N- CS259

Parallel Computing in Image processing

Roshni Velluva Puthanidam
Department of Computer Science
San Jose State University
San Jose, CA 95192-0249
roshni.velluvaputhanidam01@sjsu.edu

***Abstract*— Processing of images with large size is attained greater importance in recent years. They are widely used in various fields including, computer vision, medical imaging, military purposes, weather forecasting, etc. Traditional sequential way of image processing is extremely costly in terms of processing time and resource utilization. Sequential image processing algorithms does not take the advantages of resources available like multi-core CPUs and many core GPUs. Recent researches suggest that using parallel algorithms in image processing improves performance in terms of time and resource utilization. In this paper, I am presenting various image processing techniques, different parallelism in image processing. Various image processing algorithms used are discussed. This paper is also discussing about one of the important image processing application – medical image processing and image processing with CUDA.**

Keywords— medical imaging; image processing; GUP; CUDA

I. INTRODUCTION

Nowadays image processing getting quite good popularity in many application fields including computer vision, film industry, weather forecasting, medical imaging, etc. Most of these fields have very large sized images, need to be processed in very small time and real-time processing might be required sometimes. The traditional approach for image processing and filtering been a time and resource consuming task as the image size increases. Recent works replace sequential image processing algorithms with parallel programming approach. For a high-performance computing model, parallel processing plays an important role. In such cases, it is very important to use many computing resources to solve a difficult problem. CPU and memory are the resources specific to parallel processing. Parallel computing will help to manage a large volume of information in acceptable time. Parallel computing is usually done through a CPU when large number of cores are available [1]. In this case, speedup rate is limited to the number of cores available in the CPU.

We can utilize non-local resources very effectively through the concurrency provided by parallel computing. This will also help to remove the limitation of serial computing. Divide the large problem into the smaller task and then complete them concurrently is the main design idea behind parallel computing. Parallel algorithms can also take all advantages of processors with many numbers of cores because they are capable to work with multiple numbers of threads. This will, in turn, helps to divide the total time among the total tasks. Parallel computing cannot be applied to all problems, in other words, there are some problems that should execute sequentially to get expected result.

Operations applied to images to make a new image is known as image processing. It requires manipulation of every pixel of the image to transform them into a new image. For example, converting a color image to gray scale needs to select each pixel in the image and then operates on its RGB values [2]. It is possible to divide large images into smaller chunks and perform same operations on each chunk. Therefore, the image processing problem is an excellent candidate for parallel computing to increase the speed of processing when multiple images chunks operate at the same time. Thus, image processing is a good parallel computing problem.

Medical imaging is one of the important areas where parallel computing of images is required. Medical image processing is known n for computationally costly and data-rich domain. It is the process of creating images of human body parts for medical needs [3]. CT, PET, MRI, etc. are some of the medical image modalities and each of them have different features and requirements. Analyzing these modalities sequentially and get a result takes a lot of time because of the size of images. So, if we can divide images into several chunks and perform parallel processing on each will give good result in reasonable time. Ie, we can able to perform basic image processing steps like image transformation, edge detection, feature calculation, etc. on an image very quickly and it will be useful for medical employers. So, parallel computing will save both time and resource in an efficient way.

An image is usually represented as a 2D matrix of pixels. In image processing, manipulation is done on each pixel as a matrix element. Parallel computing allows each thread to work on each of the pixels, giving a greater computational performance speed, and resource utilization. However, there are a lot of trade-offs and overhead that must be counted. One of the important reason for the wider acceptance of CUDA (Compute Unified Device Architecture) introduced by NVidia is that overhead and trade-off due to parallel computing is improved. In this paper, we are also discussing CUDA technology to improve image processing performance.

I. IMAGE PRE-PROCESSING TYPES

In this section, I am covering some of the widely use image processing types in various application fields.

1) Color to Gray Scale Conversion: When there is an application like face recognition, medical imaging color of the image does not add any extra information. Processing of color images is generally time- consuming and complex task. By converting color to grayscale provides an easy way to process image if the color information does not add any value. One of the common ways of converting color to grayscale is by doing a modification to each pixel of the image. For converting color to grayscale on needs to obtain RGB (Red, Green, and Blue) component of a pixel and do a mathematical operation [2]. The mathematical equation for calculating grayscale value is:

Gray Scale Value = red * 0.3 + green * 0.59 + blue * 0.11 [2]

Through converting color to gray scale, we should perform operations only on single image plane instead of three (R, G, and B). Figure 1 shows sample conversion from color to gray scale.

Figure 1. Converting color image to gray scale

2) Geometrical Transformation: Geometrical transformation includes scaling and translation of images. It is possible to up or down scale the images according to requirement using a scaling factor. For resizing images, we can either specify scaling factor or we specify the size manually [4]. In some of the application like face recognition, it is a good idea to have down-sampled image (downscaling image without losing important information) to reduce the amount of data and complexity of computation (see Figure 4). It is also possible to shift object location using translation. Flipping of an input image can also be done through translation. Images can be flipped vertically or horizontally by simply copying pixels from top to bottom, left to right or vice versa. The flipping of image makes a mirror copy of the original image (see Figure 3.) [4].

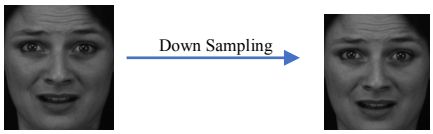

Figure 2. Down sampling Example. Image on the left is 562 * 762 pixels and image on the right is 256 * 256 pixel

Figure 3. Flipping of Image Example [4].

3. Smoothing Images: Smoothing of images are often used to reduce noise in the image. It also helps to generate less pixelated image. This can be usually achieved by using filters. Gaussian Blur, Average Blur and Median Blur are some of the filters used generally (see Figure 4.) [5].

Figure 4. Smoothing of Image Example [4].

4. Edge Detection: Canny Edge detector is the one of the most commonly used edge detection algorithm. It is a multi-stage algorithm which includes Noise Reduction, Finding Intensity gradient of image, Non- maximum suppression and finally Hysteresis thresholding [6]. Initially, smoothen image using any of the filter (generally Gaussian is used). Using gradient equation shown below intensity gradient of the image is calculated.

$$\text{Edge\_Gradient } (G) = \sqrt{G_x^2 + G_y^2}$$

$$\text{Angle } (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

[2]

Gradient always gives perpendicular direction to edges. It is formed to one of four angles representing horizontal, vertical and two diagonal directions (see Figure 5.) [6].

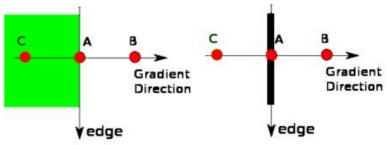

Figure 5. Non-Maximum Suppression [5]

Once gradient magnitude and direction is calculated, to eliminate any undesired pixels which may not form the edge a complete scan of the image is performed. For this, at each pixel, a pixel is checked if it is a local maximum in its neighborhood in the direction of gradient [6]. Then by means of hysteresis, the threshold of the edges is calculated. A sample canny edge detected image is shown in figure 6:

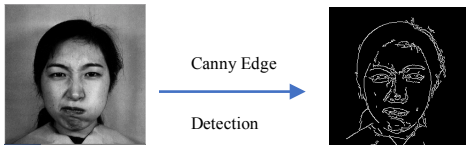

Figure 6. Canny Edge Detection

5. Morphological Transformation: Grayscale morphological operators (e.g., dilation and erosion) can be designed to highlight darker and brighter pixels in the luma component around eye areas. These operations generally used for frontal face authentication - Eye detection algorithm and Mouth detection. In morphological operation, value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors. Dilation makes object in white pixel bigger and erosion makes dark pixels bigger [7] (see Figure 7.).

Figure 7. Sample morphological operation

5. Object Detection: Object detection is one of the important field in computer science. It is commonly used to detect and identify the existence of certain object [8]. Detecting objects like eyes, faces, smiles, cars, and license plates, are all relevant nowadays. In computer vision quite often use face and eye detection (see Figure 8.).

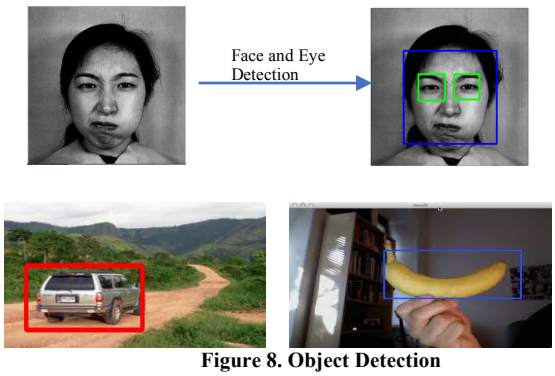

Figure 8. Object Detection

II. PARALLEL COMPUTING

Parallel computation is a type of computation which does multiple computations or executions of programs simultaneously using multiple resources. Problems will be broken into several independent parts that can be solved parallel. Following are the steps to be followed in parallel computing [8].

- Divide problem into discrete parts that can be solved concurrently
- Further divide each part down to a series of instructions
- Each part of the instructions executes concurrently on separate processors
- An overall control/coordination mechanism is employed

1) Features of a parallel program: There are some features to be satisfied for a program to run them parallel and get an accurate and effective result. Otherwise, the problem does not give the expected runtime or performance result [10]. Some of the features are following:

a) Granularity: Granularity is the number of basic units and it is further classified as:
Coarse-grain: Few tasks of higher intense computing.
Fine grain: Many small parts and less intense computing.
b) Synchronization: Synchronization used to limit the overlapping multiple processes.
c) Latency: This is the transition time of data from the request to receipt.
d) Scalability: Scalability is determined as the capability of an algorithm to maintain efficiency by increasing the number of processors and the size of the problem in the same proportion [11].

2) Type of parallel processing [10]:
Explicit: The algorithm contains instructions to specify which processes to build and execute parallel.
Implicit: Compiler have the responsibility to add required instructions to run program on a parallel computer.

3) Methods of Parallel Process [12]: Generally there are three types of parallel image processing methods are available: Task Parallelism, Data Parallelism and Pipelining. The image processing system and method includes multiple image providers configured to transmit images. Multiple destination processors receive this image data and convert them to usable form of image data.

Task parallelism also known as function or control parallelism. It used to parallelize computer codes across multiple processors in parallel processing environments. Ie, different task running on the same data. It concentrates on dividing tasks—concurrently executed by processes or threads—across multiple different processors. There should not be any kind of dependency between the task, so that they can run parallel. In simple words, Task parallelism is the simultaneous execution on multiple cores of many different functions across the same or different datasets. Task parallelism is shown in Figure 9 [12].

Data parallelism(SIMD) is another form of parallelization where data is distributed across multiple processors in parallel computing environments. Data parallelism can apply to any regular data structure like matrices and arrays by working parallel on each element. It can divide data into different parts and apply on different tasks and these tasks can run in parallel. One of the key things to be noted is that dependencies between the tasks that cause their results to be ordered should not be allowed. In simple words, Data parallelism is the simultaneous execution on multiple cores of the same function across the elements of a dataset. Data parallelism is shown is figure 10.

Pipeline in a computing is a set of data processing elements connected in series, where the output of one task is the input of the next. Each of the task can run in parallel. The elements of a pipeline are often executed in parallel or in time-sliced fashion; in that case, some amount of buffer storage is often inserted between elements. Throughput impacted by the longest-latency element in the pipeline [13]. See Figure 11 for pipelining design.

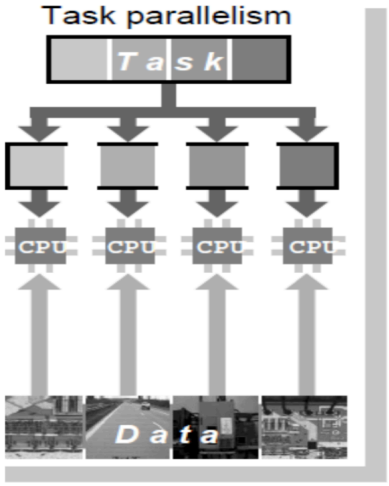

Figure 9. Task parallelism [12]

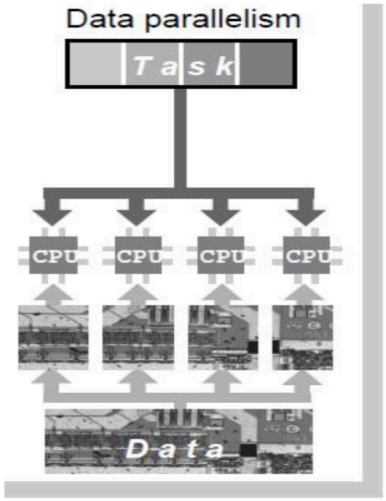

Figure 10. Data parallelism [12]

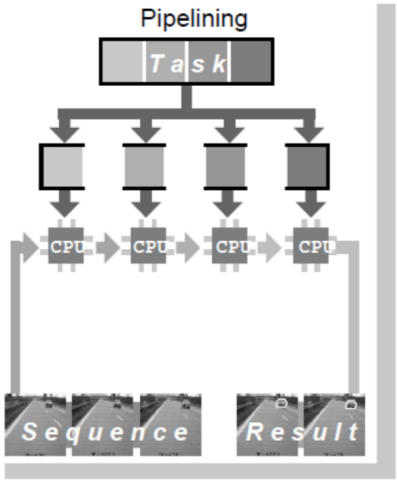

Figure 11. Pipelining [12]

IV. ALGORITHMS FOR PARALLEL IMAGE PROCESSING

Once we have the input image to be processed, defining the term number of tiles to be created is the principal step for this algorithm. The number of threads to be used is corresponding to these number of tiles. If there is only one thread/tile exist in our problem computation will execute sequentially.

Consider figure below 11. a) as our example image and 11. b), 11.c) and 11.d) are the possible way of image division (choose according to the requirements) then send each part to separate processors. If there are more threads than required (ie, number of threads greater than image division into different areas as shown in Figure 11.a), Figure 11.b), Figure 11.c)) that does not give any better result. Only one thread can work on a tile.

Each thread will take care of all task to be executed on their responsible tile pixel. One thing to ensure is that keep all the processors in Synchronization else there will be a deadlock between the processors [10].

Figure 11. a)

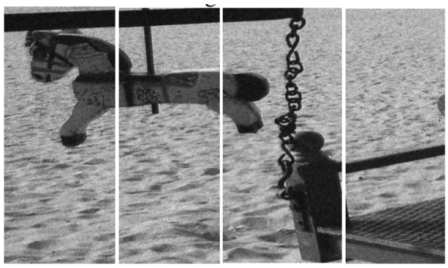

Figure 11. b)

Figure 11. c)

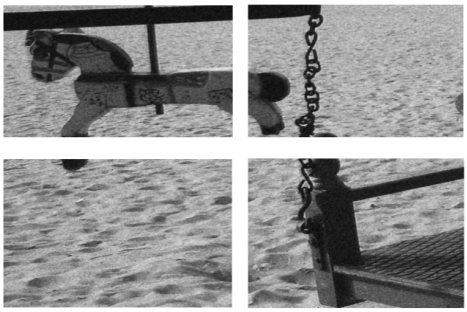

Figure 11. d)

Parallel algorithms used for image processing and result from each of the algorithm in [10] is explained below. They performed all tests on an " Intel Core i3 – 2350M Processor 2.30 GHz, 3 GB of RAM, Hard Disk Drive 320 GB". "MATLAB R2011a and JAVA JDK 1.6.0_21" is used for software implementation, the operating system is 64 bit. They have implemented multi-threaded and explicit parallelism in MATLAB.

Here is some of the parallel algorithms [10] used in image processing:

a. Parallel Segmentation by Region Growing Technique and Calculation of different Features of Segmented Regions

Region growing is one of the very famous techniques of segmentation. Main drawback of the region growing is that it is time consuming. This algorithm is designed to reduce timing of this algorithm. These are the steps involved in it.

1. Take Input image on client processor.
2. Selection of the Seed pixels(It is based on some user criterion like pixels in a certain gray level range, pixels evenly spaced on a grid, etc.) for different regions on client processor.[5]
3. Count number of seed points and activate same number of kernels.
4. Send copy of image and respective seed pixels on each kernel processor.
5. Regions are then grown from these seed pixels to adjacent depending on pixel intensity, gray level texture, or color, etc on each kernel processor for this image information is also important because region grown is based on the membership criteria.
6. Calculation of different features of different ROI (region of interest) on individual kernel.
7. Send segmented regions and information of ROIs to the client processor and deactivation of worker kernels.
8. Reconstruct the image on client processor and Display of all the information.

b. Parallel Segmentation by Global Thresholding and Calculation of Fourier and Regional Descriptors of an Image

This algorithm consists of the following steps.

1. Divide the image into Quad Tree Structure in client processor.

2. Send a flag from client processor to different worker processor and activation of four worker processors.

3. Send all the parts of the image on different kernels or different labs or different worker processors.

4. Chose Initial Thresholds T0, T1, T2, T3 in different labs individually.

5. Calculate the mean value of each lab μ of the pixel below the threshold and the pixel above the threshold value.

6. Compute a new threshold as

$T = (\mu 1 + \mu 2)/2$ in each lab.

7. Repeat steps 5 and 6 until there is no change in threshold in each lab or worker.

8. Send segmented region to client from the different labs.

9. Deactivate worker processors.

10. Reconstruct the Segmented Image.

11. Calculate the Regional and Fourier descriptors of Segmented Image on client processor.

c. Complex Noise reduction using Parallel Computing:

This algorithm consists of the following steps:

1. Take input Image on Input Image on client processor.

2. Generate Copies of the Image on client processor.

3. Activate Number of Kernels or Workers or Labs as per requirement.

4. Send Copy of the Image into different Kernels.

5. All the labs consist of filters for e.g.

Lab 1 – Median filter
Lab 2 - Wiener Filter
Lab 3 – Order statistical filter
Lab 4 – Min – Max filter etc.

6. Perform Convolution and find PSNR and MSE of the image on individual kernels.

7. Send back Values of MSE and PSNR value to the client

8. Compare the result of received data on client processor.

9. Display the image with Highest PSNR and lowest MSE Value.

10. Deactivation of kernels

d. Histogram Equalization of an Image by Parallel Computing

1. Divide the image in quad tree format or in row wise or in column wise format and activate required number of kernel processors.
2. Form the cumulative histogram on each part of image on each kernel processor.
3. Normalize the value by dividing it by total number of pixels.
4. Multiply these value by the maximum gray level value and round off the value.
5. Map the original value to the result of step 3 by a one to one correspondence.
6. Send all equalized histogram to the client processor.

Table 1 shows performance result from images of different size using above explained parallel image processing algorithms. It is evident from the table result is that parallel image processing algorithms performs on an average of 2.5 times better than sequential algorithm.

Table 1. Performance Evaluation[10]

Image	Size	Algorithms	Time(T1) Taken by Sequential Algorithm (in sec)	Time (T2) Taken by Parallel Algorithm (in sec)	Observed Speedup T1/T2
Cameraman.tiff	256 * 256	a	0.3267	0.15691	**2.08**
		b	0.268	0.0956	**2.80**
		c	0.9132	0.9005	1.01
		b	0.259	0.0823	**3.1567**
Bluredtext.jpg	256 * 768	a	1.0896	0.00783	01.08
		b	4.61	0.10005	**2.196**
		c	0.1394	0.06983	1.9965
		d	0.9857	0,3279	**3.006**
Colorleaf.jpg	1281 * 843	a	3.98154	2.991	1.33
		b	4.4722	3.9612	1.129
		c	0.3197	3.665	1.9972
		d	1.793	0.8932	**2.007**

IV. IMAGE PROCESSING USING CUDA (GPU) TECHNOLOGY

As image size and resolution capabilities of multimedia devices increases, we need to improve the overall performance of Image Processing. Then parallel computing for image processing being introduced. Parallel computing can be implemented in graphic processor (GPU) as well as central processor (CPU). In the case of CPUs speed up factor is restricted by the number of coresavailable in CPU.

Traditionally GPU has been used to accelerate computer graphics computations in applications like high-end 3D rendering and video gaming, etc. However, recent researchers used GPUs "in reverse" [14] in image processing part of computer vision application. As shown in figure 12, there is greater difference in speed up when we use GPUs in image processing. There are several fundamental differences between CPU and GPU. To Maximize performance, CPUs optimized to give the higher degree of instruction level parallelism. More than that currently, CPUs have multiple cores to process data in parallel. However, GPUs composed of multiple cores which gives highly parallel architecture. They are responsible for higher degree of data level parallelism. GPUs are designed for SIMD (single instruction, multiple data) machines [16]. Chip design of both CPU and GPU is shown in Figure 13.

Recent studies show that image processing using parallel algorithms on CUDA (compute unified device architecture)-

accelerated CPU/GPU gave potential improvement in time to process images [17]. CUDA provides multi-threaded parallel programming capabilities which increase the speedup factor into 100 to 1000 (restriction on this factor is number cores in GP-GPU). When we process image, processing should be done on the entire image. ie, each pixel of the image must perform the same operation frequently. CUDA is designed for such operations. So, image processing programming model is an excellent candidate for fully utilizing CUDAs architecture and computation capabilities. NVIDIA provides an efficient CUDA architecture to deliver better performance and resource efficiency by means of massive parallel threaded GPUs.

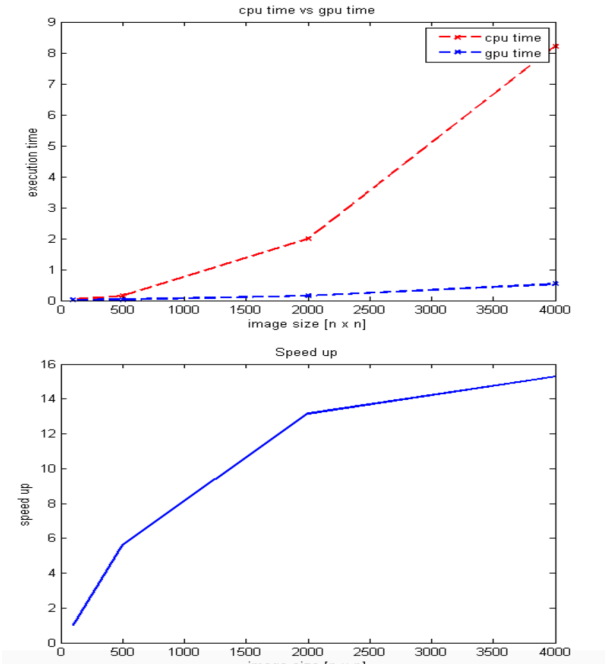

Figure 12. Performance comparison CPU vs GPU [15]

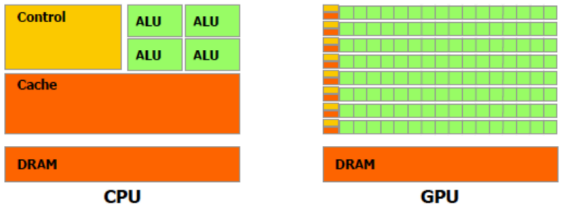

Figure 13. CPU and GPU chip design [18]

1. NVIDIA CUDA:

GPU have large number of smaller cores compared to cores in CPU. They can execute multiple tasks in parallel. NVIDIA CUDA is a parallel computing platform and an API designed to work with multiple programming languages such as C, C++, FORTRAN, etc. It is developed for parallel computing on GP-GPU [17]. Since GPU can execute large number of tasks in parallel through CUDA, software programmers can perform general purpose computation on GPU cards. In CUDA, kernel functions are called from host/CPU and execute on device/GPU. Sometimes transferring of data among the host and device can cause overhead in performance. The kernel functions can simultaneously execute by each thread. One must set the value of number of blocks and number of threads can in each of the block for completing the calculations to call the kernel function.

2. CUDA Events:

It is used for computing time taken between two events. Since CUDA events are handled by GPU itself, they do not add any overhead to that given by CPU. CUDA event time stamp is generally used by programmers to measure the execution time for specific code. It requires two steps to calculate execution time [19]:

- Event Creation
- Event Recording

Two events (start and stop) must be created to calculate execution time of specific code. Subtracting start and end time will give us the elapsed time.

3. Image Processing:

Image processing involves manipulation of images to obtain desired image. This resultant image can be used widely for image recognition, image retrieval, etc. Four major steps involved in image processing methodology is:

1. Image file importing
2. Information reading from the file
3. Image data modification
4. Output saving.

Generally, images are represented as 2D matrix of pixels. Each of these pixels repeatedly perform same steps (steps will be different according to the application), so it completely leverages CUDA architecture.

4. Measurement Parameters:

Two parameters/metric for CUDA technology study are:

1. Speedup: Ratio of sequential to parallel execution time.
2. CUDA overhead: This is the measure of time taken to allocate memory on device/GPU plus time required to copy initial values to device from host and time required to copy results back to host.

CUDA overhead is one of the main disadvantage of working with CUDA. But this delay will be there for both sequential and parallel execution of programs. Graphical representation of speed up and overhead is shown in Figure 14, and Figure 15.

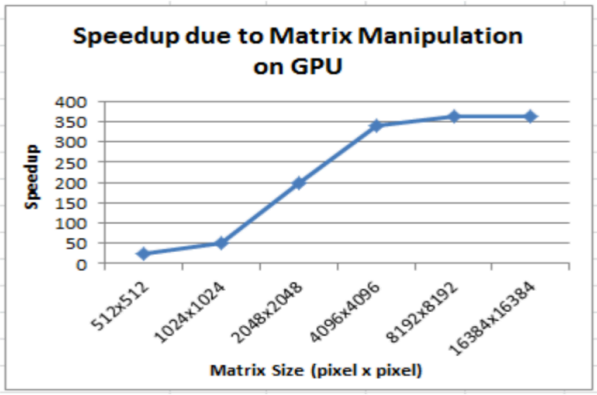

Figure 14. Speedup factor for image processing on GPU cards[17]

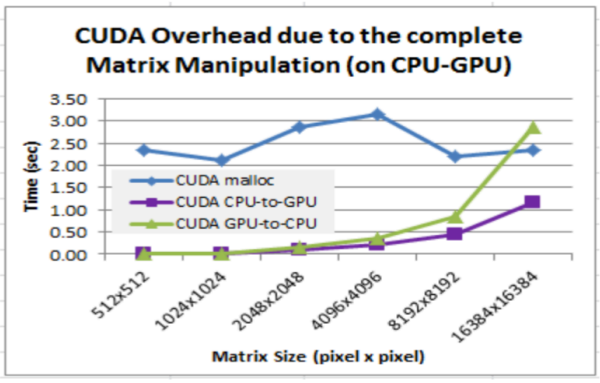

Figure 15. CUDA overhead time[17]

Important observation from Figure 14 is that as image size increases speedup factor also increases (speedup of 512*512 image is 25 times and 16,384 * 16,384 is about 365 times). Another

important observation is speedup factor does not increase after certain image size. Image with 8192 * 8192 taken almost same time as 16,384 * 16,384 image. Experiment in [17] uses 65,535 blocks and 512 threads in each Block, in 16,384 * 16,384 image each thread works on 8 pixels but in 8192 * 8192 each thread works on 2 pixels. This is reason for almost same time of execution. Similarly, there is proportional increase between size of the image and overhead in Figure 15. Time taken to transfer data between host and device is the major performance bottleneck in CUDA [17]. Steps to be followed is shown below [17]:

A. *Proposed Image Processing Algorithm for Multicore/Manycore Computing*

In this subsection, we introduce the image processing algorithm that is implemented in a parallel CUDA/C program. For traditional C and CUDA/C programs, the same image files are used as inputs. The algorithm considers the following general steps:

Image file related steps:

i) Get the input bitmap file name to be processed.

ii) Get the output file name to be generated.

a) This file contains the original pixels modified after applying the invert filter.

iii) Open the image file in read mode.

a) Parse the bitmap file header to get its properties including the pixels matrix width and height and the number of colors.

b) Update the file pointer to skip the header bytes.

c) Start a timer on the CPU.

- Load the pixel's bytes in a one dimensional array with length equal to matrix width * matrix height * 3 (3 bytes per pixel)

d) Stop the timer started on step 3.

iv) Create the output file with the provided name.

v) Save the bitmap file header.

CUDA related steps:

vi) Allocate memory on host and device as needed. Then

a) Raise a CUDA Event (timestamp start).

- Allocate an array of integers with the same size as the one in step iii)c.

b) Raise a CUDA Event (timestamp end).

c) Raise a CUDA Event (timestamp start).

- Copy the loaded pixels into the space reserved on step vi)a.

d) Raise a CUDA Event (timestamp end).

vii) Raise a CUDA Event (timestamp start).

a) Kernel call with parameters including the number of blocks and threads.

b) Call thread synchronize from the CPU as a POSIX threads join equivalent [11].

viii) Raise a CUDA Event (timestamp end).

ix) Raise a CUDA Event (timestamp start).

a) Copy the modified pixels (pixels with applied filter) back to the CPU.

x) Raise a CUDA Event (timestamp end).

xi) Store the pixels in the output header.

xii) Free allocated memory on the device and the host.

xiii) Close both file handlers.

VI. CONCLUSION

Traditional sequential image processing methods does not take advantages of multi/many cores available in CPU/GPU and they are much time intensive. Parallel computing algorithms using multicore CPU and GPU can process large sized images very fast. Image processing involves complex repetitive computation which can exploits CUDA platform efficiently. One of the important point to be noted is that there is considerable overhead due to data transfer between host and device in CUDA technology. Sometimes this might cause to reduce the speedup.

REFERENCES

[1] E. Young and F. Jargstorff, "Image processing and video algorithms with cuda," 2008.

[2] D. Visariya , S. Govindankutty, V. Goyal, Parallale Image Processing

[3] [Online]: http://en.wikipedia.org/wiki/Medical_imaging

[4] [Online]: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_geometric_transformations/py_geometric_transformations.html#geometric-transformations

[5] [Online]: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_filtering/py_filtering.html#filtering

[6] [Online]: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_canny/py_canny.html#canny

[7] [Online]: https://www.mathworks.com/help/images/morphological-dilation-and-erosion.html

[8] [Online]: https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html

[9] [Online]: https://computing.llnl.gov/tutorials/parallel_comp/#Whatis

[10] S. Saxena, N. Sharma, S. Sharma, "Image Processing Tasks using Parallel Computing in Multi core Architecture and its Applications in Medical Imaging", International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 4, April 2013.

[11] R. T. Rasúa, "Algoritmos paralelos para la solución deproblemas de optimización discretos aplicados a ladecodificación de señales," Ph.D. dissertation, Departamento de Sistemas Informáticos y Computación. Universidad Politécnica de Valencia, Valencia, España, 2009.

[12] D.M. Bhosale, P.E. Panchal, "Parallel Image Processing & its Applications in Medical Imaging", International Journal of Computer Science and Management Research, October 2013.

[13] [Online]: https://en.wikipedia.org/wiki/Pipeline_(computing)

[14] Using Graphics Devices in Reverse: GPU-Based Image Processing and Computer Vision. James Fung, Steve Mann.

[15] [Online]: https://blogs.mathworks.com/steve/2013/12/10/image-processing-with-a-gpu/

[16] Ben Baker, Using GPUs for Image Processing. [Online]: https://fhtw.byu.edu/static/conf/2011/baker-gpus-fhtw2011.pdf

[17] A. Asaduzzaman, A. Martinez, and A. Sepehri, "A Time-Efficient Image Processing Algorithm for Multicore/Manycore Parallel Computing", In Proceeding of the IEEE Southeast Conference, Florida, April -9, 2015.

[18] [Online]: https://pdfs.semanticscholar.org/597e/6bede840a1a3f2b938f4eb4cf50c00173f08.pdf

[19] J. Sanders and E. Kandrot, "CUDA by Example: An Introduction to General-Purpose GPU Programming," Addison-Wesley, 2010.