Computer Science and Information Systems 12(4):1345–1359 DOI: 10.2298/CSIS140815058C

A Parallel SRM Feature Extraction Algorithm for Steganalysis Based on GPU Architecture

Kaizhi Chen[1,2], Chenjun Lin[1], Shangping Zhong[1,2], and Longkun Guo[1]

[1] College of Mathematics and Computer Science, Fuzhou University
Fuzhou, 350108
{ckz, N110320080, spzhong, lkguo}@fzu.edu.cn
[2] Fujian Provincial Key Laboratory of Networking Computing and Intelligent Information Processing
Fuzhou, 350108

Abstract. The Spatial Rich Model (SRM) generates powerful steganalysis features, but has high computational complexity since it requires calculating tens of thousands of convolutions with image noise residuals. Practical applications dealing with a massive amount of image transferred through the Internet would suffer a long computing time if using CPU. To accelerate the steganalysis, we present a parallel SRM feature extraction algorithm based on GPU architecture. We exploit parallelism of the algorithm, modify the original SRM extraction algorithm and employ some strategies to avoid the disadvantage of its sequentiality. Some OpenCL optimization technologies are also used to accelerate the extraction process, such as convolution unrolling, combined memory access, split-merge strategy for co-occurrence matrix calculation. The experimental results show that the speed of the proposed parallel extraction algorithm for different size images is 25~55 times faster than the original single thread algorithm. In addition, when using AMD GPU HD 6850, our algorithm runs 2~4.2 times faster than using a Intel Quad-core CPU. This indicates our algorithm makes good use of the GPU cores.

Keywords: Parallel computing, Steganalysis, SRM feature, OpenCL.

1. Introduction

Steganography is an information security technology by hiding the secret data in multi-media data without perceptible modifications to be unnoticed to the third eye 1. It has been used for digital watermarking, Data Rights Management (DRM), and covered communications, etc. However, protecting data via covered communications are also used in crime, or even terrorism 23. Therefore, it is also necessary to develop inverse algorithmic techniques, such as steganalysis methods, to discover covered data in order to detect hidden information. Since images are the most used material in covered communications, steganalysis for images is a valuable theme.

The modern steganalysis paradigm applies machine learning techniques for detecting stego-images based on features extracted from each image. Early features were almost simple and with low dimensions. However, with the increased sophistication of steganographic algorithms, such as Highly Undetectable steGO (HUGO) 4, the

dimensionality of feature extracted from image for steganalysis increases greatly for improving detection rates, e.g. 24993-dimensional Higher-Order Local Model Estimators of Steganographic (HOLMES) feature 5, 34761-dimensional Spatial Rich Model (SRM) feature 6. High-dimensional feature brings detection performance improvements, but also leads to great computational cost that drops the speed performance. Actually, feature extraction takes most of the computational complexity in a modern steganalysis system. Paper 7 gives the time consumed by some popular steganalysis algorithms executed in single-thread. For example, the implementation of symmetrized co-occurrence features $CF*$ takes about 1.3 seconds for a 1Mpix image on author's benchmark machine 8; SRM features take about 12s, and JPEG Rich Model (JRM) features 36s 9. Calculation of Projected Spatial Rich Model (PSRM) features, for a single 1Mpix image, takes 20–30 minutes 1011. So when the steganalysis is carried out against massive images, a fast feature extraction is critical to boost the entire steganalysis system. So decreasing processing times is nowadays necessary due to the large number of images transferred through the Internet.

For high computational complexity in the theme of multimedia processing, parallel computing is a good way to improve the performance. In that, the first way is to use large-scale cluster computer servers, which, however, always suffers the cost of million dollars to build the infrastructure. Another low-cost way is to use GPU for parallel computing. In recent years, universal parallel computing technology on GPU develops rapidly. GPU has overwhelming superiority to CPU in the capability of floating point computing and memory bandwidth. Such as the latest NVIDIA Tesla K20X 12, it has 2688 CUDA cores, 1.31T flops (double-precision), 3.95T flops (single-precision), and 250 GB/s memory bandwidth. The price of computing card with K20X core is just \$3k~4k, that is much cheaper than the super computer (large-scale cluster computer servers). Therefore, GPU provides a feasible solution for the large-scale data computing with low cost, and more and more applications have used GPU as a co-processor of CPU to accelerate the calculation.

2. Related Work

There exist many literature giving successful examples in the theme of image parallel processing on GPU 131415. The first of them worth to address are steganalytic hardware implementations. Paper 16 developed an FPGA-based architecture for the RS algorithm, a specific steganalysis method proposed by Fridrich et al. 17 which recognizes LSB (Least Significant Bit). The proposed architecture uses a three stage pipeline and was synthesized in a Xilinx Virtex II FPGA. In 2013, Gutierrez-Fernandez et al. introduced an FPGA-based architecture for transform domain universal steganalysis in JPEG images 18. This architecture is based on JPEG's compatibility algorithm proposed by Fridrich et al. 19. Authors proposed a pipeline scheme implemented in VHDL and synthesized in a Xilinx Virtex 6 FPGA. In 2014, Tiwary et al. proposed faster and intelligent steganography detection, and used Graphics Processing Unit in cloud for faster operations 20. In 2015, Rodriguez-Perez et al. accelerated the Subtractive Pixel Adjacency Model (SPAM) model 22 calculation on the CUDA architecture 21. In 2014, Andrew developed a GPU-based architecture for an

implementation of the PSRM features 7. This PSRM feature is only marginally more powerful than SRM, but consumes far larger computation than SRM. The author also suggests it may be more valuable to settle for optimized of SRM or JRM. Moreover, to the best of the authors' knowledge, there is not any literature about SRM features parallel extraction algorithm on GPU. Therefore, we try to develop GPU-based architecture for SRM extraction (*GPU-SRM*).

In this paper, we shall first analyze the single thread SRM extraction algorithm proposed by Fridrich, and then accordingly design a parallel algorithm for SRM feature extraction using the parallel program framework of OpenCL based on GPU architecture. At last, we shall give experimental results to show our proposed algorithm speed up the SRM feature extraction well.

3. Parallelism of SRM Feature Extraction

In this section, we exploit parallelism of SRM Feature Extraction. SRM features is proposed by Fridrich and Kodovsky in 2012 [6], and arises from applying the rich model to extract the spatial domain information of the images for steganalysis 6. It is still one of the state-of-the-art steganalysis features to the best of our knowledge. In Fridrich and Kodovsky's work, the feature extraction is composed of many sub-models of rich model, each of which are used to compute a co-occurrence matrix for images by going through three processes: 1) Computing Residuals; 2) truncation & quantization; 3) co-occurrence matrix calculation. The difference of each sub-model mainly exists in the model of residual computing and the quantization parameter. The computed matrixes can be merged into a final SRM feature set for steganalysis.

3.1. Step 1: Residuals Computing

Noise residuals computing is essentially a convolution process used a high-pass filter. Different sub-model corresponds to different filter coefficients. The author defines 19 filters (sub models). Noise residual is generated by the following formula:

$$R_{ij} = \hat{X}_{ij}\left(N_{ij}\right) - cX_{ij} \tag{1}$$

where $R = \left(R_{ij}\right) \in \Re^{n_1 \times n_2}$ is noise residuals, $c \in N$ is the residual order, N_{ij} is a local neighborhood of pixel X_{ij}, $X_{ij} \notin N_{ij}$, and $\hat{X}_{ij}\left(.\right)$ is a predictor of X_{ij} using the filter. The advantage of modeling the residual instead of the pixel values is that the image content is largely suppressed in **R**, which has a much narrower dynamic range and a more compact and robust statistical description.

As the above analysis, each pixel in the image will be convert to a residual by the similar and independent convolution computing, which is suitable for processing in parallel.

3.2. Step 2: Truncation &Quantization

In each sub model, the residual is quantized and truncated as following formula:

$$R_{ij} \leftarrow truncT\left(round\left(\frac{R_{ij}}{q}\right)\right) \tag{2}$$

where q > 0 is a quantization step. The experimental results in the literature [3] show that it is best to set $q \in [c, 2c]$ for the best performance, as follows:

$$q \in \begin{cases} \{c, 1.5c, 2c\} & for \quad c > 1 \\ \{1, 2\} & for \quad c = 1 \end{cases} \tag{3}$$

In this step, every residual is processed similarly and independently. It is also suitable for parallel processing.

3.3. Step 3: Co-occurrences matrix calculation

In this step, the residual matrix will be scanned to generate horizontal and vertical co-occurrences of four consecutive samples processed using formula (2) with q=2. Formally, each co-occurrence matrix **C** is a four-dimensional array indexed with **d**=(d_1, d_2, d_3, d_4)$\in\{-q,\ldots,q\}^4$, which gives the $(2q+1)^4$=625 elements. The d-th element of the horizontal co-occurrence for residual is formally defined as the (normalized) number of groups of four neighboring residual samples with values equal to d_1, d_2, d_3, d_4, as follows:

$$C_d^{(h)} = \frac{1}{Z}\left|\left\{\left(R_{ij}, R_{i,j+1}, R_{i,j+2}, R_{i,j+3}\right) \mid R_{i,j+k-1} = d_k, k = 1,\ldots 4\right\}\right| \tag{4}$$

where Z is the normalization factor ensuring that $\sum_{d \in T_\Delta} C_d^{(h)} = 1$. The vertical co-occurrence $C_d^{(v)}$ is defined analogically.

As shown above, every group of four neighboring residual samples are scanned, and then the results are written to a global co-occurrence matrix. These processes are similar but not independent because of writing in the same co-occurrence matrix, which may cause memory conflicts in parallel program. So we need to do some modifications to achieve good parallel ability.

4. GPU and OpenCL

In this section, we shall give the necessary information of GPU and OpenCL, since we shall use the AMD GPU HD 6850 to accelerate the SRM extraction algorithm in latter sections. In general, the GPU hardware contains 12 multithreaded Single Instruction Multiple Data (SIMD) Processors (also called as CU, Compute Unit), which can execute

tasks independently. Each CU is composed by 16 PE (Processing Element) that contains 5 ALU (Arithmetic Logical Unit) for computing. PE is the smallest unit that can run a thread independently. The basic principle of parallel computing on GPU is that a large number of PEs execute same task to process different data blocks simultaneously, that is data-based parallelism.

Parallel program on the GPU needs appropriate program framework to support. OpenCL (Open Computing Language) is the first free parallel program framework for writing programs that execute across heterogeneous platforms consisting of CPU, GPU, DSP and other processors. OpenCL provides parallel computing using data-based parallelism. An OpenCL program usually consists of two parts: a program runs on the host computer and a program called kernel runs on an OpenCL device (such as GPU). Parallel computing is implemented by launching kernels that maps each work-item to a portion of the input data as opposed to a specific task. The map is processed by index ID of work-item. A work-item corresponds to a thread runs in a PE. Multiple work-items can form a work-group that runs in a SIMD engine (CU). The index ID of work-item and work-group form a N-dimensional index space, called NDRange as shown in Fig. 1. If image data is segment as NDRange, parallel processing is achieved by each of the work-item processing corresponding image data with same index ID.

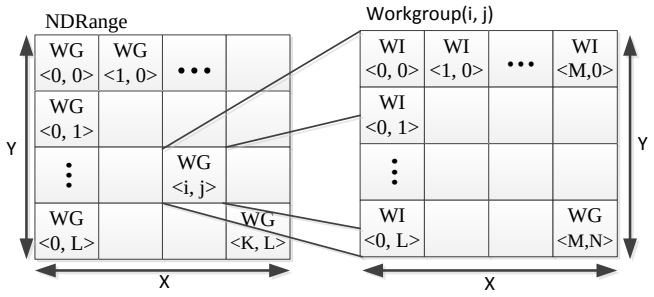

Fig. 1. OpenCL NDRange

5. The Parallel GPU-SRM Algorithm

The main steps of our proposed algorithm is depicted as in Fig. 2. In general, it contains two parts: the host program on CPU and the kernel program on GPU. The former is responsible for process control operation, such as OpenCL initialization, image input, sub models selection, while the latter executes the task of feature extraction from image with the help of powerful parallel computing capability on GPU. According to the parallel program framework OpenCL, we define some functions run on work-item for each step in SRM extraction, mainly including residual computing, truncation & quantization, co-occurrence matrix calculation. Then a collection of work-items are executed simultaneously in GPU, and use the ID to load corresponding block of image to process. Images and final SRM feature extracted are transmitted between CPU and GPU by the PCI-E bus.

As the analysis in section 3, in the GPU side, residual computing, truncation & quantization can be easy parallelized, but the co-occurrence matrix calculation needs to do some modifications to achieve good parallel computing. Moreover, because of concurrent characteristics in parallel program with large-scale multi-thread, the optimization for parallel program has great influence on the performance of algorithms. On analysis of the SRM extraction process, we employ optimization technology including convolution unrolling, combined memory access, split-merge strategy for Co-occurrence matrix calculation in parallel program to accelerate SRM feature extraction.

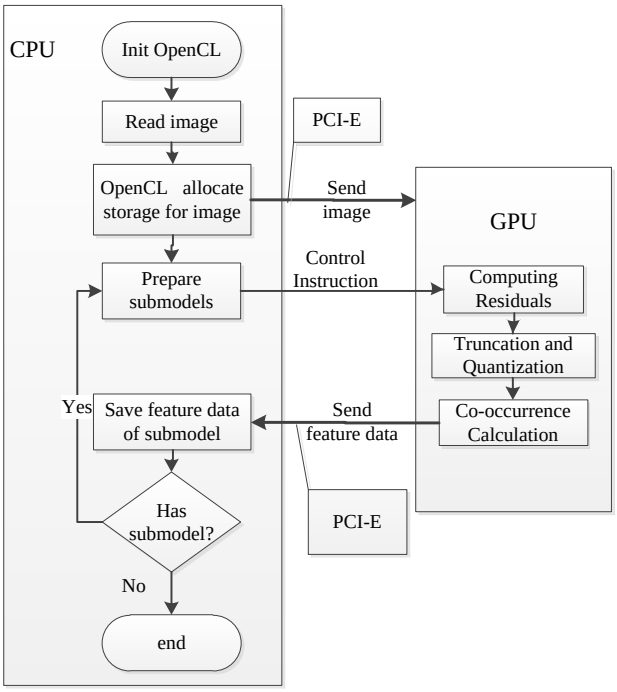

Fig. 2. GPU-SRM algorithm

6. Parallel Optimization

In three steps of SRM feature extraction, the style of parallel computing and memory access are different, so the parallelization and optimization method also have some differences. The following are the specific parallelization and optimization methods in each step.

6.1. Selection of NDRange Size

NDRange size that includes the number of workgroup and work-item, will affect the efficiency of parallel processing. Its size is limited by many factors, such as target architecture, the design of algorithm and the quantity of memory. Moreover, too many workgroups and work-items always result in great cost in resource scheduling, memory access, and communication. Usually, *the optimal number of workgroups is equal to or an integral multiple of CUs*, because local memory in each CU can be used most effectively. On the other hand, the size of image block that is processed by each work-item is important to the performance. If each work-item processes just one pixel, an image with (*row* x *col*) pixels (such as 512x512) should need great quantity of work-items. The execution cost will be great. Moreover, on AMD GPU, the number of work-item in one workgroup is limited under 256. So a more practical approach is that each work-item processes a sequence or a block of pixels.

Considering the above factors, we assign the size of workgroup and work-item in the three step as following:

- In the step 1, we assign the number of workgroups equal to CUs, for example, 12x1 workgroups for AMD HD6850 GPU. Residual computing contains the process of image convolution that needs to access these pixels around according to the size of the filtering window, so it means 2-D range of work-item is suitable. Therefore, we use 16x16 work-items in one workgroup.
- In the step 2, Truncation &Quantization is independently process one by one pixel. So it is suitable to assign the same NDRange size as step 1.
- In the step 3, for best utilizing the GPU resource, we still assign the number of workgroups equal to CUs. But as the processes are similar but not independent, the work-item size will be affect by many factors that will be explored in section 5.3. For optimal access of memory, we assign 1-D range, 256x1 work-items in a work-group.

6.2. Convolution Unrolling

As the residuals computing code (listing 1) shows, convolution occupies the main portion of computation. Each work-item contains four loops to processes a block with (*row* x *col*) pixels. Two inner loops just use one row of code to execute convolution computation. In work-item, compact convolution code is not good for effectively parallel computing, because the utilization of ALU calculation units is too low. We analyze the code of convolution in the GPU kernel function by AMD CodeXL profiler tool, and find that less than half of ALUs are in use, because the compiler cannot find enough instructions to take full advantage of VLIW (Very Long Instruction Word) units.

For increasing ALU instructions, we use convolution unrolling to fast the speed. Convolution filter size in all the sub models has been fixed, so we can unroll convolution computing. However, the number of registers required in convolution after unrolling will rise greatly. In order to control the number of required registers in a reasonable scale, all the sub models are sorted into some kernel functions according to

their filter size. *If filter size is not large than 8, just one inner loop be unrolled, as listing 2 shows*. Otherwise, both of inner loops are unrolled as listing 3 shows.

Listing 1. Residual computing model

```
function work-item-residuals-computing
input:  L_I;                    //localImage
        F;                      //filter
output: O_I;                    //outputImage
 /* just kernel code is listed */
For col := 0 To width - 1           // convolution
      For row := 0 To height - 1
      {
         O_I(col, row) := 0;
         For F_Col := 0 To F_Width - 1
            For F_Row := 0 To F_Height – 1
            {
                O_I(col, row) += L_I (col + F_Col,
                row + F_Row) * F(F_Col, F_Row);

            }
      }
```

Listing 2. Inner loop is unrolled (when filter size is not larger than 8)

```
...
{ /* convolution */
O_I(col, row) = 0;
For F_Col := 0 To F_Width - 1{
   O_I(col, row) += L_I(col + F_Col, row + 0)
   * F(F_Col, 0);
   ...
   O_I(col, row) += L_I(col + F_Col, row + F_Height - 1)
   *F(F_Col,  F_Height - 1);
   }
}
```

Listing 3. Two inner loops are unrolled (when filter size is larger than 8)

```
...
{  /* convolution */
   O_I(col, row) = 0;
   O_I(col, row) += L_I(col + 0, row + 0) * F(0, 0);
   ...
   O_I(col, row) += L_I(col + F_Width - 1,
   row + F_Height - 1) * F(F_Width - 1, F_Height - 1);
}
```

6.3. The Split-merge Strategy for Co-occurrence Matrix Calculation

In the step of Co-occurrences matrix calculation, the most troublesome problem is that the global co-occurrence matrix needs to be read/write by all the work-items, which will result in many memory read/write conflicts unavoidably. To solve this problem, we firstly divided the residual noise matrix into lots of blocks for each work-item, and a local co-occurrence matrix for each block is calculated to store in local memory in the work-items. Then all of the local co-occurrence matrixes in the same workgroup are merged into a co-occurrence matrix by the first work-item to avoid the band conflict. Finally, these co-occurrence matrixes from each workgroups are merged into a global co-occurrence matrix in GPU or in the host CPU.

This strategy seems perfect, but it has a great bottleneck that the size of local memory in SIMD is limited, just 32KB (32 bank) in AMD GPU. It means that if a local co-occurrence matrix contains 256 elements with 32bit (4B) value, it just supports 32 work-items for use. More work-items will result in bank conflict (conflict of read/write local memory) unavoidably to hang up the read/write operation for sequential process. Moreover, local memory is also used for other task. Too much more memory used for local co-occurrence matrix will affect the speed of the whole algorithm. *So it is need tradeoff the number of local co-occurrence matrixes.* We will determine the quantity by the test in experiments.

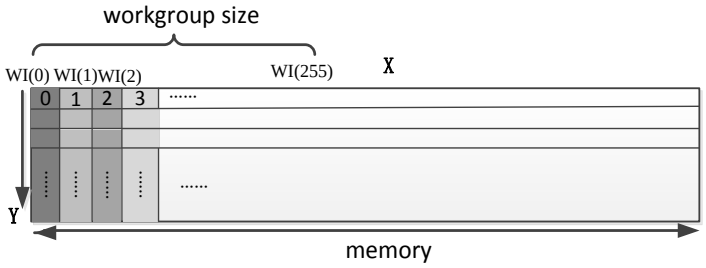

Fig. 3. Memory request coalescing

6.4. Optimization for Global Memory Access

In step 3, the memory that stores the global co-occurrence matrix will be repeatedly executed lots of read-modify-write operations. On the CPU platform, an efficient solution is to use high speed cache to store the matrix, but the GPU has no cache to use. If we use global memory, it can greatly fast the speed by coalescing memory access requests by multiple consecutive work-item into a single memory access. For effective use of memory bandwidth, AMD GPU supports 16 consecutive work-item reads the 128-bit aligned memory address, while the size of work-item is 32 on the NVIDIA GPU. This means that the most ideal access mode is 32 consecutive work-items sequentially access 4-bit. Thus, for the efficient memory access without concerning of the difference between two platforms, the width of workgroup(X dimension) should be set as

integral multiple of 32. The number of work-items in one workgroup is limited under 256. *Therefore, we set the workgroup size as 256x1 work-items in step 3.* As shown in Fig. 3, memory access requests from 256 work-items will be coalesced to improve the bandwidth utilization. Thus, each work-item can process a whole column of element.

7. Experimental Results

We now measure the time consumption of GPU-SRM extraction, against the original implementation of single thread SRM extraction, as well as multi-threads in a Multi-core CPU. We also compare the time consumption in each step to show which step is still need to be improved. In addition to this, we also determine some optimal parameters used in GPU-SRM by the experiment.

7.1. Experimental Setup

We carried out the experiments to compare the performance of proposed algorithm with the original single thread SRM algorithm. The configuration of computer and the GPU used in experiment are shown in Table 1 and 2. Algorithm is coded with C++ language, and uses Visual studio 2010 SP1 combined with Intel Parallel Studio 2013 XE plug-in, and Intel C++ x64 compiler for compiling 64-bit program, and OpenCL SDK AMD-APP-SDK v2.8.1.0. In order to eliminate the interference of time consumption in the compilation process, OpenCL kernel is pre-compiled into binary code in the experimental test.

Table 1. Experimental platform

CPU	Intel i5-2310（4 cores 4 thread）
CPU frequency	2.90 GHz
Memory	8G
OS	Windows 7 SP1 x64
PCI-E version	2.0

Table 2. GPU specifications

GPU	Radeon HD 6850
Memory Size	1 GB
Core Clock	775 MHz
SM units	12
Stream pocessors	960
FLOPS	1.5 TFLOPs
Memory Bus	256 bit
Bandwidth	128 GB/s

The experiment collected respectively 512 gray scale steganographic images with different resolution, included 512x512, 1024x768, 1280x1024, 2048x1536, 3072x2304,

and 4000x3000. These images are used to test the time consumption of three different platforms for SRM feature extraction.

7.2. The Optimal Size of Local Co-occurrence Matrix

Co-occurrence matrix calculation takes over most of the time in the whole SRM extraction, and is also the most difficult to parallel processing. In Section 5.3, we propose a split-merge strategy to accelerate its calculation. In this strategy, it needs to determine the optimal number of local co-occurrence matrix from the sub-models of *spam* and *minmax* in one workgroup. We use three different sizes of images for calculating the Co-occurrence matrix in our benchmark platform. The experimental results in Fig. 4 show that, too many local Co-occurrence matrixes in local memory will degrade the speed of process, especially in large size (3078x2304) image, because it need more local memory for other task. When the number of local Co-occurrence matrixes is 1 to 6, the process time is least.

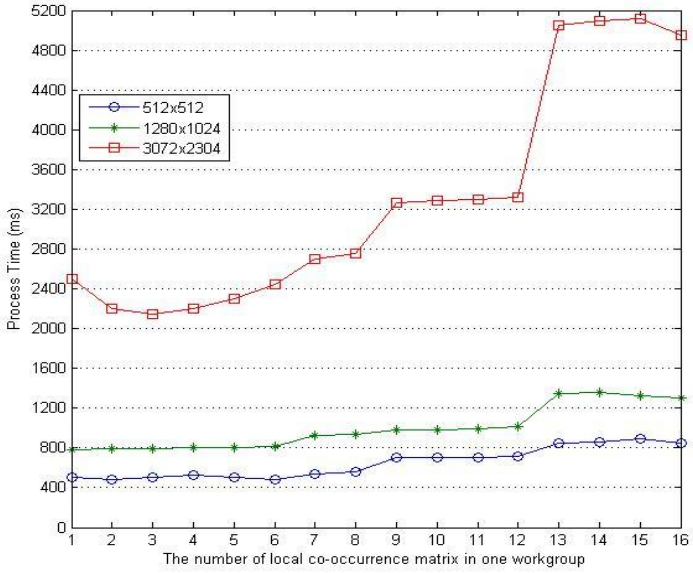

Fig. 4. The optimal number of local co-occurrence matrix

But this number is much less than the number of work-item. It results in bank conflict unavoidably. We use AMD CodeXL 23 profile to see the relation in the number of local Co-occurrence matrix, time consumption and bank conflict rate. The results are listed in Table 3. As the table shows, when the number of local Co-occurrence matrix increases, bank conflict rate decreases. But the process time also increases. It shows that time consuming cause by bank conflict is much less than that more local memory occupied by Co-occurrence matrix.

According to above analysis, it is suitable to assign the number of local Co-occurrence matrixes as 3, especially for large size of image.

Table 3. Local co-occurrence matrix and bank conflict

Number	*Spam*		*Minmax*	
	Time (ms)	Bank conflict	time(ms)	Bank conflict
1	0.30423	20.80%	1.39756	9.44%
2	0.31355	10.88%	1.38455	5.01%
3	0.30967	8.38%	1.40288	3.83%
4	0.31512	6.51%	1.45188	2.68%
5	0.31533	6.47%	1.48811	5.39%
6	0.31622	5.34%	1.45112	2.25%
7	0.31844	5.31%	1.77189	1.88%
8	0.31844	4.21%	1.77989	1.00%
9	0.31566	4.32%	2.20522	1.24%
10	0.31622	4.52%	2.29400	1.14%
11	0.31811	4.49%	2.30833	1.18%
12	0.31834	4.70%	2.33622	0.97%
13	0.38889	3.62%	2.29067	0.82%
14	0.39167	3.87%	3.32989	0.76%
15	0.39478	3.69%	3.40600	0.79%
16	0.41144	5.28%	3.21123	0.00%

Table 4. Time-consumption of three steps in SRM extraction (s)

	Single thread SRM in a CPU			GPU-SRM		
Image size	Step1	Step2	Step3	Step1	Step2	Step3
512x512	0.2398	0.3024	3.3126	0.0043	0.0072	0.3037
1024x768	0.7566	0.9153	10.4054	0.0104	0.0196	0.4379
1280x1024	1.2483	1.5100	17.5234	0.0207	0.0294	0.5800
2048x1536	2.9844	3.6049	42.6065	0.0464	0.0567	0.9147
3072x2304	6.9139	8.1981	95.1779	0.0979	0.1223	1.5909
4000x3000	11.3726	13.6937	153.2185	0.2058	0.3378	3.9255

7.3. Time-Consumption of Three Steps

Table 4 compares show that time-consumption of three steps in single thread SRM on CPU and GPU-SRM. Where Step1 is Residuals Computing, Step2 is truncation&quantization, Step3 is co-occurrence matrix calculation. As the table shows, in each size pictures, Residuals Computing on GPU is 50 times faster than single thread SRM, and truncation&quantization is increased by about 40 times. The Co-occurrence matrix calculation is accelerated from 11 to 60 times, where large-size images are

improved more than the small-size images. It is because of the large size image can be divided into more blocks for parallel computing.

In addition, Table 4 also shows that either single thread or GPU-SRM, The Co-occurrence matrix calculation occupies the most portion of the time consumed by SRM feature extraction, which is due to co-occurrence matrix calculation is memory intensive, high computation, and difficult to parallel program. Therefore, optimization on this part has a decisive role in the entire feature extraction speed.

7.4. Time-Consumption Comparison

Table 5 show that, except for in the small size of 512x512 image, the proposed algorithm is 25~55 times faster than single thread run on CPU, and is 2~4 times faster than the parallel computing on quad-core CPU. It proves that our parallel algorithm on GPU can greatly accelerate SRM extraction. But in the 512x512 pixel image, the proposed algorithm advantage is not obvious, even a little slower than the case on multi-core CPU. This is because the small size image with few blocks can not maximize the use of GPU parallel capabilities, especially in the step of co-occurrence matrix calculation.

Table 5. Time-consumption of SRM extraction on different platform (s)

Image size	Single thread in a CPU	Multi-threads in multi-core CPU	GPU-SRM
521x512	3.93	0.28	0.35
1024x768	12.29	0.92	0.49
1280x1024	20.62	1.54	0.66
2048x1536	49.56	3.64	1.10
3072x 2304	110.68	8.18	1.94
4000x3000	181.39	13.5	3.34

8. Conclusion

This paper presents a parallel algorithm for SRM feature extraction using GPU technology. Through the steps of computing noise residual, quantization and truncation, and the calculation of co-occurrence matrix, we give our implementation of the algorithm based on GPU architecture. The experimental results show when using AMD GPU 6850, the extracting speed is significantly improved comparing to the original single thread algorithm. In addition, it is also shown when using AMD 6850 our algorithm runs 2~4.2 times faster than using a Intel Quad-core CPU, which indicates our algorithm efficiently uses the GPU cores.

Acknowledgments. The research is supported by the National Science Foundation of China (Grant 61300025), the Ministry of Education of Doctoral Fund Project (Grant 20123514120013), the Educational Research Project for Middle-aged and Young Teachers of Fujian Province (Grant JA15066), and the Science and Technology Development Fund of Fuzhou University (Grant 2014-XY-20). The corresponding author is Longkun Guo.

References

1. Westfeld, A.: F5-A Steganographic Algorithm-Information Hiding. Springer Berlin Heidelberg, 289-302. (2001)
2. Robertson, N. Cruickshank, P. Lister, T.: Documents Reveal al Qaeda's Plans for Seizing Cruise Ships, Carnage in Europe. Available at: http://www.cnn.com/2012/04/30/world/al-qaeda-documents-future. (Accessed 2014)
3. Maney, K.: Bin Laden's Messages Could be Hiding in Plain Sight. Available at: http://usatoday30.usatoday.com/tech/columnist/2001/12/19/maney.htm. (Accessed 2014)
4. Pevny, T., Filler, T., Bas, P.: Using High-Dimensional Image Models to Perform Highly Undetectable Steganography. In: Information Hiding, 161–177. (2010)
5. Fridrich J., Kodovský J., Holub V., et al.: Steganalysis of Content-Adaptive Steganography in Spatial Domain. In: Information Hiding, 102-117. (2011)
6. Fridrich, J., Kodovsky J.: Rich Models for Steganalysis of Digital Images. IEEE Transactions on Information Forensics and Security 7(3), 868–882. (2012)
7. Ker, A.D.: Implementing the Projected Spatial Rich Features on a GPU. In: IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics, vol.9028, 1801–1810. (2014)
8. Kodovsky, J., Fridrich, J., and Holub, V.: Ensemble Classifiers for Steganalysis of Digital Media. IEEE Transactions on Information Forensics and Security 7(2), 432–444. (2012)
9. Kodovsky, J. and Fridrich, J.: Steganalysis of JPEG Images Using Rich Models. In: IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, vol.8303, pp. 1-13. (2012)
10. Holub, V., Fridrich, J., and Denemark, T.: Random Projections of Residuals as An Alternative to Co-occurrences in Steganalysis. In: IS&T/SPIE Electronic Imaging, International Society for Optics and Photonics, vol. 8665, 1–11. (2013)
11. Holub, V. and Fridrich, J.: Random Projections of Residuals for Digital Image Steganalysis. IEEE Transactions on Information Forensics and Security 8(12), 1996–2006. (2013)
12. TESLA K20X GPU ACCELERATORS. NVIDIA Corporation. Available at: http://www.nvidia.com/content/PDF/kepler/Tesla-K20X-BD-06397-001-v07.pdf. (2014)
13. Li, P.H.: A Novel Color Based Particle Filter Algorithm for Object Tracking, Chinese Journal of Computers, vol. 32, 2454-2463. (2009)
14. Cao, J. Xie, X. Liang, J.: GPU Accelerated Target Tracking Method. In: Advances in Multimedia, Software Engineering and Computing, vol. 1, pp. 251-257. (2012)
15. Akhloufi, M. A., Gariepy, F. Champagne, G.: GPGPU Real-time Texture Analysis Framework. In: SPIE Electronic Imaging, International Society for Optics and Photonics, vol.7872, 1-9. (2011)
16. Sun, K. Pan, X. Wang, J.: Hardware Based Steganalysis. In: Signal Processing for Image Enhancement and Multimedia Processing, 269–78. (2008)
17. Fridrich, J. Goljan, M. Du, R.: Reliable Detection of LSB Steganography in Color and Gray-scale Images. In: Proceedings of the ACM Workshop Multimedia Security, 27–30. (2011)
18. Gutierrez-Fernandez, E. Portela-García, M. Lopez-Ongil, C. Garcia-Valderas, M.: FPGA-based Implementation for Steganalysis: A JPEG-Compatibility Algorithm. In: SPIE

Microtechnologies. International Society for Optics and Photonics, vol.8764, pp. 1-7. (2013)
19. Fridrich, J., Goljan, M., Du, R.: Steganalysis Based on JPEG Compatibility. In: International Symposium on the Convergence of IT and Communications (ITCom). International Society for Optics and Photonics, 275-280. (2001)
20. Tiwary, M., Priyadarshini, R., Misra, R.: A Faster and Intelligent Steganography Detection Using Graphics Processing Unit in Cloud. In: International Conference on High Performance Computing and Applications (ICHPCA), 1-6. (2014)
21. Rodriguez-Perez, M. Morales-Reyes, A. Cumplido, R.: An Analysis of Computational Models for Accelerating the Subtractive Pixel Adjacency Model Computation. Computers & Electrical Engineering, vol.43, 9-16. (2015)
22. Pevny, T., Bas, P., Fridrich, J., Steganalysis by Subtractive Pixel Adjacency Matrix. IEEE Transactions on information Forensics and Security 2(5), 215–224. (2010)
23. AMD Develpoer Central.: CodeXL – Powerful Debugging, Profiling & Analysis. Available at: http://developer.amd.com/tools-and-sdks/ heterogeneous-computing /codexl/. (Accessed 2014)

Kaizhi Chen graduated at the College of Information Science and Engineering, Southeast University in Nanjing, China in 2011, where he received a Ph.D. in information and communication engineering. Now, he is a lecturer at the College of Mathematics and Computer Science, Fuzhou University in Fuzhou, China. His research interests include intelligent image analysis and machine learning.

Chenjun Lin graduated at the College of Mathematics and Computer Science, Fuzhou University in Fuzhou, China in 2010, where he received a B.Sc. degree in computer science and technology. He finished his M.Sc. studies in 2013. Currently, he is working at a department in the government. His research interests include parallel processing and machine learning.

Shangping Zhong received his B.Sc. in mathematics science from Fuzhou University, Fuzhou, China, in 1991, and his M.Sc. in computer science and technology from Fuzhou University, Fuzhou, China, in 1997, and his Ph.D. in computer science and technology from Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2005. Now, he is a Professor at the College of Mathematics and Computer Science, Fuzhou University in Fuzhou, China. His research interests include information security, intelligent image analysis and machine learning.

Longkun Guo received his BEng degree in computer science and his PhD degree in computer software and theory from the University of Science and Technology of China, in 2005 and 2011, respectively. Currently, he is an associate professor in the College of Mathematics and Computer Science, Fuzhou University, China. His research interests include approximation algorithms and optimization, parallel computing, computational complexity and graph theory.

Received: August 15, 2014; Accepted: September 10, 2015.