Int J Parallel Prog (2017) 45:1108–1127
DOI 10.1007/s10766-016-0459-9

Implementation of Digital Watermarking Algorithms in Parallel Hardware Accelerators

Andrzej Głowacz[1] · **Marcin Pietroń**[2]

Received: 29 February 2016 / Accepted: 22 September 2016 / Published online: 21 October 2016

Abstract The paper is focused on computing acceleration for hybrid multiprocessor environment. A considered algorithm of content authentication used for digital images and video sequences is implemented and tested in diverse scenarios. Particular goal is capability of the authentication system to process high-resolution digital images or FullHD video sequences in real time. Aim of this work was to explore and take advantage of mixed CPU and GPU processing approach and to investigate possibilities to develop optimal authentication algorithm. Chosen algorithm is based on robust hashes and semi-fragile digital watermarking. Parallelization was achieved using combined OpenCL and OpenMP. Results for time execution were measured for both images and videos. Based on collected results acceleration rates were calculated along with maximum frames per second values for video processing. It can be concluded that parallelism contributes significantly in reducing the computation time by making optimal use of resources. Depending on the test scenario, the rate of acceleration is even thirty times higher comparing with single-core solutions. Introduced modifications reduce execution time while maintaining detection effectiveness.

Keywords GPU processing · Image authentication · Semi-fragile watermarking

✉ Marcin Pietroń
pietron@agh.edu.pl

Andrzej Głowacz
aglowacz@agh.edu.pl

[1] Department of Telecommunications, AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Cracow, Poland

[2] AGH University of Science and Technology, ACK UST Cyfronet, Supercomputing Center, ul. Nawojki 11, 30-950 Cracow, Poland

1 Introduction

Content authentication is a method used to protect digital works from unwanted modifications. It can be implemented using digital watermarking technology [16,17,20] that inserts specific data stream into multimedia data stream in specific domain. Authentication methods are diverse, yet they can be subdivided into strict and selective techniques [1]. Algorithm tolerance against modifications is a key factor in above classification. Strict authentication permits no changes in original content (e.g. digital image), whereas some changes are tolerated in selective authentication. In the latter case, the changes cannot directly concern the information content of the image. Depending on defined application and requirements a selected method is used, and there are various approaches. Literature methods in the area of strict authentication are based on fragile watermarking [2] or cryptographic algorithms [3], whereas selective methods employ semi-fragile watermarking [4] or robust hashes [5].

Specifically, authentication of video content is a challenging task, due to amount of data that needs to be processed both for watermark encoding and detection. Multicore architectures can be applied to speed up algorithms and optimize the use of resources. Such systems include multicore processors (CPU), multiple processing nodes, computing using general-purpose graphics processing units (GPU) or other dedicated processing components (e.g. FPGA units). There are some content authentication approaches that focus on improving the performance of GPU processing [6].

Due to physical limitations on hardware frequency, significant computing acceleration is usually achieved using multiple computing cores or nodes in high-performance clusters. There are number of software technologies spread across the world in recent years that support parallel computing in various platforms. One of the solutions is OpenCL (Open Computing Language) [7] which allow for performing calculations with the use of GPU or CPU. What is important not only computations strictly related to processing graphic are supported. These calculations include, but are not limited to bioinformatics, chemistry, financial, seismology, physics or special imaging. According to computer architecture taxonomy, OpenCL follows SIMD (Single Instruction Multiple Data) architecture and it enables optimal use of resources for the cost of complexity of implementation. OpenMP (Open Multi-Processing) [8] is a multi-platform API based on shared memory multiprocessing and explicit programming model. Team of parallel threads are usually created by master thread, synchronized and terminated in *fork-join* model.

The remainder of the paper is organized as follows. Section 2 refers to robust hash functions and number of techniques that were implemented in order to improve the processing performance and research towards finding optimum variant of the algorithm. In Sect. 3 state-of-the-art technologies that were used for achieving parallelization are presented. Proposed algorithm and its modifications and theirs parallel implementation in GPU and CPU are described in Sect. 4. Focus is put on achieved performance results in Sect. 5. Considered scenarios include comparison of algorithm steps, characteristics in different watermark embedding domains and algorithm speed up in diverse architectures, among others. Conclusions are provided in the last section. Detailed results are presented in tables at the end of the paper.

2 Digital Watermarking

Robust Hash Functions algorithm for digital watermarking has been presented by Fridrich and Goljan [9,18]. This algorithm has been identified as a good basis for development of content authentication system. It fits in selective authentication methods and provides resistance to lossy compression. Thus, it can be applied to protection of most video content transmitted over internet connections, since compression is often used due to limited network resources.

The idea behind the algorithm is calculating robust hash, which depends on image information content. Video sequence is treated as a set of individual frames, that are processed independently. Pixel colors are converted from RGB to YCbCr color space. Digital watermark is obtained on the base of generated hash and embedded in the image providing resistance to non-malicious modifications. In case of image tampering, the watermark becomes unreliable and indicates that the image is not authentic.

It is worth pointing out that robust hashes depend on image characteristics, but not on exact values of the image pixels. Thus the method is sensitive only to significant changes in image that affect information content. In case of modifications not changing the image content, the resulting robust hash will be the same as in original image.

Algorithm consist of three major steps. In first one, robust hash is generated. In order to obtain hash values, projections of pseudorandom sequences on image blocks are performed and output values are sorted. Binary robust hash is then obtained using thresholding. In second step, watermark is created based on robust hash, after performing values permutations together with block number and secret key. Finally, resulting digital watermark is embedded in the image using spread spectrum techniques. Detailed description of the method is presented in [10].

Authentication is verified by comparison of watermark extracted from the image and results of first and second step performed on tested image. Thus, two watermarks are contraposed against each other, and result indicates whether the image is valid or not.

To sum up, described method can be classified as selective authentication and it reveals features of both semi-fragile watermarking and robust hashes. Digital watermark is embedded in the image using classical methods, and watermark is generated based on robust hash information.

It was important to introduce modifications to the original algorithm for the needs of our work and to improve overall processing performance—both for image protection and authentication detection.

Since the original method only supported spatial embedding domain [19], our implementation included frequency domain support. This way, we could compare spatial and two-dimensional DCT domain [11]. In addition, in both spatial and frequency domain additive spread spectrum technique for embedding watermark was added, in order to compare its effectiveness.

In detection process the correlation coefficient of generated and received watermark was calculated in spatial domain. For new frequency domain it was replaced by asymptotically optimal correlator which offers significantly better detection performance in frequency domain [12].

In order to further improve the processing performance, a reduction in calculation time was necessary, that maintains high level of detection efficiency. Implementation was redesigned so digital watermark based on robust hash is replaced by raw hash bits [13]. QIM technique [14] was used for that purpose. Corresponding changes were implemented in detector. For given spatial or frequency domain, embedding mask is chosen that define the number of pixels to embed each bit of hash. Blocks of 64×64 pixels were chosen for all scenarios, in result of trade-off between security level and localization capabilities.

Finally, implementations were prepared for single and multicore CPU platforms, general purpose graphic cards and heterogeneous cluster system. The implementation of algorithms is based on OpenCV library [23].

3 Parallel Software Environments

Parallel hardware platforms can be programmed by high-level programming frameworks. These programming frameworks are based on high-level languages like C language, with built-in mechanisms to exploit parallelism from specific hardware platforms. In our implementation, the OpenMP environment and OpenCL framework were used.

3.1 OpenCL Programming Model

OpenCL [7] is a software architecture that enables the graphics processing unit (GPU), to be programmed using high-level language. OpenCL can be used in programming other parallel hardware platforms e.g. multicore processors. The main advantage of OpenCL is its portability. Programs written in OpenCL can be run on AMD and NVIDA GPU cards (for example CUDA programs run only on NVIDIA hardware [22]). OpenCL provides three key mechanisms to parallelize programs in GPU: thread group hierarchy, shared memories, and barrier synchronization. These mechanisms provide fine-grained parallelism nested within coarse-grained task parallelism. Creating the optimized code on GPU cards is not a trivial task, so thorough knowledge about this hardware accelerator architecture is needed. The main issues to solve are usage of memory, efficiency of dividing code to parallel threads, and thread communications. Programmers should optimally use them to speed up access to data on which an algorithm operates. Another important aspect is to optimize synchronization and communication between the threads. Synchronization of the threads between blocks is much slower than within a single block. If necessary, it should be solved by the sequential running of multiple kernels.

3.2 OpenMP

OpenMP [8] is a concurrency platform for multi-threaded, shared-memory parallel processing multi-core architectures for C, C++ and Fortran languages. By using OpenMP, the programmer does not need to create the threads nor assign tasks to each thread. The programmer inserts directives to assist the compiler into generating threads for the parallel processor platform. OpenMP is a higher-level programming

model compared to *pthreads* in the POSIX library. The OpenMP consists of the following major components:

- *Compiler directives* which enable the programmer to instruct the compiler thread creation, data management, thread synchronization, etc. Most popular are: *atomic* (memory location that must be updated atomically), *barrier* (synchronization of all threads in the same region), *critical* (defines critical section executed by single thread at a time), *for* (defines for loop iterations should be run in parallel), and *parallel* (defines region of the code that will be run by multiple threads). Each OpenMP directive can be followed by a collection of clauses which mainly define thread variables and their access policy:
- *Runtime library functions* control the parallel execution environment, control and monitor threads, control and monitor processors;
- *environment variables* variables to alter the execution of OpenMP applications.

The most important advantage of the OpenMP framework is that the programmer does not have to restructure the sequential source code. The process of making parallel version only consists of insertion appropriate compiler directives to restructure the serial program to a parallel one.

4 Parallel Implementation of Watermarking Algorithms

The implemented algorithm can be divided into few steps. The most important ones are:

- generation of resistant hash
- watermark creation based on generated hash
- adding the watermark into the image

The detection process is similar apart from the last step, where apart from adding watermark into the image the detection algorithm is performed to decide if image was modified. At the beginning the image is divided to separate 64×64 blocks. All implemented operations (apart discrete cosine transformation) are run on these blocks. The smaller blocks then the quality of correlation (as a detection method) is and the security (less combinations to fraud) are worse. Each block of image is projected to vector (with 4096 pixels). For each vector N_h pseudo random sequences S_i (normal distribution with between (0,1)) are generated using specified key K. The low-band signal filter is run on each sequence. From the output the average value is subtracted to removed constant component. Then projection to single bit of each block sequences is performed (Eq. 4.1). Therefore we receive N_h bit hash for each data block.

$$h_j = \begin{cases} 1 & \text{when } \frac{1}{M}\left|x^T \cdot s^j\right| < T_e \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

value T_e is a quantization threshold computed as median from blocks hashes (set of binary values with equal number of zeros and ones).

The next stage is a generation of watermark from computed hash. The input values are: hash, predefined key K and number of block. These values allow creating N_h new

unique keys N_ξ. From computed keys N_h pseudo random sequences ξ are generated. Sequences are used to create final watermark η. Process of keys K_ξ and watermark creation was proposed in [9]. All operations are performed on 64×64 blocks. Firstly, hash is permuted N_p times (usually five times). Hash and its permutations are written in row order. Each column consists of N_p elements (each element from different permutation) used for computing each $K_\xi\,(i)$ key. Then these chosen elements from permutations are combined with key K, number of block and number of generated key (number from N_h). Usually it is done by XOR operations (Eq. 4.2). After that, set of N_h keys K_ξ is received. The keys are used by generator to create new pseudo random sequences ξ with values of uniform distribution from (-1, +1).

$$\xi^{(i)} = PRNG\,(K \oplus B \oplus BitNo \oplus b_1 \oplus b_{21} \oplus b_{48} \oplus b_{12} \oplus b_9) \tag{4.2}$$

where K—key, B—number of block, *BitNo*—bit number in hash $b_i - N_p$ bits from chosen from permutations.

Final watermark is generated by adding sequences in element-wise manner (Eq. 4.3).

$$\eta = \sqrt{\frac{3}{N_h} \sum\nolimits_{i=1}^{N_h} \xi_i} \tag{4.3}$$

The received watermark independently from N_h value has number of elements equals number of block elements (4096). Then watermark can be embedded in source image. In case of detection process algorithm checks if watermark was changed.

In presented algorithm watermark embedding is done in spatial domain. In case of color images the compression to YCbCr scale is performed and further operations are executed on Y channel. After watermark insertion the reverse transformation is done. The method of watermark insertion is additive Spread Spectrum method (Eq. 4.4), insertion force is the same on whole image.

$$y = x + \alpha \cdot \eta \tag{4.4}$$

where: y—input image, x—output image, α—insertion force η—watermark.

Detection process run in similar way. Steps with generating hash and creating watermark are repeated. Then new watermark and watermark from received image are used to compute theirs mutual correlation (correlation coefficient, equation). If value of correlation is bigger than threshold D_{th}, block of the image is approved to be unchanged. Correlation ensures resistance from modifications of constant component of image and adding constant value to each pixel (Eq. 4.5).

$$\begin{aligned} \tilde{v} &= v - \bar{v} \\ \tilde{w_r} &= v - \bar{w_r} \\ z_{cc}\,(v, w_r) &= \frac{\tilde{v} \times \tilde{w_r}}{\sqrt{(\tilde{v} \times \tilde{v})\,(\tilde{w_r} \times \tilde{w_r})}} \end{aligned} \tag{4.5}$$

where w_r—signal which is checked, v—received signal, $\bar{x}$—average value. The presented above algorithm was changed for using Frequency Domain. Therefore Discrete

Cosine Transform was implemented. The DCT runs on 8×8 blocks, so the 64×64 blocks are divided to 64 blocks (JPEG standard). After the two dimensional transform watermark is embedded and the reverse transform is performed. Detector to presented frequency domain is used from [12]. It ensures better detection quality of modified blocks than previous one (Eq. 4.6).

$$D_R(Y, W) = \frac{\left(\sum_{i=1}^{N} sgn(Y_i) \times W_i \times |Y_i|^{\frac{1-\gamma}{1+\gamma}}\right)^2}{\sum_{i=1}^{N} |Y + i|^{2(1-\gamma)/(1+\gamma)}} \tag{4.6}$$

where: γ—shape parameter, N–number of elements in block, Y—received image, W—watermark.

The next modification is change of watermark generation and its embedding. Therefore method presented in [13] was used. In this method watermark is not generated from hash. Hash is embedded directly in image. It is used by QIM technique independently of domain. In QIM algorithm the whole step of watermark generation is omitted (permutation, Gaussian sequence generation). The embedded watermark is a bit series received from tresholding N_h pseudo random sequences (in this case hash bits). The N_h value in QIM algorithm is constant and equals 64. The 64×64 block is divided to 8×8 blocks in which one bit is embedded. The rule of embedding is described in equation. The force of watermark insertion can be changed by Δ parameter (parameter responsible for quantization step). The higher value of this parameter then force of embedding increases. The process of watermark embedding consists of modification of pixel by using two quantizers (Eq. 4.7).

$$\mathrm{y} = Q(x, \Delta, b) = \begin{cases} \left\lfloor \frac{x}{\Delta} \right\rfloor \cdot \Delta, & dla\, b = 0 \\ \left\lfloor \frac{x}{\Delta} \right\rfloor \cdot \Delta + \frac{\Delta}{2}, & dla\, b = 1 \end{cases} \tag{4.7}$$

where: y—output value, x—input value, Δ—quantization step, b—hash bit, $\lfloor x \rfloor$—rounding down of value x.

The number of modified pixels in 8×8 blocks in implemented QIM algorithm depends on domain in which watermark is embedded. In spatial domain it is 32 elements, in case of frequency domain it is 5 pixels in low-band frequencies. It compromise between force of insertion and resistance changes.

The detection process in QIM is quite different in case of Spread Spectrum method. The chosen pixel is quantized for bit value 0 and 1. The received value closer to input value was signed as more likely. One bit is embedded in each 8×8 block. Therefore number of 1 and 0 values are counted in each protection block 64×64 and the higher is selected. An the end it is compared with appropriate watermark bit and the decision is made if received watermark is correct.

Parallel implementation should be preceded by analysis of the source code. Firstly, control and data flow graphs should be known to evaluate the chance to speed up computations of given algorithm. DFG (data flow graph) can help to extract degree of hidden parallelism. It is worth to mentioned that loops are parts of code in which the programs spend about 90 % of its whole time of execution, furthermore they are main source of parallelism. After that adaptation of the code to parallel units should be

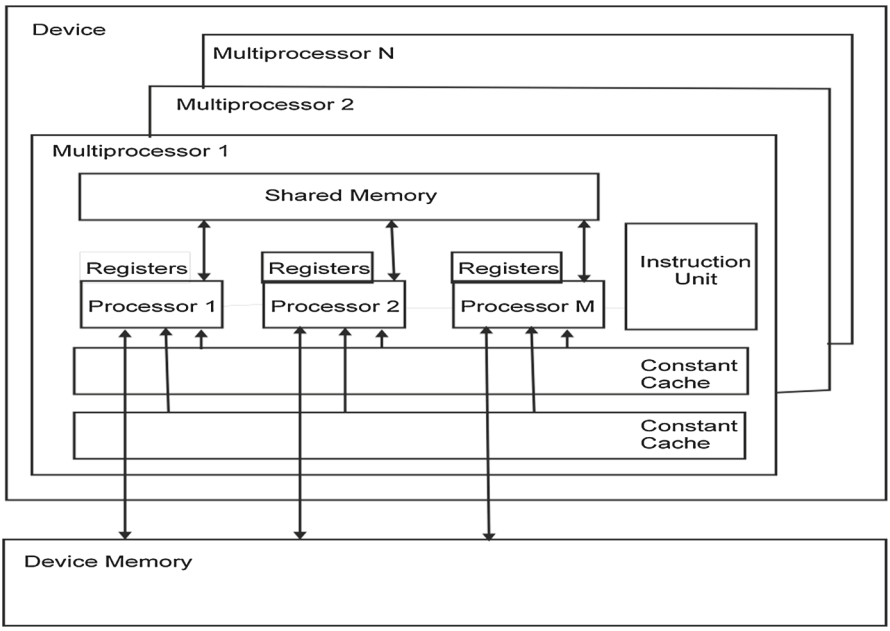

Fig. 1 GPU architecture

performed. The crucial aspects is projecting sequential control flow to parallel threads and mapping data to memory hierarchy in particular hardware accelerator.

4.1 GPU Implementation

The architecture of a GPU card is described in Fig. 1. GPU is constructed as N multiprocessor structure with M cores each. The cores share an Instruction Unit with other cores in a multiprocessor. Multiprocessors have dedicated memory chips which are much faster than global memory, shared for all multiprocessors. These memories are: read-only constant/texture memory and shared memory. The GPU cards are constructed as massive parallel devices, enabling thousands of parallel threads to run which are grouped in blocks with shared memory. A dedicated software architectures as described in previous section OpenCL or CUDA makes possible programming GPU using high-level languages such as C and C++.

Each part of the algorithm described above was implemented as a separate module in GPU. It can be run as a separate kernel function. Hash generation can be mapped in such way that single thread is responsible for processing one block of size 64×64. Pseudo random sequences are generated and stored in thread local memory. Then low-bound filtration, constant subtraction and projection are performed on sequences. This process is run sequentially by the thread and output bit is generated and stored in device memory. The second approach of hash creation can be done by mapping sequence generation among block threads. In results section the first implementation is measured.

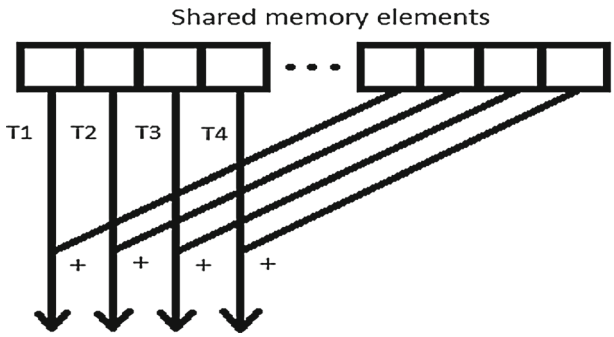

Fig. 2 Parallel reduction in GPU

The watermark generation is divided to following device procedures:

- Function which permutes hash and stores them in shared memory (each thread responsible for single image block), the permuted sequences are stored in a manner to avoid bank conflicts in shared memory,
- Function that generates from permuted sets the new keys and sequences and sums theirs elements (Eq. 4.3) to create watermark

The watermark embedding is implement as element-wise multiplication and addition which is fully parallelized in GPU. Each thread processes single pixel of input image, force of insertion vector and watermark (Eq. 4.4). The same vector multiplication and subtraction is used in detection process (Eq. 4.5). Firstly element wise operation are used then parallel reduction is executed (Fig. 2). The reduction is implemented that each thread process single pixel, addition is performed without bank conflicts and in the last stage (less elements than 64 to sum) all threads in warp are unrolled (no need of synchronization).

In case of frequency domain the Discrete Cosine Transform was implemented. Each block is responsible for one 8×8 block processing. Each thread computes single output value in frequency domain. The current version of DCT is with no complexity reduction. Therefore there is no need of synchronization of block threads. The discrete values of cosine function are pre-computed and stored in constant memory. The frequency domain correlation (Eq. 4.6) is based on distance computing (vector multiplication and reduction).

The last module QIM method divide 8×8 blocks among threads. Each thread read one bit of watermark and executes quantization (Eq. 4.7) on chosen pixels of block (32 pixels in spatial domain and 5 pixels in frequency domain).

In all cases images are read from device memory to shared memories in coalesced manner to avoid memory access delays.

4.2 Multicore CPU Implementation

Multicore implementation is based on previous analysis. All mentioned parallel sections are covered by OpenMP directives (*parallel for*). The source data are stored in aligned manner. The code is also fully vectorized for SSE and AVX modules.

5 Experimental Results

In this section results of testing the described watermarking methods will be presented. The execution time of whole algorithms were measured and selected most interesting parts of them. The hardware platforms on which algorithms were run are single and multicore CPU platforms, general purpose graphic cards and heterogeneous cluster system. The main steps of the algorithm measured by the systems are:

- Sequence generation—based on key there are generated pseudo-random sequences which are used for projection hash to image. In case of video material generation is executed only once at the beginning of algorithm,
- Pre-processing—preprocessing computing e.g. buffers and variables initialization, image format conversion,
- Sequence projection—computing values of hash by multiplying (projection) image with generated pseudo-random sequences,
- Median computing—median computing for each independent block from values received from projection process,
- Value thresholding—thresholding of hash values based on computed median to receive binary sequence,
- Permutations—permutation of received hash for key generation,
- Key generation—process based on values received from earlier permutations,
- Creating watermark,
- Watermark embedding/detection,
- end processing—process of releasing allocated memory, format conversion etc.

In case of QIM implementation steps of creating watermark have not been executed because only resistant hash is embedded. In case of GPU implementation in OpenCL additional steps must be executed like a kernels creation (reading, building and creating final kernel object), data transfer between host and GPU's global memory and data structure initialization.

In the Table 1 there are presented results of running of spread spectrum algorithm with spatial domain on eight core cluster node. It describes the relation (linear dependency) between number of sequences used for generating watermark and time of protection/detection process. All results are presented for two different image sizes (1280×720 and 1920×1080 pixels). The protection is faster in both cases (spread spectrum method) than the detection process. The reason of that fact is additional time of mutual correlation computation which is need to detect watermark. Table 2 presents the times of each part of the algorithm in case of different numbers of generated sequences. Additionally we can notice strong linear dependence of sequence

Table 1 Time of protection and detection processes (ms)

Size of image	1280 × 720				1920 × 1080			
Number of sequences	16	32	50	64	16	32	50	64
Protection	82.64	135.57	193.79	240.89	179.67	280.32	429.09	527.36
Detection	114.01	192.49	242.08	292.89	267.28	358.57	553.83	648.03

Table 2 Time of execution of parts of the algorithm (ms)

Size of image	1280 × 720				1920 × 1080			
Number of sequences	16	32	50	64	16	32	50	64
Sequences generation	202.75	331.28	484.26	546.86	307.85	587.46	884.1	1017.81
Pre-processing	9.84	10.4	9.93	10.51	24.4	22.21	22.39	22.41
Sequences projection	10.53	20	30.97	39.76	18.9	38.2	63.22	80.35
Median computing	0.13	0.17	0.22	0.24	0.24	0.32	0.43	0.48
Tresh calculation	0.21	0.23	0.31	0.31	0.4	0.49	0.55	0.62
Creating watermark	42.47	84.86	133.94	169.68	97.18	192	294.45	385.25
Adding watermark	8.86	8.78	8.81	8.89	19.93	19.76	19.64	26.65
End processing	7.35	7.67	7.35	7.65	16.89	16.87	16.91	16.94

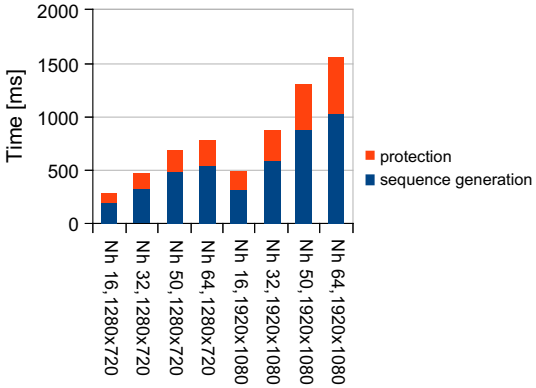

Fig. 3 The relation of protection and sequence generation times (ms)

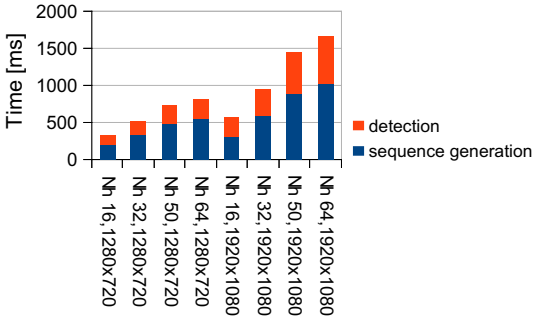

Fig. 4 The relation of detection and sequence generation times (ms)

projection, median computing, watermark creation and thresholding. In Table 2 we can also read percentage time execution of each step in whole algorithm. Therefore we can estimate which steps are most time consuming and should be taken in comparative studies between different algorithms. Figures 3 and 4 describe detection/protection times in relation with sequences generation. The process of sequence generation takes significant part of total algorithm time (more than 50 % of all the algorithm time). In

Table 3 The partial times of spread spectrum technique on 1280 × 720 image (ms)

Technique	Spread spectrum					
Domain	Spatial			DCT		
Implementation	Single core	Eight cores	OpenCL	Single core	Eight cores	OpenCL
Sequence generation	2242	546.86	544.4	2274	604.72	531.14
Kernels creation	–	–	1008	–	–	270.65
Preprocessing	10.17	10.51	19.74	10.15	10.21	15.21
Writing sequence	–	–	82.2	–	–	74.7
Writing image	–	–	0.7	–	–	0.67
Sequence projection	204.5	39.76	7.95	202.45	39.5	7.7
Median computing	1.18	0.24	0.6	1.19	0.26	0.6
Tresh computing	1.3	0.31	0.05	1.29	0.3	0.05
Permutation	–	–	0.51	–	–	0.48
Key creation	–	–	0.18	–	–	0.18
Watermark creation	1000	169.68	380.67	999.7	168.37	636.65
Adding watermark	51.38	8.89	0.29	94.11	15.99	19.8
Buffer reading	–	–	1.8	–	–	1.8
Realease OpenCL objects	–	–	0.3	–	–	0.67
End processing	7.2	7.65	7.68	7.28	7.35	7.37

further analysis number of sequences will be 64. This value is a trade-off between quality and security of embedding and detection of watermarks in the image.

Tables 3, 4, 5, and 6 present results of running four different algorithms (spread spectrum and QIM techniques in spatial and DCT domain) on images of size 1280×720 and 1920 × 1080 on single core, multicore and GPGPU platform, respectively. We can notice that each step execution time depends on technique and domain of the algorithm and hardware platform. The time of projection operation and watermark creation (executed only for spread spectrum algorithm) is constant and depends on hardware platform only. The watermark creation is fastest on the multicore platform, the projection process on the GPU platform.

Table 7 describe efficiency of protection and detection process in all configuration. As it can be seen the GPU implementation enables fastest execution of these steps. After adding additional steps (data transfer, write buffers etc.) to protection and detection time it turns out that OpenMP on the multicore platform gives the best results (faster than GPGPU). The worst case is while executing the algorithm on the single core (Figs. 5, 6).

In case of video input the influence of preprocessing (additional operation) can be minimalized because they are run only once at the beginning of the algorithm (it does not depend on number of frames). The following parameters were measured while running watermarking algorithms on video sources:

- Catching the frame,
- Protection/detection proces,

Table 4 The partial times of QIM technique on 1280 × 720 image [ms]

Technique	QIM					
Domain	Spatial			DCT		
Implementation	Single core	Eight cores	OpenCL	Single core	Eight cores	OpenCL
Sequence generation	2299	600.5	574.53	2276	571.48	556.16
Kernels creation	–	–	266.35	–	–	268.36
Preprocessing	10.2	9.9	11.37	10.37	10.2	13.61
Writing sequence	–	–	72.08	–	–	73.53
Writing image	–	–	0.62	–	–	0.69
Sequence projection	204.22	39.25	7.53	204.85	39.81	7.61
Median computing	1.2	0.26	0.62	1.2	0.26	0.62
Tresh computing	1.32	0.3	0.05	1.26	0.33	0.06
Permutation	–	–	–	–	–	–
Key creation	–	–	–	–	–	–
Watermark creation	–	–	–	–	–	–
Adding watermark	50.35	12.73	0.44	8.76	18.05	1.01
Buffer reading	–	–	1.95	–	–	2.14
Realease OpenCL objects	–	–	0.3	–	–	0.43
End processing	7.29	7.33	9.5	7.27	7.35	10.5

Table 5 The partial times of spread spectrum technique on 1920 × 1080 image (ms)

Technique	Spread spectrum					
Domain	Spatial			DCT		
Implementation	Single core	Eight cores	OpenCL	Single core	Eight cores	OpenCL
Sequence generation	4693	1026	1033	4718	1000	1038
Kernels creation	–	–	504.09	–	–	338.25
Preprocessing	22.03	22.39	22.27	22	22.26	32.54
Writing sequence	–	–	154.88	–	–	147.52
Writing image	–	–	0.88	–	–	0.89
Sequence projection	417.24	79.93	83.9	42	79.05	13.21
Median computing	2.68	0.5	0.55	2.51	0.49	0.82
Tresh computing	2.71	0.63	0.61	2.69	0.61	0.05
Permutation	–	–	0.34	–	–	0.33
Key creation	–	–	0.34	–	–	0.33
Watermark creation	2107	378.22	383.17	2111	380.13	636.7
Adding watermark	110.08	26.48	19.68	200.45	36.13	47
Buffer reading	–	–	3.49	–	–	3.42
Realease OpenCL objects	–	–	0.43	–	–	0.74
End processing	16.88	16.9	17.06	16.87	16.88	16.94

Table 6 The partial times of QIM technique on 1920 × 1080 image (ms)

Technique	QIM					
Domain	Spatial			DCT		
Implementation	Single core	Eight cores	OpenCL	Single core	Eight cores	OpenCL
Sequence generation	4676	1021.5	1001	4680	1029.5	993.3
Kernels creation	–	–	332.1	–	–	337.42
Preprocessing	22	22.2	32.74	22.02	22.41	24.62
Writing sequence	–	–	138.36	–	–	142.47
Writing image	–	–	0.98	–	–	0.91
Sequence projection	419.09	79.59	13.49	414.58	71.63	13.21
Median computing	2.45	0.55	0.82	2.46	0.48	0.82
Tresh computing	2.68	0.6	0.05	2.77	0.63	0.05
Permutation	–	–	–	–	–	–
Key creation	–	–	–	–	–	–
Watermark creation	–	–	–	–	–	–
Adding watermark	107.73	27.81	0.72	185.74	39.83	1.42
Buffer reading	–	–	3.43	–	–	3.43
Realease OpenCL objects	–	–	0.34	–	–	0.44
End processing	17.11	16.88	16.99	16.84	17.07	16.99

Table 7 The times of protection and detection processes with additional operations (ms)

Size of image	1280 × 720				1920 × 1080			
Algorithm	Spread spectrum		QIM		Spread spectrum		QIM	
Domain	Spatial	DCT	Spatial	DCT	Spatial	DCT	Spatial	DCT
Protection single-core	3539.65	3590.58	2358.64	2259.42	7381.03	7509	5263.87	5352.68
Protection muti-core	786.89	861.31	682.19	668.98	1572.91	1565.94	1183.63	1128.11
Protection GPGPU	2328.03	1590.14	978.18	954.25	2409.84	2301.34	1579.59	1546.94
Detection single-core	3726	3767.26	3621.08	2607.29	7847.03	7897	5354.73	5373.66
Detection-multi-core	825.24	916.34	697.17	673.59	1662.09	1667.35	1222.17	1118.55
Detection GPGPU	2315.19	1559.22	955.33	937.59	2401.65	2256.84	1546.12	1523.91

- Writing the results,
- Video creation (in case of protection process).

The whole system enables run described algorithms on several available nodes in a cluster. The algorithm can be run in multiple configurations. The first solution is the trial of parallelization the GPU implementation by running the algorithm on two graphic cards controlled by independent cores on a single node. The example results of such solution is described in Table 12. The second proposed solution is to execute the algorithm using multiple cluster nodes. It can be realized by MPI API. The one node is a master divides and broadcast the images and videos to other nodes. Then all nodes

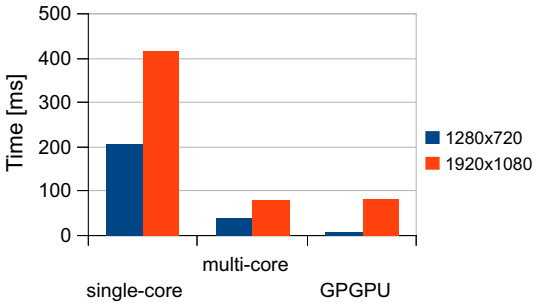

Fig. 5 The comparison of the sequence generation times on each hardware platform

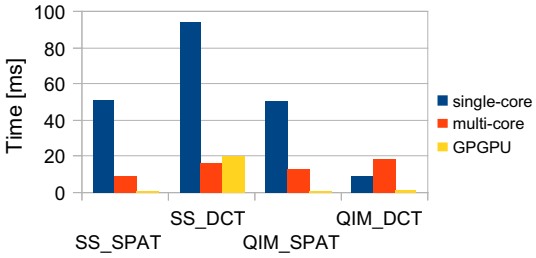

Fig. 6 The comparison of the sequence generation times in each domain

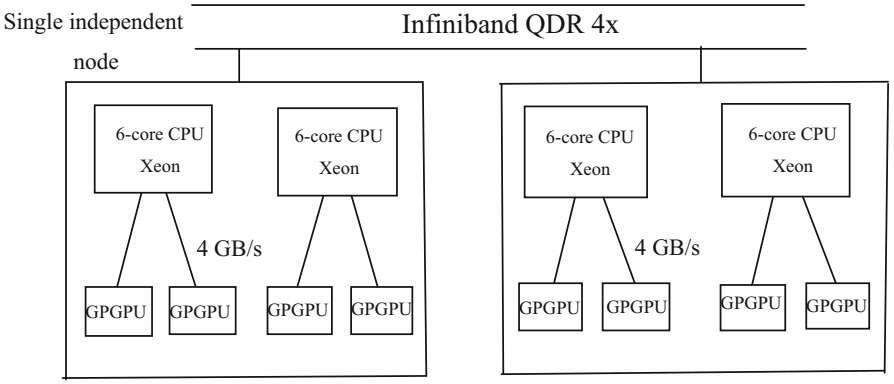

Fig. 7 Architecture of cluster

execute algorithm write the results and send to the master node (scatter and gather function are used). The transfer time and initializations operations in MPI increases significantly the total time because of size of image and video data. The example transfer times are presented in Table 11. The other way is to prepare input data for each node in its local memory. Then the achieved speed up of running the algorithm were linear. The delay of transferring the data before running the algorithm depends on hard disk parameters used in particular cluster system.

Watermarking algorithm were adapted and run on cluster system. The cluster consists of several computing nodes [21]. Each node has two six core CPUs. The single

Table 8 The times of protection and detection processes (ms)

Size of image	1280 × 720				1920 × 1080			
Algorithm	Spread spectrum		QIM		Spread spectrum		QIM	
Domain	Spatial	DCT	Spatial	DCT	Spatial	DCT	Spatial	DCT
Protection single-core	1287.48	1306.43	275.44	312.05	2666	2771	565.87	650.66
Protection muti-core	229.69	246.41	71.79	76.38	524.52	543.68	149.93	76.21
Protection GPGPU	672.03	699.14	54.18	43.25	709.84	751.34	75.59	69.94
Detection single-core	1473.83	1483.11	311.88	320.92	3132	3159	656.73	671.64
Detection-multi-core	268.04	301.44	86.77	80.99	613.7	645.09	188.47	66.65
Detection GPGPU	659.19	668.22	31.33	26.59	701.65	706.84	42.12	46.91

Table 9 The times of protection process of video sources [ms]

Size of image	Algorithm	Platform	Time of execution	Image catch	Image writing	Video creation
1280 × 720	SS_SPAT	Single-core	132.98	7.5	0.05	150.8
		Twelve-cores	30.38	1.2	0.05	35.7
		GPGPU	69.9	1.19	0.11	76.6
	SS_DCT	Single-core	137.275	7.57	0.04	146.5
		Twelve-cores	31.39	1.19	0.06	32.3
		GPGPU	71.81	1.2	0.11	72.4
	QIM_SPAT	Single-core	32.2	7.5	0.04	64.9
		Twelve-cores	14.13	1.25	0.05	17.8
		GPGPU	8.48	1.22	0.11	10.6
	QIM_DCT	Single-core	70.68	8.1	0.06	75.4
		Twelve-cores	16.4	1.24	0.05	18.3
		GPGPU	8.45	1.26	0.1	11.2
1920 × 1080	SS_SPAT	Single-core	277.29	13.3	0.25	320.8
		Twelve-cores	63.21	2.54	0.33	74.6
		GPGPU	77.01	2.56	0.23	83.8
	SS_DCT	Single-core	307.59	12.6	0.26	315.6
		Twelve-cores	67.43	2.55	0.32	65.6
		GPGPU	79.21	2.55	0.27	82.5
	QIM_SPAT	Single-core	68.37	12.89	0.26	315.6
		Twelve-cores	25.92	2.3	0.32	65.6
		GPGPU	16.88	2.54	0.27	82.5
	QIM_DCT	Single-core	75.13	13.3	0.26	315.6
		Twelve-cores	26.56	2.54	0.32	65.6
		GPGPU	16.98	2.56	0.27	82.5

Table 10 The times of detection process of video sources [ms]

Size of image	Algorithm	Platform	Time of execution	Image catch	Image writing	Video creation
1280 × 720	SS_SPAT	Single-core	148.66	7.5	0.05	150.8
		Twelve-cores	29.32	1.2	0.05	35.7
		GPGPU	67.15	1.19	0.11	76.6
	SS_DCT	Single-core	151.97	7.57	0.04	146.5
		Twelve-cores	31.39	1.19	0.06	32.3
		GPGPU	71.81	1.2	0.11	72.4
	QIM_SPAT	Single-core	32.55	7.5	0.04	64.9
		Twelve-cores	11.43	1.25	0.05	17.8
		GPGPU	4.61	1.22	0.11	10.6
	QIM_DCT	Single-core	31.82	8.1	0.06	75.4
		Twelve-cores	11.34	1.24	0.05	18.3
		GPGPU	4.31	1.26	0.1	11.2
1920 × 1080	SS_SPAT	Single-core	315.36	13.3	0.25	320.8
		Twelve-cores	65.12	2.54	0.33	74.6
		GPGPU	70.62	2.56	0.23	83.8
	SS_DCT	Single-core	320.83	12.6	0.26	315.6
		Twelve-cores	67.37	2.55	0.32	65.6
		GPGPU	72.84	2.55	0.27	82.5
	QIM_SPAT	Single-core	65.08	12.6	0.26	315.6
		Twelve-cores	18.46	2.55	0.32	65.6
		GPGPU	9.1	2.55	0.27	82.5
	QIM_DCT	Single-core	65.41	12.6	0.26	315.6
		Twelve-cores	18.99	2.55	0.32	65.6
		GPGPU	9.15	2.55	0.27	82.5

CPU core has access to 4 graphic cards which are available on each cluster node. Therefore single node enables to parallelize applications to be run in multicore environment with many core GPUs. In presented implementation each node is programmed by OpenMP environment. In first mode only CPU cores are used to make watermarking computations on independent input images and video sequences. The second solution is based on using graphic cards for accelerating protection and detection process of watermarking image data. Running application on more than four graphic cards or more than twelve cores MPI library must be used to enable communication between cluster nodes. One cluster node is chosen as a root node to coordinate and supervising other nodes. Each node receives input data and then start executing algorithm in a single process which is forked by OpenMP to given number of available core or to delegate the computations to graphic card connected to the node. Figure 7 describes process (root), which uses for executing the task maximal available resources on single node (CPU unit, run 4 independent watermarking computations on

Table 11 Data transfers measured on infiniband by MPI API

Size of data	Transfer time (ms)
10 MB	0.6
20 MB	1.1
100 MB	4.5
1000 MB	35

Table 12 Speedup the algorithm (Spread Spectrum SS_SPAT) run on two GPUs available in a cluster node

Size of data processed	Single GPGPU time (s)	Two GPGPUs time (s)
1 MB	12.1	7.5
2 MB	23.8	13.1
4 MB	46.8	24.2
8 MB	94.1	47.6

parallel cores which delegate algorithm to graphical cards available on single cluster node).

The cluster has peak performance about 136.7 TFlops. Each node consist of Intel Xeon 56XX family processors and NVIDIA Tesla M2050 and M2090 graphic cards. The bus between CPU and GPU enables 4 GB/s bandwidth. The interconnection between nodes cluster is based on Infiniband QDR 4x (peak about 40 GB/s).

6 Conclusions and Future Work

The possibilities of improving the performance of content authentication system using multiprocessor systems were investigated in the presented work. Two key parallelization technologies – OpenMP for CPU systems and OpenCL for GPU computing—have been used for implementation of redesigned hash-based watermarking method. Developed hybrid processing architecture aided by MPI protocol takes advantages of both processing domains and enables building advanced computing solution. Considered algorithm is based on semi-fragile watermarking and hash function in diverse embedding domains. The idea behind the proposed solution is to speed up video processing and maintain high embedding efficiency.

Execution time were examined both for digital images and video sequences. Performance of authentication system with computing parallelization was measured for single and multicore CPU platforms, general purpose graphic cards and heterogeneous cluster system. Analysis of results reveals that compared to single-core implementation in parallel processing environment a speed up of watermark algorithms in case of images can rise to over 30 times for protection (maximal speed up for QIM_SPAT) and over 5 times for detection can be achieved (Tables 7, 8). In case of video sources the achieved speed up is about 5–7 times for protection and detection processes (Tables 9, 10) (the best times measured in GPU). In addition a real-time processing is possible especially for QIM methods where the best speed up was achieved. Detailed inves-

tigation performed on algorithm steps helped to identify crucial parts of the whole processing chain. The results also enabled verification of the introduced algorithm modifications, that have contributed to reduction in processing times and to increasing the performance of the system. Future work is aimed at providing end-user authentication services together with efficient storage and distribution system. Further research will also concentrate on optimization of algorithms and theirs implementations in OpenCL with running them in other hardware accelerators like FPGA (Tables 11, 12).

The main benefit of our work was to show that watermarking algorithm as one of the video authentication method can be significantly speed up parallel hardware accelerators. Nowadays, GPU and multi-core processor are common in modern mobile devices and embedded systems. Therefore they can be used for real time image and videos authentication and authorization in many applications and devices like cameras, bank systems etc.

References

1. Haouzia, A., Noumeir, R.: Methods for image authentication: a survey. Multimedia Tools Appl. **39**(1), 1–46 (2008)
2. Fridrich, J.: Methods for tamper detection in digital images. In: Proceedings of the Multimedia and Security Workshop at ACM Multimedia (1999)
3. Wen, C.-Y., Yang, K.-T.: Image authentication for digital image evidence. Forensic Sci. J. (2006)
4. Zhu, B., Tewfik, A.H.: Low bit rate near-transparent image coding. In: Proceedings of SPIE (1995)
5. Fawad, A., Siyal, M.Y.: A secure and robust DCT-based hashing scheme for image authentication. 10th IEEE Singapore International Conference on Communucation Systems (2006)
6. Lin, C., Zhao, L., Yang, J.: A high performance image authentication algorithm on GPU with CUDA. Int. J. Intell. Syst. Appl. **3**(2), 52–59 (2011)
7. OpenCL—Open Computing Language. www.khronos.org/opencl (2014)
8. OpenMP—Open Multi-Processing. www.openmp.org (2014)
9. Fridrich, J., Goljan, M.: Robust hash functions for digital watermarking. International Conference on Information Technology: Coding and Computing (2000)
10. Dominguez-Conde, G., Comesa, P., Perez-Gonzalez, F.: Performance analysis of the Fridrich-Goljan self-embedding authentication method. 16th IEEE International Conference on Image Processing (2009)
11. Ahmed, N., Natarajan, T., Rao, K.R.: Discrete cosine transform. IEEE Trans. Comput. **23**(1), 90–93 (1974)
12. Nikolaidis, A., Pitas, I.: Asymptotically Optimal detection for additive watermarking in the DCT and DWT domains. IEEE Trans. Image Process. **12**(5), 563–571 (2003)
13. Zarnowiec, K., Korus, P., Dziech, A., Glowacz, A.: Practical implementation of visual hash functions for CCTV footage authentication. Commun. Comput. Inf. Sci. **368**, 309–323 (2013)
14. Chen, B., Wornell, G.: Quantization index modulation: a class of provably good methods for digital watermarking and information embedding. IEEE International Symposium on Information Theory (2000)
15. Open MPI: Open source high performance computing. www.open-mpi.org (2014)
16. Cox, I., Miller, M., Bloom, J., Fridrich, J., Kalker, T.: Digital Watermarking and Steganography, 2nd edn. Morgan Kaufmann, Burlington (2007)
17. Seitz, J.: Digital Watermarking For Digital Media. Information Science Publishing, Charlotte (2005)
18. Fridrich, J.: Visual Hash for Oblivious Watermarking, Security and Watermarking of Multimedia Contents II, Vol. 3971 (2000)
19. Cox, I. J., Kilian, J., Leighton, T., Shamoon, T.: Secure spread spectrum watermarking for images, audio and video. International Conference on Image Processing (1996)
20. Chen, B., Wornell, G.: Quantization index modulation: a class of provably good methods for digital watermarking and information embedding. IEEE International Symposium on Information Theory (2000)

21. Flynn, M. J.: Very high-speed computing systems. In: Proceedings of the IEEE (1966)
22. Nvidia, Nvidia CUDA C Programming Guide, version 4.2. www.nvidia.com
23. OpenCV—Open Source Computer Vision. www.opencv.willowgarage.com (2014)