memory, as in other implementations of the in-place FWT algorithms.

The componentwise multiplication of vectors is also performed by the corresponding OpenCL kernel which is executed by N threads in parallel, with each thread multiplying the two corresponding elements of the input vectors.

After the multiplication, the inverse FWT is performed with the same kernel that is used for the direct transform, followed by scaling with 2^{-n}. The scaling is also performed in parallel through the execution of the corresponding OpenCL kernel.

Before executing any of the kernels, both input vectors are transferred from the main memory to the GPU global memory. After the calculations, the resulting convolution coefficients are copied back to the host. These memory operations take a significant share of the total GPU running times as reported in Section 5.

5. Experimental Environment and Results

The test platform used to perform the experiments is an HP Pavilion dv7-4060us notebook computer (see Table 1). The OpenCL kernels are developed using MS Visual Studio 2010 Ultimate and ATI APP SDK 2.3 [1]. ATI Stream Profiler 2.1 is used for GPU performance analysis, in accordance with instructions provided in [2]. The source code is compiled for the x64 platform. The referent C/C++ implementation is compiled with the maximum level of performance-oriented compiler optimizations.

As in all FFT implementations over vectors, the resulting performance is independent of the function values. Therefore, we perform the experiments using randomly generated binary vectors. All values in Table 2 are an average for 10 program executions.

We show the GPU calculation time and, also, the time for transfer of data to/from the GPU which offers a more complete perspective to the reader. To estimate the performance of the proposed method, we performed a comparative analysis with respect to the classical C/C++ implementation of the same algorithm processed on the CPU.

Table 1. Specification of the experimental platform.

CPU	AMD Phenom II N830 triple-core (2.1 GHz)
RAM	4GB DDR3
OS	Windows 7 (64-bit)
GPU	ATI Mobility Radeon 5650
- engine speed	650 MHz
- global memory	1 GB DDR3 800 MHz
- compute units	5
- processing elements	400
- price	~ 100$

The results of the experiments are presented in Table 2 and Fig. 3. The OpenCL implementation processed on an inexpensive commodity GPU (Table 1) clearly outperforms the referent CPU implementation, by a factor of up to $5.5\times$ when only calculation times are compared, and by a factor of up to $4.5\times$ when total times, including memory transfers to/from GPU, are taken into account. The execution of the same kernels on a more powerful GPU (with more stream cores and a larger memory bandwidth) would directly lead to much larger speedups, which would not be the case if a more powerful CPU was used for the referent C/C++ implementation [3, 12].

Table 2. Running times for the referent CPU and the proposed GPU implementation

n = log*₂*N	**Time [ms]**		
	CPU	**GPU**	
		Computation time	Memory time
4	0	0	0
8	0	0	1
10	0	1	1
16	15	3	2
18	53	6	3
20	108	24	9
23	1201	220	45
24	2512	469	86
25	5002	998	147

n – Number of input variables
CPU – Referent C/C++ implementation on the CPU
GPU – Proposed OpenCL C implementation on the GPU

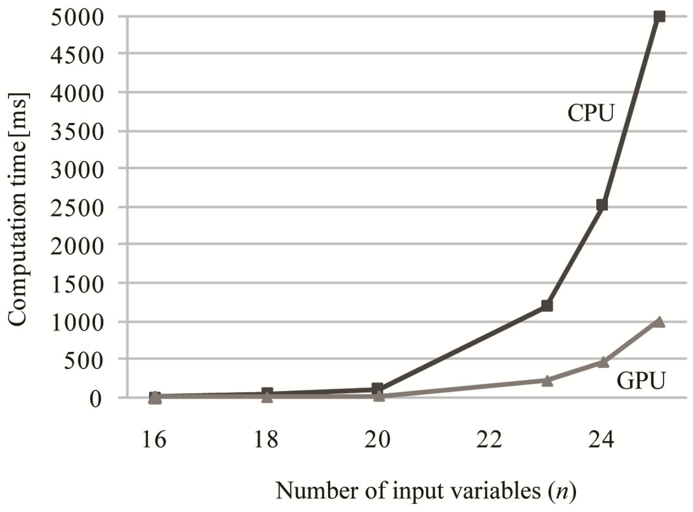


Fig. 3. Computation times for the referent CPU and the proposed GPU implementation

6. CONCLUSIONS

The dyadic systems are a particular class of linear translation invariant systems on finite groups modeled in terms of the dyadic convolution. Efficient computation of the dyadic convolution is therefore essential in practical applications of such systems.

The OpenCL parallel algorithm implementation for computing the dyadic convolution, which is processed on the GPU, provides considerable speedups, as documented by experiments over randomly generated binary functions and a comparison with the referent C/C++ CPU implementation of the algorithm.

The same OpenCL implementation of the dyadic convolution can also be used for the fast calculation of the dyadic autocorrelation. The proposed method could, therefore, extend the area of applications of these operations to problems where algorithm running time is an essential and, often, limiting factor.

REFERENCES

1. AMD Accelerated Parallel Processing SDK, http://developer.amd.com/gpu/amdappsdk, AMD Inc., website last visited on 10/06/2011.
2. ATI Stream Computing OpenCL Programming Guide, AMD Inc., 2010.
3. T. M. Aamodt, "Architecting graphics processors for non-graphics compute acceleration", in Proc. of the 2009 IEEE Pacific Rim Conf. on Communications, Computers and Signal Processing, Special Session on Computer Architecture, Victoria, BC, Canada, August 23-26, 2009.
4. J. Arndt, Matters Computational: Ideas, Algorithms, Source Code, Springer, 2010.
5. S. Cohn-Sfetcu, "On the theory of linear dyadic invariant systems", in Proc. Symp. Theory and Applic. Walsh and other Non-sinus. Functions, Hatfield, England, 1975.
6. S. Cohn-Sfetcu, Topics on generalized convolution and Fourier transform: Theory and applications in digital signal processing and system theory, Ph.D. dissertation, McMaster University, Hamilton, Ontario, Canada, 1976.
7. S. Cohn-Sfetcu and S. T. Nichols, "On the identification of linear dyadic invariant systems", IEEE Transactions on Electromagnetic Compatibility, Vol. EMC-17, No. 2, May 1975, 111-117.
8. J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series", Mathematics of Computation, No. 90, 1965, 297-301.
9. Y. Endow, "Walsh harmonizable processes in linear system theory", Cybernetics and Systems, 1996, 489-512.
10. Y. Endow and R. S. Stanković, "Gibbs derivatives in linear system theory", Cybernetic and Systems, Vol. 26, 1995, 665-680.
11. D. B. Gajić and R. S. Stanković, "Calculation of dyadic convolution using graphics processing units and OpenCL", paper accepted for presentation at the ICEST 2011 Conference, Niš, Serbia, June 29 - July 1, 2011.
12. D. B. Gajić and R. S. Stanković, "Computing fast spectral transforms on graphics processing units using OpenCL", in Proc. of the Reed-Muller 2011 Workshop, Tuusula, Finland, May 25-26, 2011, 27-36.
13. T. H. Frank, "Implementation of dyadic correlation", IEEE Trans. Electromagnetic Compatibility, Vol. EMC-13, No. 3, 1971, 111 – 117.
14. J. E. Gibbs, J. E. Marshall, and F. R. Pichler, "Electrical measurements in the light of system theory", in IEEE Conference on the Use of Computers in Measurement, University of York, Sept. 24-27, 1973, ii+5.
15. N. K. Govindaraju et al., "High performance discrete Fourier transforms on graphics processors," in Proc. of the 2008 ACM/IEEE Conf. on Supercomputing, IEEE Press, Austin, Texas, USA, November 15-21, 2008.
16. H. F. Harmuth, "Die Orthogonalteilung als Verallgemeinerung der Zeit-und Frequenzteilung", Archiv der Elektrischen Übertragung, vol. 18, 1964, 43-50.
17. H. F. Harmuth, Sequency Theory - Foundations and Applications, Academic Press, New York 1977; Russian edition: MIR, Moscow 1980; reprinted in PR China - Chinese edition: Publishing House of the People's Post and Telecommunications, Beijing 1980.
18. M. G. Karpovsky, R. S. Stanković, and J. T. Astola, Spectral Logic and Its Applications for the Design of Digital Devices, Wiley-Interscience, 2008.
19. NVidia CUDA: Compute Unified Device Architecture, http://developer.nvidia.com/object/ gpucomputing.html, NVIDIA Corp., website last visited on 21/06/2011.
20. NVidia CUDA CUFFT Library, NVIDIA Corp., 2007.

21. J. Pearl, “Optimal dyadic models of time-invariant systems”, IEEE Trans. Comput., Vol. C-24, 1975, 598-603.
22. F. Pichler, “Walsh functions and linear system theory”, in Proc. Applic. Walsh Functions, Washington, D.C., 1970, 175-182.
23. F. Pichler, J. E. Marshall, and J. E. Gibbs, “A system-theory approach to electrical measurements”, in Colloquium on the Theory and Applications of Walsh Functions, The Hatfield Polytechnic, Hatfield, Hertfordshire, June 28-29, 1973.
24. F. Pichler, “Fast linear methods for image filtering”, in Applications of Information and Control Systems, D. G. Lainiotis and N. S. Tzanues, (Eds.), D. Reidel Publishing Company, 1980, 3-11.
25. F. Pichler, “Experiments with 1-D and 2-D signals using Gibbs derivatives”, in Theory and Applications of Gibbs Derivatives, P. L. Butzer and R. S. Stanković, (Eds.), Belgrade, Serbia: Matematički institut, 1990, 181-196.
26. F. Pichler, “Some historical remarks on the theory of Walsh functions and their application in information engineering”, in Theory and Applications of Gibbs Derivatives, P. L. Butzer and R. S. Stanković, (Eds.), Belgrade, Serbia: Matematički institut, 1990.
27. F. Pichler, “Realizations of Priggogine's λ-transform by dyadic convolution”, in Cybernetics and Systems Research, R. Trappl and W. Horn, (Eds.), Austrian Society for Cybernetic Studies, 1992, ISBN 385206127X.
28. F. Pichler, “On state-space description of linear dyadic-invariant systems”, in IEEE Transactions on Electromagnetic Compatibility, Vol. EMC-13, No. 3, 1971, 166 – 170.
29. B. T. Polyak and Y. A. Shreider, “Applications of Walsh functions in approximate calculations”, in *Voprosy Teorii Matematichskikh Mashin*, 1962, 74-190.
30. M. M. Radmanović, “Calculation of dyadic convolution through binary decision diagrams”, Facta Universitatis: Automatic Control and Robotics, Vol. 8, No. 1, 2009, 89 – 97.
31. R. S. Stanković, M. Bhattacharaya, and J. T. Astola, “Calculation of dyadic autocorrelation through decision diagrams”, in Proc. of the European Conference on Circuit Theory and Design, August 2001, 337-340.
32. R. S. Stanković, M. S. Stanković, and C. Moraga, “Remarks on systems and differential operators on groups”, Facta Universitatis: Electronics and Energetics, Vol. 18, No. 3, 2005, 531-545.
33. E. A. Trachtenberg and M. G. Karpovsky, “Optimal varying dyadic structure models of time invariant systems”, in Proc. 1988 IEEE Int. Symp. Circuits and Systems, Espoo, Finland, June 1988, 1111-1115.
34. M. A. Thornton, “Spectral analysis of digital logic circuit netlists”, in Proc. of the International Conference on Computer Aided Systems Theory (EUROCAST), February 6-11, 2011, 414-415.
35. The OpenCL Specification 1.1, Khronos OpenCL Working Group, 2010.
36. F. E. Weisner, “Walsh function analysis of instantaneous nonlinear stochastic problems”, Ph.D. dissertation, Polytechnic Institute of Brooklyn, Brooklyn, June 1964.

IZRAČUNAVANJE DIJADIČKE KONVOLUCIJE NA GRAFIČKIM PROCESNIM JEDINICAMA ZA EFIKASNO MODELOVANJE DIJADIČKIH LTI SISTEMA

Dušan B. Gajić, Radomir S. Stanković

Konvolucioni sistemi predstavljaju važnu klasu sistema koji se često koriste u mnogim praktičnim primenama u različitim oblastima nauke i inženjerstva. U slučaju kada su ulazni i izlazni signali binarno kodirani, obično se kao algebarska struktura na kojoj su definisani odgovarajući sistemi uzimaju konačne dijadičke grupe. Modelovanje i rad sa ovakvim sistemima zahtevaju veoma česta izračunavanja dijadičke konovlucije. U ovom radu predstavljen je metod za efikasno izračunavanje dijadičke konvolucije na grafičkim procesnim jedinicama (Graphics Processing Units - GPUs) pri čemu je odgovarajući algoritam implementiran primenom programskog jezika Open Computing Language (OpenCL). Razmotreno je nekoliko veoma značajnih pitanja koja se tiču efikasnog mapiranja algoritma na arhitekturu grafičkih procesnih jedinica.

Performanse predložene implementacije upoređene su sa referentnom C/C++ implementacijom koja se izvršava na centralnoj procesnoj jedinici (Central Processing Unit - CPU). Eksperimentalni rezultati potvrđuju da primena predloženog metoda za izračunavanje dijadičke konvolucije na GPU donosi značajna ubrzanja u odnosu na referentnu CPU implementaciju.

Ključne reči: *Linearni translaciono nezavisni sistemi, Dijadička konvolucija, Brza Walsh-ova transformacija, GPU paralelno programiranje, OpenCL*