DOI: 10.2352/ISSN.2470-1173.2016.8.MWSF-077

Accelerating the DCTR Features Extraction for JPEG Steganalysis Based on CUDA

Chao Xia, Qingxiao Guan, Xianfeng Zhao, Yong Deng
State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences,
Beijing 100093, China
{*xiachao,guanqingxiao,zhaoxianfeng,dengyong*}*@iie.ac.cn*

Abstract

Nowadays, steganalysis of JPEG images is increasingly popular because of their widespread usage. The DCTR set (Discrete Cosine Transform Residual) is a significant steganalysis feature set designed for the JPEG images. Its main advantage is its low computational complexity, while providing high accuracy in detection. However, it is desirable to further accelerate it, especially for some real-time applications. In this paper, we accelerate the DCTR features extraction on a GPU device, and some optimization methods are presented. Firstly, we utilize the separability and symmetry of the two-dimensional discrete cosine transform to enhance the computing efficiency in decompression and filtering images. Secondly, when computing phase-aware histograms, in order to achieve a good coalesced access and avoid the serious collisions between atomic operations, we add different offsets to the elements of the residual according to their positions, which implies the computation of phase-aware histograms can be converted into the computation of ordinary 256-dimensional histograms. By this means, we can fully exploit the GPU's parallelism. The experimental results show that the speed of our parallel method for images with different sizes is 150-200 times faster than the original serial method on our machine. Our method can also be applied to other phase-aware feature sets, such as PHARM (PHase Aware pRojection Mode).

Introduction

Steganography is to embed the secret information into cover objects without arousing a warder's suspicion. Steganalysis, the counterpart of steganography, aims to detect the presence of hidden data. The modern steganalysis paradigm is based on detectors trained with features extracted from cover and stego images. In early steganalysis, the dimensionality of feature spaces was low (about a few hundred) and hence there was no significant computational cost. With the increasingly more sophisticated steganographic schemes, the past decade has witnessed the requirement to build high-dimensional image models. However, the increased dimensionality of the feature space may result in problems with the curse of dimensionality, which restricted the development of useful high-dimensional feature models. Ensemble classifiers [5] using Fisher Linear Discriminants allow steganalysts to work with high-dimensional models and large data sets because of its low computational complexity. In [1], the 33963-dimensional HOLMES features are proposed, which achieve a detection of rate of 83.90% against HUGO[2] ($T = 90$), one of modern steganographic schemes, on BOSSbase at 0.4 bpp. In [3], a group of textural features of dimensionality 22153 are utilized to obtain an average detection rate of 83.92%. In [4], the Spatial Rich Model (SRM) improves the performance further in breaking HUGO. With 12753 features, the detection rate has been increased to 86.45% on HUGO with $T = 255$. Similarly, the steganalysis in JPEG domain also benefits from the framework based on high-dimensional models and ensemble classifiers. In [6], the systematically constructed 11255-dimensional JPEG Rich Model (JRM) and 22510-dimensional Cartesian-calibrated version outperform other low-dimensional feature sets, such as the 216-dimensional LIU[7] and the 548-dimensional CC-PEV[8]. In [9], the 8000-dimensional DCTR (Discrete Cosine Transform Residual) feature set is designed for JPEG steganalysis. The historical overview clearly shows that building high-dimensional feature models is a growing tendency to capture as many statistical dependencies among individual image elements as possible.

However, the high-dimensional feature set inevitably brings the increased time cost. Calculation of SRM features takes about 26 second for a 2000×3000 image on our machine. JRM features take about 83 second and DCTR features 35 second. The cost of using high-dimensional features makes them impractical for both researchers and practitioners[10]. Parallel computing is a feasible way to make high-dimensional features practicable in some real applications, such as on-line detection. Thus, much attention has been paid to implementing high-dimensional features on the GPU in recent years. In [10], Ker first adjusts the definition of the PSRM (Projected Spatial Rich Model)[11] and proposes the GPU-PSRM features with fixed-size kernels and fixed projections, and then optimizes their implementation on a GPU. It is found that the reduced time is significant and the cost of detection power is negligible. In [12], the authors employ some technologies including convolution unrolling, combined memory access and aversion of bank conflicts when accelerating the extraction of SRM features.

But so far, the high-dimensional features for JPEG Steganalysis have not been implemented on GPU hardware. It is necessary to compute these features in parallel due to the fact that there are numerous steganographic schemes specifically designed for the most common image format – JPEG. In [10], Ker argues that the trade-off between speed and accuracy would have to be adjusted considerably and it may be more valuable to implement

[0]This work was supported by the NSFC under 61170281, 61303259 and U1536105, and the Strategic Priority Research Program of Chinese Academy of Sciences under XDA06030600.

DOI: 10.2352/ISSN.2470-1173.2016.8.MWSF-077

much simpler features. DCTR is one of the most effective feature sets for JPEG steganalysis. With the dimensionality of 8000, DCTR provides about the same level of detection as the 22510-dimensional CC-JRM and even obtains a more accurate detection against J-UNIWARD. In this paper, we accelerate the extraction of DCTR on the GPU device in order to make it more efficient in practice.

The rest of this paper is organized as follows. After a brief review of DCTR in the next section, we describe the optimized implementation of DCTR on GPU. Then, we present experimental results. Conclusions are drawn in the final section.

The DCTR Features

The DCTR features are computed from pixels of the decompressed JPEG image. DCTR provides a far better detection performance than the early low-dimensional features and even obtains a more accurate detection than CC-JRM against J-UNIWARD. Before optimizing the implementation of DCTR on GPU hardware, we briefly describe how the DCTR features are built to make this paper self-contained. For simplicity and clarity, we do not go into details and the detailed information can be seen in the original publication [9]. Given that most of steganographic schemes embed secret messages in the luminance component, without loss of generality, we only discuss grayscale JPEG images in this paper. DCT coefficients of an $M \times N$ JPEG image will be denoted as a matrix $\mathbf{D} \in \mathbb{Z}^{M \times N}$. We will also assume that both M and N are multiples of 8. Let $D_{ij}^{x,y}$ denote the (i,j)th DCT coefficient in the (x,y)th 8×8 block, $0 \leqslant i,j \leqslant 7$, $x = 1,\ldots,M/8$, $y = 1,\ldots,N/8$.

(1) Decompression

The JPEG image is first decompressed to the spatial domain without quantizing the pixel values to $\{0,\ldots,255\}$ to avoid any loss of information. The decompressed JPEG image $\mathbf{X} \in \mathbb{R}^{M \times N}$. Let $X_{kl}^{x,y}$ denote the (k,l)th pixel in the (x,y)th 8×8 block in the spatial domain, $0 \leqslant k,l \leqslant 7$:

$$X_{kl}^{x,y} = \sum_{i,j=0}^{7} \frac{w_i w_j}{4} \cos\frac{\pi}{16} i(2k+1) \cos\frac{\pi}{16} j(2l+1) D_{ij}^{x,y}, \quad (1)$$

where $w_0 = 1/\sqrt{2}$, $w_i = 1$ for $i > 0$.

(2) Computing Residuals

We can obtain residuals by computing 64 convolutions of the decompressed JPEG image with 64 8×8 DCT basis patterns $\mathbf{B}^{(a,b)}$, $0 \leqslant a,b \leqslant 7$:

$$\mathbf{U}^{(a,b)} = \mathbf{X} \star \mathbf{B}^{(a,b)}, \quad (2)$$

where $\mathbf{U}^{(a,b)} \in \mathbb{R}^{(M-7)\times(N-7)}$ and $'\star'$ denotes a convolution without padding. $\mathbf{B}^{(a,b)} = \left(B_{mn}^{(a,b)}\right), 0 \leqslant m,n \leqslant 7$:

$$B_{mn}^{(a,b)} = \frac{w_a w_b}{4} \cos\frac{\pi}{16} a(2m+1) \cos\frac{\pi}{16} b(2n+1), \quad (3)$$

and $w_0 = 1/\sqrt{2}$, $w_a = 1$ $(a > 0)$. These 64 residual images can capture different types of dependencies among pixels of the decompressed JPEG.

(3) Truncation and Quantization

We first take the absolute values of all elements in the residual and then form a quantized and truncated residual image $\mathbf{U}^{(a,b)}$, $0 \leqslant a,b \leqslant 7$:

$$\mathbf{U}^{(a,b)} = \mathrm{trunc}_T \left(\mathrm{round} \left(\frac{\left|\mathbf{U}^{(a,b)}\right|}{q} \right) \right), \quad (4)$$

where $0 \leq a,b \leq 7$, $T = 4$ is an integer threshold and q is a quantization step which can be described as:

$$q = \begin{cases} \min\left\{8 \times \left(\frac{50}{QF}\right), 100\right\} & QF < 50 \\ \max\left\{8 \times \left(2 - \frac{50}{QF}\right), 0.2\right\} & QF \geq 50 \end{cases}, \quad (5)$$

where QF is the JPEG quality factor. The purpose of truncation with a small T is to give the features a lower dimension and keep the feature vector more populated.

(3) Histogram Features Extraction

After quantization and truncation, each residual image $\mathbf{U}^{(a,b)}$ is subsampled by step 8 to get 64 subimages. This is the so-called phaes-awareness which contributes to better detection performance because in a JPEG decompressed image the pixels' statistical properties depend on their positions with respect to the 8×8 grid. To enhance the features' diversity and reduce the feature dimensionality, the 64 histogram features of a residual image are merged into 25 according to symmetry of projection vectors, which gives the DCTR set the dimensionality of $64 \times 25 \times 5 = 8000$.

Implementing DCTR on The GPU

The flow diagram of the implementation is shown in Figure 1. Due to the Huffman decoding, it is fitted to read a JPEG file on the CPU. Then the image data is transferred from the host to the GPU device via PCI Express. Quantization and truncation are natural to turn to a GPU implementation and therefore we do not describe them in details. When implementing the decompression and convolution on the GPU, we leverage the property of the two-dimensional DCT to increase the productivity. When computing histograms, we can obtain dozens of 5-dimensional histograms in a batch from a 256-dimensional histogram to remove some of the difficulties in computing phase-aware histograms in parallel. In the following, the further description is given for these optimization methods.

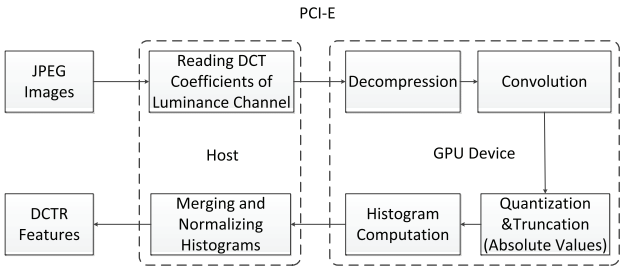

Figure 1. *The flow diagram of the implementation of DCTR on the GPU.*

DOI: 10.2352/ISSN.2470-1173.2016.8.MWSF-077

IDCT and Convolution

Because of the independence among different DCT data blocks, it is possible to perform IDCT on the GPU to take full advantage of its huge arithmetic capability. Due to the property of separability, two-dimensional DCT transform can be computed in two steps by performing successively a one-dimensional DCT on the columns and a one-dimensional on the rows[13]. The argument can be identically applied to inverse discrete cosine transform. Therefore, IDCT can be describe as $\mathbf{X} = \mathbf{A}^T\mathbf{D}\mathbf{A}$, where $\mathbf{D}$ is denoted as an inverse quantized 8×8 DCT block and $\mathbf{A}$ can be viewed as a two-dimensional array containing values of one-dimensional cosine basis functions one per column[14]:

$$\mathbf{A}^T = \begin{bmatrix} s_0 & s_0 & s_0 & s_0 & s_0 & s_0 & s_0 & s_0 \\ s_1 & s_3 & s_4 & s_6 & -s_6 & -s_4 & -s_3 & -s_1 \\ s_2 & s_5 & -s_5 & -s_2 & -s_2 & -s_5 & s_5 & s_2 \\ s_3 & -s_6 & -s_1 & -s_4 & s_4 & s_1 & s_6 & -s_3 \\ s_0 & -s_0 & -s_0 & s_0 & s_0 & -s_0 & -s_0 & s_0 \\ s_4 & -s_1 & s_6 & s_3 & -s_3 & -s_6 & s_1 & -s_4 \\ s_5 & -s_2 & s_2 & -s_5 & -s_5 & s_2 & -s_2 & s_5 \\ s_6 & -s_4 & s_3 & -s_1 & s_1 & -s_3 & s_4 & -s_6 \end{bmatrix}, \tag{6}$$

where $\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} \cos\frac{\pi}{4} \\ \cos\frac{\pi}{16} \\ \cos\frac{\pi}{8} \\ \cos\frac{3\pi}{16} \\ \cos\frac{5\pi}{16} \\ \cos\frac{3\pi}{8} \\ \cos\frac{7\pi}{16} \end{bmatrix}$.

We further utilize the symmetry of $\mathbf{A}^T$ to reduce calculation amount. Thus, $\mathbf{Y} = \mathbf{A}^T\mathbf{D}$ can be expressed as:

$$\begin{bmatrix} y(0) \\ y(2) \\ y(4) \\ y(6) \end{bmatrix} = \begin{bmatrix} s_0 & s_0 & s_0 & s_0 \\ s_2 & s_5 & -s_5 & -s_2 \\ s_0 & -s_0 & -s_0 & s_0 \\ s_5 & -s_2 & s_2 & -s_5 \end{bmatrix} \begin{bmatrix} x(0)+x(7) \\ x(1)+x(6) \\ x(2)+x(5) \\ x(3)+x(4) \end{bmatrix}$$
$$\begin{bmatrix} y(1) \\ y(3) \\ y(5) \\ y(7) \end{bmatrix} = \begin{bmatrix} s_1 & -s_3 & s_4 & -s_6 \\ s_3 & s_6 & -s_1 & s_4 \\ s_4 & s_1 & s_6 & -s_3 \\ s_6 & s_4 & s_3 & s_1 \end{bmatrix} \begin{bmatrix} x(0)-x(7) \\ -x(1)+x(6) \\ x(2)-x(5) \\ -x(3)+x(4) \end{bmatrix}. \tag{7}$$

When the JPEG image is being decompressed to the spatial domain, each thread performs the whole one-dimensional 8-tap IDCT. Hence, 8 threads are needed for a single 8×8 DCT block. As shown in Figure 2, an image is divided into a number of macroblocks containing $2 \times 4 = 8$ DCT blocks. In this case, the number of threads in a CUDA block is $2 \times 4 \times 8 = 64$, a multiple of 32 (the number of threads within the warp), which renders GPU work efficiently. We need to load the macroblock into shared memory because it has much higher bandwidth and much lower latency than global memory [15].

After decompression, the next step is to compute convolutions with 64 DCT kernels. Since the DCT kernel is separable, the two-dimensional convolution can be split in two steps[16]. First, convolve each row in the decompressed JPEG image with one row in $\mathbf{A}^T$ $(\mathbf{A}^T(a,:), 1 \leqslant a \leqslant 8)$ in Equation 7, resulting in an intermediate image. Next, convolve each column of this intermediate image with one column in $\mathbf{A}$ $(\mathbf{A}(:,b), 1 \leqslant b \leqslant 8)$ in Equation 7 to obtain the residual $\mathbf{U}^{(a,b)}$.

While computing the convolution with a row in $\mathbf{A}^T$, the decompressed JPEG image is split into some macroblocks of size 8×64. Figure 3 shows there is an apron of pixels (the shadow area) that is required to filter the image block. Here, eight one-dimensional convolutions are performed by a thread and thus each CUDA block runs 64 threads across a macroblock. When computing the convolution of the intermediate image with a column in $\mathbf{A}$, the image is analogically split into some macroblocks of size 64×8. The shared memory is also utilized in this process.

Phase-Aware Histogram Features

Given that the phase-aware features offer a more accurate detection, each residual image is subsampled to form 64 subimages based on their positions with respect to the 8×8 pixel grid. The histogram features of subimages then are combined to form the final DCTR features. However, phase-aware histogram features are challenging to compute efficiently on the GPU. This has two causes. Firstly, because each residual image is subsampled by step 8, the consecutive threads fail to access the neighboring elements of the residual image in global memory. This greatly reduces the efficiency of data access. What is more, although atomic operations provided by CUDA are the only possible way to implement histograms on parallel architectures, serious collisions between atomic operations may occur when computing 5-dimensional histograms (using 5 bins) of subimages. The collisions could render some threads waiting, which blocks GPU's effective productivity. The less bins mean that the threads are more likely to increment the same bin simultaneously. Thus, using more bins is a feasible way to improve the parallelism of histogram computation. This paper proposes a novel method to compute the phase-aware histogram features. Instead of directly computing 5-dimensional histograms of the subimages, we first compute a high-dimensional

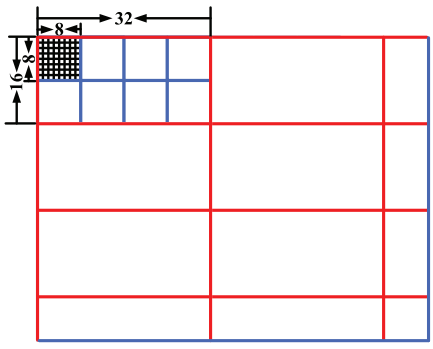

Figure 2. *When computing IDCT, a image is split into macroblocks and each macroblock contains 8 DCT blocks.*

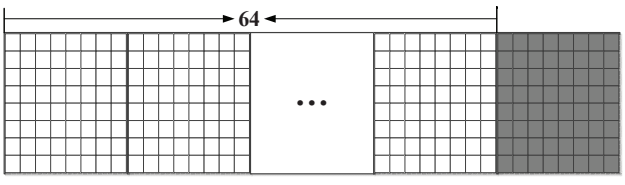

Figure 3. *When computing the convolution with a row vector, each macroblock contains* 8×64 *elements.*

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.8.MWSF-077

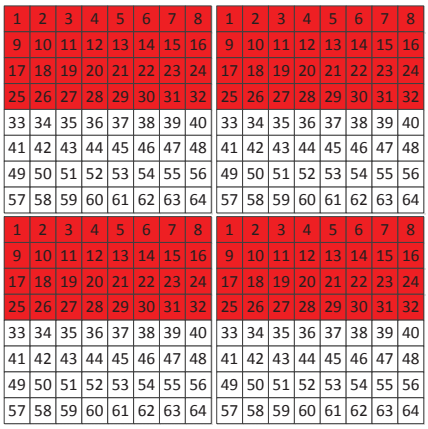

Figure 4. *The 64 subimages according to their positions within the* 8×8 *grid.*

histogram of the whole residual image whose pixel values have been truncated by a larger threshold, and then split it into low-dimensional histograms. Our method not only achieves a good coalesced access but also avoids the serious atomic collisions.

For the convenience of the following description, although we do not subsample the residual images, we still index 64 subimages with $I^1_{sub}, I^2_{sub}, ..., I^{64}_{sub}$ according to the positions of their elements within each 8×8 grid, as Figure 4 shows. In fact, our scheme of calculating phare-aware histograms is carried out twice respectively for histograms of $\{I^1_{sub}, I^2_{sub} ..., I^{32}_{sub}\}$(the red area in Figure 4) and $\{I^{33}_{sub}, I^{34}_{sub} ..., I^{64}_{sub}\}$(the white area in Figure 4). The detailed procedures are described as follows:

Step 1: After quantizing the absolute values of the filtered images, truncation threshold is set $T = 7$, instead of $T = 4$ in [9], thus the absolute value of each element belongs to $\{0, 1, ..., 7\}$. Although the threshold is changed here, we do not increase the dimensionality of DCTR. And in Step 3, we will show that after a combining operation, the final result of our scheme is unchanged as before.

Step 2: After truncation, we first deal with the elements corresponding to the first 32 subimages in the truncated image. For $\{I^1_{sub}, I^2_{sub} ..., I^{32}_{sub}\}$, elements of different subimages are added with different offset values. Figure 5 shows that the offset value depends on the position of elements in the truncated image and we denote it as $\varphi(k,l)$. More precisely, $\varphi(k,l)$ can be written as $\varphi(k,l) = (l \bmod 4) \times 64 + (k \bmod 8) \times 8$. This makes the values of elements of subimages $I^1_{sub}, I^2_{sub} ..., I^{32}_{sub}$ respectively fall into 32 non-overlapping ranges $\{0, 1, ..., 7\}$, $\{8, 9, ..., 15\}$, $\{16, 17, ..., 23\}$, ... , $\{248, 249, ..., 255\}$. As shown in Figure 6, we successfully obtain the histogram features of the first 32 subimages by constructing a 256-dimensional histogram ($8 \times 32 = 256$). Note that the elements of the first 32 subimages are aligned consecutively in rows, which can avoid reading inconsecutive locations in global memory. And further more, after adding offset values, 32 elements in each 8×8 grid belong to different ranges, which dramatically reduces the collisions when computing the histograms.

Step 3: As depicted in Figure 7, the 256-dimensional histogram can be spilt into 32 8-dimensional histograms, and then the last four bins in each 8-dimensional histogram are merged into one in order to give the histogram of each subimage the dimensionality of 5. This 5-dimensional histogram equals to the 5-dimensional histogram calculated with threshold $T = 4$, which is consistent with the original DCTR in [9].

Step 4: Similarly, for elements of the last 32 subimages $\{I^{33}_{sub}, I^{34}_{sub} ..., I^{64}_{sub}\}$, Step 2 and Step 3 are repeated again. The 5-dimensional histograms corresponding to the last 32 subimages

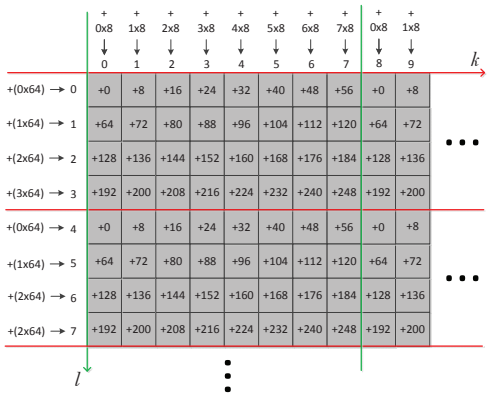

Figure 5. *The offset according to the position within the* 8×8 *grid.*

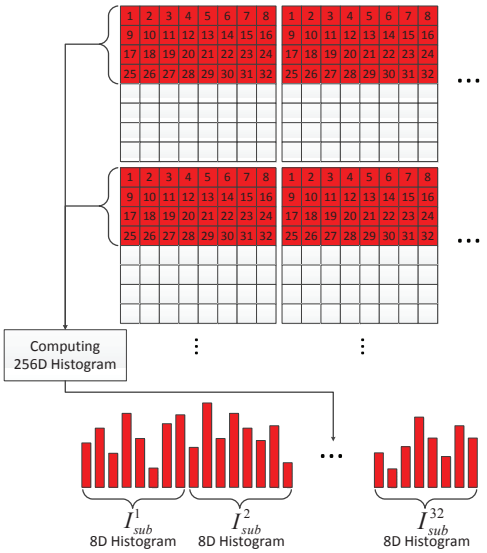

Figure 6. *Computing the histograms of the first 32 subimages.*

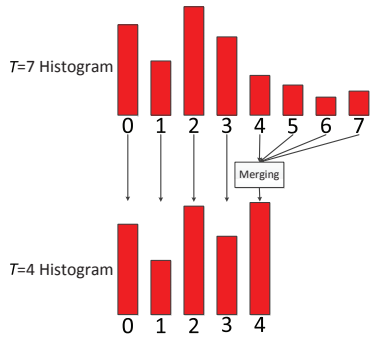

Figure 7. *Merging the last four bins into one.*

©2016 Society for Imaging Science and Technology
DOI: 10.2352/ISSN.2470-1173.2016.8.MWSF-077

can also be computed.

Step 5: According to the method in literature [9], all the histograms calculated in Step 3 and Step 4 are merged and normalized to obtain the DCTR set of dimensionality 8000.

To trade off the parallelism against shared memory exhaustion, a residual image is divided into a number of macroblocks. Each macroblock contains 4×32 pixels. Each thread is responsible to count the value of a pixel, so there are $4 \times 32 = 128$ threads in a CUDA block. Step 1 $\sim$ Step 4 are implemented on GPU device. But Step 5 is done using CPU programming, because it is trivial.

Experiments

Our implementation of DCTR is in CUDA, which is an extension of the C language. Therefore, we call the proposed implementation CUDA-DCTR.

In this section, we measure the speed of CUDA-DCTR extraction with respect to existing implementations. The configurations of our computer and the GPUs used in test are shown in Table 1 and Table 2. When implementing CUDA-DCTR, only one GPU is used at a time. Our experiment uses three kinds of images with different sizes, including 512×512, 2000×3000 and 3744×5616, and 200 images for each size. The images of size 512×512 and 3000×2000 are randomly selected from BOSSBase which has become a standard image database in steganalysis. The images of size 3744×5616 are from our image set (captured by Canon EOS 5D Mark II). These images are used to test the computation time of various implementations, including sigle-thread MATLAB implementation, multi-thread MATLAB version, CUDA-DCTR on GTX 780 and CUDA-DCTR on GTX 980. The MATLAB implementations are available online `http://dde.binghamton.edu/download/feature_extractors/`.

Experimental Platform

CPU	Intel(R)Xeon(R) E5-1620
CPU Frequency	3.7GHz
Memory	16G
OS	Windows 7 SP1 x64

GPU Specifications

GPU	NVIDIA GeForce GTX 780	NVIDIA GeForce GTX 980
Global Memory	3072 MB	4096 MB
Shared Memory per Block	49152 bytes	49152 bytes
CUDA Cores	2304	2048
GPU Clock rate	941 MHz	1279 MHz
FLOPS	4.0 TFLOPs	4.6 TFLOPs
Memory Bus	384 bit	256 bit

From Table 3, it can be seen that in the CUDA-DCTR implementation we have fully exploited the parallelism of the GPU. With only a single GPU, the speed is about 150$\sim$200 times than the single-thread MATLAB implementation and 55$\sim$80 times than the multi-thread version on a wallclock basis. The large-size images have a better speedup than the small-size, because large-size images can be split into more macroblocks for parallel processing. In addition to being fast, the low standard deviation of our GPU implementation suggests that our CUDA-DCTR runs steadily.

Table 4 shows the time consumed by step 1 and step 2 in DCTR. In the implementation of DCTR, step 1 includes decompression, residual computation, truncation and quantization; step 2 is histogram computation.

Compared to the sigle-thread MATLAB implementation, the time for step 1 of CUDA-DCTR is reduced by three to four orders of magnitude. This is because that the convolution is suited for parallel computing and fully utilizes the GPU's arithmetic capability. The computation time MATLAB spends on step 2 is 60$\sim$90 times than CUDA-DCTR. This indicates that our method successfully makes the phase-aware features favorable for parallel processing.

Time of DCTR extraction for various implementations

	Image Size	Average Time(s)	STDEV (s)	Min Time(s)	Max Time(s)
CUDA-DCTR GTX 980	Size1	0.010	0.001	0.008	0.012
	Size2	0.154	0.004	0.144	0.167
	Size3	0.528	0.009	0.510	0.557
CUDA-DCTR GTX 780	Size1	0.010	0.001	0.009	0.012
	Size2	0.134	0.004	0.127	0.145
	Size3	0.448	0.007	0.436	0.469
MATLAB single-thread	Size1	1.562	0.059	1.422	1.703
	Size2	30.990	1.175	28.435	33.988
	Size3	105.580	2.441	100.729	114.632
MATLAB 4-thread	Size1	2.300	0.130	1.855	2.836
	Size2	47.841	2.247	39.096	50.981
	Size3	169.916	6.651	139.110	179.231

Size1 = 512×512, Size2 = 2000×3000 and Size3 = 3744×5616.

Time of two steps in DCTR

	Image Size	STEP1(s)	STEP2(s)
CUDA-DCTR GTX 980	512×512	0.001	0.005
	3000×2000	0.002	0.112
	5616×3744	0.004	0.402
MATLAB single-thread	512×512	1.076	0.483
	3000×2000	23.752	7.232
	5616×3744	80.949	24.626

Conclution

With the advent of the high-dimensional features, increasingly more attention has been paid to the GPU implementation of steganalysis features. But there has been little research on how to implement JPEG domain high-dimensional features on the GPU. In this paper, we accelerate the DCTR features extraction based on CUDA and GPU execution. To this end, some optimization methods have been proposed. We first utilize the separability and symmetry of the two-dimensional discrete cosine transform to

DOI: 10.2352/ISSN.2470-1173.2016.8.MWSF-077

simplify the calculation of decompression and convolution. When computing the phase-aware histograms, we encounter two problems – serious atomic collisions and inefficient reading of images from global memory. To address these two problems, we specifically design a novel method to calculate phase-aware histograms, which can efficiently calculate histograms of subimages without subsampling. The experimental results show that the proposed method can greatly accelerate the DCTR features extraction, especially for images of large size. The method can also be applied to other phase-aware features, such as PHARM[17] and GFR[18]. For steganalysis in the spatial domain, SRM is a significant feature set. Therefore, we will further optimize the implementation of SRM on GPU in future work to make it more usable in practice. To further decrease the time needed to extract the features, we will investigate the performance of the multi-GPU system in future research.

References

[1] Jessica Fridrich, Jan Kodovský, Vojtěch Holub, Miroslav Goljan, Steganalysis of Content-Adaptive Steganography in Spatial Domain, Proc. IH, pg. 102. (2011).

[2] Tomáš Pevný, Tomáš Filler, Patrick Bas, Using High-Dimensional Image Models to Perform Highly Undetectable Steganography, Proc. IH, pg. 161. (2010).

[3] YunQ. Shi, Patchara Sutthiwan, Licong Chen, Textural Features for Steganalysis, Proc. IH, pg. 63. (2013).

[4] Jessica Fridrich, Jan Kodovský, Rich Models for Steganalysis of Digital Images, IEEE Trans. Inf. Forensics Secur., 7, 3 (2012).

[5] Jan Kodovský, Jessica Fridrich, Vojtěch Holub, Ensemble Classifiers for Steganalysis of Digital Media, IEEE Trans. Inf. Forensics Secur., 7, 2 (2012).

[6] Jan Kodovský, Jessica Fridrich, Steganalysis of JPEG Images Using Rich Models, Proc. SPIE, pg. 83030A. (2012).

[7] Qingzhong Liu, Steganalysis of DCT-embedding Based Adaptive Steganography and YASS, Proc. MMSec, pg. 77. (2011).

[8] Tomáš Pevný, Jessica Fridrich, Merging Markov and DCT Features for Multi-Class JPEG Steganalysis, Proc. SPIE, pg. 650503. (2007).

[9] Vojtěch Holub, Jessica Fridrich, Low Complexity Features for JPEG Steganalysis Using Undecimated DCT, IEEE Trans. Inf. Forensics Secur., 10, 2 (2015).

[10] Andrew D. Ker, Implementing the Projected Spatial Rich Features on a GPU, Proc. SPIE, pg. 90280K. (2014).

[11] Vojtěch Holub, Jessica Fridrich, Tomáš Denemark, Projections of Residuals as An Alternative to Co-Occurrences in Steganalysis, Proc. SPIE, pg. 86650L. (2013).

[12] Kaizhi Chen, Chenjun Lin, Shangping Zhong, Longkun Guo, A Parallel SRM Feature Extraction Algorithm for Steganalysis Based on GPU Architecture, Proc. PAAP, pg. 178. (2014).

[13] Syed Ali Khayam, The Discrete Cosine Transform (DCT): Theory and Application, Texts in Computer Science 41, 1 (2003).

[14] Anton Obukhov, Alexander Kharlamov, Discrete Cosine Transform for 8×8 Blocks with CUDA, Nvidia White Paper (2008).

[15] NVIDIA Corporation, CUDA C Programming Guide v7.5, 2015, pg. 82.

[16] Steven W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing, CA, 1997, pg. 404.

[17] Vojtěch Holub, Jessica Fridrich, Phase-Aware Projection Model for Steganalysis of JPEG Images, Proc. SPIE, pg. 94090T. (2015).

[18] Xiaofeng Song, Fenlin Liu, Chunfang Yang, Xiangyang Luo, Yi Zhang, Steganalysis of Adaptive JPEG Steganography Using 2D Gabor Filters, Proc. IH, pg. 15. (2015).