REAL TIME FACE DETECTION ON GPU USING OPENCL

Narmada Naik[1] and Dr Rathna.G.N[2]

[1,2]Department of Electrical Engineering
Indian Institute of science
Bangalore-560012
India
[1]nnreema22@gmail.com
[2]rathna@ee.iisc.ernet.in

ABSTRACT

This paper presents a novel approach for real time face detection using heterogeneous computing. The algorithm uses local binary pattern (LBP) as feature vector for face detection. OpenCL is used to accelerate the code using GPU[1]. Illuminance invariance is achieved using gamma correction and Difference of Gaussian(DOG) to make the algorithm robust against varying lighting conditions. This implementation is compared with previous parallel implementation and is found to perform faster.

KEYWORDS

Heterogeneous Computing, OpenCL, Image Processing, LBP, Histogram, Classifier.

1. INTRODUCTION

Face detection finds its application in various research discipline such as image processing, computer vision, pattern recognition. Face detection can be done using many algorithms such as Eigen faces, Fisher faces, Local Binary Pattern etc. Applications of human face detection algorithms, such as computer assisted surveillance, need to be in real time. These algorithms, being highly parallel, are more suited for platforms like GPU which have an inherent parallel execution architecture and higher computing capacity compared to CPU. In this paper we have implemented a parallelized variant of LBP on GPU using OpenCL, a heterogeneous computing platform.

In this paper Face detection is done based on efficient grey scale invariant texture Local Binary Pattern (LBP) Histogram, which is parallelized and processed by using Heterogeneous CPU-GPU based platform. The work is based on converting colored images(captured from camera) to grey scale, preprocessing the image and extraction of LBP feature and its histogram on GPU to reduce the computation time. LBP operator is a computationally simple and efficient texture analysis operator which labels the pixels of an image by thresholding the neighbourhood of each pixel and considers the result as a binary number. In real time application the success of LBP is due to its uncomplicated computation and robustness due to illumination variations [2].

Rest of the paper is organized into five more sections. Section 2 gives a brief overview about Heterogeneous computing and OpenCL. In Section 3, we have discussed the LBP algorithm.

David C. Wyld et al. (Eds) : CCSIT, SIPP, AISC, PDCTA, NLP - 2014
pp. 441–448, 2014. © CS & IT-CSCP 2014 DOI : 10.5121/csit.2014.4238

Section 4 discusses the implementation details on GPU. Experimental results are shown in section 5. In Section 6, we give a brief conclusion of the paper.

2. HETEROGENEOUS COMPUTING WITH OPENCL

OpenCL[3] is an industry standard writing parallel programs targeting heterogeneous platforms. In this section a brief overview of heterogeneous computing with OpenCL programming model is given. Programming model of OpenCL is classified into 4 models[4] Platform model, Execution model, Memory model, Programming model.

2.1. platform model

The OpenCL platform model defines a high-level representation of any heterogeneous platform used with OpenCL. This model is shown in the Fig 1. The host can be connected to one or more OpenCL devices(DSP, FPGA, GPU, CPU etc), the device is where the kernel execute. OpenCL devices are further divided into compute units which are further divided into processing elements(PEs), and computation occurs in this PEs. Each PEs is used to execute an SIMD.

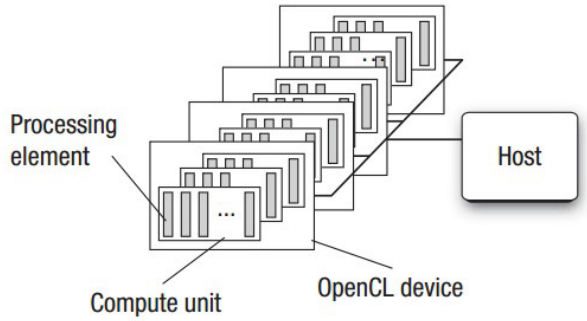

Fig. 1. OpenCL Platform Model .

2.2. Execution model

OpenCL execution consist of two parts - host program and collection of kernels. OpenCL abstracts away the exact steps for processing of kernel on various platforms like CPU, GPU etc. Kernels execute on OpenCL devices, they are simple functions that transforms input memory object into output memory objects. OpenCL defines two types of kernel, OpenCL kernels and Native kernels.

Execution of kernel on a OpenCL device:

1. Kernel is defined in the host,
2. Host issues a command for execution of kernel on OpenCL device,
3. As a result OpenCL runtime creates an index space.
4. An instance of the kernel is called work item, which is defined by the coordinates in the indexspace (NDRange) as shown in Fig 2.

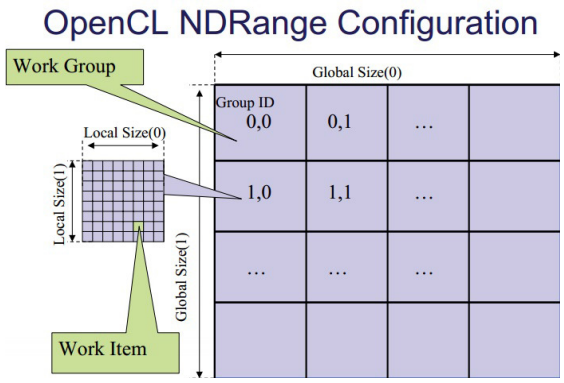

Fig. 2. Block diagram of NDRange.

2.3. Memory model

OpenCL defines two types of memory objects, buffer objects and image objects. Buffer object is a contiguous block of memory made available to the kernel, whereas image buffers are restricted to holding images. To use the image buffer format OpenCL device should support it. In this paper buffer object is used for face detection. OpenCL memory model defines five memory region:

- **Host memory:** This memory is visible to host only.
- **Global memory:** This memory region permits read/write to all work items in all the work groups.
- **Local memory:** This memory region, local to the work group, can be accessed by work items within the work group.
- **Constant memory:** This region of global memory remains constant during execution of the kernel. Workitems have read only access to these objects.
- **Private memory:** This memory region is private to a work item i.e variables defined private in one work item are not visible to the other work item. Block diagram of memory model is shown in Fig 3.

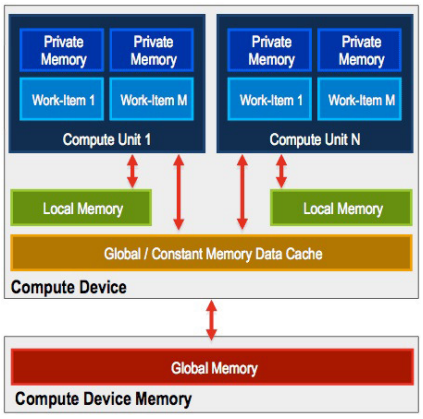

Fig. 3. Block diagram for memory model

2.4. Programming model

Programming model is where the programmer will parallelize the algorithm. OpenCL is designed both for data and task parallelism. In this paper we have used data parallelism which will be discussed in section 4.

Basic work flow of an application in OpenCL frame work is shown below in block diagram Fig.4

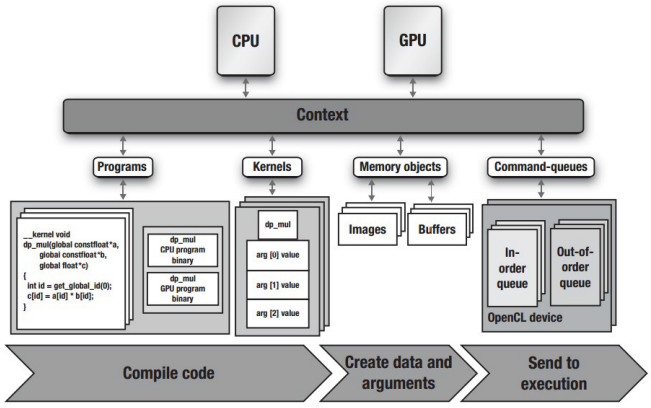

FIG. 4. WORK FLOW DIAGRAM OF OPENCL.

Here we start with the host program that defines the context. The context contains two OpenCL devices, a CPU and a GPU. Next the command queue is defined, one for GPU and the other for CPU. Host program then defines the program object to compile and generate the kernel object both for OpenCL devices. After that host program defines the memory object required by the program and maps them to the arguments of the kernel. Finally the host program enqueue the commands to the command queue to execute the kernels and then the results are read back.

3. OVERVIEW OF THE ALGORITHM

In this section it is discussed about how the image is captured from camera and converted to grey scale used gamma operation and DOG operation in preprocessing[1] and LBP feature extraction, Histogram and classifier.

3.1 Face detection using LBP:

LBP operator labels the pixel by thresholding the 3x3 neighbourhood of each pixel with the center value. This generic formulation of the operator puts no limitations to the size of the neighbourhood [5]. Here each pixel is compared to its 8 neighbours (on its left-top, left-middle, left bottom, right-top, right-middle, right-bottom) followed in clockwise direction. Wherever the center pixel value is greater than the neighbour write 1 to the corresponding neighbour pixel otherwise write 0. This gives an 8-digit binary number as shown in Fig.5. This 8-digit binary number is then converted to a decimal.

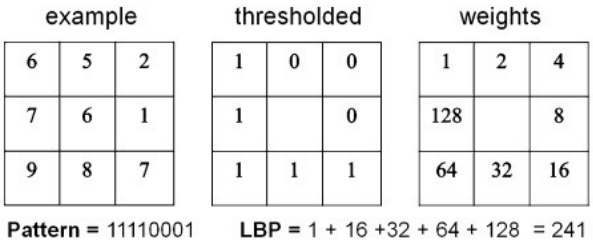

Fig. 5. LPB Thresholding

After getting the LBP of the block , histogram of each block is calculated in parallel and is concatenated as shown in Fig.6. This gives the feature vector used for training the classifier. LBP operator[6] is defined as

$$LBP_{P,R}(x_c, y_c) = \sum_{p=0}^{p-1} s(g_p - g_c)2^p$$

where g_p is the intensity of the image at the p_{th} sample point where p is the total number of the sample point at a radius of R denoted by(P,R). The P spaced sampling points of the window are used to calculate the difference between center gc and its surrounding pixel[5]. The feature vector of the image obtained after cascading the histogram is,

$$H_s(p,k) = \sum_{i=1}^{I}\sum_{j=1}^{J} f(LBP_{P,R}, P)$$

where k is an integer to represent the sub histogram that is obtained from each block k=1,2...K. K is the total no of histograms ,and f(x,y) = $\begin{pmatrix} 1, x = y \\ 0\ otherwise \end{pmatrix}$ where f(x; y) is the LBP calculated value at pixel (x,y).

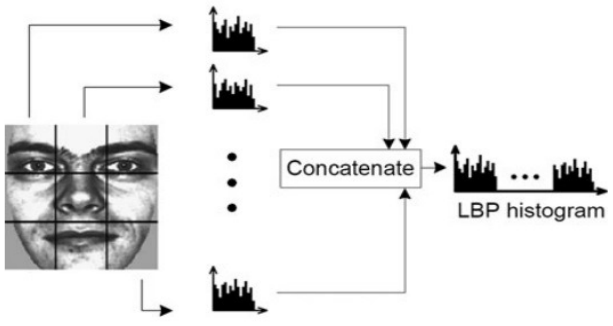

Fig. 6. LBP in Face detection

3.2. classifier

There are many methods to determine the dissimilarity between the LBP pattern, here chi-square method is used and further work is going on with SVM for training and classifying the LBP feature. The chi-square distance used to measure the dissimilarity between two LBP images S and M is given by

$$D(S,M) = \sum_{i=1}^{L} (S_x - M_x)^2/(S_x + M_x)$$

where L is the length of the feature vector of the image and S_x and M_x are respectively the sample and model image in the respective bin.

4. IMPLEMENTATION

The goal of this paper is to implement LBP algorithm on a cpu, gpu based heterogeneous platform using OpenCL, to reduce computation time. The process of LBP feature extraction and histogram calculation from the image is computationally expensive (N^2xW^2, where NxN is size of image and WxW is size of LBP block) and it is easy to figure out that the extraction in different parts are independent as discussed in section 3. Thus it can be efficiently parallelized [7].

For real time implementation, first the image is captured using OpenCV and is converted to a grey scale image. Then preprocessing of the image is done. To figure out different features in the image, various algorithms are implemented on the image. For parallel processing of the algorithm, the image is subdivided into smaller parts. In this case the image is divided into 16 x 16 pixels blocks. The task of calculating LBP for each block is given to work items. So there are different work items processing different blocks in parallel. Each work item processes 256 pixels and different work items work in parallel, reducing the processing time up to a significant extent. So, to calculate the LBP of 16x16 pixels blocks the image is converted to an one dimensional array. A global reference of the image is used by each work item for creating one 18 x 18 two dimensional matrix to find out the LBP for each block. From the 18 x 18 matrix, LBP is calculated as discussed in section 3 for 16x16 pixels not considering the boundary pixel of 18 x 18 matrix . Afterwards the calculated values for LBP are processed to form a histogram ranging from 0 to 255. Each work item formulates a histogram accordingly for one block and the different histograms are cascaded to form the actual histogram for the complete image in order to get the LBP feature vector as shown in Fig.7. This feature vector is then classified using nearest neighbourhood method.

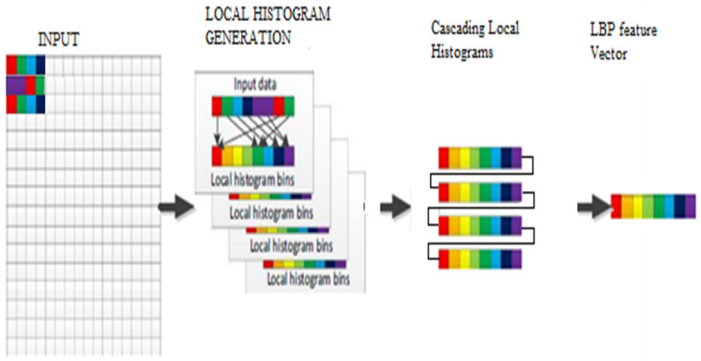

Fig. 7. Histogram Calculation on GPU

The Block Diagram Of The Overall Method Is Shown In Fig.8

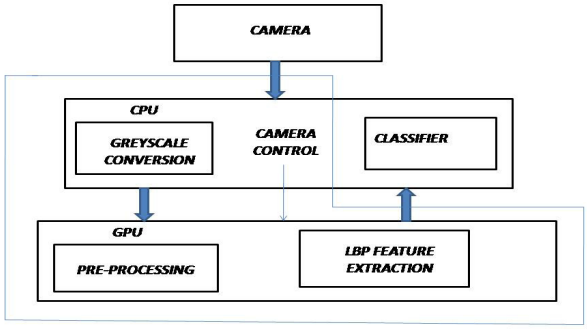

Fig. 8. Block diagram of the overall method

5. RESULTS

In the proposed paper we have calculated each block in different compute units. Since the calculation of histogram depends on all the pixels within a block thus it is better to do the whole calculation within one compute unit. Additionally, the amount of computation per compute unit shouldn't be too small otherwise the overhead associated with managing a compute unit will be more than the actual computation. Since the whole computation is done in the GPU and only the input image and the final histogram are transferred between the CPU and GPU thus overheads associated with data transfer are minimal. As a result the computational time is 20 ms. The performance table of the implementation is shown in Table 1.

Table 1. Performance Table

Input Resolution	640x480
Sub Histograms	256
CPU(i5 3rd generation)	109 ms
AMD(7670M)	20 ms

Table 2. Comparison With Previous Work

	PREVIOUS WORK[8]	**OUR WORK**
Image size	512x512	640x480
Sub Histograms	256	256
Feature Extraction	36.3 ms	20 ms

Compared to previous work[8], the image is grabbed from camera and processed in real time. As can be seen from TABLE 2, computational time for our implementation is less. Performance of the proposed paper is tested on AMD 7670M GPU, i5 3rd generation CPU based system. Total time for feature extraction on CPU was 108 ms and on GPU was 20 ms for 640X480 resolution input image. Thus we get a 5x improvement in speed using GPU implementation.

6. CONCLUSION

In this paper real time face detection using LBP feature extraction is done and is classified using nearest neighbourhood method. We have parallelized the existing LBP algorithm to make it suitable for implementation on SIMD architecture such as GPGPU. Performance gain has been achieved over previous implementations.

REFERENCES

[1] Xiaoyang Tan; Triggs, B., "Enhanced Local Texture Feature Sets for Face Recognition Under Difficult Lighting Conditions," Image Processing, IEEE Transactions on , vol.19, no.6, pp.1635,1650, June 2010 doi: 10.1109/TIP.2010.2042645

[2] Computational Imaging and Vision, Vol. 40 Pietikinen, M., Hadid, A.,Zhao, G., Ahonen, T. 2011, XVI, 212 p.

[3] KHRONOS:OpenCLoverviewwebpage, http://www.khronos.org/opencl/,2009.

[4] Aaftab Munshi, Benedict Gaster, Timothy G. Mattson, James Fung, Dan Ginsburg ISBN: 978-0-3217 4964-2

[5] TY - JOUR T1 - Facial expression recognition based on Local Binary Patterns: A comprehensive study JO - Image and Vision Computing VL - 27 IS - 6 SP - 803 EP - 816 PY - 2009/5/4/ T2 - AU - Shan, Caifeng AU - Gong, Shaogang AU - McOwan, Peter W. SN - 0262-8856 DO http://dx.doi.org/10.1016/j.imavis.2008.08.005
UR-http://www.sciencedirect.com/science/article/pii/S0262885608001844 KW - Facial expression recognition KW - Local Binary Patterns KW - Support vector machine KW - Adaboost KW – Linear discriminant analysis KW - Linear programming ER

[6] Caifeng Shan; Shaogang Gong; McOwan, Peter W., "Robust facial expression recognition using local binary patterns," Image Processing, 2005. ICIP 2005. IEEE International Conference on , vol.2, no., pp.II,370-3, 11-14 Sept. 2005 doi: 10.1109/ICIP.2005.1530069

[7] Miguel Bordallo Lpez ; Henri Nyknen ; Jari Hannuksela ; Olli Silvn and Markku Vehvilinen "Accelerating image recognition on mobile devices using GPGPU", Proc. SPIE 7872, Parallel Processing for Imaging Applications, 78720R (January 25, 2011); doi:10.1117/12.872860; http://dx.doi.org/10.1117/12.872860

[8] Parallel Implementation of LBP Based Face recognition on GPU Using OpenCL Dwith, C.Y.N. ; Rathna, G.N. Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2012 13th International Conference on Digital Object Identifier: 10.1109/PDCAT.2012.107 Publication Year: 2012 , Page(s): 755 - 760