Scene Recognition Acceleration using CUDA and OpenMP

Yuxin Wang, Zhen Feng, He Guo, Changqin He, Yuansheng Yang
Department of Computer Science and Technology
Dalian University of Technology
Dalian, 116024, China
wyx@dlut.edu.cn

***Abstract*—Scene recognition has become a remarkable field in image processing area, and many methods have been proposed in recent years, in which the idea of extracting the scene gist from global features has been proved to have higher retrieval accuracy compared with many other methods. However, the process of extracting gist is heavily time-consuming and not suitable for real-time application. In this paper, the CUDA architecture is deployed to accelerate this process. The influences of data transfer and memory access pattern on performance have also been investigated. Further more, when all the gist descriptors are got, parallel image feature similarity comparison is implemented with the OpenMP architecture. Experiments illustrate obvious speedup compared with implementation using only CPU.**

Keywords-Scene Recognition; gist Descriptor; CUDA; OpenMP

I. INTRODUCTION

Scene recognition is the process of retrieving the features of scenes and evaluating the semantic similarity between scenes. Features are properties of the image extracted with image processing algorithms, such as color, texture, shape, and edge information [1]. Many methods based on these features have been presented for effective scene recognition. In this paper, we introduce scene gist method, which is proposed by Oliva and Antonio [2] [3]. This method is based on the Gabor wavelet transform of texture feature and it is reported to have a high performance in scene recognition [4].

However, the process of extracting gist includes many convolution operations, which are heavily time-consuming, and this shortcoming badly limits this method's application in real time. In this paper, we deploy the CUDA architecture, which utilizes the powerful parallel computation capacity of GPU. Convolution operation is transformed into series of Fourier transform, inverse Fourier transform and matrix multiplication operations according to convolution theorem. After that, CUDA can be applied to accelerate these operations, consequently accelerate gist extraction process. In this paper, we also investigate the influences of data transfer and memory access pattern on performance in the CUDA-acceleration process.

Moreover, when the gist descriptors of all images in database are got and stored. The next step is to compute the degree of similarity between the gist of query image and each one in the gist database so that we can obtain some images that have the similar scene to the query image. This stage is essential for real-time application and should be finished in a very short time. As a result, .the multi-threads on multi-cores strategy is implemented with OpenMP.

II. CUDA-ACCELERATED GIST EXTRACTION

A. Building the Gist of a Scene

Scene gist method is based on the Gabor [5] wavelet transform of texture feature, the two-dimension Gabor kernel is defined as:

$$g(x,y)=(\frac{1}{2\pi\sigma_x\sigma_y})\exp\left[-\frac{1}{2}(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2})+2\pi jW_x\right] \quad (1)$$

Trough making Gabor kernel do proper scaling and rotation transformation, we can get a set of Gabor kernels, as shown in (2):

$$g_{mn}(x,y)=a^{-m}g(x',y'),a>1,m,n\in Z \quad (2)$$

where $x=a^{-m}(x\cos\theta+y\sin\theta)$, $y=a^{-m}(x\sin\theta+y\cos\theta)$. In (2), $\theta=n\pi/k$ and k is the number of orientations, $n\in[0,k)$, a^{-m} is the scale factor to ensure energy invariance.

The Gabor kernels of a gist descriptor is built from 8 oriented edge responses at 3 scales, which results in a total of 24 kernels.

Given an image I, its Gabor wavelet transform can be defined as:

$$W_{mn}(x,y)=\iint I(x,y)g^*_{mn}(x-x_1,y-y_1)dx_1dy_1 \quad (3)$$

It is a convolution operation about image I and Gabor kernel g_{mn}, this operation costs lots of time as mentioned above. We generalize (3) into:

$$W_{mn}=I*g_{mn} \quad (4)$$

where * presents the convolution operation.

After all the W_{mn} from 8 oriented edge responses at 3 scales are computed, the PCA dimensionality reduction is applied to get the gist descriptor of the image.

B. CUDA Acceleration Strategy

According to convolution theorem [6], time-consuming convolution operation can be transformed into series of Fourier transform, inverse Fourier transform and matrix multiplication operations. In this way, (4) can be simplified into:

$$W_{mn} = I * g_{mn} = ifft(fft(I) \cdot fft(g_{mn})) \qquad (5)$$

Then we can deploy CUDA to accelerate these operations, consequently accelerate gist extraction process.

CUDA is a general purpose parallel computing architecture that leverages the parallel compute engine in NVIDIA graphics processing unit (GPU) to solve many complex computational problems in a fraction of the time required on a CPU [7]. It includes the CUDA Instruction Set Architecture (ISA) and the parallel compute engine in the GPU. To program to the CUDA architecture, developers can, today, use C, one of the most widely used high-level programming languages, which can then be run at great performance on a CUDA enabled processor.

In the CUDA architecture, serial code executes on the Host (CPU), while parallel code executes on the Device (GPU) [8]. Then utilizing the powerful parallel computation capacity of GPU, CUDA can obviously accelerate the Fourier transform, inverse Fourier transform as well as matrix multiplication, as shown in Figure 1-2.

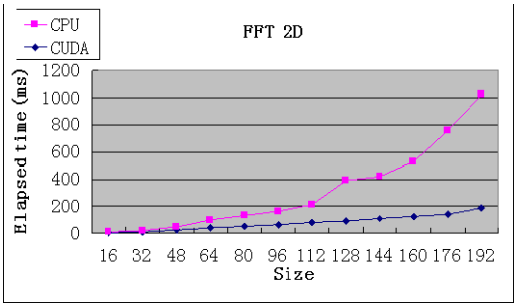

Figure 1. FFT 2D Computation Efficiency Comparation

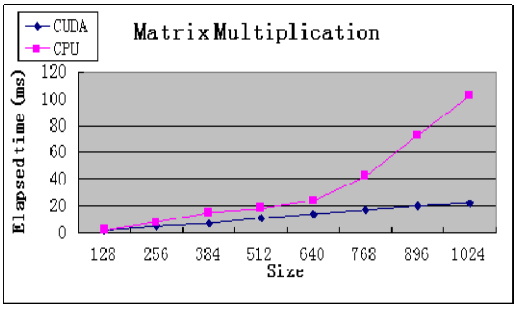

Figure 2. Matrix Multiplication Computation Efficiency Comparation

All the results are got on a computer of Intel Core 2 Quad Q8200 CPU (2.33 GHz), 2GB memory and NVIDIA GeForce 9800 GTX+.

In the acceleration process, another aspect we investigate is the cost for data transfer between host and device, especially when we intend to process millions of images for their gist vectors. The bandwidth between the device and the device memory is much higher than the bandwidth between the device memory and the host memory, and also because of the overhead associated with each transfer, batching many small transfers into a big one always performs much better than making each transfer separately. In our experience, when we transfer 1000 images with the size of 1024×1024 one by one from host memory to device memory, it costs 7032 ms, while when all the data is transferred in batch, the time it costs is so tiny that it can be neglected totally.

The third point to take into consideration is the device memory access pattern. Since device memory is of much higher latency and lower bandwidth than on-chip memory, device memory accesses should be minimized. A typical programming pattern is to stage data coming from device memory into shared memory.

For example, a CUDA-implemented matrix multiplication with the memory access pattern strategy is illustrated in Figure 3, C_{sub} is equal to the product of two rectangular matrices: the sub-matrix of A of dimension (w_A, block_size) that has the same line indices as C_{sub}, and the sub-matrix of B of dimension (block_size, w_A) that has the same column indices as C_{sub}. In order to fit into the device's resources, these two rectangular matrices are divided into as many square matrices of dimension block_size as necessary and C_{sub} is computed as the sum of the products of these square matrices. Each of these products is performed by first loading the two corresponding square matrices from global memory to shared memory with one thread loading one element of each matrix, and then by having each thread compute one element of the product. Each thread accumulates the result of each of these products into a register and once done writes the result to global memory. By blocking the computation this way, we take advantage of fast shared memory and save a lot of global memory bandwidth since A and B are read from global memory only (w_A / block_size) times.

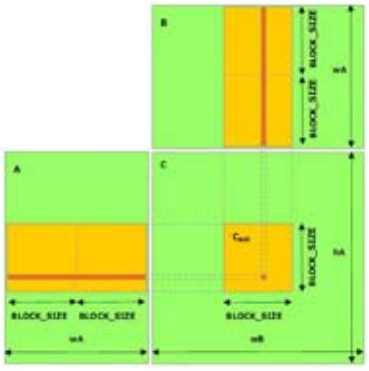

Figure 3. Memory Access Pattern for Matrix Multiplication

Thus the CUDA-accelerated strategy for gist extraction is given by:

Step 1: Compute all the Gabor kernels on the host CPU.

Step 2: Read all the images to the host CPU.

Step 3: Send Gabor kernels and image data to device memory in batch.

Step 4: Compute gist vectors of all the images on the device GPU.

Step 5: Send gist vectors back to host memory in batch.

Then all the gist vectors are stored in gist database for further scene recognition applications.

III. OPENMP-ACCELERATED SCENE MATCHING

When all the gist vectors are stored in image database, a real-time scene recognition application can be performed afterward.

Given a query image, in order to find the most semantically similar scenes, its gist descriptor should be computed at first, this process can be accelerated by CUDA. And then the Euler distances between its gist and all the gist vectors stored in image database are evaluated. The images with the smallest Euler distances can be considered as semantically similar with the query image. As demonstrated in Figure 4, a set of semantically similar images with the query image are selected and demonstrated in the background.

Figure 4. Scene matching results, the query image is highlighted in center.

It is essential that it should be a real-time process to compare the relativity between the gist of query image and each one in the gist database so that we can obtain some images that are most similar to the query image. In this stage, Euler distances are computed and some sorting algorithms should be applied to find semantic similar scenes. In this paper, the OpenMP architecture is introduced to accelerate these operations.

The OpenMP is a set of directives for C, C++, and Fortran programs that make it easier to express shared-memory parallelism, which was released in 2005[9][10]. The advent of commodity inexpensive multi-core processors and corresponding OpenMP-capable complier has recently increased the popularity of OpenMP. The OpenMP consists of two teams: Master and Slave, and an implementation of multithreading whereby the master "thread" forks a specified number of slave "threads" and a task is divided among them. The threads then run concurrently, with the runtime environment allocating threads to different processors. Figure 5 shows an illustration of multithreading where the master thread forks off a number of threads that execute blocks of code in parallel.

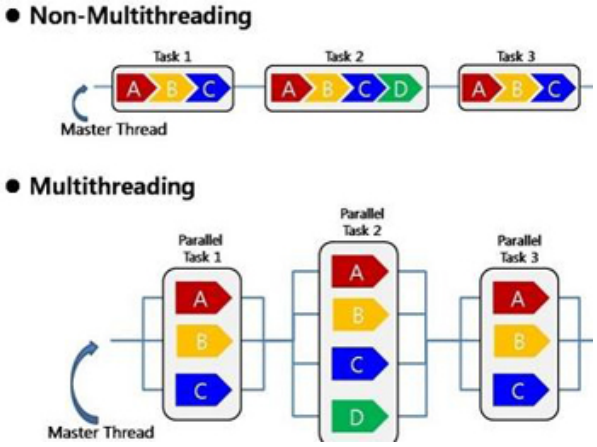

Figure 5. Architecture of OpenMP

We simulated the process of similarity comparison with difference number of images. Results are demonstrated in Figure 6. From Figure 6 we can concluded that the acceleration effect with OpenMP is very obvious especially when the number of images is huge, such as when there are millions gist vectors to be analyzed, just as applications in practice, it takes only 3.11s to complete this process, which is suitable for real-time requirements. On the whole, the similarity comparison with OpenMP shows nearly four times faster than implementation with CPU only.

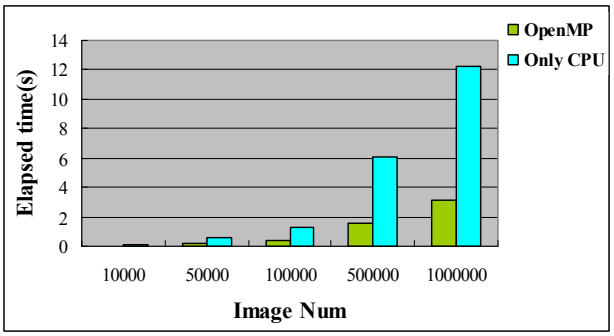

Figure 6. Elapsed times for Scene Matching

IV. EXPERIMENTS RESULTS

A. Gist Extraction

We measured the elapsed times of computing gist vectors of images with the size of 128×128. Results are demonstrated in Figure 7. All the results are got on a computer of Intel Core

2 Quad Q8200 CPU (2.33 GHz), 2GB memory and NVIDIA GeForce 9800 GTX+.

From Figure 7, we can observe that gist extraction with CUDA architecture shows about 3 times faster than implementation with CPU only.

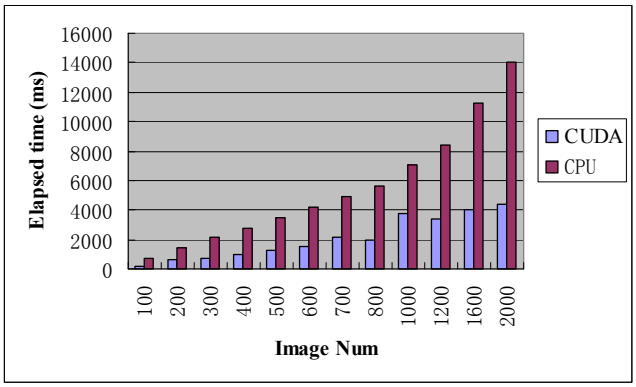

Figure 7. Computational times for Gist Extraction

B. Scene Matching

We suppose a gist database has been constructed and a query image is submitted to find semantic similar images in the database. Gist of the query image can be extracted with or without CUDA, and then the scene matching process is performed by using OpenMP or not using OpenMP. Results are illustrated in Figure 8. When the number of gist vectors in database increases to a certain degree, OpenMP helps to reduce computational times greatly. We can see when image number is 10^6, the elapsed time using only CUDA is relatively high, which obviously proves OpenMP's significance.

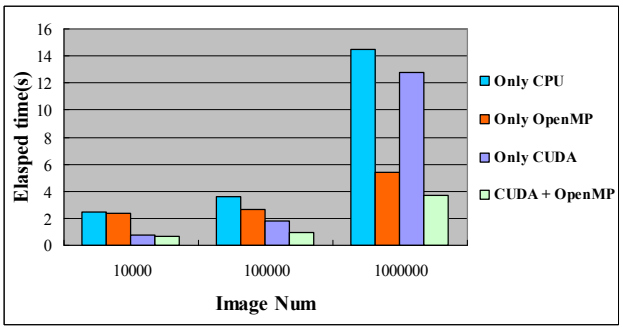

Figure 8. Computational times for Scene Matching.

In this experiment, we also observed that the CPU utilization ratio is below 5% when CUDA architecture is deployed, and it increases to above 25% when the multi-core CPU is used only.

V. CONCLUSION

This paper proposes a faster and more efficient multithreaded implementation on both general purpose graphic processing units and multi-core CPU. Based on the proposed architecture, we implement the Gist Scene Recognition Method where CUDA is deployed for gist extraction and OpenMP is applied for real-time similarity comparison stage. The experiments evaluate the proposed implementation, and show faster computational times on the proposed architecture than on only CPU.

REFERENCES

[1] John M. Zachary, Jr. and Sitharama S. Iyengar, "Content Based Image Retrieval Systems," ASSET '99. Proceedings. 1999 IEEE Symposium on, pp. 136-143.

[2] Torralba A., Murphy K.P., Freeman W.T., AND Rubin M. A., "Context-based vision system for place and object recognition," In ICCV, 2003. Vol. 1, pp. 273-280.

[3] OLIVA A, TORRALBA A., "Building the gist of ascene: The role of global image features in recognition," In Visual Perception, Progress in Brain Research, vol. 155, 2006.

[4] James Hays, Alexei A. Efros, "Scene Completion Using Millions of Photographs," Communications of the ACM, Vol. 51, pp. 87-94, October 2008.

[5] Manjunath, B.S. Ma, W.Y. "Texture features for browsing and retrieval of image data, ", Pattern Analysis and Machine Intelligence, IEEE Transactions on, Vol. 18, pp. 837-842, August 1996.

[6] H. M. Ozaktas, B. Barshan, D. Mendlovic, and L. Onural, "Convolution, filtering, and multiplexing in fractional Fourier domains and their relationship to chirp and wavelet transforms," J. Opt. Soc. Amer. A, vol. 11, pp. 547-559, 1994..

[7] David Luebke. NVIDIA GPU Architecture: Implications & Trends. SIGGRAPH 2008 Beyond Programmable Shading Course. August 14, 2008.

[8] NVIDIA CUDA Compute Unified Device Architecture Programming Guide, V2.1, Decmber 2008.

[9] Honghoon Jang, Anjin Park, Keechul Jung, "Neural Network Implementation using CUDA and OpenMP," Computing: Techniques and Applications, 2008, pp. 155-161.

[10] Shah Sanjiv, Bull Mark, "OpenMP ," 2006 ACM/IEEE Conference on Supercomputing, SC'06.