Parallelized Computation for Edge Histogram Descriptor Using CUDA on the Graphics Processing Units (GPU)

Alireza Ahmadi Mohammadabadi
Department of Computer Engineering
Razi University
Kermanshah, Iran
alireza.ahmadi@pgs.razi.ac.ir

Abdolah Chalechale
Department of Computer Engineering
Razi University
Kermanshah, Iran
chalechale@razi.ac.ir

Hadis Heidari
Department of Computer Engineering
Razi University
Kermanshah, Iran
h.heidari@pgs.razi.ac.ir

***Abstract*—Most image processing algorithms are inherently parallel, so multithreading processors are suitable in such applications. In huge image databases, image processing takes very long time for run on a single core processor because of single thread execution of algorithms. GPU is more common in most image processing applications due to multithread execution of algorithms, programmability and low cost. In this paper we show how to implement the MPRG-7 Edge Histogram Descriptor in parallel using CUDA programming model on a GPU. The Edge Histogram Descriptor describes the distribution of various types of edges with a histogram that can be a tool for image matching. This feature is applied to search images from a database which are similar to a query image. We evaluated the retrieval of the proposed technique using recall, precision, and average precision measures. Experimental results showed that parallel implementation led to an average speed up of 14.74×over the serial implementation. The average precision and the average recall of presented method are 67.02% and 55.00% respectively.**

Keywords—content based image retrieval; CUDA; edge histogram descriptor; GPU

I. INTRODUCTION

Image retrieval as one of the interesting applications of image processing is an appropriate case to be implemented in parallel. Because in this application, images are usually divided into several parts and each part is processed separately and similarly. The main process in image retrieval is to search images from a database based on low-level visual features such as color, texture, and shape. Typically, image retrieval systems extract features in the offline phase and use these features for matching in the online phase. For this, the visual contents of the images in the database are extracted and described by multi-dimensional feature vectors, which are more compressed and easier to process.

Content based image retrieval (CBIR) is a research area, which targets to develop tools for retrieval of visual information using it's perceptual contents and has been used for the first time by Kato et al. [1] to describe his experiments into automatic retrieval of images from a database. CBIR has very important applications in many areas including medical science, education, architectural design, the justice department and agriculture and uses the visual contents of an image such as color, shape and texture to represent and index the image.

Color is one of the most widely used low-level visual features and is popular in many CBIR applications. Many CBIR systems employ color to retrieve images, such as Query by Image Content (QBIC) system that allows users to refine queries based on multiple properties such as color, texture and shape. The most common method for color based image retrieval is the color histogram that describes the global color distribution in an image. The main drawback of color histogram is that it only gives an overall indication of the color content of an image but does not provide information about the spatial distribution of the colors.

Several literature are reported for the purpose of texture based image retrieval, for example, Zhang et al. [2] extracted the rotation invariant texture feature extraction based on Gabor texture features by a circular shift of the feature elements so that all the images have the same dominant direction. Kokare et al. [3] applied dual tree rotated complex wavelet filter and dual tree complex wavelet transform for effective rotation invariant texture feature extraction.

Object shape features can provide powerful information for image retrieval, because humans can recognize objects solely from their shapes. The edges in images are an essential feature to represent their contents. Also, human eyes are sensitive to edge features for image perception [4]. There is a descriptor for edge distribution in the image that is called Edge Histogram Descriptor (EHD). The Edge Histogram Descriptor is one of the descriptors specified by MPEG-7 Visual Standard for measuring similarity in images [4]. An edge histogram in the image space represents the frequency and the directionality of the brightness changes in the image [4]. The Edge Histogram Descriptor can express the local edge distribution in an image with a histogram.

To extract edge features, the different parts of the image must be processed separately, thereby consuming long time especially when processing high resolution images on large databases. So it is essential to investigate speed up techniques, satisfying time constraints for the various applications. Since EHD extraction is a data-parallel task described in flowing, it is expected to reduce execution time by using multi-core processors. For extracting edge features for an image, the image is divided into 16 non-overlapping image blocks. Each image block is divided into several sub-images as well, such that each sub-image is processed similar to other sub-images. So in each image block, the type of edge is recognized independently. In fact, the same instruction is executed by

978-1-4799-0565-2/13/$31.00 ©2013 IEEE

IEEE COMPUTER SOCIETY
IEEE

multiple processors using different data streams (SIMD). Consequently these processes can be performed in parallel. To accelerate of EHD extraction, multi-core processors can be useful because of multiple thread execution of algorithms.

Graphics Processing Units (GPUs) play important role to speedup processing of database images matching algorithms because it has more inbuilt execution cores [5]. Many researchers have already been applied GPUs to implement many algorithms in various areas such as image processing, computational geometry, and scientific computation, as well as computer graphics [6]. Parallel implementations on GPUs have been applied to various numerical problems to reduce the computation time without sacrificing the degree of accuracy. Compute Unified Device Architecture (CUDA) programming model released by NVIDIA provides a set of minimal extensions to the C programming language that allow the programmer to write kernels executed in parallel on the GPU [7]. In this paper, parallel implementation is performed for Edge Histogram Descriptor using CUDA to run on GPU. Our work uses extensive usage of highly multithreaded architecture of multi-core GPU.

The rest of this paper is organized as follows. Section II describes the GPU architecture. Section III presents the basic concepts of the CBIR domain and the proposed image retrieval procedure. Section IV reports experimental results to evaluate the robustness and retrieval effectiveness of the proposed scheme. Finally, conclusion and future work are provided in Section V.

II. THE GPU ARCHITECTURE

In recent years, GPUs have become increasingly attractive for general purpose parallel computation [8]. Fig. 1 illustrates the architecture of a GPU. A GPU has a series of Multiprocessors (MP), and each multiprocessor has a number of Scalar Processors (SP). The memory bank of a MP can be accessed by all its SPs and a large global memory is shared across all the MPs [9]. CUDA is a parallel computing architecture developed by NVIDIA to implement algorithms in their GPUs. A typical CUDA program consists of a host code and a device code, where CPU and GPU are the host and the device respectively. The host performs the nonparallel computations and passes data to the global memory in the GPU and launches a kernel and data-parallel portions of an application are implemented as kernels. The kernel executes the computations using parallel threads on the SPs. The threads are grouped into blocks and blocks are grouped further into grids. A thread block is a 3, 2 or 1-dimensional group of threads. Threads within a block can cooperate among themselves by sharing data through some shared memory and synchronizing their execution to coordinate memory accesses. Threads in different blocks cannot cooperate. The number of threads per block is constrained by the limited memory resources of a processor core.

The multiprocessor partitions its thread blocks into groups of 32 parallel threads called warps. If threads of a warp diverge via a data-dependent conditional branch, the warp executes each branch path serially; disabling threads that are not on that path, and when all paths complete, the threads converge back to the same execution path [10].

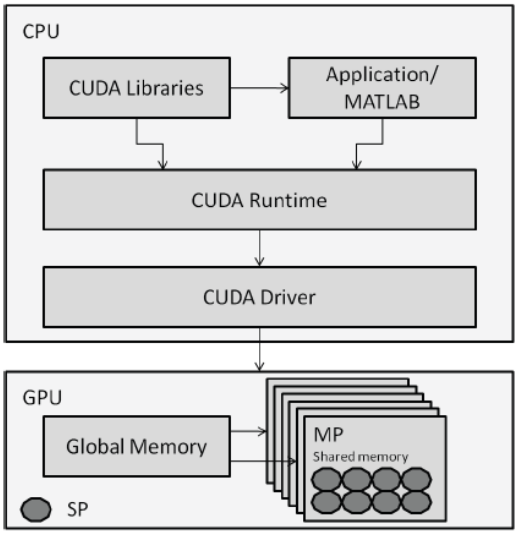

Fig. 1. The NVIDIA GPU architecture [9].

Due to the large amount of threads, CUDA introduces the concept of grid, block and thread to manage them. In a grid, blocks are managed in x-, y- and z-directions. The dimensions of a grid are given by the variable gridDim, whose components are gridDim.x, gridDim.y and gridDim.z. The dimensions of a block are given by the variable blockDim, whose components are given as blockDim.x, blockDim.y and blockDim.z. The block index is given by blockIdx, which contains the components of blockIdx.x, blockIdx.y and blockIdx.z. So that threads are managed in a 3D way in a block and the thread index is given by threadIdx (threadIdx.x, threadIdx.y, threadIdx.z).

III. THE PROPOSED IMAGE RETRIEVAL

In this section, we provide some information about the basic concepts of CBIR domain and the proposed image retrieval procedure.

A. The Basic Concepts of CBIR Systems

The process of image retrieval consists of two tasks include indexing and retrieval. Features are the representatives of the images. Indexing means characterization of images based on image properties. The percentages of retrieval efficiency fully rely on selection of proper image features. Two main requirements of image retrieval are high retrieval accuracy and less computational complexity. The image retrieval process consists of calculating a feature vector for characterizes some image properties, and stored in the image feature database. The user provides a query image, and the image retrieval system computes the feature vector for it, and compares it with the particular image feature database images. The relevance comparison is performed by using some distance measurement technique, and the minimum or permissible distances are the metrics for the matched or similar images.

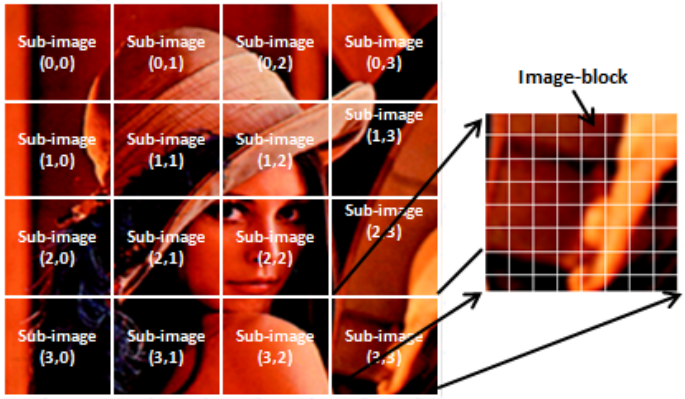

Fig. 2. Definition of sub-image and image block.

Fig. 3. Five types of edges.

B. *The MPEG-7 Standard Method*

Content based image retrieval is defined as a process that searches and retrieves images from a large database on the features such as color, texture and shape. There are various methods which can be used for retrieval that one of them is Edge Histogram Descriptor. The EHD express the local edge distribution with a histogram. The EHD represents the distribution of 5 types of edges in each local area called a sub-image. As shown in Fig. 2, the sub-image is specified by separating the image space into 4×4 non overlapping blocks.

To describe the sub-image, a histogram of edge distribution for each sub-image is created. As shown in Fig. 3, edges in the sub-images are classified into 5 types: vertical, horizontal, 45-degree diagonal, 135-degree diagonal and non-directional edges. Each local histogram includes 5 bins. Each bin corresponds to one of 5 edge types. Each histogram bin value should be normalized, for normalization; the number of edge events for each bin is divided by the total number of image-blocks in the sub-image. Each image-block is classified into one of the 5 types of edge blocks or a non-edge block. Although the non-edge blocks do not contribute to any histogram bins, each histogram bin value is normalized by the total number of image-blocks including the non-edge blocks. To extract directional edge features, defining small square image-blocks in each sub-image is performed. The image space is divided into non-overlapping square image-blocks and then the edge information from them is extracted. A method to extract an edge feature in the image-block is to apply digital filters in the spatial domain.

In first, the image-block into four sub-blocks is divided. Then, by assigning labels for four sub-blocks from 0 to 3, the average gray levels for four sub-blocks at (i,j)th image-block as $a_0(i,j)$, $a_1(i,j)$, $a_2(i,j)$, and $a_3(i,j)$, respectively is presented. Also, Won et al. [4] represented the filter coefficients for vertical, horizontal, 45-degree diagonal, 135-degree diagonal, and non-directional edges as $f_v(k)$, $f_h(k)$, $f_{d\text{-}45}(k)$, $f_{d\text{-}135}(k)$, and $f_{nd}(k)$, respectively, where k=0,…,3 represents the location of the sub-blocks. Now, the respective edge magnitudes $m_v(i,j)$, $m_h(i,j)$, $m_{d\text{-}45}(i,j)$, $m_{d\text{-}135}(i,j)$, and $m_{nd}(i,j)$ for the (i,j)th image-block can be obtained as follows:

$$m_v(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_v(k) \right| \quad (1)$$

$$m_h(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_h(k) \right| \quad (2)$$

$$m_{d-45}(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_{d-45}(k) \right| \quad (3)$$

$$m_{d-135}(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_{d-135}(k) \right| \quad (4)$$

$$m_{nd}(i,j) = \left| \sum_{k=0}^{3} a_k(i,j) \times f_{nd}(k) \right| \quad (5)$$

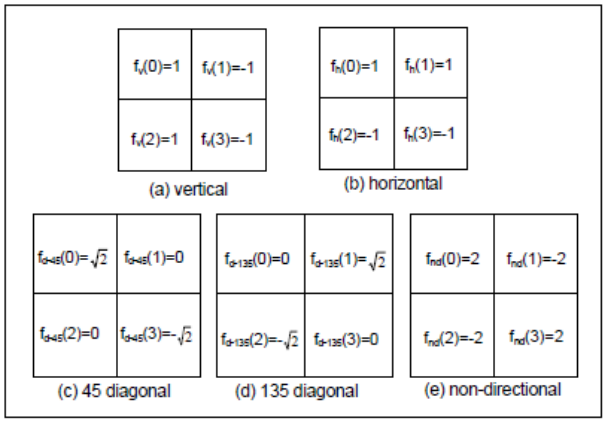

Fig. 4. Filter coefficients for edge detection [4].

If the maximum value among five edge strengths obtained from (1) to (5) is greater than a threshold T_{edge} (T_{edge} = 11), then the image-block is considered to have the corresponding edge in it [4], otherwise, the image-block contains no edge. In MPEG-7 XM Document [11], a set of filter coefficients, depicted in Fig. 4, is recommended.

For the similarity matching, we calculate the distance D(A, B) of two image histograms A and B using the following measure.

$$D(A,B)=\sum_{i=0}^{79}\left|Local\_A[i]-Local\_B[i]\right| \qquad (6)$$

Where Local_A[i] represents the reconstructed value of Bin-Count[i] of image A. Similarly, Local_B[i] is the reconstructed value of BinCount[i] of image B.

C. Parallel Implementation Using CUDA

According to CUDA, data-parallel portions of an application are implemented as kernels. The main CPU, acting as the host, can initiate one kernel at a time. Each kernel is executed in parallel by several threads. The programmer groups threads into blocks, which are logically aggregated into a grid. The host and the device (i.e., the GPU) communicate by copying data from/to the CPU's memory to/from the so-called GPU global memory. Therefore, by default, threads operate on data stored in the GPU's global memory. This kind of memory is off-chip, and features huge latency times. A kernel call has the following form:

```
kernelName<<< gridDim, blockDim >>> ( .. parameter list..);
```

Where gridDim and blockDim respectively contain the dimensions of the CUDA grid and of the blocks.

The image is two dimensional and so execution of the kernel creates gridDim.x*gridDim.y thread blocks, and blockDim.x*blockDim.y threads within each block. Each thread has its own id, thus making it possible to assign a specific cell to each thread. Each image is two-dimensional that can be mapped to CUDA threads. In our problem, there are 16 blocks in the grid and also 64 threads in each block. The image contains 16 sub-images and the total number of image-blocks within the sub-image is set to 64. In order to get this aim, gridDim.x and gridDim.y can be set to 4 and blockDim.x and blockDim.y are set to 8. So for EHD extraction, each sub-image is mapped to one block and each image-block is processed by one thread. Therefore edge type recognition is performed in all image-blocks in parallel by CUDA threads. In fact, the kernel is responsible for edge detection in the image-block as explained in previous sub section. The image itself and the Bin-Count are input arguments of the kernel. All the threads run the same kernel instructions but with different data. Thread position and image-block which should be processed by the thread can be determined by the blockIdx and threadIdx as :

```
data_per_thread_x = image_height / (gridDim.x*blockDim.x);
data_per_thread_y = image_width / (gridDim.y*blockDim.y);

row_index = (blockIdx.x*blockDim.x +
             threadIdx.x)*data_per_thread_x;

column_index = (blockIdx.y*blockDim.y +
                threadIdx.y)*data_per_thread_y;
```

Where image_height and image_width are the horizontal and vertical image resolutions, and row_index, column_index are two indexes to the image matrix that specify the beginning of the image-block. Threads of each block compute number of edge types in a sub-image. When a thread specifies an edge, it increases relative elements in Bin-Count array. So each image is processed by 16×64 parallel threads to extract edge features.

It is important to note that there are factors limiting the speed up such as:

- The image should be copied to GPU global memory before execution of the kernel and Bin-Count array should be copied to CPU's memory after execution of the kernel. The data communication overhead has a significant impact on the speed up.
- Since edge type counters (Bin-Count) is shared among threads of within a block, these data serially should be updated. However this bottleneck is improved using shared memory.
- Logical operations may force the threads within a warp to be serialized, reducing the speed up.

IV. EXPERIMENTAL RESULTS

The performance of retrieval of the system can be measured in terms of its recall and precision. Recall measures the ability of the system to retrieve all the models that are relevant, while precision measures the ability of the system to retrieve only the models that are relevant. Precision and recall are defined as:

$$\text{Pr}\,ecision=\frac{\text{Number of relevant images retrieved}}{\text{Total number of images retrieved}} \qquad (7)$$

$$\text{Re}\,call=\frac{\text{Number of relevant images retrieved}}{\text{Total number of relevant images}} \qquad (8)$$

Flower query

Fig. 5. One query response example of the presented system.

Man query

Fig. 6. The top 4 retrieved images in the presented system.

The average precision for the images that belongs to the qth category (A_q) has been computed by:

$$AP = \sum_{k \in A_q} \frac{p(i_k)}{|A_q|} \tag{9}$$

We use MPEG-7 database which consists of 230 images. Serial implementation of the image retrieval technique is done in C language using a PC with 3rd Generation Intel Core i5 Pentium Processor (2.5GHz) and 4 GB RAM. Parallel implementation obtained an average speed up of 14.74×over the serial implementation when running on a GPU named NVIDIA GeForce GT610M. The GeForce GT610M is an entry-level card with 48 CUDA cores and 900MHz core clock speed. Fig. 5 and Fig. 6 show the results generated from our proposed system that show the efficiency of our proposed approach and have an average retrieval time as 0.036 seconds. These results show that the performance of the proposed method is better than the other methods. The average precision and the average recall of our proposed method are 67.02% and 55% respectively. One graph that describes the performance of the system is the Precision-Recall graph.

It provides a meaningful result when the database is known and has been used be some earlier systems. To evaluate our presented system, we use the Precision-Recall graph. We select all images from each class in the database to use them as queries to calculate the precision and recall. For each image, the precision of the retrieval result is obtained by increasing the number of retrieved images. Fig. 7 show the Precision-Recall graph for the proposed method. From the Figure, we notice that the system has good precision results over the different values of recall. For the proposed method, the maximum average precision of 100% at recall value is 10%, and the precision value decreases to 42.55% at 100% of recall. To evaluate our proposed system, we use each image in our database to be a query image and submit it to the system. We calculate the precisions for each query in all classes. Then we take the average of all calculated precisions as shown in Table I. Recall, precision, Average Precision (AP) and Average Recall (AR) for the proposed method are in Table I. Content based retrieval method manages to find images similar to query image in database but with a drawback of a lot of time consumption.

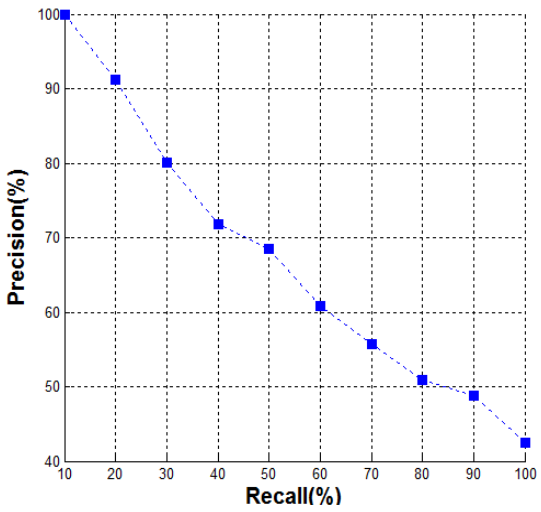

Fig. 7. The average precision/recall chart of the presented system.

TABLE I. PRECISION AND RECALL OF THE PROPOSED METHOD

Recall (%)	Precision (%) for the proposed method
10	100
20	91.15
30	80.13
40	71.74
50	68.43
60	60.78
70	55.73
80	50.88
90	48.78
100	42.55
AR = 55%	**AP = 67.02%**

To make faster the method, we parallelized it on CUDA and achieved an average speed up of 14.74×(approx.) over the serial implementation when running on a GPU.

TABLE II. EXECUTION TIME SERIAL OVER PARALLEL IMPLEMENTATION

Resolution (a×a)	CPU Runtime [s]	GPU Runtime [s]	Speed up	Speed up Average
300	0.323	0.027	11.96	14.74
400	0.514	0.034	15.12	
500	0.822	0.048	17.13	

The comparison of serial implementation over parallel is shown in Table II. Table II also shows that execution time depends on the image resolution for 1000 time iterations, with the optimized code. This Table reports the speed up and the speed up average for varying the problem sizes.

V. CONCLUSION AND FUTURE WORK

In this paper, parallel implementation was performed for Edge Histogram Descriptor using CUDA programming model to run on GPU. This descriptor describes the distribution of various types of edges with a histogram that can be a tool for image matching. Experimental results showed that parallel implementation obtained an average speed up of 14.74×over the serial implementation. Large color database of 230 images were used for check the retrieval performance. Our method was evaluated using precision, recall, and average precision measures.

The main purpose of this research is parallel implementation of EHD extraction on the GPU (GeForce GT610M), as compared to serial implementation on the CPU. Since our case study is an intensive data-parallel application, it is well-suitable to implement on a many-core architecture like the GPU. However, CPU code can be implemented using Open-MP to run on a multi-core CPU in parallel, achieving higher performance. This can be considered as a new research direction.

Furthermore, the work can be extended for translation, rotation, and scale invariance properties, so that the retrieval efficiency can be more increased.

REFERENCES

[1] J. Yue, Z. Li, L. Liu, and Z. Fu, "Content-based image retrieval using color and texture fused features", Mathematical and Computer Modeling 54, pp. 1121-1127, 2011.

[2] D. Zhang, A. Wong, M. Indrawan, and G. Lu, "Content-based Image Retrieval Using Gabor Texture Features", pp. 1-4.

[3] M. Kokare, P. Biswas, and B. Chatterji, "Texture image retrieval using new rotated complex wavelet filters", Journal of IEEE Transaction on Systems, Vol. 35, No. 6, pp. 1168-1178, 2005.

[4] C. Won, D. Park, and S. Park, "Efficient use of MPEG-7 edge histogram descriptor", ETRI Journal, Vol. 24, No. 1, pp. 23-30, February 2002.

[5] H. Jang and K. Jung, "Neural network implementation using CUDA and OpenMP", In Proceedings of Computer: Techniques and Application, pp. 155-161, 2008.

[6] F. Yi, I. Moon, J. Lee, and B. Javidi, "Fast 3D Computational Integral Imaging Using Graphics Processing Units", IEEE Journal & Magazins, pp. 714-722, 2012.

[7] S. Walsh, M. Saar, P. Bailey, and D. Lilja, "Accelerating geoscience and engineering system simulations on graphics hardware", Computer & Geoscience 35, pp. 2353-2364, 2009.

[8] D. Donno, A. Esposito, L. Tarricone, and L. Catarinucci, "Introduction to GPU Computing and CUDA Programming: A Case Study on FDTD", IEEE Antennas and Propagation Magazine, Vol. 52, No. 52, pp. 116-122, June 2010.

[9] P. Sattigeri, J. Thiagarajan, K. Ramamurthy, and A. Spanias, "Implementation of a fast image coding and retrieval system using a GPU", Emerging Signal Processing Applications, pp. 5-8, 2012.

[10] NVIDIA Corporation, NVIDIA CUDA C Programming Guide 4.1, 2011.

[11] M. Swain and D. Ballard, "Color Indexing", International Journal of Computer Vision, pp. 11-32, 1991.