OpenCL application to accelerate the lossless image compression algorithm based on cascading fragmentation and the pixels sequence ordering

A. Khokhlachev[a], V. Smirnov[a], A. Korobeynikov[a]

[a] *Kalashnikov Izhevsk State Technical University, 426069, 7 Studencheskaya street, Izhevsk, Russia*

Abstract

The previous papers of authors describes approach to building the ordered sequence of image pixels at lossless compression, which comprises methods of cascading fragmentation, and code book. For fragment sized 6*6 the code book contained 22144 various bypasses, every must be estimated for the cost of coding. The search of optimal bypass is an exhaustive search. Present paper describes increasing the image lossless compression speed by using parallel computing based on OpenCL. Algorithm functions with great runtime were changed, in order to transfer calculations to OpenCL using GPU/CPU. Gained in experiments acceleration degree for different algorithm functions ranging in 3..32.

Keywords: lossless image compression; cascading fragmentation; pixels sequence ordering; optimal bypass; code book; acceleration calculations; parallel computing; open computing language (OpenCL); graphics processing unit (GPU); central processing unit (CPU); integral-valued Haar wavelets; interchannel decorrelation

1. Introduction

At the moment there are a large number of compression algorithms particular data classes, and universal compression algorithms. This work will address the lossless image compression algorithm based on cascading fragmentation and optimization of bypass image developed by the authors and described in [1...3]. In processing test images [7], the algorithm shows the compression ratio is on average equal to 1.54. Let us consider test results by groups of images: 1) in group «2.1.*.tiff» by 1.426 2) in group «2.2.*.tiff» by 1.547 3) in group «4.1.*.tiff» by 1.622 4) in group «4.2.*.tiff» by 1.522, which is comparable with the analogues [5]. In addition, the algorithm has some additional advantages [5].

To achieve a high compression ratio it is necessary to use a number of demanding algorithm functions, so that the processing program of the image takes a long time. To date, there is the possibility of parallel computing. The aim of this work is the use of OpenCL for faster image compression algorithm without losses. To achieve this goal it is necessary: to analyze duration of program execution; find the algorithm functions with time-consuming calculations; to consider transfer of these functions on the GPU. In the compression algorithm image processing based on working with particular fragments, so in the general case a calculation for fragments can execute in parallel. Also it is possible to perform in parallel the preprocessing functions for image fragments.

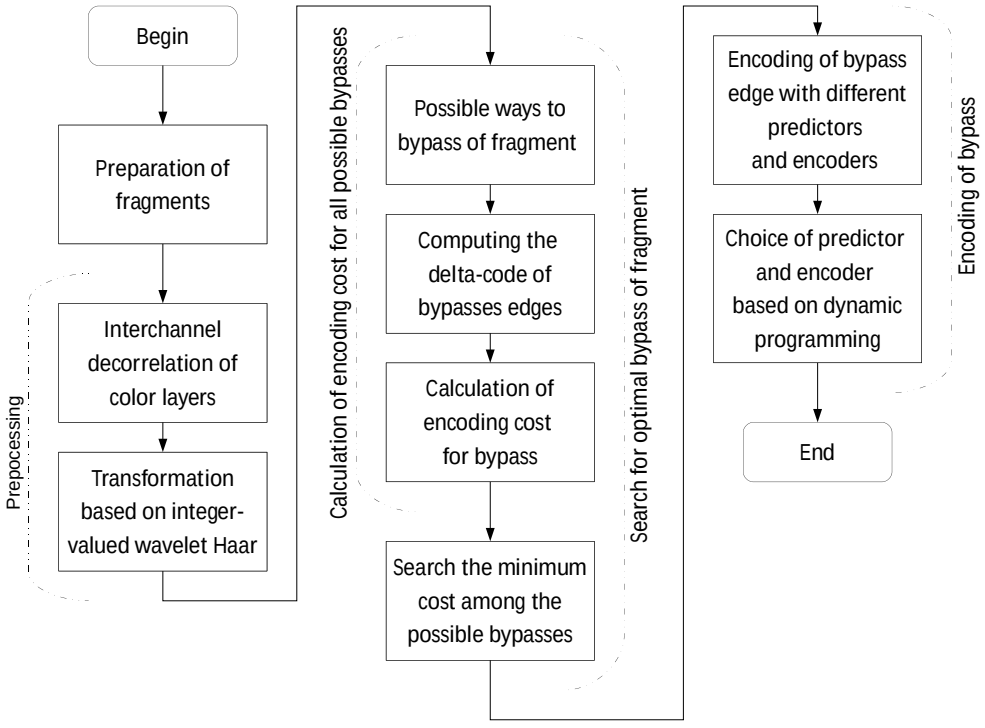

Fig. 1. Structure of lossless image compression program.

2. Basic algorithm

The basic algorithm - it is a cascading fragmentation of image [1], the search for the optimal bypass (path) of fragment [2] and dynamic programming for a pixels delta-code encoding on bypass [6]. After encoding the obtained data is further compressed using the standard library for Deflate algorithm. The compression ratio depends on the class of compressible image, the average is equal to 1.54 on array of test images [5].

The image compression duration depends on the processed image size. Due to a number of algorithmic solutions such as the cascading fragmentation, and the use of bypasses codebook instead of calculating the possible bypasses for each image fragment was able to increase the speed. Despite this, the image compression duration is still high enough [5]: 1) in group «2.1.*tiff» for 101 seconds 2) in group «2.2.*tiff» for 404 seconds 3) in group «4.1.*tiff» for 24 seconds 4) in group «4.2.*tiff» for 141 seconds.

The most complex in computational terms of the basic algorithm function is to estimate the encoding cost for all the possible bypasses. Meanwhile, this algorithm is suitable for parallelization, since the optimal bypass choice uses an exhaustive search in encoding cost estimates. For a fragment size 6*6 total bypasses number from the upper left corner is 22144.

To use all the multi-core CPU resources it is necessary to effectively implement paralleling of functions between all cores. In the basic program is a parallel execution of finding the optimal fragment bypass was performed using .Net Framework standard classes (SSE instructions). It is possible to use a more powerful CPU, but even in this case, the speed increase will not be significant.

In recent years more and more programs with parallel data processing use the GPU computing [7]. This is dictated by an increasing difference in performance between the CPU and GPU.

Due to the above it was decided to move part of the compression algorithm functions on the GPU. Of course, this will require some significant changes in the functions, but it will allow expect on a significant increase of the program speed without changing the basic algorithm.

Currently there are several approaches to the programs execution on the GPU. OpenCL is an open standard [8], which can execute programs on the CPU and GPU of different manufacturers. Therefore, in this work to speed the algorithm, we chose the OpenCL.

At the moment there are quite a number of different compression algorithms in general and in particular for images. Successfully used everywhere image compressed as lossy and lossless. Lossless compression is used, for example, in PNG algorithm where the actual compression is implemented by the Deflate algorithm [9, 10], which is a combination of the LZ and Huffman algorithms. There are no available free programs of lossless image compression with use OpenCL. WinZip an example of the compression program based on universal algorithm using OpenCL, where the performance increasing is about 45% [11].

In addition to the basic algorithm implemented the image preprocessing which described in the authors works: interchannel decorrelation of image color layers [12] and the transformation pixels matrix based on the integer-valued Haar wavelets [13]. These functions can be easily threaded for the implementation on OpenCL.

3. Methods of acceleration

A common task of this work, there was a change compression software in order to transfer part of calculations in OpenCL. Diagram of image compression program is shown in Fig. 1.

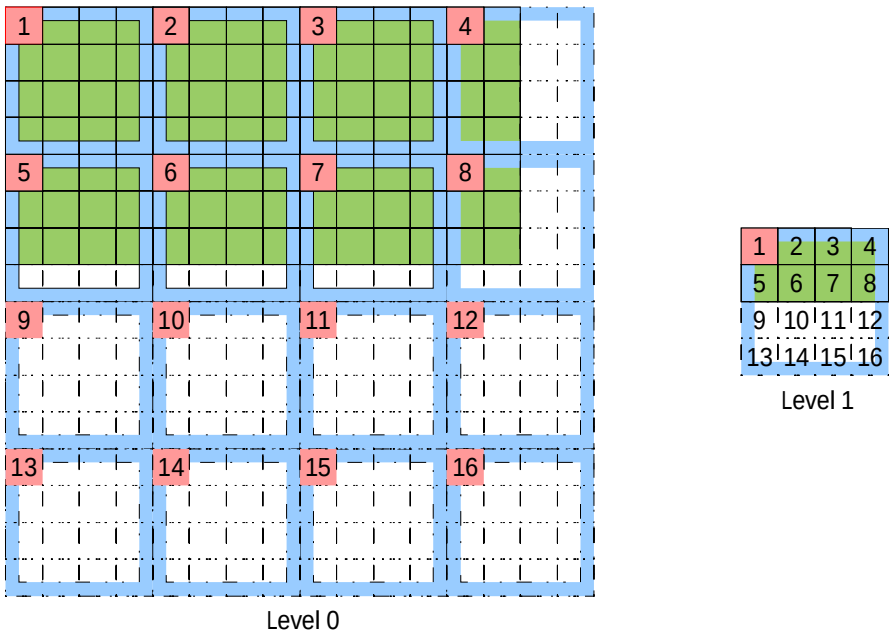

Fig. 2. Preparation of fragments.

3.1. Preparation of fragments

The function receives a image separate color layer. The function output is arrays of separate fragments of fixed size. Pixels of the fragment beyond the image borders are virtual pixels and the values of this pixels are set as a constant (white pixels on Fig. 2). The top left pixels of each fragment on level 0 constitute the fragments on level 1 and so on while the fragments number on level is more then one. The data structure passed to the OpenCL kernel is the matrix of image values, the output structure is the array of separate fragments [1].

3.2. Preprocessing

3.2.1. Interchannel decorrelation of color layers

This function is designed to calculate the interchannel decorrelation between groups of the color channels (layers) in the original image and to find the best variant to group them [12].

When function starting it need to pass the arrays containing the pixels values of all color channels of the fragment, and the number of channels (Fig. 3). In addition, are needed data on the possible grouping of channels.

The formula for calculating interchannel de-correlation for any channels number based on the mean and interchannel differences is apply [12]:

$$P^1 = Round\left(\frac{\sum_{i=1}^{n} X^i}{n}\right)$$
$$P^i = X^1 - X^i, i = \overline{2,n} \quad (1)$$

where *Round* is the rounding operation to the nearest integer; X^i – pixels value on each of the channels; k – channel index; n – the number of processed channels.

The color channels can be independent from each other, therefore, it necessary to select grouping variant based on a minimum estimating of encoding costs. It necessary to implement the decorrelation formulas for all dependent channels groups. The minimum channels number in the group is equal to 2. If the image consists of 3 channels (24 bits per pixel), we get the following grouping variants:

$$(X^1X^2X^3)\ X^1(X^2X^3)\ X^2(X^1X^3)\ X^3(X^1X^2)\ X^1X^2X^3 \quad (2)$$

where for groups of channels marked with parenthesis it is necessary to implement the decorrelation formulas.

The calculation of the decorrelation is performed for all possible groups (g=1..G). The result is the index of grouping g:

$$g = \underset{g}{argmin}\left(\sum_{g=1}^{G}\left(\sum_{i=1}^{n}\sum_{j=1}^{k} Cost\left(P^i{}_j\right)\right)\right) \quad (3)$$

where $P^i{}_j$ – is the pixel value after the interchannel decorrelation for the grouping index g; i – channel index; j – pixel index; n – number of channels; k – pixel number in the fragment.

Cost is the some estimation function of encoding cost, for example, the length of the Fibonacci code which coding the value $P^i{}_j$, or the estimated length of binary coding:

$$log_2\left|P^i{}_j + 1\right| + 1 \quad (4)$$

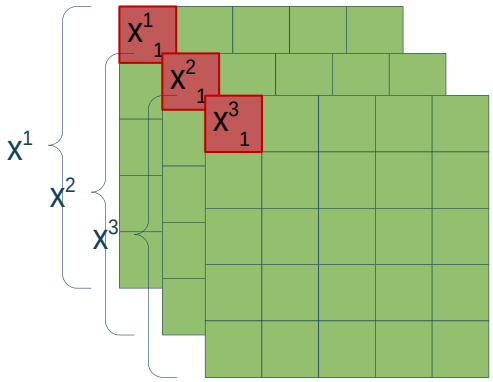

Fig.3. Interchannel decorrelation of color layers.

3.2.2. Transformation based on integer-valued wavelet Haar

This function is designed for cascading processing of each image fragment by applying an integer-valued version of the Haar wavelet transform [13].

The function is passed an array containing a single channel of image and the number of fragments in width and height which need to processed.

In the course of the algorithm it uses additional arrays for each executable OpenCL kernel with a size equal to one fragment, for storing the intermediate results of the cascading conversion.

The function result is an array of size equal to original image.

Image matrix should be divided into blocks of size 2*2. Then calculated the values for *a*, *h*, *v*, *d* by the formula [13]:

$$\begin{gathered} c_2 = x_1 - x_2 \\ c_3 = x_1 - x_3 \\ c_4 = x_1 - x_4 \\ c_1 = x_1 - Round\left(\frac{1}{4}\sum_{i=2}^{4} c_i\right) = x_1 - z_1 \approx \frac{1}{4}\sum_{i=1}^{4} x_i \end{gathered} \quad (5)$$

$$\begin{gathered} a = c_1 \\ h = -Round(d/2) + c_3 \\ v = -Round(d/2) + c_2 \\ d = c_3 - c_4 + c_2 \end{gathered} \quad (6)$$

where *Round* is the rounding operation to the nearest integer; x_i – original pixels of the block.

The obtained values of a, h, v, d should be recorded in positions of the matrix, as shown in Fig. 4. Calculation should be carried out as multiresolution, repeating the transform on the blocks consisting of grouped values a^i, each time reducing the size in 2 times in each coordinate. Cascading transform will stop then the block a^i with size 2*2 is absent.

It should be noted that when the fragment size of 2^m*2^m it is possible to use preprocessing (interchannel decorrelation, and Haar transform) after the function of dividing on the fragments. In this case, the Haar transformation is possible only within the same fragment. With this approach, the fragment encoding completely independently of the other fragments and therefore the decoding possible for a single fragment.

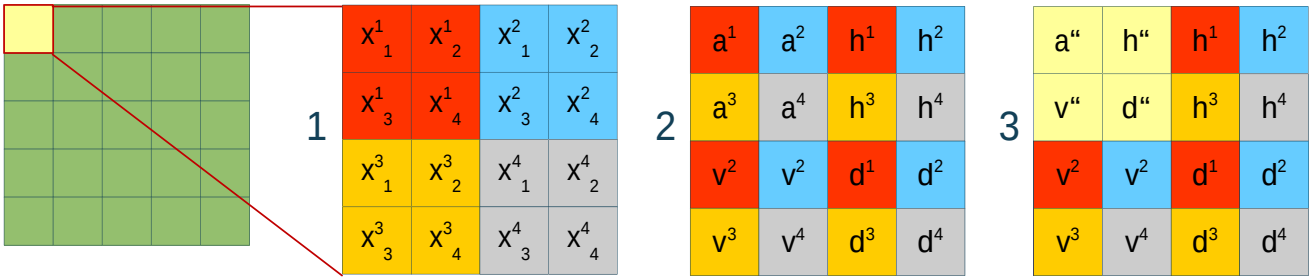

Fig. 4. Multiresolution transformation based on wavelet Haar.

3.3. Search for optimal bypass of fragment

The function receives the fragments obtained after preprocessing functions (integer-valued Haar transform, interchannel decorrelation of color layers).

3.3.1. Calculation of encoding cost for all possible bypasses of fragment

3.3.1.1. Possible ways to bypass of fragment

Information about all the possible bypasses for a given fragment size is known for encoding and decoding algorithms.

All bypasses (paths) have been calculated in advance and constitute the bypasses codebook [2].

Therefore, in encoding and decoding only need to know the bypass index, but not the edges of bypass (path).

3.3.1.2. Computing the delta-code of bypasses edges

This function is designed to calculate the difference values for all pairs of nodes that make up the edge on a given fragment bypass [2]. In the course of the function uses the previously prepared fragments.

The result is a list of arrays containing the delta-code of all edges for each fragment.

For each fragment you need to make an array of differences between the nodes values (delta-code) connecting the edge *e* is calculated according to the formula:

$$\Delta_e = x_{start(e)} - x_{stop(e)} \quad (7)$$

where $x_{start(e)}$, $x_{stop(e)}$ is the pixel values are connected by an edge *e*; *start*(*e*) and *stop*(*e*) - nodes indixes of edge *e*.

3.3.1.3. Calculation of encoding cost for bypass

For each fragment, estimates the encoding cost for each of the possible bypass. The function receives the previously prepared fragments and details of all the fragment bypasses in a codebook.

The result is the array containing estimated encoding cost for each bypass of fragment (Fig. 5, Table 1).

Estimating the encoding cost of bypass through all edges (with its delta-codes) it is possible to produce in different ways. For each fragment is need to find the cost of the bypass:

$$\Sigma_s = \sum_{e=1}^{E} Cost(\Delta_e) \cdot z^{e_s} \quad (8)$$

where E – length of bypass (number of edges); e - edge index; S - number of bypasses; s - bypass index; Δ_e - delta-code of edge; z^e_s – presence an edge e in bypass s; *Cost* – is the some estimation function of encoding cost, it is similar to the *Cost* in interchannel decorrelation function.

It should be noted that the estimate of bypasses encoding cost based on the table 1 is effective from the point of view of parallel computing. In this function there is parallel processing of all possible bypasses downloaded from the codebook, for all fragments, forming the processed image.

3.3.2. Search the minimum among the possible bypasses

After the estimates of encoding costs of all paths is selected and the save path of each fragment with the smallest estimate.

$$s = \underset{s}{argmin}(\Sigma_s) \quad (9)$$

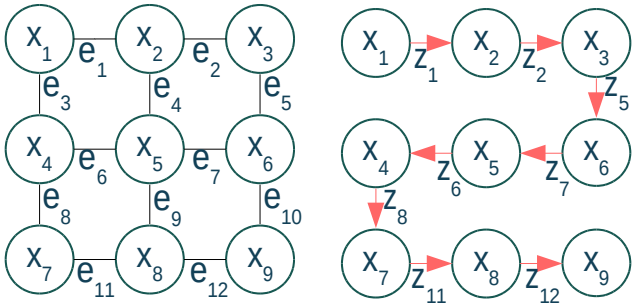

Fig. 5. Example of bypass on fragment 3*3.

Table 1. Calculation of encoding cost for bypass in fragment 3*3

e	Δ_e	z^e_1	z^e_2	…	z^e_8
1	Δ_1	1	1	…	0
2	Δ_2	1	1	…	1
3	Δ_3	0	0	…	1
4	Δ_4	0	0	…	1
5	Δ_5	1	1	…	1
6	Δ_6	1	1	…	0
7	Δ_7	1	0	…	0
8	Δ_8	1	1	…	1
9	Δ_9	0	1	…	1
10	Δ_{10}	0	1	…	1
11	Δ_{11}	1	0	…	1
12	Δ_{12}	1	1	…	0

3.4. Encoding of bypass

This function is designed to encode the previously found optimal bypass. That is, the obtained array of bypass nodes values with a minimum encoding cost, must be handled by the predictor and encoded by the encoder.

Note that the encoding of bypass which previously was found as optimal bypass can be performed in various ways. In particular, there can be used known generic methods: Huffman algorithm or arithmetic coding.

In suggested algorithm it is offered to perform the encoding of bypass using more sophisticated method [6]: 1) using a set of predictors and encoders for encode the bypass edge; 2) using dynamic programming for choice of predictor and encoder for edge in purpose to optimize (to minimize) of total encoding cost of bypass.

3.4.1. Encoding of bypass edge with different predictors and encoders

In the simplest case, as the predictor uses the edge delta-code described above, and as encoder uses the Fibonacci codes [3]. In this case, the applying of dynamic programming to select the predictor and the encoder is not required. In a more complex cases number of choices of predictors and encoders may be more than one. For example, it is possible case with the predictors on the basis of not only the finite difference of the first degree, but higher degrees, and with the coders with use Rice codes with different bases. The use of a set of predictors and a set of encoders increase the resulting image compression ratio, but this raises the problem of choosing the best predictor and encoder for the current section of the bypass array.

3.4.2. Choice of predictor and encoder based on dynamic programming

In this embodiment, compression algorithm to encode each bypass edge use the most optimal encoding parameters (predictor/encoder) based on dynamic programming [6]. In start of the function it is loaded the table of encoding cost for every encoder for values of every predictors for all pixels on edges of the optimal bypass.

In the result the function creates a data file containing information with the encoded using encoders predictors values for edges and additional information – optimal switching of the predictors/encoders for edges of bypass [6].

Due to the complexity of the dynamic programming algorithm to run on OpenCL managed to transfer only a part, responsible for the coding. This part contains branching, and is switching large sections of the algorithm. These operations are an integral part of the algorithm, or change the calculations flow in aim of parallel execution without the use of branches is not possible.

4. Results and Discussion

Screen form of the developed software shown in Fig. 6. The program displays the following information: used hardware processor device and software platform; the number of files to process; the current processed file; the execution duration of the compression program particular functions; the execution duration of compression; the compression ratio.

To use OpenCL acceleration requires the presence of GPU or CPU with support of OpenCL 1.2 [8]. You must install the appropriate OpenCL support software which distributed with equipment.

To compile the developed image compression program requires the following software components [8]: 1) DotNetZip library, for the final compression results using the Deflate algorithm; 2) to provide OpenCl bindings for C#, use the Cloo library from OpenTk; 3) to compile and execute kernels on the GPU you must have the required header files for OpenCL support. To compress the encoding results used library Ionic.Zip.dll. In addition is used a set of libraries to support work OpenCL, bindings these libraries to .Net Framework and the source code of the OpenCL kernels.

The basic program required about 250 MB of RAM. When algorithm was adapted for accelerating on OpenCL was added using arrays of large volume and memory required increased to 900 MB.

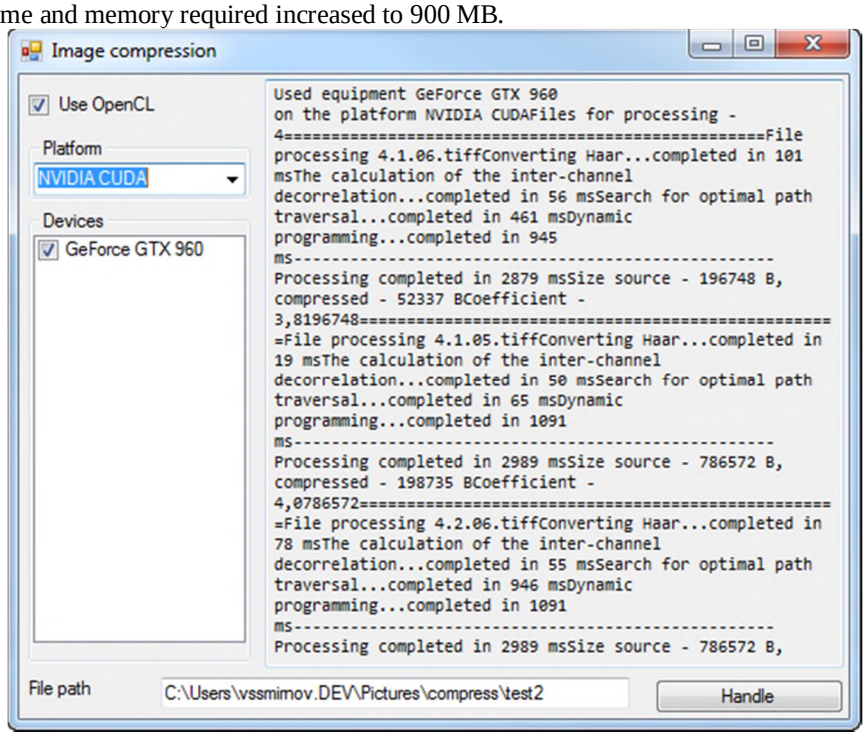

Fig. 6. Screen form of the developed software.

Test batch of images designed to assess acceleration of all core functions of modified program in comparing with basic. To estimate the dependence of the program speed to images size the batch have the images of different sizes. To check used 4 images from the standard set provided by the Institute of signal processing and images: 4.1.06.tiff, 4.2.05.tiff, 4.2.06.tiff, 4.2.07.tiff. The color depth of the images are 24 bit. Image sizes: 256*256, 512*512, 1024*1024, 2048*2048.

Testing was performed using image compression as the basic program on the CPU AMD Phenom II X4 955 and a program using OpenCL. For testing OpenCL parallel processing was used different 4 devices: 1) GPU AMD Radeon HD6850; 2) GPU Nvidia GTX 960; 3) CPU AMD Phenom II X4 955; 4) CPU AMD FX-4300. The time spent on the particular functions of the algorithm, and the total processing time for each image are given in Table 2 and Fig. 7, where F1 - integer-valued Haar transform, F2 - interchannel decorrelation of color layers F3 - search of the optimal bypass, F4 - encoding bypass using dynamic programming.

When testing for each fragment was fixed size: 6*6 pixels, and the number of bypasses: 22144.

It should be noted that when using the GPU Nvidia GTX 960, while the high number of processing devices and high work frequency, according to Profiler, the load does not exceed 60%. Compression image size 2048*2048 pixels on the GPU AMD Radeon HD6850 failed to produce due to the lack of graphics memory. In the future, to avoid this situation, the necessary modification of the program: to run the calculations flow should be divided into several groups and processed sequentially.

In the basic program were not implemented functions the integer-valued Haar transform and the interchannel decorrelation, and therefore, testing of these functions was carried out.

Testing showed approximately the same reduction in the compression total time when using both CPU and GPU. The larger the size of the processed image, the greater the acceleration obtained as long as there is memory available to OpenCL.

GPU showed the best results for searching the optimal bypass. CPU well with the function of dynamic programming, because of presences a large number of branches in the function, despite the small number of processor cores.

Calculations of interchannel decorrelation and integer-valued Haar transform is performed using OpenCL for a short time compared to total compression time.

Table 2. Results processing of test images

Used program / device		Image size, pixels	Execution time of compression particular functions, miliseconds					Acceleration, Times		
			F1	F2	F3	F4	Total	F3	F4	Total
Program on OpenCL	AMD Radeon HD 6850	256*256	71	65	677	528	2581	2,2	9,0	2,8
		512*512	43	63	913	1129	4643	6,1	13,4	5,2
		1024*1024	66	187	2017	4492	15217	12,9	13,5	6,4
		2048*2048	151	438	—	—	—	—	—	—
	Nvidia GeForce GTX 960	256*256	23	20	160	360	2057	9,4	13,2	3,5
		512*512	11	36	411	1153	4402	13,6	13,1	5,5
		1024*1024	33	148	1474	4226	13416	17,7	14,4	7,3
		2048*2048	122	675	5354	14530	50031	31,8	15,9	8,6
	AMD FX-4300	256*256	34	30	350	499	2581	4,3	9,5	2,8
		512*512	18	48	913	1146	4207	6,1	13,2	5,7
		1024*1024	45	206	3324	4093	15514	7,9	14,8	6,3
		2048*2048	172	920	13164	13629	58567	12,9	17,0	7,4
	AMD Phenom II X4 955	256*256	31	32	455	664	2407	3,3	7,2	3,0
		512*512	21	56	1 378	1222	5981	4,0	12,4	4,0
		1024*1024	67	239	4571	4477	16777	5,7	13,6	5,8
		2048*2048	227	992	18432	12804	61350	9,2	18,1	7,0
Basic program / AMD Phenom II X4 955		256*256	—	—	1504	4751	7273	1,0	1,0	1,0
		512*512	—	—	5570	15155	24183	1,0	1,0	1,0
		1024*1024	—	—	26107	60698	97297	1,0	1,0	1,0
		2048*2048	—	—	170161	231398	431555	1,0	1,0	1,0

5. Conclusion

In the course of this work was modified the basic program of image compression without losses, with the aim of increasing the speed. The parallel processing based on OpenCL was used for program acceleration. This solution significantly affected the processing speed, enabling to reduce computational time. This modification will allow more efficient use of the program in the future, will facilitate future research aimed at improving the compression ratio.

The changing of optimal bypass search function allowed to obtain the acceleration up to 32 on the large images. This acceleration has been achieved because of executing OpenCL functions almost linear, and branching, even where they are, have only a few simple operations. For future program modification to accelerate this function is important because it is possible to use fragments of larger size that was previously impossible due to too much execution time. Among other things fragments with the size of 2^k*2^k will effectively apply the integer-valued Haar transformation for the fragment, and will allow to compress every fragment separately.

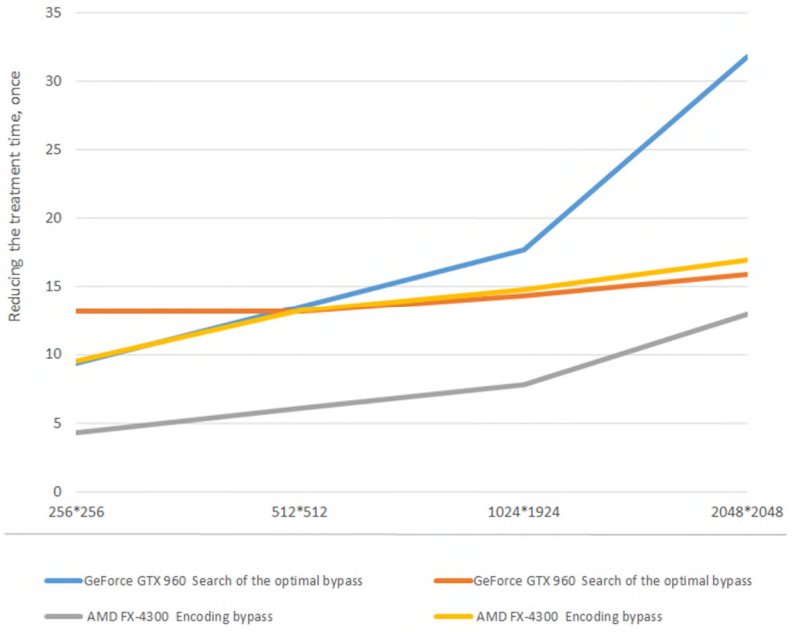

Fig. 7. Comparison of image compression acceleration.

Somewhat worse is the situation with dynamic programming during encoding fragment bypass. Speed managed to increase mostly by ordinary parallel execution of some operations, shutdown of operations which need only for debugging purposes and the use of the packet data read operations. The part that runs on OpenCL gives the increase in performance is only about 30% compared with ordinary parallel computing. On the other hand, even this result is good enough, given the fact that OpenCL function has a large enough branching. Note that the bypass encoding can be performed in various ways, for example, Huffman algorithm or arithmetic coding.

References

[1] Smirnov, V. S. Cascade Image Splitting into Fragments at Lossless Compression on Basis of Image Bypass Optimization / Smirnov, V. S., Korobeynikov, A.V. // Bulletin of Kalashnikov ISTU. – Izhevsk: ISTU. – 2012. – Vol. 2. – P. 143-144.

[2] Korobeynikov, A.V. Optimal Bypass Definition with Code Book Application at Images Lossless Compression / Korobeynikov, A.V., Smirnov, V. S. // Bulletin of Kalashnikov ISTU. – Izhevsk: ISTU. – 2012. – Vol. 3. – P. 114-115.

[3] Smirnov, V. Ordering the numeric sequence of image pixels at lossless compression / Smirnov, V., Korobeynikov, A. // I International Forum “Instrumentation Engineering, Electronics and Telecommunications (November, 25–27, 2015, Izhevsk, Russian Federation). – 2015” – P. 175-180.

[4] Sample images from the site of University of Southern California (URL: http://sipi.usc.edu/database/database.php?volume=misc). Retrieved 2017-01-10.

[5] Smirnov, V. The results of testing lossless compression algorithm based on cascade fragmentation method and ordering pixels sequence / Smirnov, V., Korobeynikov, A. // II International Forum “Instrumentation Engineering, Electronics and Telecommunications (November, 23–25, 2016, Izhevsk, Russian Federation). – 2016”.

[6] Korobeynikov, A.V. The Use Of Dynamic Programming And Fibonacci Codes For Interchannel Decorrelation In The Three-Channel Signals Lossless Compression // Bulletin of KIGIT. . – Izhevsk: KIGIT. – 2010. – Vol. 1. – P. 72-81. (URL: http://elibrary.ru/item.asp?id=18348092). Retrieved 2017-01-10.

[7] Denny Atkin. "Computer Shopper: The Right GPU for You" (URL: http://www.computershopper.com/feature/the-right-gpu-for-you). Retrieved 2017-01-10.

[8] Official webpage of the standart OpenCL (URL: https://www.khronos.org/opencl/). Retrieved 2017-01-10.

[9] Portable Network Graphics (PNG) Specification (Second Edition) (URL: https://www.w3.org/TR/PNG/). Retrieved 2017-01-10.

[10] PNG Home Site (URL: http://www.libpng.org/pub/png/). Retrieved 2017-01-10.

[11] WinZip official webpage (URL: http://www.winzip.com/win/ru/index.htm). Retrieved 2017-01-10.

[12] Franchenko, R.S. Interchannel Decorrelation for Any Number of Channels at Lossless Compression of Multichannel Signals / Franchenko, R.S., Korobeynikov, A.V. // Bulletin of Kalashnikov ISTU. – Izhevsk: ISTU. – 2010. – Vol. 1. – P. 87-88.

[13] Smirnov, V. S. Lossless Image Compression Based On Integral-Valued Haar Wavelets / Smirnov, V. S., Korobeynikov, A.V. // Intelligent Systems in Manufacturing. – Izhevsk: ISTU. – 2013. – Vol. 2. – P. 158-160.