GPGPU Based High Performance Computation Support For Client-Server Framework

BY
Poorna Banerjee
(11MCEC23)

Nirma University, Ahmedabad
May 2013

GPGPU Based High Performance Computation Support For Client-Server Framework

Major Project

Submitted in partial fulfilment of the requirements

For the degree of

Master of Technology in Computer Science and Engineering

By:
Poorna Banerjee
(11MCEC23)

Guided By:

Mr. Amit Dave (SAC, ISRO)
Dr. Madhuri Bhavsar

Nirma University, Ahmedabad
May 2013

Declaration

This is to declare that

1. The thesis comprises my original work towards the degree of Master of Technology in Computer Science at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgement has been made in the text to all other material used.

- Poorna Banerjee (11MCEC23)

Certificate

This is to certify that the Major Project entitled "GPGPU Based High Performance Computation Support for Client-Server framework" submitted by Ms. Poorna Banerjee (11MCEC23), towards the partial fulfilment of the requirements for the degree of Master of Technology in Computer Science and Engineering of Nirma University of Science and Technology, Ahmedabad, is the record of work carried out by her under my supervision and guidance. In my opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of my knowledge, haven't been submitted to any other university or institution for award of any degree or diploma.

Mr. Amit Dave
External Guide, PCSVD Division,
Space Applications Centre (SAC), ISRO
Ahmedabad - 380015

Dr. Madhuri Bhavsar
Guide and Section-Head (IT)
Institute of Technology,
Nirma University, Ahmedabad

Prof. Vijay Ukani
PG Coordinator, M.Tech (CSE)
Institute of Technology,
Nirma University, Ahmedabad

Dr. Sanjay Garg
Professor and Head,
Department of Computer Engineering,
Institute of Technology,
Nirma University, Ahmedabad

Dr. Ketan Kotecha
Director,
Institute of Technology,
Nirma University, Ahmedabad

Acknowledgements

It gives me immense pleasure in expressing my gratitude towards **Mr. Amit Dave**, Sr. Scientist, Space Applications Centre (SAC), ISRO - Ahmedabad, for his valuable guidance, support and motivation throughout the Major Project. I am thankful to him for the valuable time he spent with me for my thesis, for suggestions that shaped this project.

I will always be grateful to **Shri D. R. Goswami**, Group Director (PCEG/SEDA) and **Mr. Ashish Mishra**, Head of PCSVD Division, Space Applications Centre (SAC), ISRO Ahmedabad, for their valuable time and highly constructive suggestions right from the very early stage of this project work.

I would like to extend my heart-felt gratitude to **Dr. Madhuri Bhavsar**, Institute of Technology, for her valuable guidance and suggestions in improving the quality of my thesis work and always directing me towards the right path to progress.

I would like to give my special thanks to **Prof Vijay Ukani**, Institute of Technology for providing constant motivation and support.

I am also thankful to **Dr. K Kotecha**, Director, Institute of Technology for his kind support in all respects during my study.

I am thankful to all faculty members of Department of Computer Science and Engineering, Nirma University, Ahmedabad for their special attention and suggestions towards the improvement of my project work.

Last, but not the least, no words are enough to acknowledge the constant support and motivation given by my family members during the course of this project work.

- Poorna Banerjee (11MCEC23)

Abstract

In today's era of computing where applications can require up to terabytes of data for processing, virtually all microprocessor vendors have switched to models where multiple processing units, referred to as processor cores, are used in each chip to increase the processing power.

This brings into focus the Graphics Processing Units (GPUs) which emphasize on the many-core paradigm of data processing rather than the multi-core approach supported by most of today's modern Central Processing Units (CPUs). With the advent of General Purpose GPUs (GPGPU), even those applications that are not directly associated with graphics operations can still harness the computing capabilities of a GPU.

The objective of this project is the development of a platform-independent client-server framework to support GPGPU based high performance computation with transparency to underlying hardware and operating systems.

The system would provide extensibility in terms of the number and type of tasks or jobs that the client can submit for processing at the remote server connected to the GPGPU. The processed results would then be returned back to the client.

Keywords: Client-Server Framework, Client Tasks, High Performance, GPGPU, Many-core, Multi-core, Parallel Computing.

Contents

List of Figures

List of Tables

Chapter 1

About Space Applications Centre (SAC), ISRO

Introduction: Space activities in the country started during early 1960s with the scientific investigation of upper atmosphere and ionosphere over the magnetic equator that passes over Thumba near Thiruvananthapuram using small sounding rockets Realising the immense potential of space technology for national development, *Dr. Vikram Sarabhai*, the visionary leader envisioned that this powerful technology could play a meaningful role in national development and solving the problems of common man. Thus, Indian Space programme born in the church beginning, space activities in the country, concentrated on achieving self reliance and developing capability to build and launch communication satellites for television broadcast, telecommunications and meteorological applications; remote sensing satellites for management of natural resources.

The objective of ISRO is to develop space technology and its application to various national tasks. Accordingly, Indian Space Research Organisation (ISRO) has successfully operationalised two major satellite systems namely Indian Na-

tional Satellites (INSAT) for communication services and Indian Remote Sensing (IRS) satellites for management of natural resources; also, Polar Satellite Launch Vehicle (PSLV) for launching IRS type of satellites and Geostationary Satellite Launch Vehicle (GSLV) for launching INSAT type of satellites.

Space Applications Centre (SAC) is one of the major centres of the Indian Space Research Organisation (ISRO) located in Ahmedabad, Gujarat. SAC focuses on the design of space-borne instruments for ISRO missions and development and operationalisation of applications of space technology for national development. The applications cover communication, broadcasting, navigation, disaster monitoring, meteorology, oceanography, environment monitoring and natural resources survey.

SAC designs and develops all the transponders for the INSAT and GSAT series of communication satellites and the optical and microwave sensors for IRS series of remote sensing satellites. Further, SAC develops the ground transmit/receive systems (earth stations/ ground terminals) and data/image processing systems.

In order to carry out the above tasks, SAC has highly sophisticated payload integration laboratories, electronic and mechanical fabrication facilities, environmental test facilities, systems reliability/assurance group, image processing and analysis facilities, project management support group and a well-stocked library.

The Centre also conducts nine-month post graduate diploma courses for students from the Asia Pacific region under the aegis of the Centre for Space Science and Technology Education (CSSTEAP) in satellite meteorology and communication. SAC works with industry for sourcing and indigenization, involves Indian universities in space research and propagates space technology and applications amongst students and public through in-house and mobile exhibitions.

History and Achievements:
Prior to 1972, research in applications of space technology was pursued by different units of ISRO in Ahmedabad - the birthplace of Dr Vikram A. Sarabhai. These were merged to form SAC in 1972. SAC is one of the major centres of Indian Space Research Organisation.

Satellite Communications:
Recognizing the possible role of space communications in nation building, an Experimental Satellite Communication Earth Station (ESCES) was established in 1967 at Ahmedabad (now part of SAC). It was an experimental Earth Station and training centre where scientists and engineers of India and other developing countries could receive training and first hand experience in the design, development and operations of an earth station for communications and broadcasting. A large number of international training courses have been conducted since then.

A unique experiment called the Satellite Instructional Television Experiment (SITE) was conducted during 1975-76 utilising the American ATS-6 satellite. It involved telecasting educational programmes aimed at socio-economic upliftment of rural India, to 2400 villages - spread over six states - through experimental Direct Reception Sets. SITE was followed by communication techniques developmental project called Satellite Telecommunications Experiments Projects (STEP), carried out with the Frenco-German satellite, Symphony.

The first experimental communication satellite APPLE, designed, fabricated and qualified at SAC, It was launched onboard the first experimental flight of the Arian. An exhaustive communications application programme called the APPLE Utilisation Programme (AUP) was also conceived and carried out simultaneously. The INSAT-1 series of satellites was custom designed and made as per the unique requirements of the country by a US company. The INSAT 2A, 2B, 2C, 2D and 2E, launched in the years 1992, 1993, 1995, 1997 and 1999 respectively, were designed, fabricated and qualified in house. These had various combinations of C, Ext. C, Ku and S band transponders with varying degrees of EIRP Some of these also carried meteorological payload VHRR, payload for Search and Rescue, etc. It is worth noting here that eleven transponders onboard the INSAT2E was leased to the international INTELSAT group even before its launch.

Remote Sensing:
The utilisation of aerial and Landsat imagery for resources application in early 70s paved way for initiation of the remote sensing activities in the country. Activities were also carried in the field of meteorology with available data from foreign satellites and from indigenously developed airborne thermal Scanner. All the remote sensing activities so far can be divided into three Phases, viz. Experimental Phase, Semi-Operational and Operational Phase.

The first phase saw the development of airborne thermal sensors such as Infrared (IR) scanner, multispectral scanner, linear Charge Coupled Device (CCD) camera, Side Looking Radar, Colour Infrared (CIR) based photographic systems and a number of photo interpretation and ground truth equipment which were later productionised through technology transfer. Landsat data were fully utilised since 1973 to learn space based Remote Sensing applications. For all these sensors, efforts were made to also define and develop data products systems.

Based on above initial work, a strong applications programme was evolved around these instruments. Foundations for space borne sensors were laid during this period. Under Satellite for Earth Observation (SEO) programme, 2 satellites were launched and called Bhaskara satellites after their Launch onboard Russian launch Vehicle. Bhaskara carried a 1 km resolution 2 band TV camera systems and a three channel microwave radiometer. These were designed, developed and successfully qualified in house. The programme formed the basis

for the advanced sensor development leading to operational applications. Users were also sensitised for utilising remote sensing data from satellite based sensors.

The second phase in 1980s witnessed the results of earlier efforts of experimental satellites. The IRS 1A programme was successfully launched and the users started receiving multispectral imagery with 36m resolution. Professionalism was brought into the design of sensors, data products and applications projects. Major applications in agriculture, hydrology, geology and other areas were defined in close interaction with user agencies and the IRS utilisation programme was carried out successfully. These efforts led to semi-operational applications of IRS 1A data

Meteorological and Oceanography:
The Meteorological activities at SAC basically involve payload design and fabrication and applications using the data received from such satellites. The successful ventures outlined in the previous paragraphs enabled ISRO to evolve an ambitious Remote Sensing and Meteorology programme for the country to have a unique constellation of satellites for resources and environmental applications. Bhaskara I and II were the first Indian Meteorological satellites which carried microwave radiometer called SAMIR to provide information on sea state and atmospheric water vapour content for use in meteorological studies.

In the initial phase of met applications data from Bhaskara were supplemented by data from NOAA, SEASAT, and ERS etc. Meteorological components from INSAT, starting from 1982 have now become fully operational. Over the years the resolution available from VHRR onboard INSATs has become better. The CCD camera also onboard some of the INSATs and inclusion of water vapour channel in the new VHRRs have added advantage. Exclusive meteorological payload would be carried ON BOARD Metsat.

The first exclusive meteorological satellite KALPANA-1 was launched by ISRO's PSLV on Sept. 12, 2002. It carried a VHRR and a data relay transmitter. ISRO has also launched the Oceansat-I in May, 1999. It has an Ocean Colour Monitor (OCM), an optical sensor with 8 narrow spectral bands with high resolution and higher dynamic range and Multi frequency Microwave Scanning Radiometer (MSMR). These sensors have high repetivity of 2 days and hence are most suited for dynamic events in coastal and mid ocean regions.

SAC has also taken up many Applications projects under Announcement of Opportunity scheme of several international missions like ERS, TRMM, ADEOS, ENVISAT, etc. SAC has state of the art General Circulation Models for experimentation with satellite data. Prediction of weather in the extended range and prediction of Ocean state in the short range are the fields of active research.

Chapter 2

Introduction

2.1 Motivation

Microprocessors based on a single central processing unit (CPU) drove rapid performance increases and cost reductions in computer applications for more than two decades. These microprocessors brought giga (billion) floating-point operations per second (GFLOPS) to the desktop and hundreds of GFLOPS to cluster servers. This relentless drive of performance improvement has allowed application software to provide more functionality, have better user interfaces, and generate more useful results. The users, in turn, demand even more improvements once they become accustomed to these improvements, creating a positive cycle for the computer industry.

Virtually all microprocessor vendors have switched to models where multiple processing units, referred to as processor cores, are used in each chip to increase the processing power. Traditionally, the vast majority of software applications are written as sequential programs. The execution of these programs can be understood by a human sequentially stepping through the code. Historically, computer users have become accustomed to the expectation that these programs run faster with each new generation of microprocessors. Rather, the applications software that will continue to enjoy performance improvement with each new generation of microprocessors will be parallel programs, in which multiple threads of execution cooperate to complete the work faster.

Over the previous decades, microprocessor design has taken two trajectories:

- The multicore trajectory seeks to maintain the execution speed of sequential programs while moving into multiple cores. The multicores began as two-core processors, with the number of cores approximately doubling with each semiconductor process generation.
- In contrast, the many-core trajectory focuses more on the execution through-

put of parallel applications. The many-cores began as a large number of much smaller cores, and, once again, the number of cores doubles with each generation. A current exemplar is the NVIDIA GeForce GTX 280 graphics processing unit (GPU) with 240 cores, each of which is a heavily multithreaded, in-order, single-instruction issue processor that shares its control and instruction cache with seven other cores. Many-core processors, especially the GPUs, have led the race of floating-point performance since 2003. This phenomenon is illustrated in Figure 2.1.

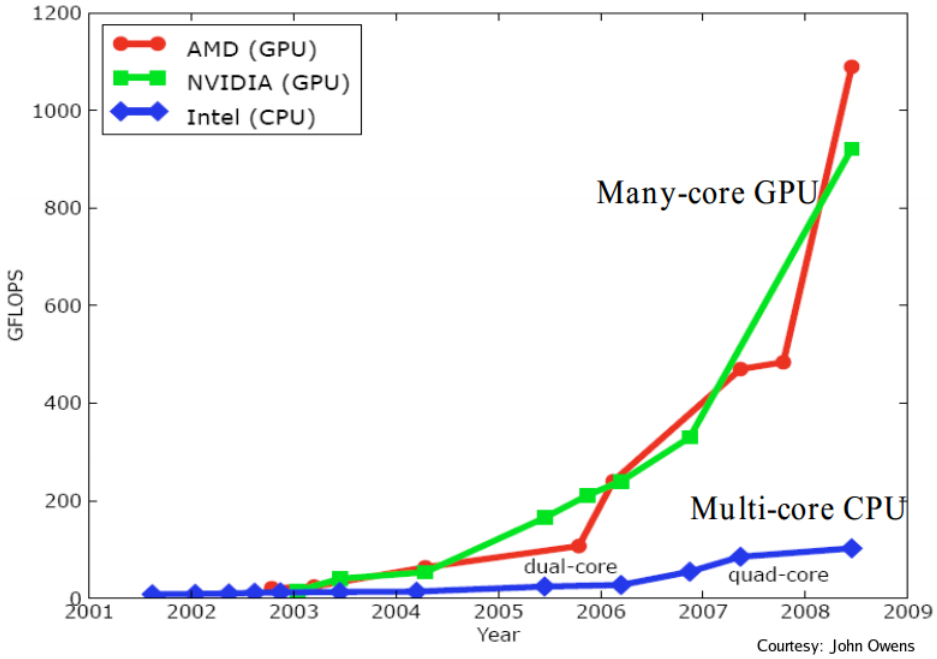


Figure 2.1: Enlarging performance gap between GPUs and CPUs. [11]

2.2 Scope of this Project

This project is aimed at the development of a platform-independent client-server framework to support GPGPU based high performance computation with transparency to underlying hardware and operating systems. The system would provide extensibility in terms of the number and type of tasks the client can submit for processing at the remote server connected to the GPGPU. Some examples of such tasks would be:

- Radiometry

- LUT Generation and Image Correction
- Bayer-Pattern based Demosaicing of an Image
- Least Squares Curve Fitting for any Polynomial Order
- Graphic Visualization

and many more.

Development of this project would involve establishment of client and server interfaces and management of distributed data structures. The user application interacts with client-side library modules which in turn interact with server-side modules associated with the GPGPU for data processing. The processed results are then sent back to the client. In the basic scenario, the client and server machines are situated on different networks as demonstrated in Figure 2.2. Thus,

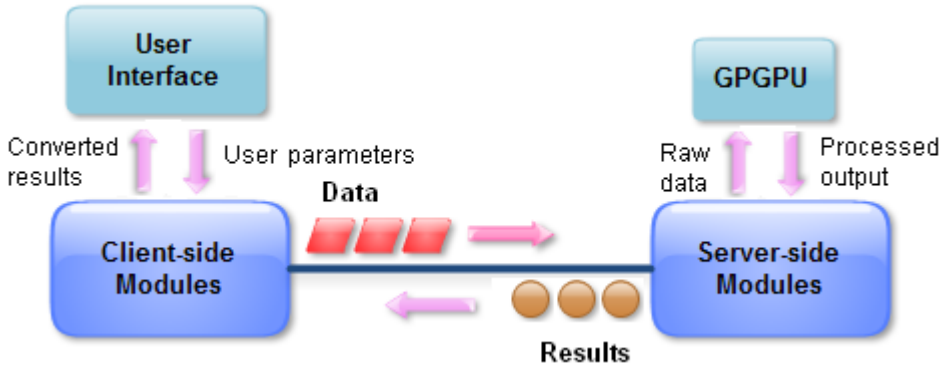


Figure 2.2: Basic Schematic Diagram.

this project involves development of customized data exchange mechanisms between the client and server, designing modules at both the client and server sides for dynamic loading and unloading of data and design of efficient parallel programs for optimized processing of data at the GPGPU.

Chapter 3

Establishment and Completion of the Client-Server Framework

Certain design issues came into perspective while developing the framework:

- *Type of application interface* that is to be provided to the client - it can be GUI or Command Line (CLI) based. Both GUI and CLI based support has been provided during the actual implementation of the framework.

- *Various parameters that are needed to be passed to the client-side processing modules.* Some such parameters could be - IP-address of the remote GPGPU server, input data file to be processed, output file in which the results are to be returned back, task flags and other options.

- *Representation of the data for transmission over the network.* Certain aspects that need to be decided are - fields and format of the data sent and received, maximum allowable data size, required meta-data to be added such as data-type indicators, matrix-dimensions, etc.
  One such template header structure, for sending client parameters to the server, is shown in Figure 3.1. Field sizes indicated in the header are in bytes. In Figure 3.1, the first field of 29 bytes in the header indicates the task flag that is generated when a client chooses a particular processing task. This task flag helps the GPGPU sever decide what kind of processing is to be carried out on the input data. The next field is a one-byte special symbol which indicates to the server whether it should expect any input data from the client. A + indicates input data follows the header. A null character indicates absence of input data from the client. The next 200 bytes in the header have been allocated for specifying a comma separated list of parameters. The set of parameters present in the list depends on what particular task the client has chosen. Some common parameters

include data-matrix dimensions, data type indicator, polynomial orders, etc. The next 30 bytes in the header specify the name of the output file in which the client wants to receive the results. All the fields in the header get populated whenever the client fires a task for processing, either via the GUI or CLI, and accordingly a suitable header structure is generated by the client-side processing modules and sent to the server.

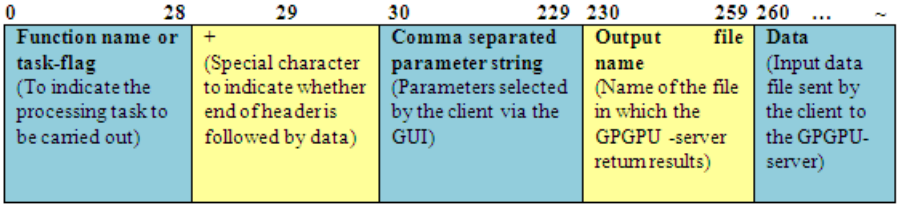


Figure 3.1: Header template for sending user parameters to the server (Field sizes in bytes)

- *Technology that should be used to establish the connection and transmit data between the client and server.* Some possible choices include Socket Programming, RPC, Java RMI, and CORBA. While the more sophisticated option such as CORBA and RMI work best with Java and introduce a layer of middleware functionalities, options such as RPC and socket-programming are more primitive, C/C++ based and can provide better performance. During actual implementation of the framework, C/C++ sockets have been used.

- Upon reception of processed results, the client-side modules would have to re-assemble the data to present it in an understandable form to the user application. This requires *data conversion methods.*

- Choice of transport layer protocol. Common choices would be TCP, UDP and XTP. While TCP is connection oriented and reliable, UDP is connectionless, unreliable but faster. Since most of the data transfer involved in the given client-server scenario involves data files which must be sent and received without any errors, TCP was chosen as the transport layer protocol.

Thus, designing the framework involved:

- Development of customized data exchange mechanisms between the client and server.

- Designing client-side and server-side modules for dynamic loading and unloading of user-submitted tasks and data.

- Development of efficient, parallelized programs for optimized processing at the GPGPU.

After establishing the system-framework, the following four functions to establish the capability of the framework and an essential utility for the user have been developed:

1. Bilinear Interpolation based Demosaicing of an Image.
2. Gradient Interpolation based Demosaicing of an Image.
3. Least-Squares Curve Fit for any Polynomial Order on Given Data.
4. Edge Detection and Edge Ranking for an Image.
5. Remote GPGPU Information Generation

A user-friendly GUI has been developed for the client so that various input parameters can be specified, as shown in Figure 3.2. The GUI has been designed so as to facilitate both command-prompt based and UI based execution of tasks submitted by the client. Additional features have been provided in the GUI for fault detection and error-log archiving.

When adding further tasks to the framework is considered, the framework has been designed keeping in mind that future developers can contribute their GPGPU library codes to the already existing task-set. The only steps that code developers have to follow is to create their library codes according to a certain generic template designed specially for the framework, following which the new libraries can be seamlessly integrated with the existing task-set through creation of shared, dynamically loaded libraries. All that is needed is a one-step compilation to generate the shared-library object.

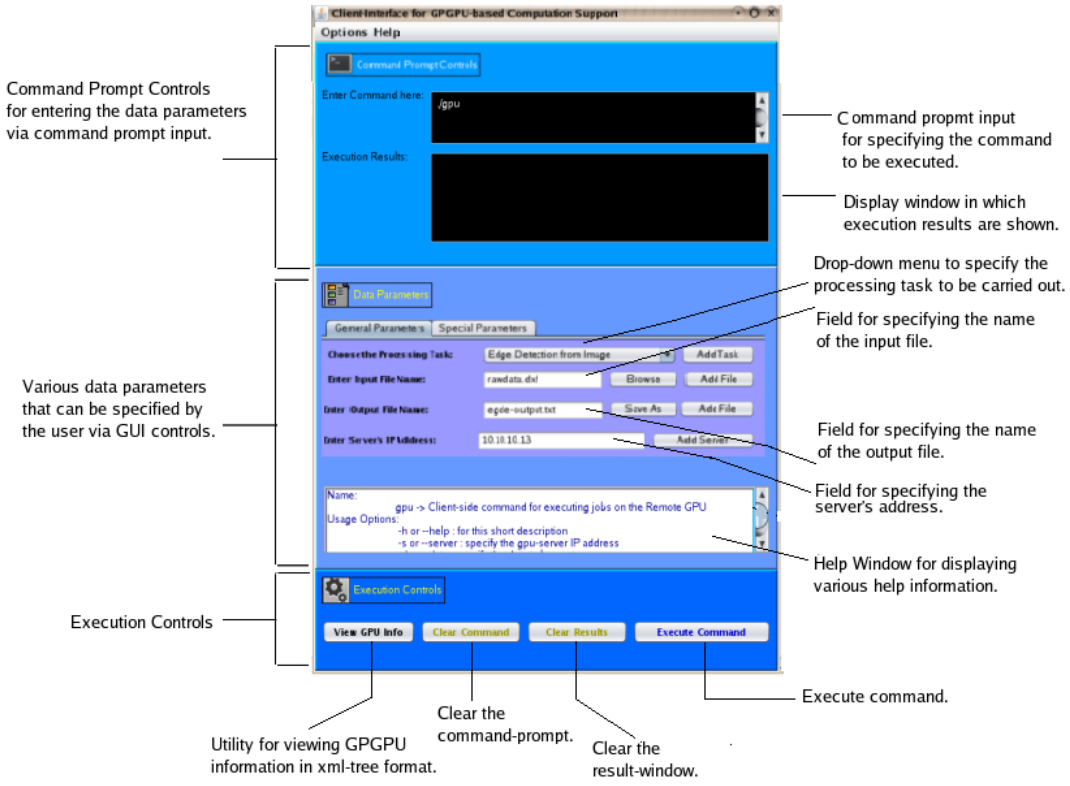


Figure 3.2: GUI Interface for GPGPU Based Computation Support

Chapter 4

Demosaicing of an Image

With the advent of technology,the resolutions for single-sensor imaging devices such as digital cameras, are becoming better by the day but the core working-principle remains the same. These cameras are based on a charge coupled device (CCD) array and each sensor on the CCD captures only one sample of the color spectrum[8].

Digital color cameras generally use a Bayer mask over the CCD. A Bayer filter mosaic is a color filter array (CFA) for arranging Red Green Blue (RGB) color filters on a square grid of photo-sensors, as shown in Figure 4.1. Each square of four pixels has one filtered red, one blue, and two green. This is because the human eye is more sensitive to green than either red or blue and the Bayer CFA is based on the knowledge of human visual perception.

We now have a mosaiced image; at each pixel,there is only one spectral measurement, which means that the other colors have to be estimated according to the neighboring colors in order to produce a high resolution color image. This process is referred to as demosaicing.

In order to produce the demosaiced image, the process of Interpolation has to be performed. There are many methods for performing such interpolation, two of them being Bilinear Interpolation and Gradient-Based Interpolation, which are discussed in the following two subsections.

4.1 Bilinear Interpolation based Demosaicing of an Image

4.1.1 Mathematical Analysis

In this method of interpolation, we calculate the average on each pixel depending on its position in the Bayer Pattern for determining the spectrum of that

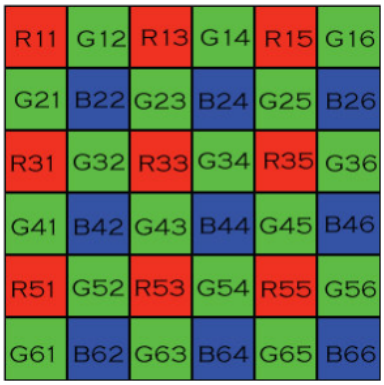


Figure 4.1: The Bayer Pattern [16]

particular pixel. For each pixel, we consider its 8 direct neighbors and then we determine the 2 missing color components of that pixel by averaging the corresponding colors of the neighboring pixels.

Depending on pixel position,we have 4 different cases of averaging:

1. the pixel is red
2. the pixel is blue
3. the pixel is green and is in a blue row
4. the pixel is green and is in a red row.

For each case, there are separate equations to calculate the average value, which are given as follows:

Case 1: Pixel R33 (red pixel):
Red = R33
Green = (G23+G34+G32+G43) / 4
Blue = (B22+B24+B42+B44) / 4

Case 2: Pixel B44 (blue pixel):
Blue = B44
Green = (R33+R35+R53+R55) / 4
Red = (R33+R35+R53+R55) / 4

Case 3: Pixel G43 (Green in a blue row):
Green = G43
Red = (R33+R53) / 2
Blue = (B42+B44) / 2

Case 4: Pixel G34 (Green in a red row):
Green = G34
Red = (R33+R35) / 2
Blue = (B24+B44) / 2

This method of interpolation allows us to determine the red, green and blue color values for each pixel. Also, Bilinear interpolation produces better results than the pixel-doubling interpolation because we calculate the average of the neighbor pixels instead of just copying the values. The missing color-component values will hence be closer to the true values.

4.1.2 Task-Module Description

In the client-server framework scenario, the client will submit raw image data over the network as input to the remote GPGPU server for performing Bilinear Interpolation based Demosaicing. The resultant demosaiced image will then be sent back to the client. Figure 4.2 shows the how the client may submit raw image data to the remote GPGPU server for performing Bilinear Interpolation based demosaicing of an image. The client needs to specify the IP address of the server, name of the input image-data file, the type of processing task to be carried out (in this case, its Bilinear Bayer Demosiacing) and the name of the output-file in which the results are to be returned. These controls can be specified either via the command prompt controls or through the GUI based data parameters. Upon successful execution and thereafter completion of the submitted task, the demosaiced image is sent back to the client in the specified output file name. As a sample output, the mosaiced input image and the demosaiced output images are shown in Figure 4.3. Fig.4.3(a) shows the original mosaiced image, 4.3(b)shows the demosaiced output image, 4.3(c) shows a zoomed out portion of the original image to highlight the mosaiced checkerboard pattern, 4.3(d) shows the same demosaiced portion from the output image.

4.1.3 Execution Performance Analysis

The task of gradient interpolation based demosaicing was performed on images of size 2048x2048, with each pixel's intensity value being represented by 16 bits (short int). The task of performing Bilinear Interpolation based demosaicing on an image requires similar actions to be performed on each pixel of the image, and these operations can be carried out on each pixel simultaneously. This provides a very good scope for parallelizing the task of Bilinear Interpolation based demosaicing on an image. A parallelized, CUDA/C based implementation has been developed which has been tested on the Nvidia Tesla C1060 GPGPU and Nvidia Quadro FX580 graphics-card. The sequential version of the same program has been implemented and tested on many different platforms. A comparative performance chart, along with the corresponding speed-ups obtained,

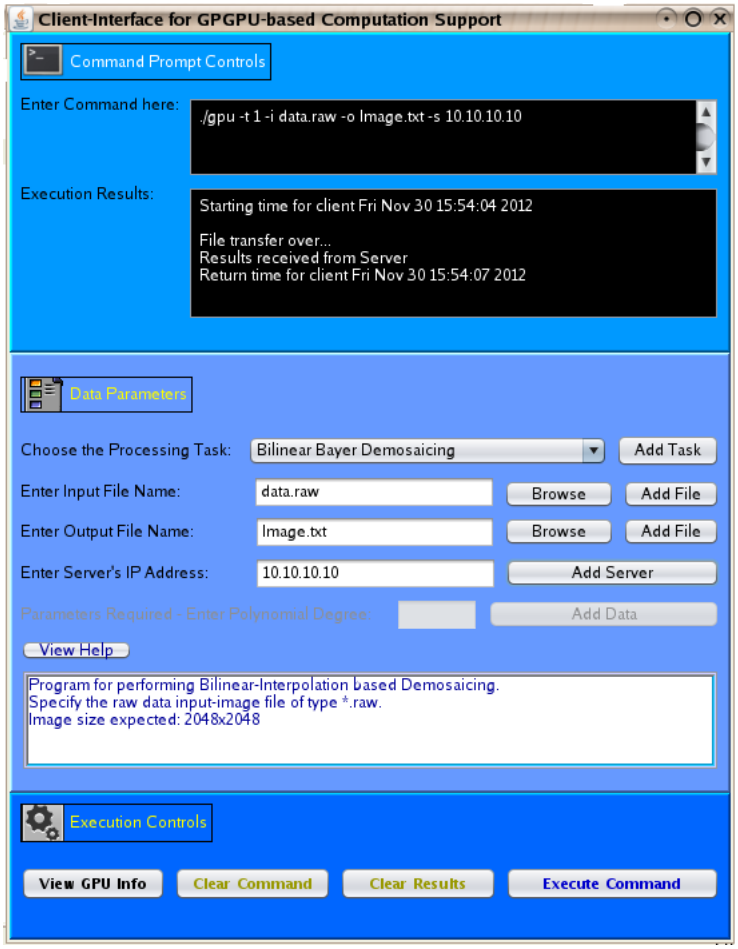


Figure 4.2: Executing Bilinear Interpolation Based Demosaicing

are shown in Figure 4.4. From the performance chart, it can be seen that significant speed-up has been obtained due to parallelization of the demosaicing process. The individual speed-up's for different platforms have been calculated keeping the performance on the Nvidia Tesla C1060 as the comparison benchmark. The speed-up's obtained, along with the specifications of each platform, are summarized in the table of Figure 4.5. From the table, it is evident that tremendous speed-ups have been achieved, the values going up to 30x.

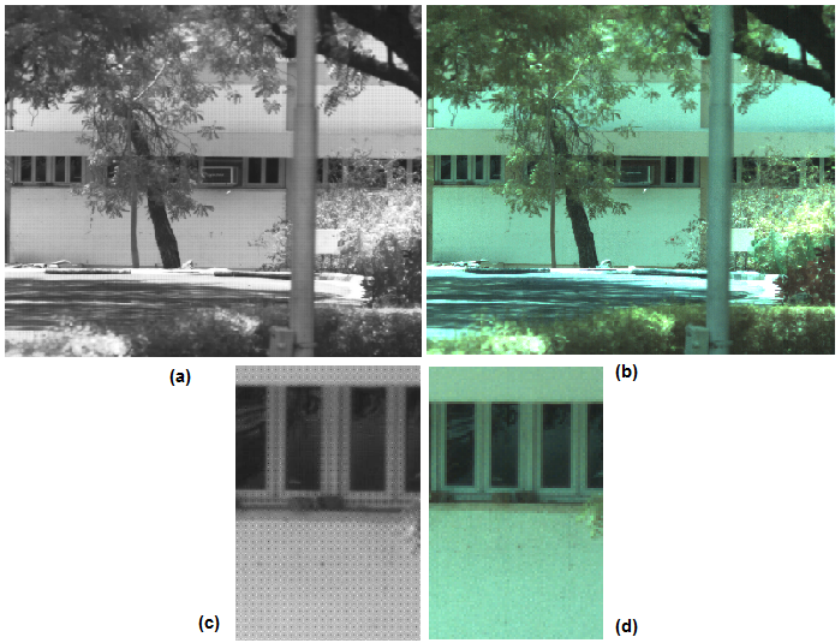


Figure 4.3: Results of Bilinear Interpolation Based Demosaicing

4.2 Gradient Interpolation based Demosaicing of an Image

4.2.1 Mathematical Analysis

This method of interpolation involves 3 steps:

1. Compute the luminance on each pixel
2. Interpolation of the color difference : Red minus Green
3. Interpolation of the color difference : Blue minus Green.

These two interpolations are needed in order to determine the red and blue channel for each pixel. This is a direct consequence of the fact that the human eye is sensitive to luminance changes. In the first step, we have only to determine the luminance for the red and blue channel. The luminance is the green channel so we consider, on the green pixel, that the luminance is actually the intensity of the pixel. On a red or blue pixel, the luminance of this pixel is determined by knowing if the pixel is on a vertical edge or on a horizontal one. In order to know if we have an edge, we can calculate 2 parameters: **alpha** and **beta**, which depends on whether we are on a red or blue pixel.

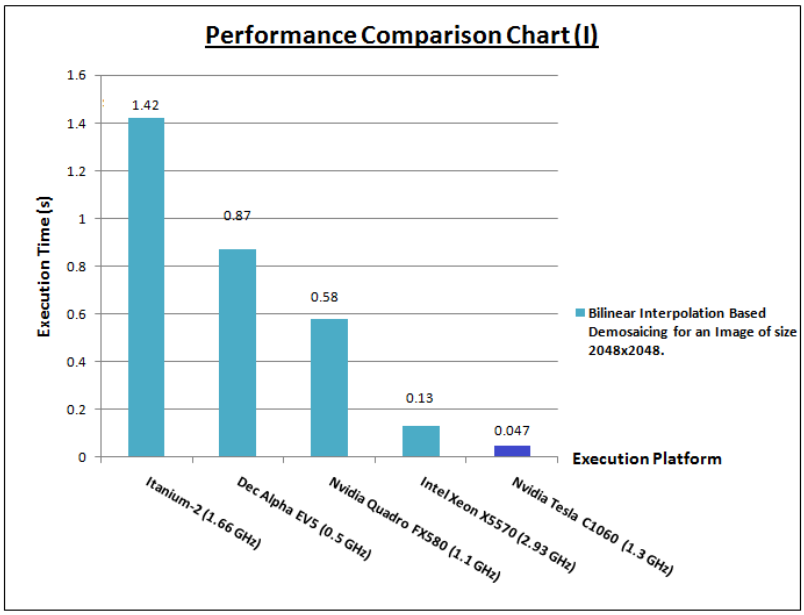


Figure 4.4: Performance Chart for Bilinear Interpolation Based Demosaicing

If we are on a blue pixel, for example B44, we have:
alpha = abs[(B42 + B46) / 2 - B44] (vertical edge)
beta = abs[(B24 + B64) / 2 - B44] (horizontal edge)
Then using those 2 values, we can estimate the missing green value as follows:

If alpha < beta (vertical edge), G44 = (G43+G45) / 2
If alpha > beta (horizontal edge), G44 = (G34+G54) / 2
If alpha = beta (no edge), G44 = (G43+G45+ G34+G54) / 4
Similarly, for R33, we have :
alpha = abs[(R31 + R35) / 2 - R33] (vertical edge)
beta = abs[(R13 + R53) / 2 - R33] (horizontal edge)
If alpha < beta (vertical edge), G33 = (G32+G34) / 2
If alpha > beta (horizontal edge), G33 = (G23+G43) / 2
If alpha = beta (no edge), G33 = (G32+G34+ G23+G43) / 4

At this point, the luminance values for each pixel (green channel) are known, we just have to determine the blue and red channel by computing the differences between those channels and the luminance channel.
Let Dxy = Cxy - Lxy. (C: Red or Blue depending on which one is to be computed, L: Luminance)
We have for red and blue channels:

Comparison Benchmark:
Nvidia Tesla C1060
Compute Capability: 1.3
Clock Rate: 1.3 GHz
No. of Processor Cores: 240
Total Global Memory: 4GB
Speed-up = Execution time on Test Platform / Execution time on Comparison Benchmark

Name of Platform	**Platform Specifications**	**Speed-up Obtained over this Platform**
Itanium-2	Model: 1 , Architecture: IA-64 Clock Rate: 1.66 GHz	30x
Dec Alpha EV5	Model: 21164, Architecture: DEC 64-bit RISC Clock Rate: 0.5 GHz	18x
Nvidia Quadro FX580	Compute Capability: 1.1 No. of Processor Cores: 32 Total Global Memory: 0.5 GB Clock Rate: 1.1 GHz	12x
Intel Xeon	Model: X5570 Architecture: Nehalem-EP, x86-64 Clock Rate: 2.93 GHz	3x

Figure 4.5: Speed-ups Obtained from Bilinear Interpolation Based Demosaicing

Red:
$R34 = G34 + (D33 + D35)/2$
$R43 = G43 + (D33 + D35)/2$
$R44 = G44 + (D33 + D35 + D53 + D55)/4$
Blue:
$B34 = G34 + (D24 + D44)/2$
$B43 = G43 + (D42 + D44)/2$
$B33 = G33 + (D22 + D24 + D42 + D44)/4$

In this method, we determine the missing color values by averaging and by considering the different edges that we encounter in the image and also by computing the luminance. This method of interpolation is expected to produce more accurate colors in output especially on the edges. When an edge is found, the pixels along the edge are considered instead of considering all the neighbors.

4.2.2 Task-Module Description

In the client-server framework scenario, the client will submit raw image data over the network as input to the remote GPGPU server for performing Gradient Interpolation based Demosaicing. The resultant demosaiced image will then be sent back to the client.

Figure 4.6 shows the how the client may submit raw image data to the remote GPGPU server for performing Gradient Interpolation based demosaicing of an image. The client needs to specify the IP address of the server, name of the input image-data file, the type of processing task to be carried out (in this case, its Gradient Based Bayer Demosiacing) and the name of the output-file in which the results are to be returned. These controls can be specified either via the command prompt controls or through the GUI based data parameters. Upon successful execution and thereafter completion of the submitted task, the demosaiced image is sent back to the client in the specified output file name. As a sample output, the mosaiced input image and the demosaiced output images are shown in Figure 4.7. Figure 4.7(a) shows the original mosaiced image, 4.7(b)shows the demosaiced output image, 4.7(c) shows a zoomed out portion of the original image to highlight the mosaiced checkerboard pattern, 4.7(d) shows the same demosaiced portion from the output image.

4.2.3 Execution Performance Analysis

The task of gradient interpolation based demosaicing was performed on images of size 2048x2048, with each pixel's intensity value being represented by 16 bits (short int). As in the case of Bilinear Interpolation, the task of performing Gradient Interpolation based demosaicing on an image requires similar actions to be performed on each pixel of the image, and these operations can be carried out on each pixel simultaneously. This provides a very good scope for parallelizing the task of Gradient Interpolation based demosaicing on an image. A parallelized, CUDA/C based implementation has been developed which has been tested on the Nvidia Tesla C1060 GPGPU and Nvidia Quadro FX580 graphics-card. The sequential version of the same program has been implemented and tested on many different platforms. A comparative performance chart, along with the corresponding speed-ups obtained are shown in Figure 4.8.

From the performance chart, it can be seen that significant speed-up has been obtained due to parallelization of the demosaicing process. The individual speed-up's for different platforms have been calculated keeping the performance

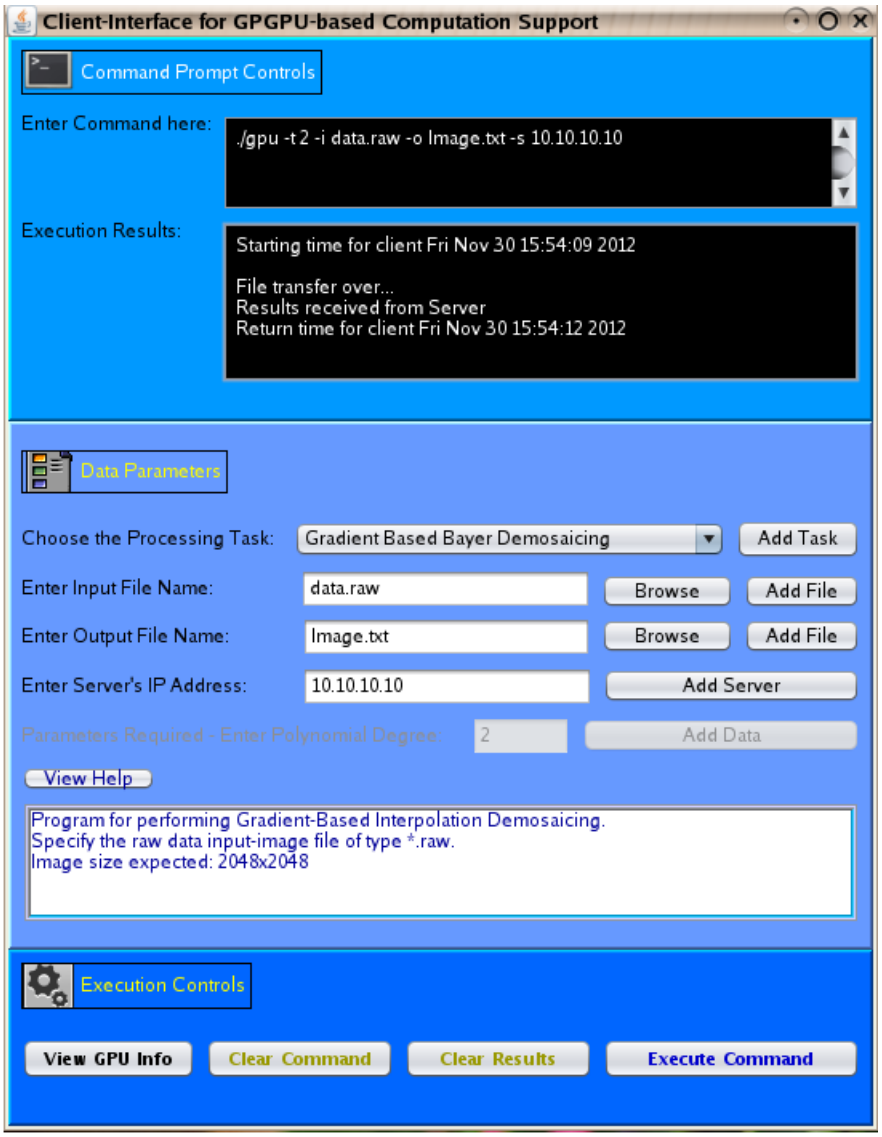


Figure 4.6: Executing Gradient Interpolation Based Demosaicing

on the Nvidia Tesla C1060 as the comparison benchmark. The speed-up's obtained, along with the specifications of each platform, are summarized in the following table. From the table in Figure 4.9, it is evident that tremendous speed-ups have been achieved, the values going up to 19x.

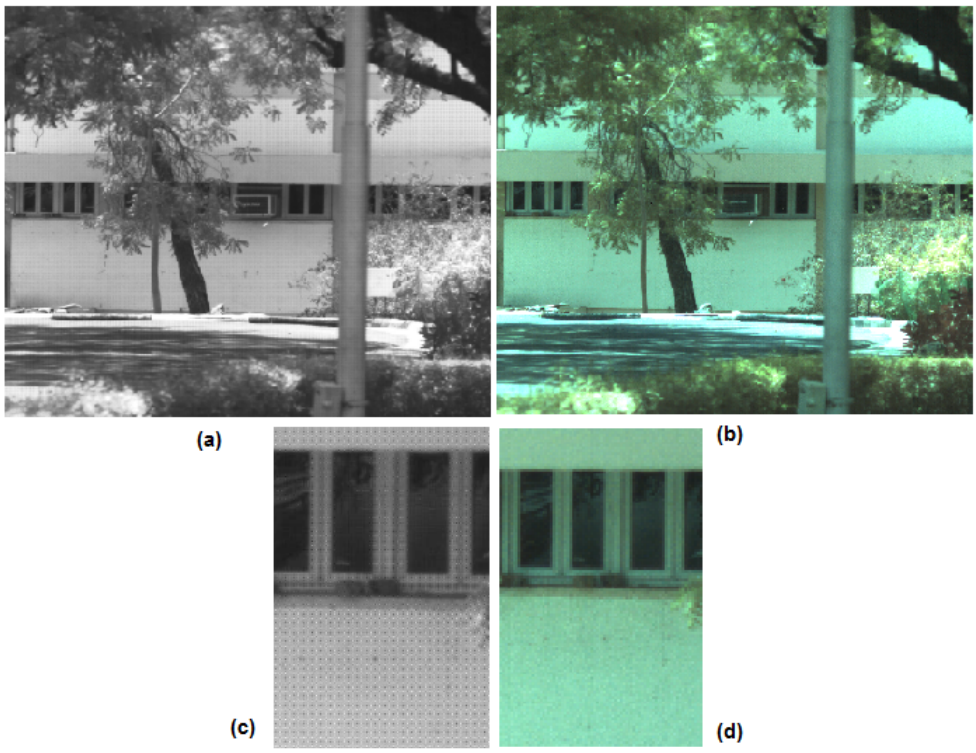


Figure 4.7: Results of Gradient Interpolation Based Demosaicing

4.3 Qualitative Comparison between Bilinear and Gradient-based Interpolation

In order to gauge the quality of demosaicing performed by both the interpolation algorithms, statistical attributes were analyzed for both the interpolation methods to compute the Peak Signal to Noise Ratio (PSNR) and they are summarized in Table 4.1. PSNR can be calculated using the following formula where X_{max} represents the peak or maximum intensity value for an image and N represents the number of pixels:

PSNR = 20 $\log_{10}$ X_{max} / RMSE
RMSE = ((demosaiced value$_i$ - actual value$_i$)^2 / N)$^{1/2}$
where i = 1 to N
The PSNR values have been computed separately for each color channel. Higher the PSNR values, better is the quality of the demosaicing performed by the interpolation algorithm. From the table, it can be seen that Gradient-based interpolation yields a higher PSNR value for images containing many edges.

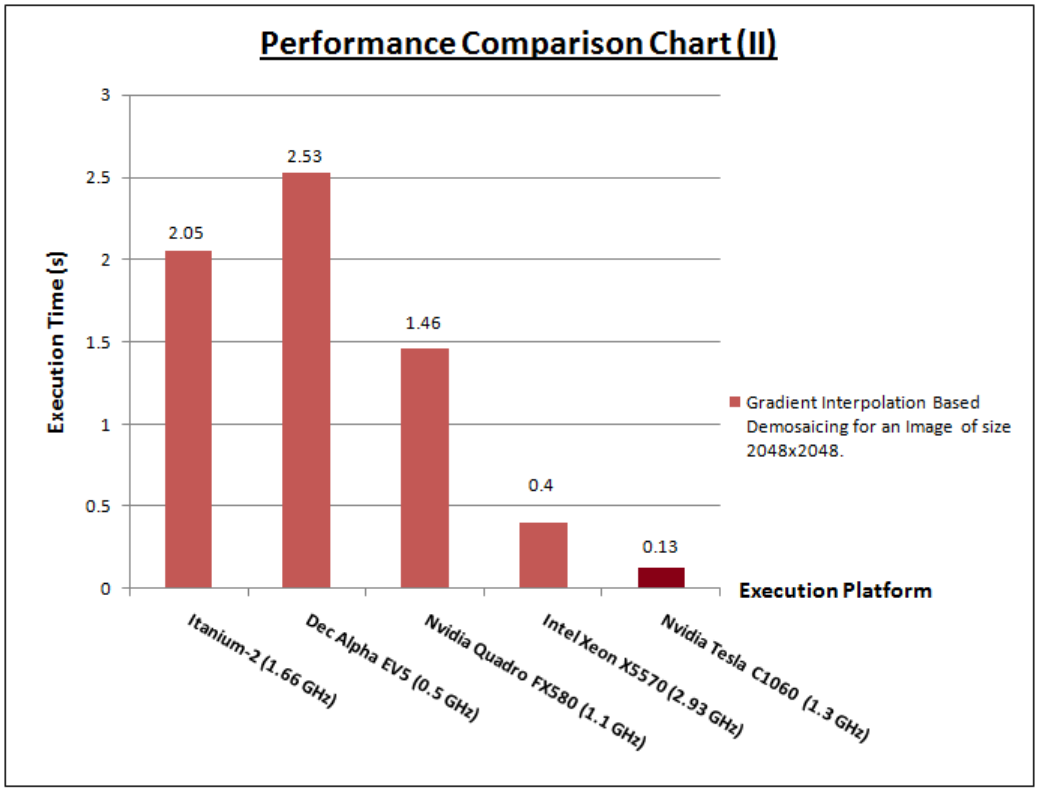


Figure 4.8: Performance Chart for Gradient Interpolation Based Demosaicing

Table 4.1: PSNR Calculation for Bilinear and Gradient-based interpolation

Interpolation method	**Color channel**	**Mean**	**Std. dev**	$\mathbf{X}_{max}$	**PSNR**
Bilinear	Red	99.3362	18.497	807	15.8579
Bilinear	Green	150.4735	35.4213	1023	27.2455
Bilinear	Blue	120.7161	33.4365	1023	30.9678
Gradient	Red	99.8825	18.7531	809	16.1532
Gradient	Green	150.5353	35.5223	1023	27.3701
Gradient	Blue	121.2844	33.6508	1032	31.5739

Comparison Benchmark:
Nvidia Tesla C1060
Compute Capability: 1.3
Clock Rate: 1.3 GHz
No. of Processor Cores: 240
Total Global Memory: 4GB
Speed-up = Execution time on Test Platform / Execution time on Comparison Benchmark

Name of Platform	Platform Specifications	Speed-up Obtained over this Platform
Itanium-2	Model: 1 , Architecture: IA-64 Clock Rate: 1.66 GHz	15x
Dec Alpha EV5	Model: 21164, Architecture: DEC 64-bit RISC Clock Rate: 0.5 GHz	19x
Nvidia Quadro FX580	Compute Capability: 1.1 No. of Processor Cores: 32 Total Global Memory: 0.5 GB Clock Rate: 1.1 GHz	11x
Intel Xeon	Model: X5570 Architecture: Nehalem-EP, x86-64 Clock Rate: 2.93 GHz	3x

Figure 4.9: Speed-ups Obtained from Gradient Interpolation Based Demosaicing

Chapter 5

Least-Squares Curve Fit for any Polynomial Order on Given Data

5.1 Estimation of Best-Fit Curve using Least Squares Method

The procedure of Least Squares curve-fitting is exquisitely used in many applications for fitting a polynomial curve of a given degree to approximate a set of data. If the input data-set is represented by pairs of the type (x_i,y_i) where $1 \leq i \leq n$, $n \geq 2$ (**n** is the number of data-points), then the best fitting curve f(x) has the least square error, i.e.[13],
$\Pi = \Sigma[y_i - f(x_i)]^2$ = minimum, where i = 1 to n

For, example if we want to fit a best-fit, Least Squares line on the given set of data, then f(x) will be given by:
f(x) = a + bx, where a and b are coefficients to be determined.
Similarly, if we want to fit a 2^{nd} degree best-fit curve, then f(x) will be given by:
$f(x) = a + bx + cx^2$, where a, b and c are coefficients to be determined.
Similarly, for a $\mathbf{m}^{th}$ **degree polynomial fit**,
$f(x) = a_0 + a_1x + a_2x^2 + ... + a_mx^m$, where a_1, a_2,...,a_m are coefficients to be determined.
To obtain the least square error, the unknown coefficients must yield zero first derivatives, which lead to the following equations:
$\delta \Pi / \delta a_j = 2\Sigma[y_i - (a_0 + a_1x + a_2x^2 + ... + a_mx^m)] = 0$, i = 1 to n, j = 0 to m

Expanding the above set of equations, we get:
$\Sigma x_i^j y_i = a_0 \Sigma x_i^j + a_1 \Sigma x_i^{j+1} + ... + a_m \Sigma x_i^{j+m}$, i = 1 to n, j = 0 to m

The unknown coefficients a_0, a_1,...,a_m can hence be obtained by solving the above set of linear equations.

5.2 Task-Module Description

In the client-server framework scenario, the client will submit the file containing the data-set over the network as input to the remote GPGPU server for performing Least Squares curve fitting. Additionally, the client must specify the order of the polynomial-curve that has to be fitted on the input data. The resultant polynomial coefficients of the best-fit curve, along with the correlation coefficients, will then be sent back to the client in the specified output file name. The following Figure 5.1 shows how the various parameters are submitted by the client via the GUI.

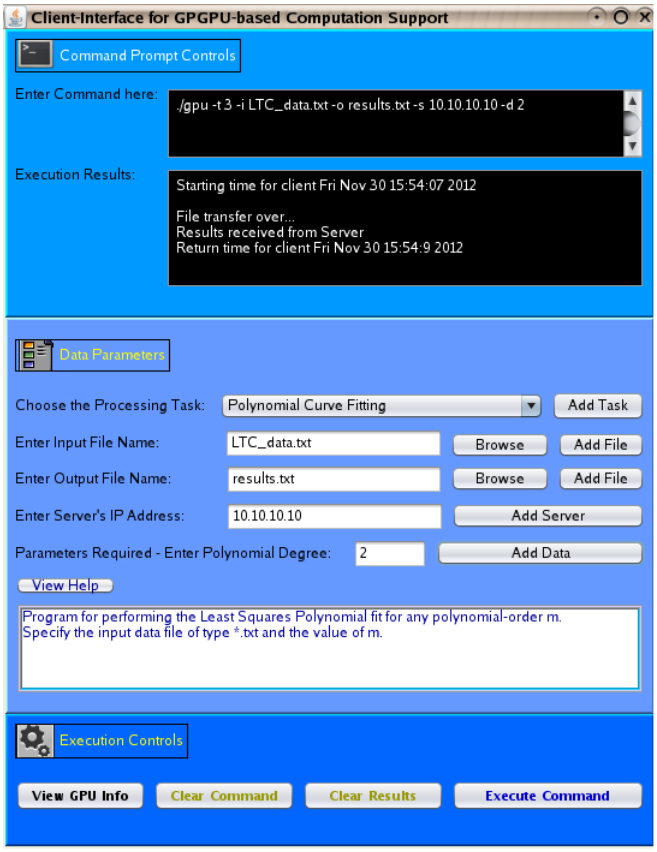


Figure 5.1: Executing Least Squares Curve Fitting

5.3 Results Accuracy Analysis: a comparison with MATLAB's *polyfit*() function

5.3.1 Methodology Adopted

The set of equations described above indicate that in order to determine the unknown coefficients a_0, a_1,...,a_m, we have to solve a system of linear equations of the form $\mathbf{AX = B}$, where the matrices A, X, B are given as shown. The matrix X can now be solved by evaluating the inverse of matrix A, i.e. $\mathbf{X = A^{-1}B}$. In this project's implementation, the matrix X has been solved for using the method of **Gaussian Elimination**.

Matrix X : $[a_0\ a_1\ a_2\ ...\ a_m]$
Matrix B : $[\Sigma y_i\ \Sigma x_i{}^j y_i\ \Sigma x_i{}^2 y_i\ ...\ \Sigma x_i{}^m y_i]$

Table 5.1: Matrix A

$$\begin{vmatrix} 1 & \Sigma x_i & \Sigma x_i{}^2 & ... & \Sigma x_i{}^m \\ \Sigma x_i & \Sigma x_i{}^2 & \Sigma x_i{}^3 & ... & \Sigma x_i{}^{m+1} \\ ... & & & & \\ \Sigma x_i{}^m & \Sigma x_i{}^{m+1} & \Sigma x_i{}^{m+2} & ... & \Sigma x_i{}^{2m} \end{vmatrix}$$

5.3.2 Comparison MATLAB's *polyfit*() function

MATLAB's *polyfit*() function uses an indirect method of determining the least-squares coefficients. This method is based on constructing the **Vandermonde matrix V** and then performing QR factorisation of V, where **Q is an orthogonal matrix** and **R is an upper triangular matrix** as shown. The QR factorisation is usually carried out using Holder Reflections[12].

Therefore we have:
V p = Y
If V = QR, where Q is an orthogonal matrix, then $QQ^T = I$. Substituting these values, we have:
(QR) p = Y or (QQ^T)R p = Q^TY
i.e. p = $\mathbf{R^{-1}(Q^TY)}$ or p = $\mathbf{(Q^TY)\backslash\ R}$, where \ is the special matrix division operator used by MATLAB. Hence, the *polyfit*() function returns the unknown coefficients in the array p.
Matrix p : $[a_0\ a_1\ a_2\ ...\ a_m]$
Matrix Y : $[y_1\ y_2\ y_3\ ...\ y_n]$
In this project's implementation, several comparisons were made for different polynomial orders and the coefficients were calculated for each order along with the corresponding least-squares errors for the fitted data. The data-set used was obtained from **Light Transfer Characteristic(LTC) data set** where

Table 5.2: Matrix V

$$\begin{vmatrix} 1 & x_1 & {x_1}^2 & ... & {x_1}^m \\ 1 & x_2 & {x_2}^2 & ... & {x_2}^m \\ ... & & & & \\ 1 & x_n & {x_n}^2 & ... & {x_n}^m \end{vmatrix}$$

variable x represents the Ground Truth Radiation(GTR) values and variable y represents the characteristic pixel response.
The calculated coefficients, along with MATLAB's coefficients for one pixel's data-set are shown in the following tables. Table 3 shows the LTC data-set for one pixel, Table 4 shows the calculated coefficients for that pixel and Table 5 shows the fitted data with calculated coefficients and the corresponding least-square errors. **R represents the Correlation Coefficient**in Table 4. In Table 5, the function $f(x)$ represents the best-fit polynomial with generated coefficients. The function $f_m(x)$ represents the best-fit polynomial with MATLAB's coefficients. From Table 5, the sum of Least Squares errors were calculated, i.e.:
Σe_f^2 **= 128.199937,** Σe_m^2 **= 129.651164**

According to the definition, the best-fit curve is the one which yields minimum least-squares errors. **It can be seen that the generated coefficients yield a lower least-squares error than the MATLAB coefficients and hence a best-fit curve is produced as compared to MATLAB. This procedure was repeated for several other data-sets, and consistent results were obtained.**

Table 5.3: LTC Data-set for one Pixel

x(GTR values)	y(Mean Response Values)
39.206	751.912
29.74	567.121
21.31	403.746
12.087	221.738
1.812	18.8418
0.001	1.88672

5.4 Execution Performance Analysis

The procedure of curve fitting was applied on LTC data with 6 scan lines and 6000 pixels per scan-line. The sequential version as well as the parallel versions were implemented on many platforms, and a comparison chart is shown in the Figure 5.3.

Table 5.4: Coefficients for Best-Fit Least Squares Curve

Order 1		**Order 2**		**Order 3**	
Generated	MATLAB	Generated	MATLAB	Generated	MATLAB
a_0=-8.356	a_0=-8.356	a_0=-6.5106	a_0=-6.5109	a_0=-4.7553	a_0=-4.7551
a_1=19.3496	a_1=19.3496	a_1=18.8735	a_1=18.8735	a_1=17.5105	a_1=17.5109
R=0.9997		a_2=0.0127	a_2=0.0127	a_2=0.1086	a_2=0.1086
		R=0.9998		a_3=-0.0016	a_3=0.0016
				R=0.9996	.

Table 5.5: Fitted Data from Generated Coeff and MATLAB Coeff for Order 3

$\mathbf{y}$	$\mathbf{y}_f$=$f(x)$	$\mathbf{y}_m$=$f_m(x)$	$\mathbf{e}_f$=$\mathbf{y}$-$\mathbf{y}_f$	$\mathbf{e}_m$=$\mathbf{y}$-$\mathbf{y}_m$
751.912	751.183960	752.285156	0.728027	-0.373169
567.121	569.500305	569.985718	-2.379333	-2.864746
403.746	402.053284	402.235626	1.692719	1.510376
221.738	219.903793	219.939758	1.834213	1.798248
18.8418	27.321678	27.321703	-8.479877	-8.479902
1.88672	-4.736779	-4.737589	6.623499	6.624309
			Σe_f^2 = 128.199937	Σe_m^2 = 129.651164

From the performance chart, it is evident that tremendous speed-ups have been achieved, the speed-up achieved going up to 73x. The various speed-up's obtained, for varying polynomial orders, are summarized in the following table of Figure 5.2.

Comparison Benchmark:
Nvidia Tesla C1060
Compute Capability: 1.3
Clock Rate: 1.3 GHz
No. of Processor Cores: 240
Total Global Memory: 4GB
Speed-up = Execution time on Test Platform / Execution time on Comparison Benchmark

Name of Platform	Platform Specifications	Speed-up Obtained over this Platform
Itanium-2	Model: 1 , Architecture: IA-64 Clock Rate: 1.66 GHz	Order 1: 47x Order 2: 66x Order 3: 73x
Dec Alpha EV5	Model: 21164, Architecture: DEC 64-bit RISC Clock Rate: 0.5 GHz	Order 1: 11x Order 2: 37x Order 3: 45x
Intel Xeon	Model: X5570 Architecture: Nehalem-EP, x86-64 Clock Rate: 2.93 GHz	Order 3: 18x

Figure 5.2: Speed-ups Obtained from Least Squares Curve Fitting

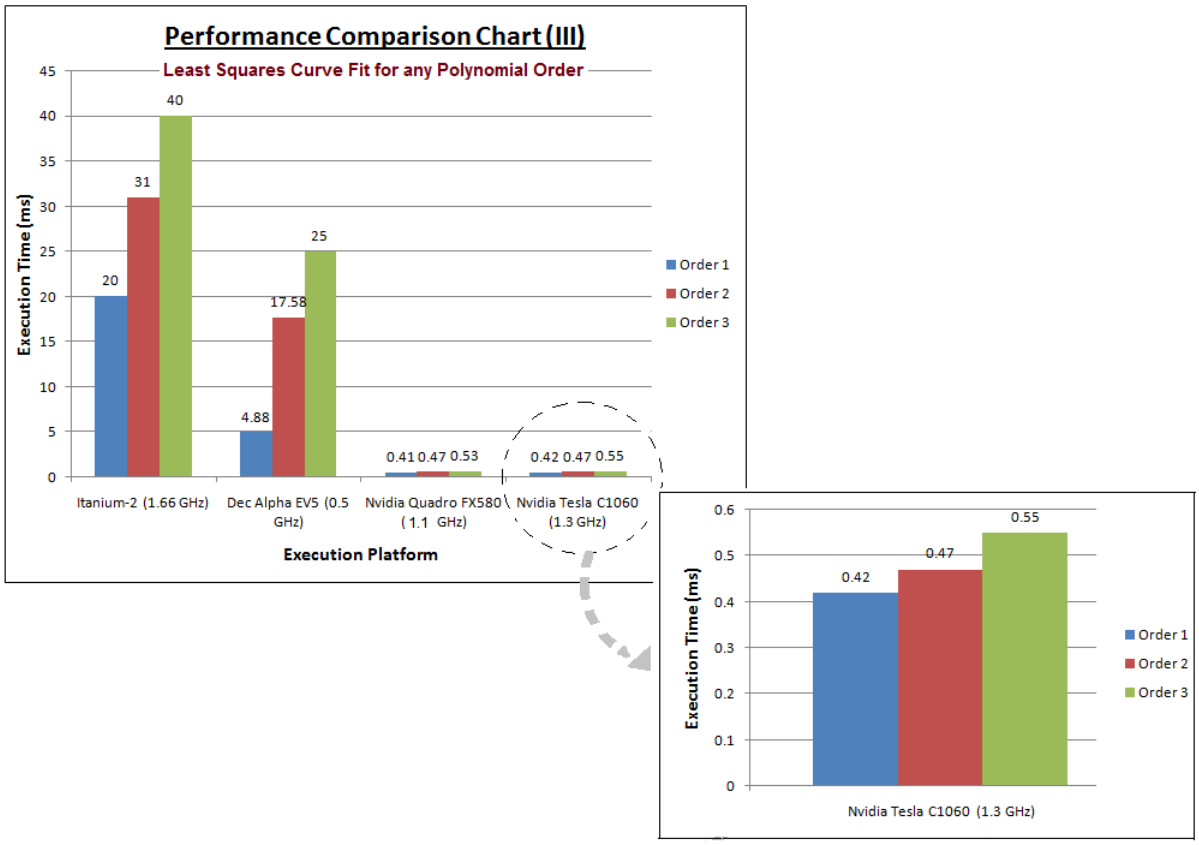


Figure 5.3: Performance Chart for Least Squares Curve Fitting

Chapter 6

GPGPU Information Generation

6.1 Task-Module Description

This module serves as a utility for the client, which when executed gives the client all the information about the various GPU resources available on the Remote GPGPU server. A complete listing with all the specifications of each GPU resource is generated in the form of an XML file which is then sent back to the client. On the GUI, the client can view this data in the form of a tree-structure with associated attributes as shown in the Figure 6.1. As the figure indicates, the client only needs to specify the IP address of the server and the name of the output file. The resultant XML file is sent back to the client and this data can be viewed in the form of an xml-tree. Various GPU specifications include:

- Name
- Compute Capability
- Warp Size
- Device Overlap
- Total Constant Memory Size
- Total Global Memory Size
- Shared Memory Per Block
- Clock Rate
- Multi Processor Count
- Registers Per Block

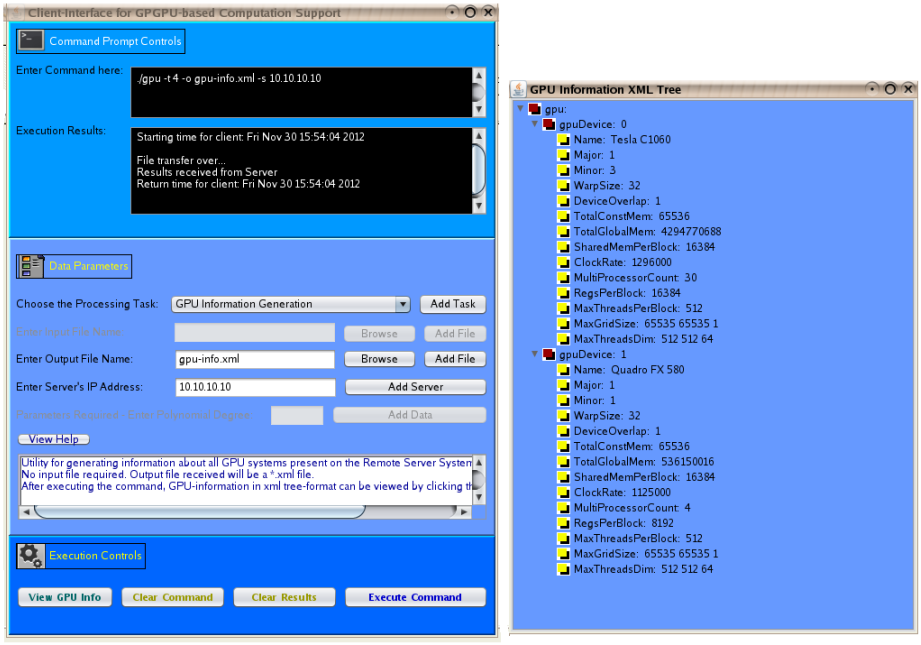


Figure 6.1: Executing GPU Information Generation Utility

- Maximum Threads Per Block
- Maximum Grid Size
- Maximum Threads Dimensions

This module has been developed with the intention that it may help the user to decide which GPU resource is the best for executing a particular task. This utility for generating information about remote GPGPU resources can be extended to enable the client to choose the GPGPU resource on which he or she wants to execute the chosen task. This would involve associating resource allocation algorithms with the framework to allocate the available GPGPU resources in the best possible way.

Chapter 7

Edge Detection from an Image and Edge Ranking for MTF Determination of an Imaging System

7.1 Introduction

Edge detection is one of the most principal techniques for detecting discontinuities in the gray levels of image pixels. An edge is a set of connected pixels that lie on the boundary between two regions. Blurred edges tend to be thick and sharp edges tend to be thin. The intensity profiles for an ideal edge and a step 'ramp' edge are shown in Figure 7.1. The task of edge detection can be carried out by convolving gradient-based filters with the input image in both spatial and frequency domains. Some simple first-derivative based filters include Sobel, Prewitt and Robert's filters while Laplacian-of-Gaussian and Canny-edge detection are more advanced edge detection techniques. The shape of the Laplacian-of-Gaussian filter, along with a 5x5 mask approximating the Laplacian-of-Gaussian filter in the spatial domain is shown in Figure 7.2 [16].
The quality of satellite camera images is regularly verified in flight. The Modulation Transfer Function (MTF) is one of the criteria of imaging quality. MTF is a parameter used for measuring the sharpness of an imaging system [14][15]. Measuring the MTF is essential to carry out the focusing of a telescope, or to implement a deconvolution filter whose goal is to enhance the image contrast or reduce the noise. Its knowledge also helps us to compare the characteristics of different known and unknown imaging systems. There are several methods for determining the MTF of an imaging system. MTF is mathematically given as the Discrete Fourier Transform (DFT) of the Line Spread Function (LSF). The LSF can be calculated as the first derivative of the Edge Spread Function (ESF).

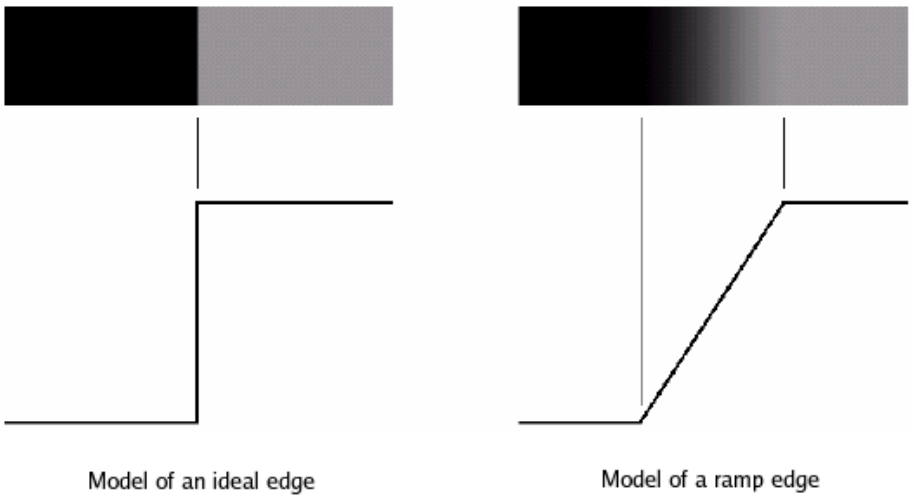
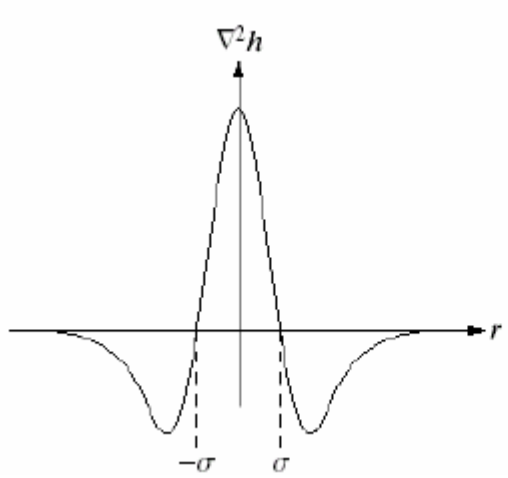


Figure 7.1: Edge profiles for an ideal and ramp edge

In order to develop the ESF, it is essential to determine the best edge from the target image so that an edge profile can be developed and then the line spread function and hence the MTF, can be computed accordingly. For regular image sizes, the human visual system is adept enough to identify suitable images from the image. But considering huge image data-sets, such as those obtained from satellites, the image size may range in few gigabytes and in such a case, manual inspection of images for determination of the best suitable edge is not plausible. In such a scenario, edge profiling tasks have to be automated and this brings into focus GPGPUs which are ideal for such tasks.

However, the process of edge profiling for huge datasets brings into focus certain issues such as:

- The edge detection algorithm that should be used depends on factors such as the image thresholding levels, acceptable noise level in the image, the target imaging system, etc.
- Once all the edges have been detected in the target image, some algorithm has to be devised for ranking the detected edges, so that the best edge can be chosen. Some such parameters for ranking edges, specifically for MTF determination include: thickness of the edge (the ideal edge is one pixel thick), length of the edge typical edges have the length of 100 or more pixels, orientation of the edge for MTF determination, vertical or near-vertical edges are preferred.
- After determining the best ranked edge, the location of such an edge in the image also has to be determined, so that the best edge, along with its



0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

Figure 7.2: The Laplacian-of-Gaussian filter

location and its edge profile can be given as output to the client. The edge ranks can be stored in a vector array. Similar arrays or matrices can be constructed for storing other edge profile information and can be supplied as output to the client.

- After the rank vector has been constructed, it is necessary to apply efficient sorting algorithms in order to find the maximum rank, and hence the best edge, from the array.

- Processing such a huge amount of data has associated issues such as: I/O time taken up in transferring the data to and from the client/server and I/O time taken up in transferring the data to and from the host CPU to the GPGPU. For example, on a gigabit network, transmitting a typical MTF data file with size 2.5GB would itself take 20 seconds! In order to reduce the transmission latency incurred, one option is to apply lossless data compression techniques to the data. One solution for minimizing the latency incurred in transferring data between the CPU and the GPGPU is to use memory-mapped I/O, where the GPU and the host CPU would share the same memory address space. This would eliminate the need for copying data frequently to and from the GPU to the CPU. NVIDIA GPUs with compute capability 2.x or higher possess such a feature.
  Another option is the use of multiple streams where transfer of data between GPU and CPU can be overlapped in time with kernel execution. A stream represents a queue of GPU operations that get executed in a specific order. Use of streams involves the allocation of page-locked or pinned memory on the host. Page-locked buffers have an important property: the operating system guarantees us that it will never page this memory out

to disk,which ensures its residency in physiscal memory, i.e. it becomes safe for the OS to allow an application access to the physical address of the memory, since the buffer will not be evicted or relocated. Knowing the physical address of a buffer, the GPU can then use direct memory access (DMA) to copy data to or from host. Overlap of such data transfers with kernel execution in an asynchronous manner is supported by certain GPU's which have the 'deviceOverlap' property enabled, such as in NVIDIA GPU's with compute capability higher than 1.1.

7.2 Mathematical analysis for Edge-Rank Determination

As of now, three prime parameters have been considered for determining the rank (R) of an edge: the edge thickness (t), edge length (l) and the edge angle made with the x-axis (Θ). Using these three parameters, the following formula has been proposed for determining the rank of an edge for MTF determination:

R = (l-t)/10 + abs(Θ*) where l, t are in units of pixels,* Θ *is in radians and* $0 \leq \Theta \leq \Pi/2$.

From the above formula it can be seen that suitable edges should be sufficiently long and should ideally be single pixel thick and vertical or near-vertical edges are preferred.
The edge angle Θ can be calculated with the help of the image gradient. The gradient of an image f(x,y) at location (x,y) is given by the vector $\bigtriangledown f$.

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

$$\nabla f = mag(\nabla f) = \left[G_x^2 + G_y^2\right]^{1/2}$$

If $\alpha(x,y)$ represents the direction angle of the image vector $\bigtriangledown f$ at location (x,y), then $\alpha(x,y) = \tan^{-1}(G_y/G_x)$.
The direction of an edge at (x,y) is perpendicular to the direction of the gradient vector at that point.
The above mentioned rank formula was used for edge detection and edge profiling was carried out. Two such sample test images, along with the corresponding edge ranks, lengths and starting coordinates of the edges found are shown in Figures 7.3 and 7.4. The first sample image depicts a typical satellite image of

a city landscape. Edge detection on such images help in detecting major roads, rivers and other important geographical features. The second sample image was chosen keeping in mind that vertical edges are preferred for MTF determination. The best edge found, i.e. the edge with the highest rank has been highlighted in a blue circle.

Original Image | Edge Detection without Thresholding | Edge Detection with Thresholding

Edge Rank	Edge Length	Starting Coordinates
1.9961	6	(123,6)
0.9976	6	(124,6)
1.6956	7	(117,18)
1.8036	6	(116,45)
2.9109	15	(115,49)
2.5943	14	(116,49)
1.9172	13	(117,49)
1.3041	12	(118,49)
2.0296	11	(119,49)
2.1978	10	(118,59)
1.8505	9	(122,74)
1.1059	8	(123,79)
....	....	....

Figure 7.3: Edge Detection and Ranking for sample-image1

From the tables of edge-rank, edge-length and starting coordinates, it can be seen that the best found edge in the first image has a rank of 2.91 and and edge length of 15 pixels and its starting coordinates is (115,49); in the second image the best found edge has a rank of 1.6156 with an edge length of 16 pixels and starting coordinates at (130,132).

7.3 Task-Module Description and Execution Performance Analysis

In order to execute or fire the task of edge detection and ranking at a remote GPGPU server, the client needs to specify the following general parameters: server IP address, name of input and output files. Also, the client needs to specify two special parameters: the user-defined *thresholding level* and the *maximum*

Original Image

Edge Detection with Thresholding

Edge Rank	Edge Length	Starting Coordinates
1.1218	8	(168,72)
0.7974	6	(163,80)
0.7384	7	(130,131)
0.6768	6	(131,131)
1.6156	16	(130,132)
1.5312	15	(131,132)
1.4121	14	(132,132)
....	....	....

Figure 7.4: Edge Detection and Ranking for sample-image2

intensity level that the user wishes to appear in the output image. Thresholding is done on the output image so that false or weak edges such as those resulting from noise can be eliminated. Upon execution, the server returns the edge-detected image and various edge-ranking parameters such as edge length, edge rank and position coordinates of the detected edge. Figure 7.5 shows the GUI configuration for executing the edge detection and ranking task.

The algorithm for edge detection and ranking was performed on an image dataset containing 25000 scanlines and 6000 pixels per scanline and each pixel value being represented by 16 bits of type short int. The algorithm has been designed keeping in mind that the datasets obtained from satellites can range up to many gigabytes and fitting such huge data directly in the memory of the GPGPU may not always be possible. Thus this algorithm takes an iterative approach and divides the input datset into sub-images that can be fitted into the GPGPU memory and processes each sub-image per iteration with the help of multiple streams so that memory transfers between the host and the GPGPU can be overlapped asynchronously with kernel execution. A parallelized, CUDA/C based implementation has been developed which has been tested on the Nvidia Tesla C1060 GPGPU and Nvidia Quadro 4000 graphics-card. The sequential version of the same program has been implemented and tested on many different platforms. A comparative performance chart, along with the corresponding speed-ups obtained, is shown in Figure 7.6. From the

Figure 7.5: GUI configuration for Edge Detection and Ranking

performance chart, it can be seen that significant speed-up has been obtained due to parallelization of the edge detection and ranking process. The individual speed-up's for different platforms have been calculated keeping the performance on the Nvidia Quadro 4000 as the comparison benchmark. The speed-up's obtained, along with the specifications of each platform, are summarized in Figure 7.7.

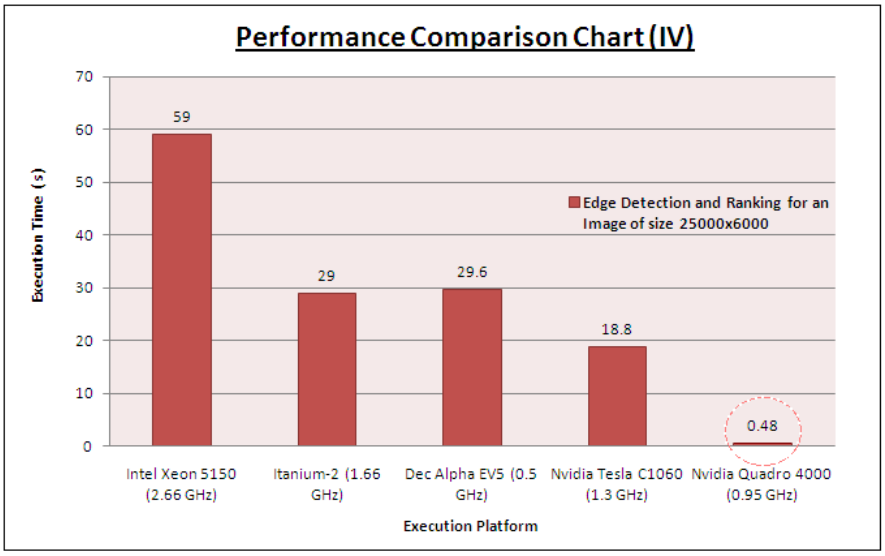


Figure 7.6: Performance Chart for Edge Detection and Ranking

Comparison Benchmark:
Nvidia Quadro 4000
Compute Capability: 2.0
Clock Rate: 0.95 GHz
No. of Processor Cores: 256
Total Global Memory: 2GB
Speed-up = Execution time on Test Platform / Execution time on Comparison Benchmark

Name of Platform	Platform Specifications	Speed-up Obtained over this Platform
Itanium-2	Model: 1 , Architecture: IA-64 Clock Rate: 1.66 GHz	60x
Dec Alpha EV5	Model: 21164, Architecture: DEC 64-bit RISC Clock Rate: 0.5 GHz	61x
Nvidia Tesla C1060	Compute Capability: 1.3 No. of Processor Cores: 240 Total Global Memory: 4 GB Clock Rate: 1.3 GHz	39x
Intel Xeon	Model: 5150 Architecture: Nehalem-EP, x86-64 Clock Rate: 2.66 GHz	122x

Figure 7.7: Speed-ups Obtained from Edge Detection and Ranking

Chapter 8

Conclusions and Future Scope for Work

As the requirements of today's scientific applications evolve, parallel data processing has become indispensable for processing applications involving huge data sets and thus the processing paradigm has shifted from multi-core to many-core processing. This brings into focus the Graphics Processing Units (GPUs) which emphasize on many-core computing. With the advent of General Purpose GPUs (GPGPU), applications not directly associated with graphics operations can also harness the computational capabilities of GPUs. Hence, it would be beneficial if the computing capabilities of a given GPGPU could be task optimized and made available to all within an organisation.

The objective of this project is the development of a platform-independent, client-server framework to support GPGPU based high performance computation with transparency to underlying hardware and operating systems. In this framework, users can choose a processing task and submit large data-sets for processing to a remote GPGPU and receive the results back, using well defined interfaces. The framework provides extensibility in terms of the number and type of tasks that the client can choose or submit for processing at the remote GPGPU server machine, with complete transparency to the underlying hardware and operating systems. The framework has been designed in such way so as to facilitate the addition of further tasks to the already existing task-set with minimum possible configuration changes. Parallelization of user-submitted tasks on the GPGPU has been achieved using NVIDIA's Compute Unified Device Architecture (CUDA).

Five major tasks or jobs can be submitted by the client to the remote GPGPU server:

- Bilinear Interpolation based Demosaicing of an Image

- Gradient Interpolation based Demosaicing of an Image
- Least-Squares Curve Fit for any Polynomial Order on Given Data
- Remote GPGPU Information Generation
- Edge Detection for an Image and Edge Ranking for MTF Determination of an Imaging System

Bilinear and Gradient Interpolation are two important interpolation techniques used to demosaic an image so that missing color components at each pixel position in an image can be recovered to generate a true color RGB image. While the Bilinear interpolation method is simpler to implement ,the Gradient-based interpolation method is more suited for images that have a lot of edges.

Least Square Errors based Curve Fitting is done in order to fit a polynomial function that best approximates a given set of data-points. Such a best-fit polynomial is then described in terms of polynomial-coefficients. In order to qualitatively assess the precision of the results obtained, a comparative analysis with MATLAB's $polyfit()$ function was done, and minimum least-squares errors and hence a best-fit curve was obtained.

The task of *Edge Detection and Ranking* was carried out for very huge image data-sets where it is not possible to manually determine edges from an image due to the mere hugeness of the data. An algorithm was proposed to rank the detected edges of an image so that the "best edge" which is ideally single pixel thick, vertical and sufficiently long can be determined. This ranking algorithm was specifically designed for determining the Modulation Transfer Function (MTF) of an imaging system. MTF is a key parameter which plays a crucial role in determining the spatial resolution of an image

The *Remote GPGPU Information Generation* module was developed so that clients can know what all GPU resources are available on a particular chosen server machine.
Each of the above mentioned tasks has been parallelized on the GPGPU and tremendous speed-ups in execution performance have been obtained as compared to the sequential versions of the same.

As future scope of work, the utility for generating information about remote GPGPU resources can be extended to enable the client to choose the GPGPU resource on which he or she wants to execute the chosen task. This would involve associating resource allocation algorithms with the framework to allocate the available GPGPU resources in the best possible way.

Also, the task of edge detection and ranking can be further extended to take into account sub-pixel accuracy while detecting edges. As a further extension, the edge detection and ranking algorithm can also be enhanced to convolve edge-detection filters in the frequency domain instead of the spatial domain.

Chapter 9

Publications

Title: GPGPU Based Parallelized Client-Server Framework for Providing High Performance Computation Support.

Authors: Poorna Banerjee, Shri Amit Dave.

Publication Journal: International Journal of Computer Science and Technology (IJCST), Vol-IV, Issue I, Jan-March, 2013.

Link to online published copy of paper: http://ijcst.com/?page_id=3679

Appendix A: Literature Review

During the course of this project, a myriad of scientific concepts and techniques had to be learned and studied, and this required an extensive survey of work done in relevant fields, study of existing trends and technologies, and future scope for enhancement. Brief summaries for some such topics have been presented in this section.

Topics covered:

- Compute Unified Device Architecture (CUDA)
- Single sensor CCD Camera and Alternate Color Filter Arrays
- Alternate Methods for Image Demosaicing
- Image Segmentation
- Modulation Transfer Function

A Overview

The following subsections provide a brief overview of each of the listed topics.

A.1 Compute Unified Device Architecture (CUDA)

References: [20] [21] [22] [23]

One of the main objectives of this project work was to carry out the parallelization of compute-intensive tasks on general purpose Graphics Processing Units (GPGPU). For achieving this goal, the programming model of Nvidia CUDA was chosen.

To a CUDA programmer, the computing system consists of a host, which is a traditional central processing unit (CPU), and one or more devices, which are massively parallel processors equipped with a large number of arithmetic execution units. In modern software applications, program sections often exhibit a rich amount of data parallelism, a property allowing many arithmetic operations to be safely performed on program data structures in a simultaneous manner. The CUDA devices accelerate the execution of these applications by harvesting a large amount of data parallelism.

A CUDA program consists of one or more phases that are executed on either the host (CPU) or a device such as a GPU. The phases that exhibit little or no data parallelism are implemented in host code. The phases that exhibit rich amount of data parallelism are implemented in the device code. A CUDA program is a unified source code encompassing both host and device code. The NVIDIA C compiler (nvcc) separates the two during the compilation process. The host code is straight ANSI C code; it is further compiled with the hosts standard C compilers and runs as an ordinary CPU process. The device code is written using ANSI C extended with keywords for labeling data-parallel functions, called kernels, and their associated data structures. The device code is typically further compiled by the nvcc and executed on a GPU device The kernel functions (or, simply, kernels) typically generate a large number of threads to exploit data parallelism. CUDA threads are of much lighter weight than the CPU threads. CUDA programmers can assume that these threads take very few cycles to generate and schedule due to efficient hardware support. This is in contrast with the CPU threads that typically require thousands of clock cycles to generate and schedule. When a kernel is invoked, or launched, it is executed as grid of parallel threads. Each CUDA grid typically is comprised of thousands to millions of lightweight GPU threads per kernel invocation.

Threads in a grid are organized into a two-level hierarchy. At the top level, each grid consists of one or more thread blocks. All blocks in a grid have the same number of threads. Each block has a unique two dimensional coordinate given by the CUDA specific keywords blockIdx.x and blockIdx.y. Each thread

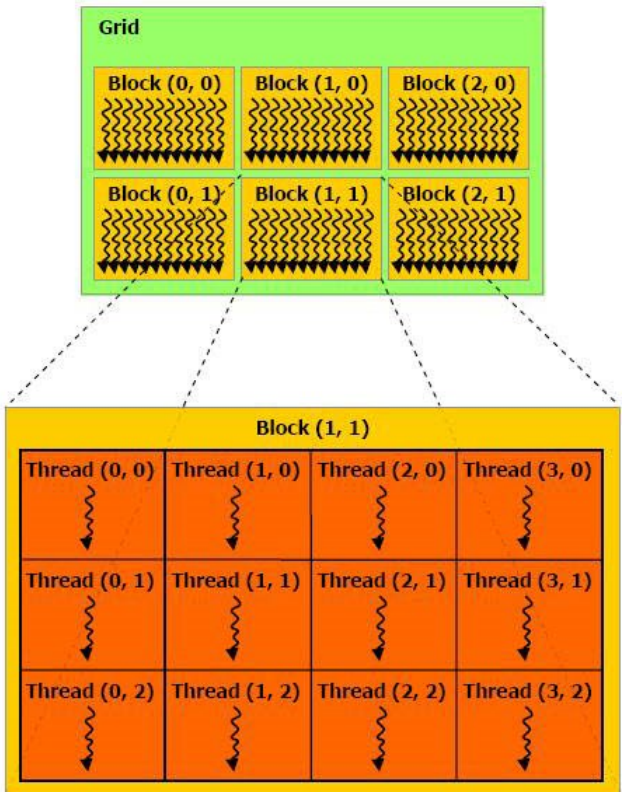


Figure A.1: The CUDA thread model

block is, in turn, organized as a three-dimensional array of threads. The coordinates of threads in a block are uniquely defined by three thread indices: threadIdx.x, threadIdx.y, and threadIdx.z. This is illustrated in figure A.1.

Figure A.2 shows an overview of the CUDA device memory model for programmers to reason about the allocation, movement, and usage of the various memory types of a device.
At the bottom of the figure, there is global memory, texture memory and constant memory. These are the memories that the host code can transfer data to and from the device, as illustrated by the bidirectional arrows between these memories and the host. Constant memory allows read-only access by the device code.

Registers and shared memory are on-chip memories. Variables that reside in these types of memory can be accessed at very high speed in a highly parallel manner. Registers are allocated to individual threads; each thread can only ac-

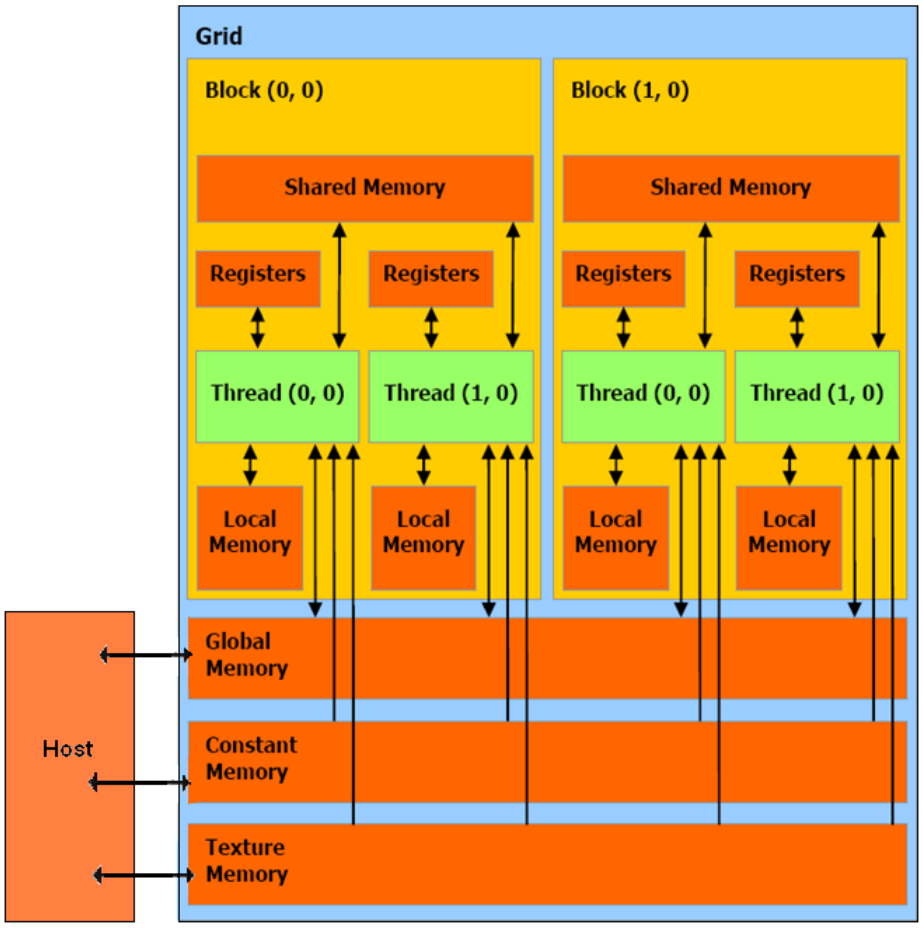


Figure A.2: CUDA device memory model

cess its own registers. Shared memory is allocated to thread blocks; all threads in a block can access variables in the shared memory locations allocated to the block. Shared memory is an efficient means for threads to cooperate by sharing their input data and the intermediate results of their work.

Like constant memory, texture memory is also read-only memory with on-chip caching and supports short-latency, high-bandwidth, read-only access by the device when all threads simultaneously access the same location.

To summarize,

- Device code can:

  - R/W per-thread registers
  - R/W per-thread local memory
  - R/W per-block shared memory
  - R/W per-grid global memory
  - Read only per-grid constant memory
  - Read only per-grid texture memory

- Host code can:
  Transfer data to/from per-grid global, constant and texture memories.

The architecture of a typical CUDA-capable GPU is organized into an array of highly threaded streaming multiprocessors (SMs). Each SM has a number of streaming processors (SPs) that share control logic and instruction cache. Although CUDA registers, shared memory, constant memory and texture memory can be extremely effective in reducing the number of accesses to global memory, one must be careful not to exceed the capacity of these memories. Each CUDA device offers a limited amount of CUDA memory, which limits the number of threads that can simultaneously reside in the streaming multiprocessors for a given application. In general, the more memory locations each thread requires, the fewer the number of threads that can reside in each streaming multiprocessor (SM) and thus the fewer number of threads that can reside in the entire processor.

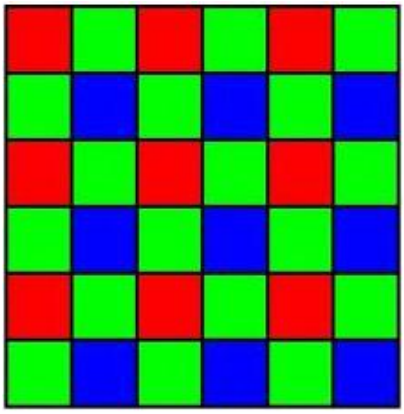


Figure A.3: The conventional Bayer Pattern color filter array

A.2 Single sensor CCD Camera and Alternate Color Filter Arrays

References: [8] [18]

With the advent of technology, the resolutions for single-sensor imaging devices such as digital cameras are becoming better by the day but the core working-principle remains the same. These cameras are based on a charge coupled device (CCD) array and each sensor on the CCD captures only one sample of the color spectrum.
Digital color cameras generally use a Bayer mask over the CCD. A Bayer filter mosaic is a color filter array (CFA) for arranging Red Green Blue (RGB) color filters on a square grid of photo-sensors, as shown in figure. Each square of four pixels has one filtered red, one blue, and two green. This is because the human eye is more sensitive to green than either red or blue and the Bayer CFA is based on the knowledge of human visual perception and is consistent with the theory of YUV color space.
Since the introduction of the Bayer color filter array in 1975 digital photography has seen dramatic progress. This progress has converted digital photography from an exclusive tool available only to the aerospace industry to widespread everyday use in consumer digital photo cameras. One feature of digital photography which has remained almost unchanged throughout the decades is the layout of color components in the photo-sensitive array. The de-facto standard is the Bayer color filter array which is proven to be technologically feasible and robust enough for wide variety of applications. The performance of digital cameras depends not only on the accuracy of methods of restoration of missed color samples (demosaicing) for a given color filter array, but also on the spatial configuration of color sensors in the color filter array (CFA) itself. Some of such alternate color filter array configurations where the YUV model has not been

assumed are described below.
These alternate CFA models are based on the following constraints

- Minimization of the atomic building block for pattern. In other words it could be a 2x2, 3x2 or 3x3 pattern.
- The frequency of color components should be similar in horizontal and vertical dimensions.
- The frequency of color samples should be consistent with an established theory of human visual perceptional sensitivity.

Based on these constraints, two such patterns are shown below in Figure A.4.

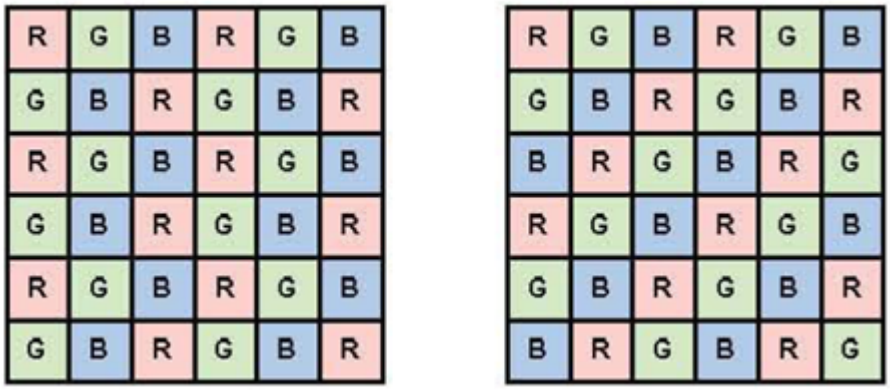


Figure A.4: Spatial distribution for alternate CFA patterns

The pattern on the left is called the 6-samples CFA pattern. It has a building block size of 3x2. On the contrary to classic Bayer CFA the colors are mixed here in equal proportions. One disadvantage of the 6 sample CFA is that it is not symmetrical to permutation of vertical and horizontal dimensions. The second pattern is called the Diagonal Bayer CFA pattern and it has a building block size of 3x3 and the colors are mixed here in equal proportions, but unlike the 6-sample CFA, it has symmetrical permutation in vertical and horizontal directions. It has been shown that alternatives to classic Bayer CFA can give good performance in some circumstances, and that there may be benefits in using a different pattern.

A.3 Alternate methods for Image Demosaicing

References: [17] [19]

In the project work carried out, two methods of image demosaicing were used: Bilinear Interpolation based demosaicing and Gradient Interpolation based demosaicing. Both these techniques are used in the spatial domain. There are other advanced techniques for image demosaicing in both spatial and frequency domains. Some of them are explained below.

High Quality Linear Interpolation based Demosaicing: Malvar et al proposed a method to demosaic an image using a Bayer Pattern. This is actually a set of 8 kernels that we need to normalize and then apply to the mosaiced image in order to demosaic it. The choice of the kernel depends on the position of the pixel.
There are 8 cases as illustrated in Figure A.5:

- 2 cases to determine the red and green values on a blue pixel.
- 2 cases to determine the blue and green values on a red pixel.
- 4 cases to determine the red and blue values on a green pixel.

To determine a color, we calculate the average of the neighbor with the same color, but we also incorporate the green channel into each of them. This creates a link between the channels, and thus the rendering of the edges should be better than a basic linear interpolation (nearest neighbor or bilinear).

Frequency Domain Methods for Image Demosaicing: The color filter array (CFA) image can be represented as the combination of an achromatic or luma component at baseband and two chrominance components modulated at high spatial frequency. CFA demultiplexing methods must try to exploit signal characteristics and correlations as much as possible to obtain the best signal quality. Let f_{CFA}[n1,n2] represent the output of the CFA sensor, sampled on a rectangular sampling lattice Λ. The horizontal and vertical sample spacings are equal, and this sample spacing X is used as the unit of length, called the pixel height (px). Thus, the lattice Λ is simply the integer lattice Z^2.

In the Bayer CFA, Λ is partitioned into three disjoint subsets that are each shifted sub-lattices of Λ, referred to as Λ_R, Λ_G and Λ_B illustrated in Figure. A.6. The origin is set at the upper left corner of the image on a green sample with a red sample to the right and a blue sample below.

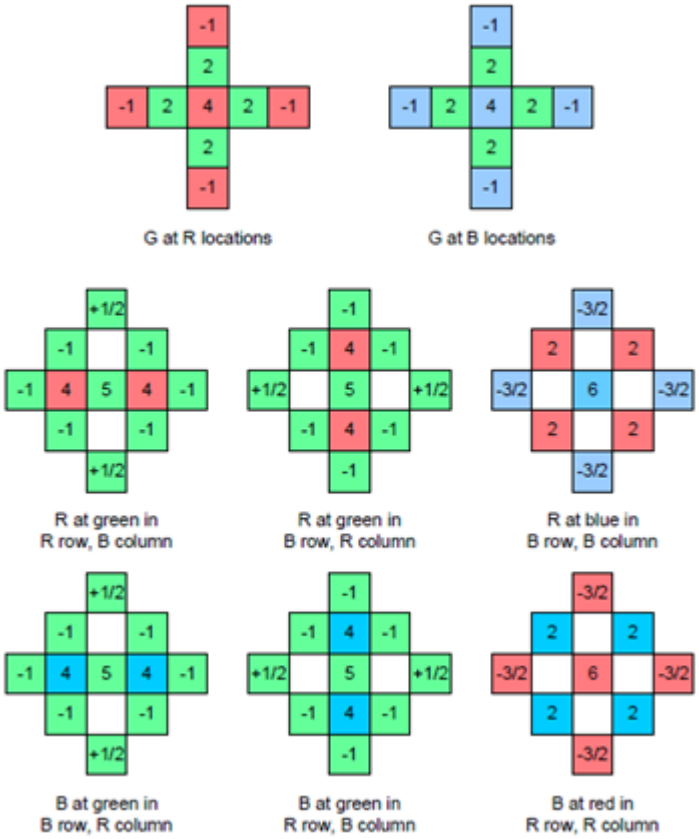


Figure A.5: Filter Coefficients for High Quality Linear Interpolation

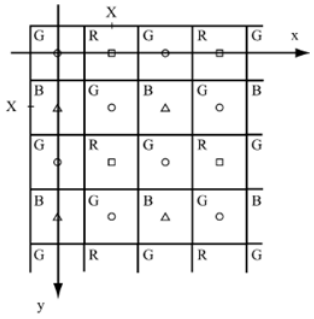


Figure A.6: Bayer CFA sampling structure

A.4 Image Segmentation

References: [16]

Segmentation subdivides an image into its constituent regions or objects. The level to which the subdivision is carried depends on the problem being solved. Image segmentation algorithms generally are based on one of two basic properties of intensity values: *discontinuity and similarity.*

- Image segmentation techniques based on the discontinuity-approach partition an image based on abrupt changes in intensity, such as points, lines or edges in an image.
- The principal approaches in the second category are based on partitioning an image into regions that are similar according to a set of pre-defined criteria. Thresholding, region growing, region splitting and merging are examples of methods in this category.

For detecting points in an image, the idea is that an *isolated point* will be quite different from its surroundings, and thus be easily detectable by running the type of mask shown below throughout the image:

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

$$R = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9$$
$$= \sum_{i=1}^{9} w_i z_i$$

Using the mask shown, we say that a point has been detected at the location on which the mask is centered if —R— $\geq$ T, where T is a pre-defined thresholding value.
For detecting *one-pixel thick lines* in an image, varieties of masks can be used for detecting lines at varying angles such as those shown in Figure A.7.
The first mask will respond most to horizontal lines in an image. The next mask is used for detecting lines inclined at positive 45 degrees in an image, the third mask gives maximum response to vertical lines in an image and the last mask is used for lines inclined at negative 45 degrees in an image.

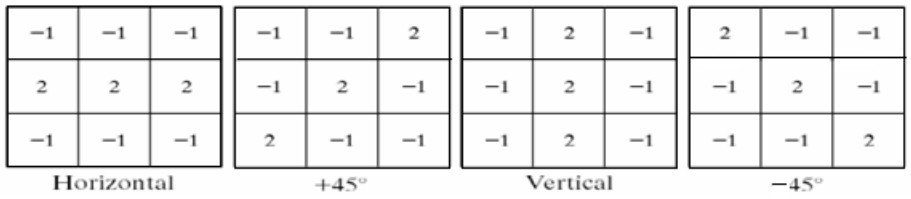


Figure A.7: Line detection masks

Thresholding: Because of its intuitive properties and simplicity of implementation, image thresholding enjoys a central position in applications of image segmentation. One way to extract the objects from the background is to select a threshold T that separates the background from the region of interest. Then any point (x,y) for which f(x,y) $\geq$ T is called an *object point*; otherwise, the point is called a *background point*. Here f(x,y) is the image on which thresholding is to be carried out. Thresholding using single and multiple threshold-levels is shown in the Figure A.8.

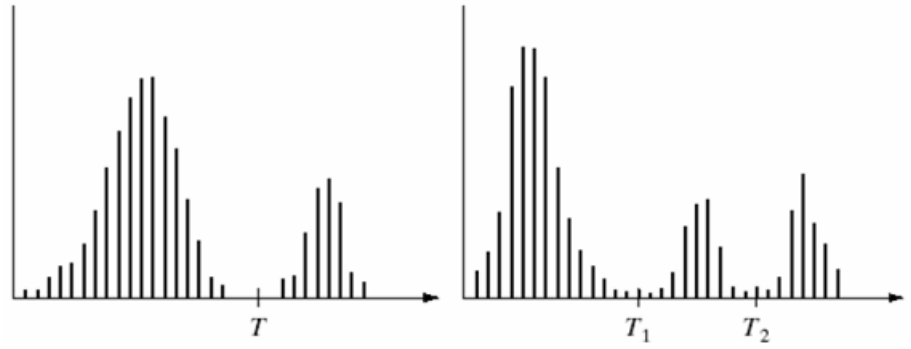


Figure A.8: Gray-level histrograms that can be partitioned by a (a) a single threshold (b) by multiple thresholds

Thresholding may be viewed as an operation that involves tests against a function T of the form where f(x,y) is the gray level of point (x,y) and p(x,y) denotes some local property of this point. When T depends only on f(x,y) (that is, only on graylevel values) the threshold is called *global*. If T depends on both f(x,y) and p(x,y), the threshold is called *local*. If, in addition, T depends on the spatial coordinates x and y, the threshold is called *dynamic or adaptive*.

Region Growing: Region growing is a procedure that groups pixels or subregions into larger regions based on predefined criteria. The basic approach is

to start with a set of seed points and from these grow regions by appending to each seed those neighboring pixels that have properties similar to the seed (such as specific ranges of gray level or color). The selection of similarity criteria depends not only on the problem under consideration, but also on the type of image data available. Another problem in region growing is the formulation of a stopping rule.

A.5 Modulation Transfer Function

References: [14] [15] [16]

The quality of satellite camera images is regularly verified in flight. The modulation transfer function (MTF) is one of the criteria of image quality. The MTF is used to assess the aptitude of the imager to distinguish details contained in a scene. The knowledge of the MTF makes possible estimation or comparison of the in-flight performances of different satellites, and is useful for calculating deconvolution filters.

For linear and spatially invariant systems, the relation between the real image *s(x,y)* and the perfect image, in the radiometric sense, *e(x,y)*, is of the form:

$$s(x,y) = \int\int_{R\times R} e(\xi,\eta)h(\xi-x,\eta-y)d\xi d\eta = (e*h)(x,y),$$

that is , the convolution of the perfect image by the impulse response *h(x,y)*.

Let *H, S, E* be the Fourier transforms of *h, s, e*.
We obtain the relation: $S(f_x, f_y) = E(f_x, f_y)H(f_x, f_y)$

The function $H(f_x, f_y)$ is called the system transfer function. This transfer function is usually normalized such that H(0,0) = 1, which is equivalent to:

$$\int\int_{R\times R} h(\xi,\eta)d\xi d\eta = 1$$

This results from an energy conservation hypothesis between the systems input and output. The transfer function is a complex function of real variables. It can then be expressed by its modulus and its phase and is called the optical transfer function (OTF). The modulus is called the modulation transfer function (MTF). If we consider perfectly centered optical instruments, the impulse response is even and the OTF is real. In this case, MTF and impulse response are two equivalent representations of the quality of an optical instrument. Several methods can be used to measure the MTF. The most intuitive one involves calculating the Fourier transform of an impulse using a point source as an object. A variant of this method is the step edge method. It consists of observing,

using the system being tested, a radiometric step edge whose luminance is a Heaviside function in the direction perpendicular to the edge. The system MTF is deduced from the Fourier transform of the image of the step edge. The utilization of artificial patterns should also be mentioned. The MTF can be calculated by measuring the contrast of periodic pattern images at different frequencies. Last, the MTF can easily be estimated from an image of any landscape if the reference landscape is known. Nevertheless, all these methods are bivariate, because they need a reference landscape. For all these methods, the main problem is the specific nature of the landscape being viewed to be able to assess the MTF correctly: point, step edge, periodic patterns, or known real landscape (bivariate method).

In the edge method, after the selection of some suitable edges in the image, at first the algorithm determines edge locations with sub-pixel accuracy; under the assumption that the chosen edges are on a straight line, the alignment of all edge locations is estimated with a least squares fitting technique.

The edge profiles, which are centered at each edge pixel and have the direction perpendicular to the edge, are interpolated with cubic spline functions. These cubic spline functions are averaged and interpolated with an analytical function in order to obtain an empirical Edge Spread Function (ESF). The ESF is then differentiated to obtain the Line Spread Function (LSF). Finally the LSF is Fourier-transformed and normalized to obtain the corresponding MTF. Finally, after the Fourier transformation, the computed MTF is scaled in the frequency axis in order to represent the calculated MTF in terms of the Nyquist frequency of the image. In addition, the Full Width at Half Maximum (FWHM) value is also computed from the estimated LSF. This process is illustrated schematically in Figure A.9.

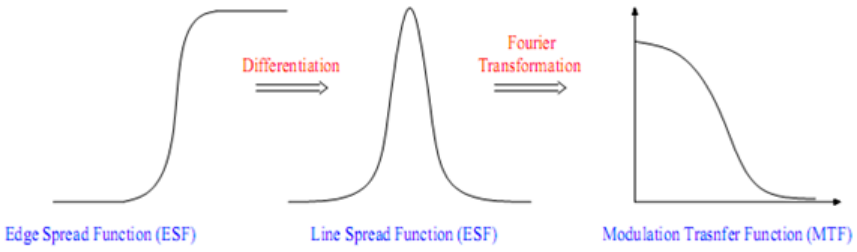


Figure A.9: Edge Based Estimation of MTF

Appendix B: Plagiarism Report

Here the plagiarism report will come.

References

Research Papers and Books

1. Gow-Hsing King, Jan-Jan Wu. "A High-Performance Multi-user Service System for Financial Analytics Based on Web Service and GPU Computation". International Symposium on Parallel and Distributed Processing with Applications, May 2009.

2. Yihan Guo, Meiping Shi, Yan Li, and Duoneng Liu. "Research on Fast Image Mosaic Based on CUDA". Fourth International Symposium on Computational Intelligence and Design, 2011.

3. Nicolas Seiller, Nitin Singhal, Kyu Park. "Object Oriented Framework For Real-Time Image Processing On GPU". Proceedings of 2010 IEEE 17th International Conference on Image Processing September 26-29, 2010, Hong Kong.

4. Shane Ryoo, Christopher I. Rodrigues, Sara S. Baghsorkhi, Sam S. Stone. "Optimization Principles and Application Performance Evaluation of a Multithreaded GPU Using CUDA". Proceedings of 13th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, Pages 73-82, 2008.

5. Naga K. Govindaraju, Jim Gray, Ritesh Kumar, Dinesh Manocha. "GPUTeraSort: High Performance Graphics Coprocessor Sorting for Large Database Management." Microsoft Research, SIGMOD Publications, Pages 325-336, December 2006.

6. Zhe Fan, Feng Qiu, Arie Kaufman, Suzanne Yoakum-Stover. "GPU Cluster for High Performance Computing". ACM / IEEE Supercomputing Conference 2004, November 06-12, Pittsburgh, PA.

7. Nicolas SeillerComputer. "Architectures for Spatially distributed Data". NATO ASI Proceedings, Centraro Holly, Italy, 1985.

8. Remi Jean. "Demosaicing with the Bayer Pattern". Department of Computer Science, University of North Carolina, 2006.

9. Rastislav Lukac, Karl Martin, Konstantinos N. Plataniotis. "Demosaicked Image Post-processing Framework". Bell Canada Multimedia Laboratory, 2009.

10. K.Gorokhovvskiy, J. A. Flint, S. Datta. "Alternative Color Filter Array Layouts For Digital Photography". IEEE 2006 Conference Publications.

11. David B. Kirk, Wen-mei W. Hwu. "Programming Massively Parallel Processors - A Hands-on Approach". Morgan Kaufman Publishers, 2010.

12. Giorgia Zucchelli , Marieke van Geffe. "Speed up numerical analysis with MATLAB". 2011 Technology Trend Seminar, MathWorks Inc.

13. S. S Sastry. "Introductory Methods of Numerical analysis". PHI Publications, 2007.

14. Jean-Marc DELVIT, Dominique LEGER, Sylvie ROQUES , Christophe VALORGE. "Modulation Transfer Function Measurement Using Non Specific Views". Image and Signal Processing for Remote Sensing, Proceedings of SPIE, Vol. 4885, 2003.

15. Mattia Crespi , Laura De Vendictis. "A Procedure for High Resolution Satellite Imagery Quality Assessment". Open Access Sensors Journal, 2009.

16. Rafael C. Gonzalez, Richard E. Woods. "Digital Image Processing". 3rd Edition, Prentice Hall Publications, 2009.

17. Henrique S. Malvar, Li-wei He, Ross Cutler. "High Quality Linear Interpolation for Demosaicing of Bayer-Patterned Color Images". IEEE International conference on Acoustics, Speech and Signal Processing, Vol-3, Pages 485-8, 2004.

18. K. Gorokhovskiy, J.A. Flint and S. Datta. "Alternative color filter array layouts for digital photography." Research in Microelectronics and Electronics, Pages 265-268, IEEE Conference Publications, 2006.

19. Eric Dubois. "Frequency-Domain Methods for Demosaicking of Bayer-Sampled Color Images". IEEE Signal Processing Letters, VOL 12, No.12, December 2005.

20. NVIDIA CUDA C Programming Guide. NVIDIA Corp.

21. NVCC CUDA Compiler Driver Guide. NVIDIA Corp.

22. NVIDIA GPU Computing Webinars Introduction to CUDA. NVIDIA Corp.

23. NVIDIA CUDA Toolkit 4.1 CUBLAS Library Documentation

Web Based References

- Calling Linux Commands from Java Applications - www.sitepoint.com
- Introduction to CORBA - www.ois.com
- Tesla C1060 board Specifications v03 - www.nvidia.com
- Remote Procedure Calls (RPC) XDR Datatypes - www.hp.com/open-vms-systems
- Passing complex data structures through XDR - www.oracle.com/onc-developers-guide
- Linux GUI Programming - http://ubuntuforums.org/index.php
- Java Native Code Programming - www.machtech.com/jni
- Socket Programming - www.thegeekstuff.com/c-socket-programming/
- Cuda Programming with Java - www.TheServerSide.com/jcuda
- Indian Space Research Organisation (ISRO) http://www.isro.org/scripts/Aboutus.aspx
- Space Applications Center (SAC) http://www.sac.gov.in/sacwebi/sacHomePage.iface