Semi-Global Matching on Compute Unified Device Architecture (CUDA)

Report for BTech 451

by

HaoYu Gan

Supervisor: T.N. Chan and Professor Reinhard Klette

Tamaki Innovation Campus
Department of Computer Science
The University of Auckland
New Zealand
October 2012

Abstract

Compute unified device architecture (CUDA) is a parallel computing architecture developed by Nvidia for graphics processing. CUDA is the computing engine in NVIDIA graphics processing units (GPUs) that is accessible to software developers through variants of industry standard programming languages.

The GPU, as a specialized processor, addresses the demands of real-time high-resolution 3D graphics compute-intensive tasks. As of 2012, GPUs have evolved into highly parallel multi-core systems allowing very efficient manipulation of large blocks of data. This design is more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel.

The project is divided into two parts. The first part is researching on CUDA and benchmarking on certain GPU cards from NVIDIA by using COTS (commercial off the shelf) applications, gathering all the results and analysis the performance in order to gain a fully knowledge of CUDA.

The second part continues testing without COTS, but also implementing semi-global matching (SGM), at first on a standard PC and then by employing CUDA as the computing tool. Semi-global matching is a very popular method in stereo analysis and thus a good candidate to be discussed for CUDA implementation.

Keywords: CUDA, GPU, CPU, Semi-Global Matching

Acknowledgments

I would like to thank everyone involved in this project – I thank T.N. Chan and Prof. Reinhard Klette for project supervision, Dr. S. Manoharan for his advice, and Ralf Haeusler for introducing me to semi-global matching.

Hao Yu, Gan
University of Auckland
October 24, 2012

Contents

Chapter 1

Introduction

This is 4th year final project for student who majoring in Bachelor of Technology (IT) degree. The project carries weight of two semester University courses, which it is Btech451 Part A and Part B for semester 1 and semester 2 respectively.

1.1 The Project

This project is an appraisal of a computing environment based on NVIDIA ARM and Intel for COTS and research applications. We will benchmark COTS (commercial off the shelf) applications including a few that have been optimized and certified for NVIDIA CUDA (Quadro and Tesla). Testing based on Quadro 2000 graphics card and Tesla C2075 companion card, also included a new technology called Maximus which combines Quadro and Tesla products. As NVIDIA likes to reiterate to their customers its not a new product, it's a new technology – a new way to use NVIDIAs existing Quadro and Tesla products together. Theres no new hardware involved, just new features in NVIDIA drivers and new hooks exposed to application developers.

1.2 Project Goal

Gain a good understanding of how CUDA Computing works, and how does semi-global matching [5] employs CUDA as the computing tool (in Semester 2). To understand today's industry emphasis, in both commercial and academically ways, also hoping to gain more knowledges about Hardware's design and architectures, not only just Software.

Semi-global matching is one way to do stereo matching, and, for example, belief propagation [2] is another option. Stereo matchers are compared on [11] for their performance.

Figure 1.1: The Compucon New Zealand

1.3 Company Information

This project is sponsored by the company Compucon New Zealand as shown in Figure 1.1.

Compucon NZ is part of an International Computer manufacturing group of companies founded in 1989 in Sydney.

The NZ operation is registered as Modern Technology NZ Ltd and has established a reputation for technical excellence based on sound engineering and other knowledge based practices. All manufacturing processes are certified by Telarc ISO 9002 quality standards at our Albany assembly plant in Auckland NZ.

The Compucon team contributes to the success of customers through their knowledge, excellence, commitment and supply of computing platforms and solutions meeting or exceeding customer expectations.

Compucon PC's are manufactured to users requirements while meeting the latest industry open systems standard. Consistency of components and prototype testing ensure seamless inter-operability with other systems.

The Compucon New Zealand:
`http://www.compucon.co.nz/`

Chapter 2

Research

Before it starts, some researches on CUDA are necessary, any topics related to CUDA are encouraged for further studying. For this project, i will need to set up a computer for benchmarking, so hardware knowledge such as motherboard, CPU, GPU, Memory RAM and Hard disk driver will be part of the area that require further study.

2.1 CPU serial vs CUDA parallel

CPU serial:

CPU serial Computing refers of a computer system that carries out the instructions of a computer program, to perform the basic arithmetical, logical, and input/output operations of the system by using central processing unit. It simply means that most of the thing is done by CPU alone.

CUDA parallel Computing:

CUDA is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU). Computing is evolving from "central processing" on the CPU to "co-processing" on the CPU and GPU. To enable this new computing paradigm, NVIDIA invented the CUDA parallel computing architecture that is now shipping in GeForce, Quadro, and Tesla products, representing a significant installed base for application developers. (NVIDIA website)

With the development of the graphics card, the GPU is more powerful, somehow it has gone beyond the general-purpose CPU. If such a powerful chip is only for processing graphics, then it is too wasted, so NVidia launched CUDA that enabled graphics card can be used for purposes other than the image calculation.

GPU works as following:

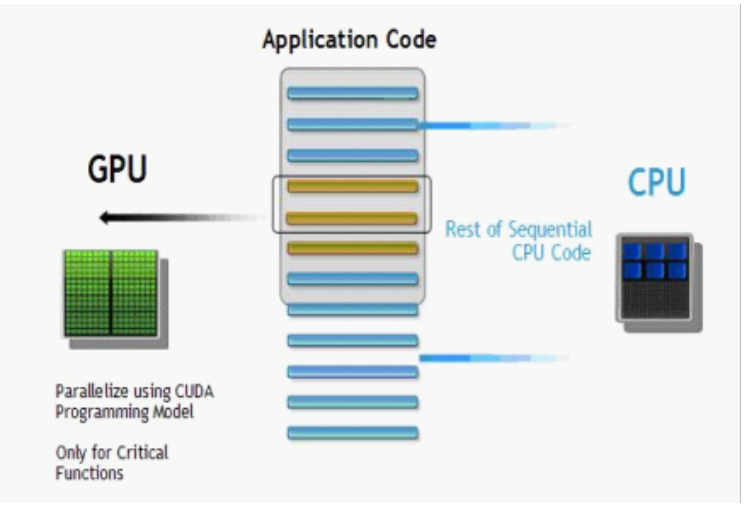

Figure 2.1: This diagram shows how an application that normally runs in the CPU of a PC is ported over to the GPU.

A software program is made up of application codes. The 1st step is to partition some sections of the code of a repeating nature to run on the GPU and other sections to remain on the CPU. Use C, C++ or other supported languages with special keywords. The sections of the code allocated to the GPU must be highly computing intensive and they tend to be the core algorithm and thus the critical parts of the application. A minor effort of code porting will result in significant performance gains. At the center of this parallel computing model is CUDA (Compute United Device Architecture) which is NVIDIAs parallel computing hardware and programming model.

So in general words, processing on CUDA is that GPU provides support for CPU.
CUDA parallel computing: To be parallel run using CPU + GPU
Example of CUDA processing flow:

1. Copy data from main memory to GPU memory

2. CPU instructs the process to GPU

3. GPU execute parallel in each core

4. Copy the result from GPU memory to main memory

2.2 Tesla C2075

GPU Specifications:	
NVIDIA Tesla GPU	Tesla C2075
CUDA Cores	448
Form Factor	9.75" PCIe x16 form factor
GPU Memory Specifications:	
Total Frame Buffer	6 GB GDDR5
Memory Interface	384-bit
Memory Bandwidth (GB/sec)	144 GB/s

Figure 2.2: NVIDIA Tesla C2075

The NVIDIA Tesla C2075 companion processor [14] is built for GPU computing. It features 448 application-acceleration cores per board, dramatically increasing performance compared to a traditional workstation. By adding a Tesla companion processor, engineers, designers, and content creation professionals can add over one Teraflop of computing potential to their workstation.

So Tesla is not like Quadro, it is compute processing intensive, Tesla is still a GPU and the cores are being used exclusively for general computing purposes to offload work from the CPU while the Quadro half of the equation handles graphical tasks.

Tesla C2075:
`http://www.nvidia.com/object/workstation-solutions-tesla.html#`

2.3 Quadro 2000

GPU Specifications:	
NVIDIA Quadro GPU	Quadro 2000
CUDA Cores	192
Form Factor	4.376" H x 7" L Single Slot
GPU Memory Specifications:	
Total Frame Buffer	1 GB GDDR5
Memory Interface	128-bit
Memory Bandwidth (GB/sec)	41.6 GB/s

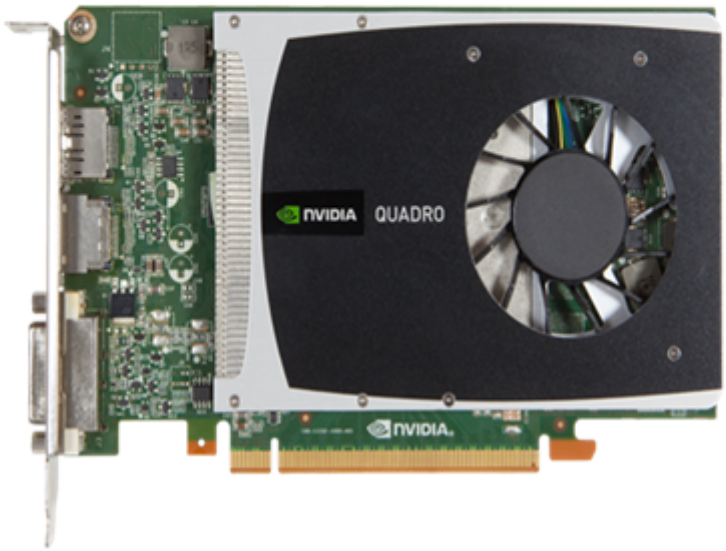

Figure 2.3: NVIDIA Quadro 2000

The Quadro 2000 is based on Nvidias Fermi architecture [12, 13], and is equipped with 192 CUDA parallel processing cores. Accompanying these is 1GB of GDDR5 RAM running over a 128-bit memory interface, and offering 41.6GB of memory bandwidth. The Quadro is compute and graphics intensive, since the algorithm/equation handles graphical tasks.

Quadro 2000:
`http://www.nvidia.com/object/product-quadro-2000-us.html`

2.4 Requirements and Set-up

2.4.1 Configuration of the Test System Model

Our test system model's information for this project is shown below:

Motherboard: Asus P9X79
http:
//www.asus.com/Motherboards/Intel_Socket_2011/P9X79/#overview

CPU: Intel i7-3930K LGA2011 http://ark.intel.com/products/63697/Intel-Core-i7-3930K-Processor-12M-Cache-up-to-3_80-GHz

Memory RAM: 8GB 16GB DDR3-1333 Quad Channel

Graphics Card: Quadro 2000, Tesla C2075, or Maximus(Q+T)

HDD: Single SATA with two different models

1. WESTERN DIGITAL WD5000AAKX Caviar Blue 500GB 7200 RPM 16MB cache SATA 6.0Gb/s 3.5" internal hard drive
2. Seagate Barracuda 7200.7 ST3120827AS 120GB 7200 RPM 8MB Cache SATA 1.5Gb/s

For a comparison purpose, we will compare the results from previous results done by Joseph, Joseph used to be one of the staff that working for the Compucon Company. He have done similar benchmarks for Quadro 2000 by using various of different system configuration.

2.4.2 Benchmarking Applications

This first half of the project is mainly on benchmarking. Hence we will use some of the benchmarking tools that are optimized and certified for CUDA graphics cards. Most of them are COTS(commercial off the shelf) which mean they are not open sources and pay to use. The following are the applications that we will be benchmark:

1. SPECviewperf 11: http://www.spec.org/
2. Cadalyst c2012 AutoCAD: http://www.cadalyst.com/benchmark-test
3. PPBM5 test for Adobe Premiere Pro CS5.5: http://ppbm5.com/

More details will be discuss when each comes to the real testing.

Chapter 3

Benchmarking

3.1 SPECviewperf 11

The first phase of testing was done using SPECviewperf 11 from the Standard Performance Evaluation Corporation. SPECviewperf is a benchmarking application that uses viewsets from various CAD applications such as Autodesk Maya, SolidWorks and Siemans NX to simulate daily CAD usage. Since testing began I have found that SPECviewperf is designed to isolate the graphics subsystem and is only reliable for comparing graphics cards and not other system components such as CPU and memory. I have since found other applications to test overall system performance for CAD work (see next section). SPECviewperf is still useful for comparing different graphics cards, the following configurations were tested and the results are below:

- Xeon x5506 / 6GB DDR3-1333 / Quadro 600
- Xeon x5506 / 6GB DDR3-1333 / Quadro 2000
- Xeon x5506 / 6GB DDR3-1333 / GeForce GTS 450
- Intel i7-3930K LGA2011 / 8GB DDR3-1333 / Quadro 2000 (our system)

The GTS 450 card was used as a comparison as it is the closest specced desktop card to the Quadro 2000 (they have the same number of active CUDA cores and the same amount of memory). Results are also included for the Quadro 600 card: this is the cheapest Quadro Fermi card available, it has 96 CUDA cores and 1GB memory. Testing was done at **1280x960** resolution with **8x multi-sampling** enabled in the benchmark application.

As expected(shown in Figure 3.1), the type of graphics card used greatly affected the benchmark score. The Quadro 2000 card received benchmarking scores up to almost 20x better than the GeForce card. There is, however, an outlier in this situation: EnSight gained a performance increase when using the GeForce card; from this we can conclude that it is not optimized for use with the Quadro model card and relies on raw performance which the GeForce has more of.

	X/6GB Quadro 600	**X**/6GB Quadro 2000	**X**/6GB Geforce
CATIA	9.26	14.13	3.19
EnSight	8.18	12.72	15.90
LightWave	23.94	23.15	5.78
Maya	20.53	26.07	2.90
Pro/ENGINEER	5.11	5.16	0.89
SolidWorks	20.35	25.41	5.53
Siemens TCVis	8.98	11.80	0.61
Siemens NX	7.52	10.03	2.35

Figure 3.1: Results

As a new configuration is added: Intel i7-3930K LGA2011 / 8GB DDR3-1333 / Quadro 2000

So a new Testing was done again at 1280x1024 resolution with 8x multi-sampling by me (HaoYu), another testing was done at 1024x768 resolution with 8x multi-sampling by Celestino.

The following things need to be aware:

Different resolution and muli-sampling was using since the SPECviewperf 11 that we installed did not include 1280x968 resolution which Joseph has done previously. I decided to choose 1280x1024 resolution that it is the closest value.

Different configuration was used for this test, a better CPU core and 2GB memory higher than before.

From the previous test, Joseph has mentioned that he have found SPECviewperf is designed to isolate the graphics subsystem and is only reliable for comparing graphics cards and not other system components such as CPU and memory. If he is right, the new configuration of Intel i7-3930K LGA2011 / 8GB DDR3-1333 / Quadro 2000 compares with Xeon x5506 / 6GB DDR3-1333 / Quadro 2000 should come out with a similar score, else we should expect a higher score.

The result is shown below on Figure (3.2):

A interesting result has came out, The color marked in green represents a higher score than before, the color marked in purple means a lower score. A significant

	1280x1024 8x multi-sampling	1024x768 8x multi-sampling	1280x968 8x multi-sampling
CATIA	15.29	14.48	14.13
EnSight	19.09	20.21	12.72
LightWave	21.79	21.42	23.15
Maya	15.57	15.21	26.07
Pro/ENGINEER	4.58	4.53	5.16
SolidWorks	22.97	23.03	25.41
Siemens TCVis	8.05	7.7	11.80
Siemens NX	24.32	24.45	10.03
Configuration	Intel i7-3930K LGA2011 / 8GB DDR3-1333 / Quadro 2000	Intel i7-3930K LGA2011 / 8GB DDR3-1333 / Quadro 2000	Xeon x5506 / 6GB DDR3-1333 / Quadro 2000

Figure 3.2: SPECviewperf 11 results

decreasing performance score for "May", it dropped from 26.07 to 15.57, which it shouldn't be the case for a system with better hardware.

As we all known that SPECviewperf 11 is a 3rd party benchmarking software, this will lead to us some bias in some situations. Bias is always a concern with testing.

In this case, we have concluded our first hypothesis of bias that it might causes this result, since the new configuration was using a brand new mother board Asus P9X79 with the newest CPU chipset LGA 2011, these hardware are only released about few months ago, the SPECviewperf 11 may has not updated to the newest version that support our hardware.

3.2 Cadalyst c2012 Auto-CAD

The next phase of testing was done using the Cadalyst benchmark test for Auto-CAD 2012. This is not an independent application like SPECviewperf and is instead run from inside a fully installed version of Auto-CAD. This way it is mimicking actual CAD usage in a proper CAD application and should give us the most consistent and realistic benchmark we can hope for. A 30 day trial of AutoCAD 2012 was used as it is compatible with this test.

There have been some instabilities experienced when running this benchmark, this would be due to the fact that it is a 3rd party benchmarking test. The first group of tests was done using the base Quadro driver supported by Auto-CAD; the second group was done with the additional Auto-CAD performance driver by Nvidia installed and there was a large increase in 3D rendering performance. The following systems (Figure (3.3)) were tested:

	DXA Workstation 1366	Superhawk Plus 1366	Superhawk 1155	Superhawk 1155	Our system for this project
Motherboard	Supermicro X8DAi	Asus P6X58D-E	Asus P8P67 LE	Asus P8P67 LE	Asus P9X79
CPU	Xeon X5680, 6C/12T,12MB, 3.33GHz	i7-950, 4C/8T, 8MB, 3.06GHz	i3-2100, 2C/4T, 3MB, 3.10GHz	i7-2600K, 2C/4T, 3MB, 3.10GHz	i7-3930K, 6C/12T, 12MB, 3.20GHz
Memory	6GB DDR3-1333 Triple Channel	6GB DDR3-1333 Triple Channel	4GB DDR3-1333 Dual Channel	4GB DDR3-1333 Dual Channel	8GB DDR3-1333 Quad Channel
Graphics Card	Quadro 2000	Quadro 2000	Quadro 2000	Quadro 2000	Quadro 2000
HDD	Single SATA 3Gbps	Single SATA 3Gbps	Single SATA 3Gbps	Single SATA 3Gbps	Single SATA 6Gbps

Figure 3.3: Testing systems

The majority of the price different comes from the CPU and motherboard used. Do not use these to calculate the price increase (e.g. 300% cost increase) as the cost of parts such as the PSU, chassis and more expensive storage solutions aren't factored in and these would change the final cost ratio.

The tests were performed on newly installed operating systems using the system configuration recommended by the Cadalyst benchmark as well as installing a tweak to remove the info center from Auto-CAD (this sub-application appears to have been adding to the instability during testing). The testing was performed at 1024x768 instead of the suggested resolution of 1280x1024 because the higher resolution was also causing instability during testing.

I have made a new table for comparing previous tests done by Joseph with my current tests:

Testing was done using the Cadalyst benchmark test for Auto-CAD 2012 instead of Auto-CAD 2011.

	i7-950 / Base	x5680 / Base	i3-2100 / Base	i7-2600K / Base	i7-950 / Perf	i3-2100 / Perf	i7-2600K / Perf
3D	762	799	842	992	4133	4140	4408
2D	305	334	298	383	300	300	377
CPU	226	244	229	289	224	232	285
HDD	147	137	145	145	144	146	144
Total Score	**360**	**378**	**379**	**452**	**1201**	**1205**	**1304**
Total Time (mins)	18	17	17	14	14	13	11

Figure 3.4: Joseph's result

The first group of tests were done using the base Quadro driver supported by Auto-CAD where we called it a"Bas" testing; the second group called "Performance" testing was done with the additional Auto-CAD performance driver by Nvidia.

*Note: the result is tested by two different Driver versions, for instance: (8.17.12.9573) vs (8.17.12.6570)

	i7-3930K / Base	i7-3930K / Base	i7-3930K / Perf
3D	1149	983	It will be test when 10th of April
2D	409	417	
CPU	294	288	
HDD	175	159	
Total Score	**507**	**462**	
Total Time (Mins)	11	12	
Driver version	8.17.12.9573	8.17.12.6570	

Figure 3.5: New result

After the first group of tests was done by using the base Quadro driver, then lately we found out that testing with performance drive is unable if we continue using Auto-CAD 2012. Since Nvidia currently does not support performance drive for Auto-CAD 2012.

The link is shown here:

`http://www.nvidia.com/object/AutoCAD_PD_workstation.html`

We suspect that Nvidia has hidden the drivers for AutoCAD 2012 this time. When Joseph found the drivers for AutoCAD 2011, Joseph did note that the driver was hidden somewhere on the Nvidia website.

Similar scores where gained despite the varying system components, with the least expensive system performing fractionally better (due to the new architecture).

The performance driver results in a 3D performance increase of over 400%. Please note, however, that the performance driver has the following limitation with Auto CAD 2011:

- The "Advanced Material Effect" option introduced with AutoCAD 2011 is not currently supported by the NVIDIA AutoCAD Performance Driver. The setting controlling this graphics mode (in the Manual Performance Tuning dialog accessed by the GraphicsConfig command) is grayed-out when the Performance Driver is active.
- Procedural Materials and Maps introduced with AutoCAD 2011 will only display with the material's diffuse colour.
- Materials and Maps used in drawings coming from earlier AutoCAD versions are supported as they would have displayed in AutoCAD 2010.

3.3 PPBM5

The next phase of testing was done using the Premiere Pro Benchmark 5 (PPBM5) for Adobe Premiere Pro CS5.5. Since one of our goal is to test out Maximus functionality (Quadro combines Tesla) that whether it can provide a cost effective configuration, and previously, the two tests that done by both SPECviewperf 11 and AutoCAD 2012 did not have the feature to support Maximus technology [10].

Complete directions are included in the ZIP file downloaded from the website. Create a directory called PPBM on your Premiere project disk and download the PPBM5 file and unzip it in that directory. The zip file also includes a timing/information gathering script which writes the Output.txt file.

There are four tests in PPBM5:

1. Render the Time-line to create Preview files (Pressing Enter). This test may have to be done twice, once with Hardware MPE acceleration and once with Software MPE only.
2. Export the Time-line with Abode Media Encoder to a MPEG2 DVD file.
3. Export the Time-line with Adobe Media Encoder to a H.264 file.
4. Export the Time-line with Adobe Media Encoder to a Microsoft DV AVI file

DISK I/O test:

The overriding factor is disk speed here. The test uses many small reads and a large sequential write (nearly 13 GB). Number of cores makes no real difference (it is not well multi-threaded), but clock speed does.

MPEG2 DVD test:

The two overriding factors here are amount of memory and number of cores. More is better here. Additionally the location and speed of the page-file can be important especially if you have a small amount of RAM.

H.264 test:

Here the speed of CPU/RAM communication is king. Number of cores, clock speed and the amount of CPU cache are very important. Dual processor systems are hampered by the 2 chip communication.

CPU / GPU Test Result:

This is almost solely based on the video card and whether hardware or software MPE is used.

MPE Gain:

This shows how much faster hardware MPE rendering is than software only rendering. The minimum score is of course 1, since if there is no hardware MPE available, there is no performance gain.

Total Time:

The Total Time is the sum total of the individual test scores, where each test score is calculated by seconds, so the lower the score the better.

Below are the results for benchmarking PPBM5 by using two different configurations:

1. Intel i7-3930K LGA2011 / 8GB DDR3-1333 / Quadro 2000 only
2. Intel i7-3930K LGA2011 / 8GB DDR3-1333 / Quadro 2000 + Tesla C2075 (Maximus technology)

The two tables shown below are the time taken for running the test; it is the score for each test and the time is measured in sec.

Quadro 2000 only	Format	Preset	Encoding time
H.264 test	H.264 Blu-ray	Custom	1.13 sec
Disk test	Microsoft AVI	PAL DV	5.44 sec
MPEG2-DVD test	MPEG2-DVD	Custom	3.46 sec

Quadro 2000 + Tesla C2075	Format	Preset	Encoding time
H.264 test	H.264 Blu-ray	Custom	1.03 sec
Disk test	Microsoft AVI	PAL DV	5.26 sec
MPEG2-DVD test	MPEG2-DVD	Custom	2.26 sec

Figure 3.6: Upper table is tested with Quadro 2000; the bottom one is Q+T)

Remember these are the scores for each test from Output.txt file (Figure 3.7): and each test have to be done twice, once with Hardware MPE acceleration(MPE-On) and once with Software MPE only(MPE-Off). Hardware acceleration is useful with rendering, previewing, and on certain parts of the export process, i.e. scaling, frame rate adjustments, blurring and blending, but not with encoding.

Our result for Quadro + Tesla(MPE-ON): Total scores = 359+140+62+8 = 569 it is still so high because of the HDD(Disk IO score) that we used was an old one from TSD. Our MPEG2-DVD(we got 140) is also quite high which I do not quite understand at the beginning. But after i did some research on the website, I found

Mercury Playback Engine GPU Acceleration	Quadro 2000	Quadro 2000 + Tesla C2075	Mercury Playback Engine Software Only	Quadro 2000	Quadro 2000 + Tesla C2075
Disk I/O	345	359	Disk I/O	345	359
MPEG2-DVD	226	140	MPEG2-DVD	226	140
H.264	73	62	H.264	73	62
MPE-On	10	8	MPE-off	99	96

Figure 3.7: Mercury On/Off

out that the score is affected by the amount of memory. See the diagram I attached below:

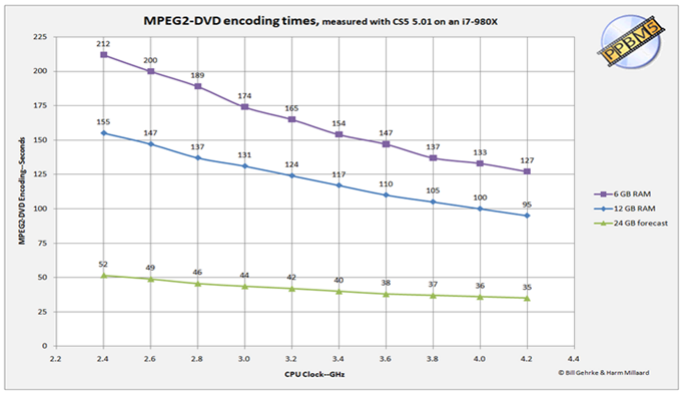

Figure 3.8: Encoding times with diff range of RAM

Resource from: `http://ppbm5.com/DB-PPBM5-1.php`

Rank	Motherboard	CPU Model	Cores	# RAM	Version	Video card	Total	Disk I/O	MPEG2 DVD	H.264 BR	MPE On	MPE Off	OS Disk	# HDD's
1	SR-2	Xeon X5680	12	48	5.03	GTX 580	131	67	21	37	6	64	SSD	6
2	P9-X79PRO	i7-3930K	6	32	5.04	GTX 570	133	71	22	35	5	56	SSD	2
3	GA-X58A-UD7 rev2	i7-970	6	24	5.04	GTX 480	134	59	18	53	4	67	SSD	16
4	GA-UD5-X79	i7-3930K	6	32	5.04	GTX 590	134	66	23	39	6	66	SSD	5
5	P6T WS Supercomputer	i7-980X	6	24	5.03	GTX 580	139	58	21	55	5	65	10000	9
6	X79-UD5	i7-3930K	6	32	5.03	GTX 570	142	75	24	38	5	60	SSD	5
7	Saberthooth X79	i7-3930K	6	32	5.5	GTX 570	144	45	58	36	4	49	10000	3
8	P8P67 EVO	i7-2600K	4	16	5.03	GTX 470	145	56	28	57	4	66	SSD	4
9	X58 Xtremen	i7-950	4	24	5.03	GTX 460	146	75	24	42	5	79	7200	4
10	X79 UD5	i7-3930K	6	32	5.5	GTX 580	146	48	57	37	4	55	SSD	4
11	X79	i7-3960X	6	32	5.5	GTX 580	146	47	58	36	5	49	7200	3
12	P8P67 EVO	i7-2600K	4	16	5.03	GTX 460	148	56	29	58	5	65	SSD	6
13	GA-X58A UDR3	i7-980X	6	24	5.5	GTX 680	149	51	55	39	4	61	SSD	5
14	SR-2	Xeon X5650	12	24	5.03	GTX 470	150	69	23	52	6	74	7200	6
15	X79 Extreme9	i7-3930K	6	64	5.5	GTX 480	150	46	62	38	4	52	SSD	16
16	SR-2	Xeon E5660	12	48	5.5	GTX 580	152	52	58	37	5	56	SSD	8
17	P6T WS Supercomputer	i7-980X	6	24	5.03	GTX 580	153	67	22	59	5	71	15000	7
18	Z67 Exreme4 Gen3	i7-2600K	4	16	5.03	GTX 570	153	65	27	56	5	74	SSD	5
19	P6X58D-E (bios 602)	i7-980X	6	24	5.03	GTX 480	154	68	25	56	5	71	SSD	7
20	Rampage III Extreme	i7-970	6	24	5.03	GTX 570	155	64	27	59	5	69	10000	9

Figure 3.9: Top 20 scores on 30/06/2012

As we can see from the top 20 results, they all had a very high amount of RAM, especially for the ranking No.1, it has number of 48 RAM for the configuration. For our set up, we had only 8GB RAM.

And for the GPU card GTX580, it has 512 CUDA cores comparing with our Quadro 2000 that contains only 192 CUDA cores. If we look further for Maximus, Quadro 2000 combines with Tesla C2075 with 448 CUDA cores. We should expect a high improvement result by using Maximus functionality. But in fact, our result comes out with not much improve overall even though our H.264 score and MPE-On score are actually very close to the top 20 result.

So after we have compared our results with the Top 20 scores, we decided to increase our 8GB RAM to 16GB RAM and test again by using exactly the same configuration. Hopefully, we should expect a better result related to MPEG2-DVD score.

**Important Note:

This test was still using the old/same Hard disk drive Model: Seagate Barracuda 7200.7 ST3120827AS 120GB 7200 RPM 8MB Cache SATA 1.5Gb/s 3.5" Hard Drive

Why this need to be pay a special attention? I will talk about this later after we had another tests on a New Hard disk drive Model.

Now, let's check the table, it shows the fact that no matter how we changed the

Mercury Playback Engine GPU Acceleration	Quadro 2000	Quadro 2000 + Tesla C2075	Quadro 2000	Mercury Playback Engine Software Only	Quadro 2000	Quadro 2000 + Tesla C2075	Quadro 2000
Disk I/O	345	359	370	Disk I/O	345	359	370
MPEG2-DVD	226	140	234	MPEG2-DVD	226	140	234
H.264	73	62	75	H.264	73	62	75
MPE-On	10	8	9	MPE-off	99	96	96
Memory RAM used	8GB	8GB	16GB	Memory RAM used	8GB	8GB	16GB

Figure 3.10: RAM increased from 8GB to 16GB

Memory RAM does not give any improvement to our scores. Hence, we did not continues on testing with Quadro2000 + TeslaC2075, it is meaningless for doing it since RAM does not apply better result. So we had one conclusion by now:

- Even though Adobe PPBM5 explained/proved that higher Memory RAM will increase performance, it does not work for our configuration with Quadro2000 graphics card.

We don't think we can expect much from them given that they are third-party solutions. After we contacted with Joseph, he then gave us some suggestions:

- one possible solution to the issue would be to bypass the benchmarks and test the performance manually. Possibly you could find a few common tasks with the application you wish to benchmark (rendering a model in AutoCAD for example) and them manually running the task and timing how long it takes to finish. This way you get a 'real world' performance result (i.e. instead of saying "it received a benchmark score of 398" which is abstract, you could say "it rendered the model 20 seconds faster" or "it took half as long to encode a 20 minute video". The problem with that I guess would be that you would wouldn't be able to compare it with scores online and would have to run the tests on older system setups to compare performance differences.

Next step we are trying to improve the performance for Disk I/O, as i mentioned previously about OLD hard disk, now we changed to new Model:

- Western Digital Caviar Blue WD5000AAKX 500GB 7200 RPM 16MB Cache SATA 6.0Gb/s 3.5" Internal Hard Drive

Mercury Playback Engine GPU Acceleration	Quadro 2000	Quadro 2000 + Tesla C2075	Quadro 2000	Quadro 2000	Mercury Playback Engine Software Only	Quadro 2000	Quadro 2000 + Tesla C2075	Quadro 2000	Quadro 2000
Disk I/O	345	359	133	370	Disk I/O	345	359	133	370
MPEG2-DVD	226	140	217	234	MPEG2-DVD	226	140	217	234
H.264	73	62	76	75	H.264	73	62	76	75
MPE-On	10	8	8	9	MPE-off	99	96	80	96
Memory RAM used	8GB	8GB	8GB	16GB	Memory RAM used	8GB	8GB	8GB	16GB
Hard Disk	Old	Old	New	Old	Hard Disk	Old	Old	New	Old

Figure 3.11: New Hard-disk's scores

This time has successfully improved the score of Disk I/O as expected, reducing the time from 340-360 sec to 133 sec, which leads to our second conclusion:

- Disk I/O is working perfectly fine with our configuration, better the Hard disk, better the result.

The next phase of testing was done using Tesla C2075 alone, Quadro 600 alone and Maximus(TeslaC2075 + Quadro 600) with different range of RAM.

*Note: Both Tesla C2075 and Quadro 600 do not support Mercury Playback engine.

However, Maximus enables MPE on (if we combined Tesla with Quadro).

Explanation of MPE from Adobe website:

'Mercury Playback Engine' is a name for a large number of performance improvements in any version above Premiere Pro CS5. Those improvements include the following:

- 64-bit application
- Multi-threaded application
- Processing of some thins using CUDA

Everyone who has Premiere Pro CS5 has the first two of these. Only the third one depends on having a specific graphics card.

Confusingly—because of one of our own early tests that was just plain unclear—a lot of people think that 'Mercury' just refers to CUDA processing. This is wrong. To see that this was not the original intent, you need look no further than the project settings UI strings 'Mercury Playback Engine GPU Acceleration' and 'Mercury Playback Engine Software Only, which would make no sense if 'Mercury' meant "hardware" (i.e., CUDA).

The official and up-to-date list of the cards that provide the CUDA processing features is here:

`http://www.adobe.com/products/premiere/systemreqs/`

As long as MPE is on, the results always come out around 10 sec, which it is an acceptable score.

With the new Hard disk, our 1st conclusion still apply to the Quadro 2000 card, RAM does not improve at all, in real, it should not be the case, because Memory RAM does effect both Tesla C2075 and Quadro 600 as we can see the following table:

*Note: The following table is MPE-OFF, since Tesla C2075 and Quadro 600 do not respond to MPE.

Tesla C2075 with 16GB RAM reduced the time from 105sec to 54 sec. Same effect apply to Quadro 600 even though we did not test Quadro 600 with 8GB RAM, but the data shows a fact that 16GB RAM working perfectly fine with Quadro 600, and leads to a result better than Tesla C2075 $(41sec < 54sec)$.

Mercury Playback Engine GPU Acceleration	Quadro 2000	Quadro 2000 + Tesla C2075	Quadro 2000	Quadro 2000	Quadro 2000	Quadro 600 + Tesla C2075
Disk I/O	345	359	133	370	138	120
MPEG2-DVD	226	140	217	234	217	126
H.264	73	62	76	75	73	62
MPE-On	10	8	8	9	9	7
Memory RAM used	8GB	8GB	8GB	16GB	16GB	16GB
Hard Disk	Old	Old	New	Old	New	New

Figure 3.12: Q600 is added to the test configuration

Mercury Playback Engine Software Only	Quadro 2000	Tesla C2075	Tesla C2075	Quadro 600	Quadro 600 + Tesla C2075
Disk I/O	133	130	124	126	120
MPEG2-DVD	217	105	54	41	126
H.264	76	98	97	96	62
MPE-off	80	78	77	81	80
Memory RAM used	8GB	8GB	16GB	16GB	16GB
Hard Disk	New	New	New	New	New

Figure 3.13: MPE-Off

This draws us to a deep thinking of why will Quadro 2000 perform lower score regarding to Quadro 600 if we do not consider the special feature MPE-On/Off. This could be the reasons where graphics card's driver version or 3rd party solution. Recently, Adobe official announces Adobe Premiere Pro CS6 is released, this could be more reliable and accuracy for testing Quadro2000 and Maximus functionality.

A Quadro 6000 and Tesla C2075 are not identical but they are very similar and you can expect similar performance. There are a few reasons you might want to use a Maximus configuration for Premiere Pro rather than a single Quadro 6000:

1. Having both a Quadro and Tesla GPU in the system means when the Tesla is cranking full-out on Mercury Playback Engine and the Quadro is unaffected, so you can, say, open After Effects or other application that may take advantage of the Quadro, and system performance on that app will be better than if it was competing for resources with MPE on a single GPU.

2. In the future, we expect many users will want to run an animation application (using the Quadro) and a simulation application (on the Tesla) at the same time to provide animators with a level of interactivity they don't have without Maximus technology. Example video is here. `http://youtu.be/_LagqqsVO28`

3. It costs less. A typical Maximus configuration has a mid-range Quadro (e.g. a Quadro 2000) and a Tesla C2075, which in that instance costs hundreds of dollars less than a single Quadro 6000 and offers similar performance plus the workflow advantage listed above.Of course, some users may want to run a Quadro 6000 and a Tesla C2075 and get maximum performance, but others can actually get the best MPE acceleration for less money with Maximus technology. (Resource from Adobe forums)

Maximus is a technology that essentially marries a graphics-intensive Quadro card with a Tesla card, which is all compute, inside a workstation to meet that challenge. Theres also a software stack at the driver level that allocates the code within any application you're using to CUDA, routing it over to Tesla to handle the compute processing and the Quadro to handle graphics.

Sum up all the conclusions so far:

1. Even though Adobe PPBM5 explained/proved that higher Memory RAM will increase performance, it does not work for our configuration with Quadro2000 graphics card.

2. Disk I/O is working perfectly fine with our configuration, better the Hard disk, better the result.

3. Maximus feature: (Q2000 + C2075) vs (Q600 + C2075), result came out as Q600 + C2075 is better.

Most of the COTS tools we have benchmarked were not Open sources that we cannot investigate deeper into the concept for CUDA computing, thus there are some bias exist.

For next step, we would like to have our own experience of applying CUDA to speed up Semi-global matching (SGM), and the comparison could also be drawn with SGM on normal CPU.

Chapter 4

SGM

4.1 Review

Stereo vision has been an intensive research area in the last decades. The solution proposed was originally split into two main categories, local and global methods. Later a third category was introduced by Hirschmüller to separate the algorithms from the global methods [5]. The author has published a more detailed paper [6]. As this third category called Semi-global matching/method which are based on global optimizations, but the computational complexity is reduced to allow real-time implementations.

Local methods use a finite support region around each point/pixel to calculate the disparities. For selecting the appropriate disparity for each pixel, we applied the "winner takes it all" that the minimum of the calculated matching costs is searched. The main advantage of local method is the small computations. The main disadvantage is that only local information is used. As these methods are not able to handle featureless regions or repetitive patterns.

Global methods are able to improve the quality of the disparity map by enforcing several global constraints in the disparity selection step. Although results show a improvement in the disparity map quality, the disadvantage is still obviously as those methods are not suitable or desirable for real-time applications [8].

4.2 Semi-global Matching

Hirschmüller's original Semi Global Matching algorithm uses a mutual information based pixel matching cost and aggregates matching costs in multiple directions, for instance, 8 or 16 directions. SGM finds a minimum cost along a column of all possible disparity levels. In contrast to traditional implementation, the cost function was based on mutual information. But in our implementation, we prefer a simple sum of absolute difference (SAD) cost function. At last, we limited the direction along which we accumulate cost to only one at the beginning since it is easier when it comes to real CUDA implementation.

4.2.1 Correspondence Problem

Now let's look at the basic algorithm that we used for our correspondence problem. The approach is called Correlation-Based Methods. It compares a window of pixels in one image with a window of pixels in the other. We assumed rectified images in canonical configuration and epipolar lines aligned with scan lines.

See Cost function in Equation (4.1).

$$f(I_1(x,y), I_2(x-d,y)) = |I_1(x,y) - I_2(x-d,y)| \tag{4.1}$$

Now if a rectified pair of stereo images, called Left image I_1 and Right image I_2. The simplest form is to calculate the absolute difference, for each pixel, at coordinate $I_1(x,y)$, Right image will compares to Left image at a disparity range d which the pixel at $I_2(x-d,y)$.

See the example in Figure (4.1)

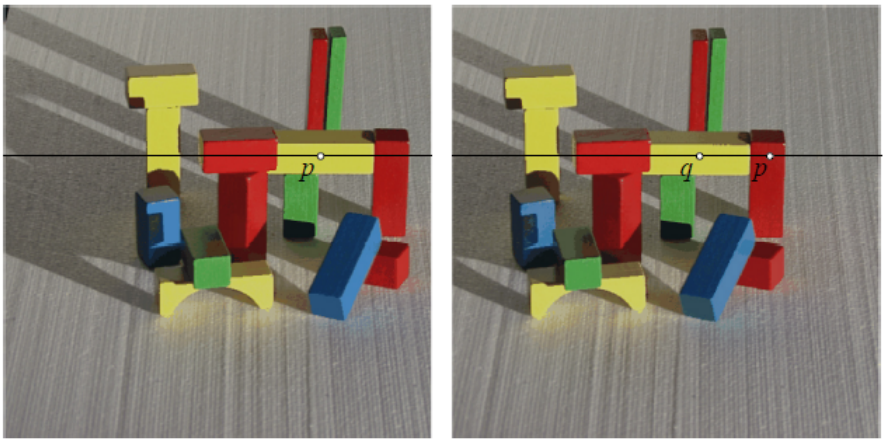

Figure 4.1: Rectified stereo image pair from [Klette, Schlüns, Koschan 1998]

Pixel p (in the left image) is on one side of a block, and the corresponding point q (in the right image) is what we want. For minimizing the sum in Equation (4.1), we may apply the following simple matching strategy for p = (x, y) in I_1:

Initialize $d = d_{min} = 0$ and $D_{min} = |I_1(x,y) - I_2(x,y)|$. While d<x increase d by one, calculate $D = |I_1(x,y) - I_2(x-d,y)|$, and, if $D<D_{min}$, replace d_{min} by d and D_{min} by D.

Hence, the corresponding point q with the same (or most similar) pixel value along the epipolar line (defined by p) will define the disparity: "the winner takes it all".

*Notes: This total independence of decisions at every pixel will create artefacts (noisy depth maps).

The simplest form may cause incorrect matching as a pixel may has lower value than the real correspondence pixel, therefore a window of pixels around (x,y) will be considered. e.g. a 3x3 window size around pixel p at (x,y).

Now we compared a region of 3x3 window size of pixels instead of one pixel, the Equation (4.1) has slightly changed to the following:

$$f(I_1(x,y), I_2(x-d,y)) = \sum_{\substack{-w \leq p \leq +w \\ -w \leq q \leq +w}} |I_1(x+p, y+q) - I_2(x-d+p, y+q)|$$

As w is the window size.

The success of correlation-based methods depends on whether the image window in one image exhibits a distinctive structure that occurs infrequently in the search region of the other image.

How to choose the size of the window, w?

- too small a window
  - may not capture enough image structure and
  - may be too noise sensitive
  - many false matches
- too large a window
  - makes matching less sensitive to noise but also
  - decrease precision (blurs disparity map)

For our implementation, we have used three different window sizes. such as 3x3, 5x5 and 7x7. The final algorithm is shown on Figured (4.2):

```
for each row, y
  for x = Δ to w
    c_min = ∞
    for d = 0 to Δ   // check each possible disparity
      c(d) = f ( I1(x, y), I2(x-d, y) )
      if  c(d) < c_min then
          d_best = d
          c_min = c(d)
    disp( x,y ) = d_best   // Save best d value
```

Figure 4.2: Correlation-Based Methods

4.2.2 Aggregation of Pixelwise Matching Costs

Of course, individual pixels do not contain enough information for unique matching. Therefore, global methods additionally use a smoothness constraint that penalizes discontinuities. This is typically formulated in a cost function:

$$E(D) = \sum_{p}(C(p, D_p) + \sum_{q \in N_p} P_1 T[|D_p - D_q| = 1] + \sum_{q \in N_p} P_2 T[|D_p - D_q| > 1]) \tag{4.2}$$

The first term sums the pixel-wise matching costs for all pixels. The second term penalizes small discontinuities with a penalty P_1, while the third term penalizes all larger discontinuities with a penalty P_2. D is the set of disparities, C is the cost function and N_p is the neighborhood of the point p in all directions. The function T returns the value of true or false.

The Semi global matching algorithm approximates the minimum of a 2D energy function (Equation 4.2) by minimizing multiple one-dimensional paths. The costs L_r are summed over paths in all direction r, The number of paths should be 8 or 16 for being sufficient. For each linear cost/path is aggregated over all pixels from each staring point p_0 (border pixel) to the opposite image border. And the computation is carried out recursively as shown in the following Equation:

$$\begin{aligned} L_r(p, d) = C(p, d) + min(&L_r(p - r, d), \\ &L_r(p - r, d - 1) + P_1, \\ &L_r(p - r, d + 1) + P_1, \\ &min_i L_r(p - r, i) + P_2) - min_i L_r(p - r, i) \end{aligned} \tag{4.3}$$

The pixelwise matching cost C will be our SAD cost for each pixel p at disparity d. The costs of paths L_r, for all (say, 16) directions r, are accumulated at a pixel p, for all disparities d with $0 \leq d \leq d_{max}$, and the disparity d with the lowest cost is finally selected.

The path costs for one pixel in one path direction are dependent only on the path costs of the predecessor pixel in each direction ($p - r$), not costs of the neighboring pixels perpendicular to the path.

For penalty P_1 and penalty P_2 in our implementation, we have chosen $P_1 = 20$ and $P_2 = 100$ respectively.

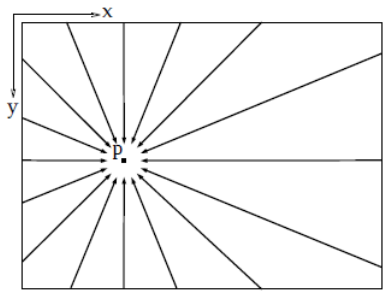

Figure 4.3: 16-Path SGM

*Note: The value of P_2 should higher than P_1.

For the next section, we will start looking at the CUDA implementation, the implementation on GPU is already published by many other people [3, 7, 15, 16]. The primary goal of each paper is to be very fast, it is the same as our goal. Some of them also want a secondary goal of being fairly flexible to allow changing of parameters for different stereo algorithms/methods

4.3 GPU Implementation of SGM

4.3.1 CUDA Programming Model

Stereo Imaging is a powerful yet seldom utilized technique for determining the distance to objects using a pair of camera spaced apart. A pair of cameras is spaced apart at roughly the same spacing as human eyes. This is the same visual system used by humans and most other animals. But as we know, the extremely high computational requirements of stereo vision limit the application to non-real time or to applications where high computational power is needed. Hence, a fine-grained, data-parallel threads provided by CUDA can achieve this.

In CUDA, it launched a "grid" of "blocks" of "threads" onto a GPU as shown in Figure (4.4)

At the top level of the hierarchy, a grid is organized as a two dimensional array of blocks. The number of blocks for each dimension is specified by the first parameter given at the kernel launch (e.g. methodName$\langle\langle\langle N, 1\rangle\rangle\rangle$ where N is number of Blocks), with gridDim.x specifying the number of blocks in the x dimension and gridDim.y the y dimension. The values of gridDim.x and gridDim.y should be between 1 and 65536. All threads in a block share the same blockId number. Each block

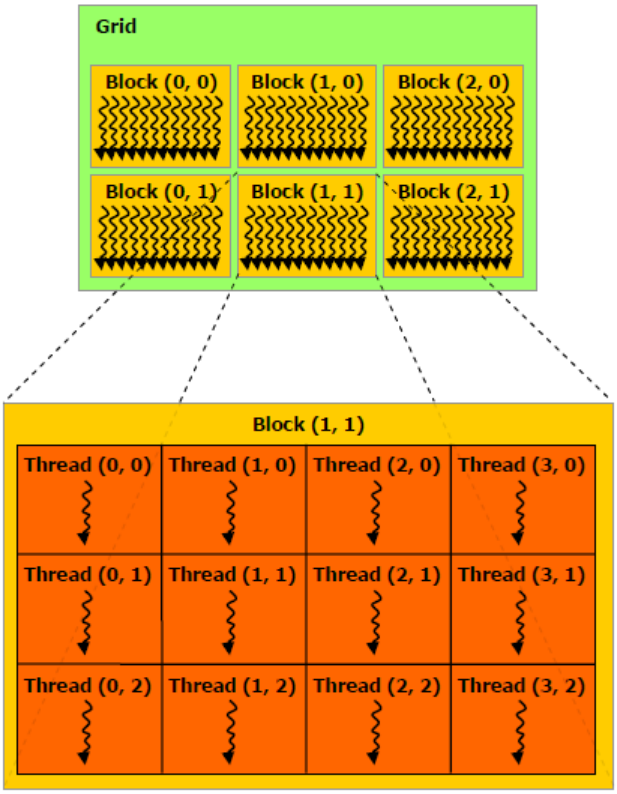

Figure 4.4: CUDA Model

in the array is labeled with (blockId.x, blockId.y). From the Figure (4.4), Block(1,1) has its blockId.x = 1 and blockId.y = 1. And this is the same principle applies to threadId.x and threadId.y.

Each thread that executes the kernel is given a unique thread ID that is accessible within the kernel through the built-in threadIdx variable. $threadIdx$ is a 3-component vector, so the threads can be identified using one-dimensional, two-dimensional, or three-dimensional thread index. This provides a better way to invoke computation across the elements in a domain such as a vector, matrix, or volume, and also good for our image processing.

Let us consider a simple example as shown on Figure (4.5):

We assumed that there is a total number of 16 elements need to be executed, and for our example, we had $blockDim = 4$ which it is 4 blocks. Inside each block, it

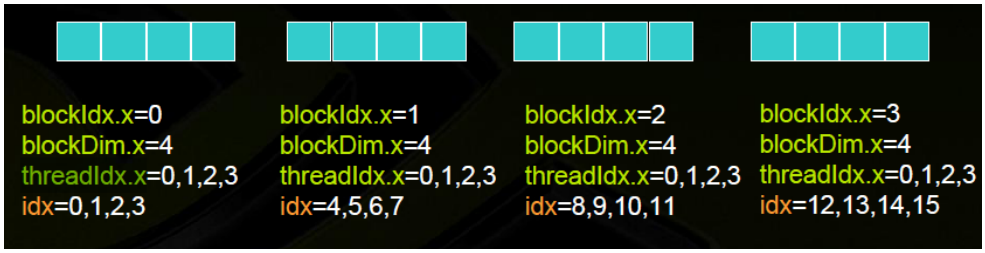

Figure 4.5: A simple example

contains 4 threads, and the index is $threadIdx.x$ = 0, 1, 2, 3. But in actual CUDA implementation, the global index should be "idx" as marked in orange color in the figure. And thus, a common pattern is used for the correct idx:

int idx = blockDim.x * blockId.x + threadIdx.x

*Note: $blockDim$ should be $\geq$ 32 in real code, it is just a example

There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 512 threads.

CUDA allows threads in the same block to coordinate their activities using a barrier synchronization function syncthreads(). When a kernel function calls syncthreads(), all threads in a block will be held at the calling location until everyone else in the block reaches the location. This ensures that all threads in a block have completed a phase of their execution of the kernel before they all move on to the next phase.

4.3.2 Heterogeneous Programming

In CUDA, CPU and GPU are separate devices with separate DRAMs, one of the main goal is to enable heterogeneous systems(i.e., CPU + GPU). So the idea is divided the code into two parts, **Serial code** executes in a host thread (i.e. CPU thread) and **Parallel kernel code** executes in many device threads across multiple processing elements (i.e. GPU threads). And this can be illustrated from Figure (4.6):

Kernels can only operate in device memory, so the CUDA library provides functions to allocate, de-allocate, and copy device memory, as well as transfer data between host memory and device memory.

For instance, Device memory management has the following library:

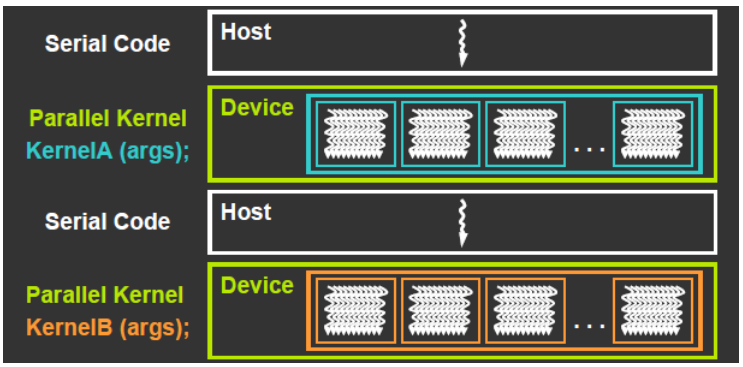

Figure 4.6: Heterogeneous Programming

$$cudaMalloc(), \quad cudaFree(), \quad cudaMemcpy()$$
$$cudaMemcpyHostToDevice, \quad cudaMemcpyDeviceToHost$$

Memory is typically allocated using $cudaMalloc()$ and freed using $cudaFree()$ and data transfer between host memory and device memory are typically done using $cudaMemcpy()$. as $cudaMemcpyHostToDevice$ and $cudaMemcpyDeviceToHost$ indicate the type of transaction.

For our implementation:

```
//Read Left image and Right image
char *readLeft = "left-image.pgm";
char *readRight = "right-image.pgm";
//Allocate host memory
unsigned char *left_image;
unsigned char *right_image;
read(readLeft, &imageWidth, &imageHeight, &grayValue, &left_image);
read(readRight, &imageWidth, &imageHeight, &grayValue, &right_image);
```

Firstly, we declared two char pointer variables called '$readLeft$' and '$readRight$' for reading the data from two images resources 'left-image.pgm' and 'right-image.pgm'; as the image is stored using .pgm format type. Hence, a further method called $read()$ is required to correctly allocate the memory into host; and it is stored in two new unsigned char pointer variables '$left\_image$' and '$right\_image$'.

Next, we need to allocate the same amount of memory onto GPU, in order to do this, we are required to know the size of the image which it is the imageWidth * imageHeight, as well as the amount of bytes for each pixel. The simple code creates a integer variable called '$size$' to store the size of memory as $sizeof(unsigned\,char) *$

$imageWidth * imageHeight$.

```
//Allocate device memory
int size = sizeof(unsigned char)*imageWidth*imageHeight;
CUDA_SAFE_CALL(cudaMalloc((void **)&d_left_image, size));
CUDA_SAFE_CALL(cudaMalloc((void **)&d_right_image, size));
```

Then, two new char pointer variables called 'd_left_image' and 'd_right_image' are declared for device. By using the CUDA library function $cudaMalloc$ to allocate the required memory space on the device (GPU).

When both host memory and device memory have correctly allocated, then it is time to copy the data from host to device by calling the function $cudaMemcpy()$. There are 4 parameters in the function method, first is the destination for device memory, second is the source for host memory, third is the total size, and last is the type of transaction which it is host to device in this example.

```
//Copy data from host to device
CUDA_SAFE_CALL(cudaMemcpy(d_left_image, left_image, size, cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy(d_right_image, right_image, size, cudaMemcpyHostToDevice));
```

Finally, the method of SGM can run parallel on GPU with multiply threads within multiply blocks. $cudaThreadSynchronize()$ ensures that all threads synchronized correctly. After the execution of SGM, remember to copy the data from device memory back to host memory by using $cudaMemcpy()$ again.

```
SGM<<<(N+THREADS_PER_BLOCK-1)/THREADS_PER_BLOCK, THREADS_PER_BLOCK>>>(d_left_image,
                                                                      d_right_image,
                                                                      imageWidth, imageHeight,
                                                                      maxDisparity,
                                                                      window_size,
                                                                      disparity, N);
cudaThreadSynchronize();
//Copy data from device to host
CUDA_SAFE_CALL(cudaMemcpy(out, disparity, size, cudaMemcpyDeviceToHost));
```

Inside the method of SGM, we used the common pattern formula to get the correct thread's ID:

Since we stored the all image pixels into one-dimensional array, the coordinate for x and y is shown above.

```
int index = threadIdx.x + blockIdx.x * blockDim.x;

int x = index%width;
int y = index/width;
```

The following is part of the code for SGM, the idea was for each pixel, which it $height \times width$, and for each disparity range from 0 to 50. We compute the SGM function from Equation (4.3); see Figure 4.7.

```
unsigned int minPrevious = 99999;
for(int y=0; y<height; y++) {
    for(int x=0; x<width; x++) {
        //get the min Cost of the previous pixel
        unsigned int min_Cost = minPrevious;
        //create a temporary int for updating the lowest cost
        unsigned int temp = 999999;
        for(int d=0; d<=maxDisparity; d++) {
            unsigned int cost = SAD_Cost[y][x][d];
            if(x == 0){
                L[y][x][d] = cost;
                minPrevious = min(cost, minPrevious);
            }
            else {
                unsigned int L0 = L[y][x-1][d];
                unsigned int L1 = L[y][x-1][d-1]+P1;
                unsigned int L2 = L[y][x-1][d+1]+P1;
                unsigned int L3 = min_Cost + P2;
                L[y][x][d] = cost + min(L0, min(L1, min(L2, L3))) - min_Cost;
                //update the current min Cost for next iteration
                temp = min(L[y][x][d], temp);
                minPrevious = temp;
            }
        }
    }
}
```

Figure 4.7: Semi-global method.

As the $cost$ is the first term of the function,the second term of the function is matched to L_0, L_1, L_2 and L_3 respectively, and the third term is min_Cost. We aggregated over the minimum 1D cost along one direction L_r from left image border to the end of right image border.

4.4 Results

4.4.1 Matching Accuracy

In this section, we compare and analyze computing performance and matching accuracy of Semi-global matching algorithms on a GPU card (Nvidia GeForce 8400 GS). And the result is compared with CPU running serially (Quad CPU Q6600 2.40 Ghz).

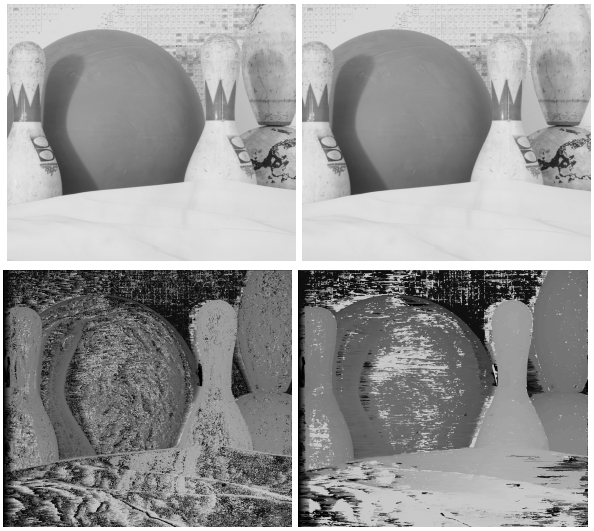

Figure 4.8: Results (bottom left is SAD and bottom right is SGM)

One of the testing result is shown on Figure (4.8), the original two image is taken by two cameras. As shown in the first row, left and right image are compared with each other using SAD (sum of absolute difference) first, then it uses the SAD cost and carries on the computation recursively with SGM algorithm (Equation 4.3 as described previously).

The image size for Figure (4.8) is width 1252 x height 1110. The disparity range that we tested from 0 to 50, and the size of SAD window is 3x3 for this example.

The bottom left result is SAD, the result is suffered from many “Peaks”, “Noise” and some area of untextured background. So the resulting disparity image still contains certain kinds of errors. there are generally ares of invalid values that required us to be fixed. If we increase the size of SAD window (e.g. 5x5, 7x7), the result for SAD can slightly improve the result; as the next result that shown on the next page.

Now it comes to the SGM, which it is the bottom right image, we can noticed that there is a significant improvement. But it still does not prove to be accurate enough, one of the reason is that number of path for L_r is limited to one path at this stage.

For different block size of windows, such as 3x3, 5x5 and 7x7, the results are in second row, third row and fourth row respectively. The image size is width 450 x height 375. Same disparity range as 50.

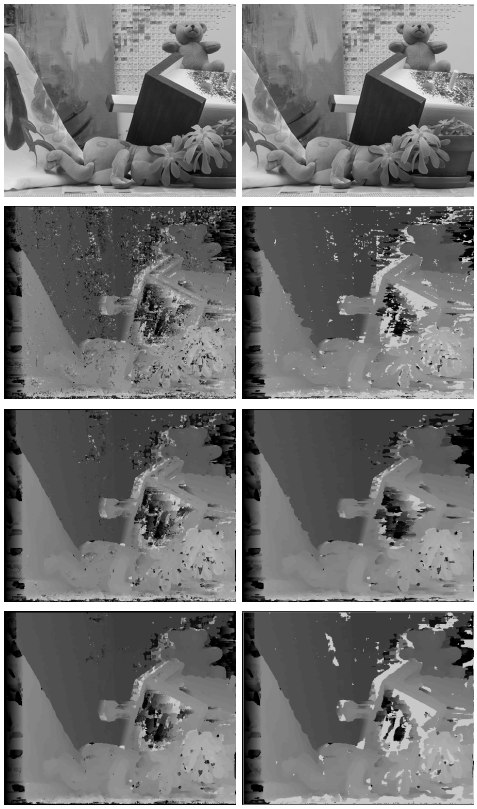

Figure 4.9: Results (3x3 and 5x5 SAD, 3x3 and 5x5 SGM)

In this example, the best result is given by 5x5 window size, for the last row of 7x7 block size, image processing with SAD perform better result than 5x5 block size SAD, but this does not apply to SGM, many outliers ("Peaks" and "Nosie") appeared in the image. Therefore, further disparity refinement for removing 'Peaks' is needed. This will be discuss more in the Conclusion and Future work section.

4.4.2 Performance Evaluation

The execution time is scaled linear with the size of window as well as the disparity range, and with the number of pixels in an image. In both tables (Figure 4.10 and Figure 4.11), we used the same image size of width 450 x height 375, but testing is done with different SAD window sizes and disparity ranges. The configuration of this testing system is not in a good condition, since the CPU is Quad Q6600 2.40Ghz and the GPU card is Nvidia GeForece 8400 GS with only 8 CUDA cores.

The reason is because the implementation of the SGM is still under development, it is considered as a testing stage which I did not think it is ready enough to test with powerful GPU cards in Chapter 3 (e.g. Quadro 2000 and Tesla C2075), those GPUs can certainly tune to produce higher performance. But without performing their maximum capability. If it is not running with its best capability, then it is meaningless to do so.

Even though the GPU card we currently using is poor condition, the execution time still gives a 2x faster than normal CPU. As mentioned before, it scaled linear with increase of size, it is clearly shown from the graph which blue color indicates CPU and red color represents Nvidia's GPU.

Image size: 450 x 375 Disparity = 50	3 x 3 Window Size	5 x 5 Window Size	7 x 7 Window Size
Quad CPU Q6600 2.40Ghz	2.0 sec	5.3 sec	9.9 sec
NVIDIA GeForce 8400 GS	1.0 sec	2.4 sec	4.2 sec

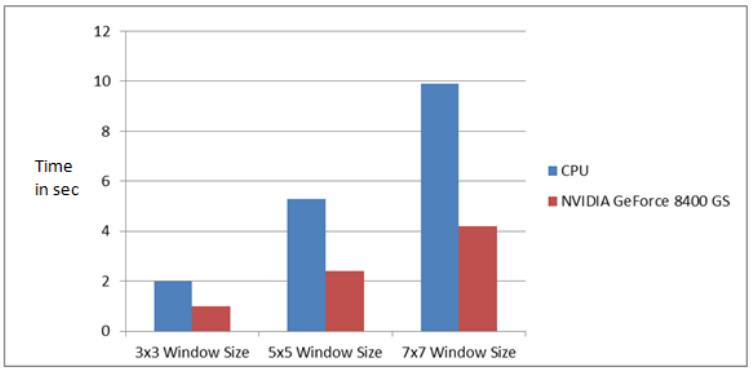

Figure 4.10: Running time for different block sizes

Image size: 450 x 375 3x3 Window Size	Disparity = 50	Disparity = 100	Disparity = 150
Quad CPU Q6600 2.40Ghz	2 sec	3.7 sec	5.1 sec
NVIDIA GeForce 8400 GS	1.1 sec	1.7 sec	2.5 sec

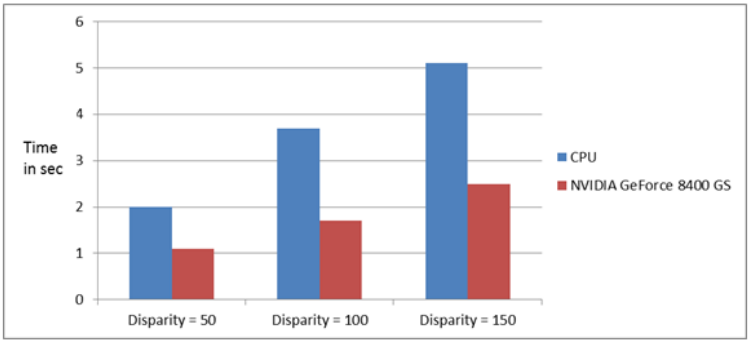

Figure 4.11: Running time for different disparity ranges

*Note: More pair of images data should be tested along with different combinations of SAD window size and disparity range.

Chapter 5

Knowledge Gained

5.1 Achievements

5.1.1 Project Goal

Most of the goals for this project have been accomplished. The project involved many technologies and algorithms, A lot of research is done in order to gain a fully understanding for each concept. And it is great to know a range of new ideas and concepts. One of the goals for myself is to find out certain areas where I was lacking, and then develops the skills that will be beneficial for me in the future.

5.1.2 CUDA Language

From this project, I have learned C++ programming language with CUDA extensions to express parallelism, data locality and thread cooperations. Using CUDA, it allows the latest Nvidia GPUs become accessible for computation like CPUs. We also found out that CUDA is compatible with most standard operating system, it is suitable to use for a range of different environments. And we understood that it is possible to solve general purpose problems on GPUs, but unlike CPUs, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing s single thread very quickly. However, we also known that CUDA still has many limitations, such as memory allocation between host an device may incur a performance hit due to latency. For a better result, it is necessary to understand the shortages/limitations of CUDA. Learning CUDA language is a great experience for me as it will be very useful for me in the future.

5.1.3 Presentation and Report Skills

This whole year-long project includes three seminars(Introduction seminar, End of semester one seminar and Final seminar) and two reports (End of semester one report and Final report). Working on this project definitely improved my presentation skills also report writing skills. After each seminar or report, feedbacks are given by

Dr. S. Manoharan and Prof. Reinhard Klette, as well as TN Chan, those feedbacks are valuable to me as it indicates my weakness and shortcomings.

5.1.4 Organisation and Time Management Skills

During the semester, along with other courses that have assignment, seminars and reports. It meant that I really need to manage myself in a scheduled time-frame. Weekly or Monthly meeting with the industrial mentor TN Chan and academic mentor Prof. Reinhard Klette made the project up to date as well as planning the next week/month's work. It does improve a lot for my time management skills.

5.1.5 Working Experience

As an great opportunity to work in the industry/company like Compucon New Zealand (CNZ). I have met many smart people, especially engineering specialist in the area of hardware, they guided me well for this project. I was involved as part of the member of CNZ, this helps me to develop experiences for working with others. Taking responsibility for my work is a very important thing. In the future, I might involve in the similar situation as working in a company. Also the ideas for commercial objectives, it enables me to understand the goals/achievements that industry is aiming for.

5.1.6 Planning Solutions

The progress is slow at the beginning of the second semester due to the implementation of SGM on CPU, it hold me back again when it comes to real CUDA. The problems throughout the project included both technical and non-technical issues. So it is a important thing for planning a solution. It may good to separate or break the project into smaller tasks, since it made us more easier to find out the problems, and if one solution failed, do what was needed or try another solution for that part without affecting other parts of the task.

Chapter 6

Conclusion and Future work

6.1 Summary

Modern graphics processing units (GPU) are usable as high-speed co-processors for general purpose computational tasks. And CUDA is the tool for this project. The first half of the project has successfully benchmarked using many well-known software such as SPECviewperf 11, Cadalyst c2012 Auto-CAD and PPBM5, etc. While testing those high-end GPU cards, we can discover the limitation and behavior of them. This helps us when it comes to the second phase of the project, which it is Semi-global matching on CUDA.

Fast stereo matching is necessary for many practical real-time applications. Solutions may be based on local approaches that perform correlation of rectangular windows (our SAD costs), followed by 'the winner takes it all' disparity selection. The advantage of correlation methods is fast, but they are known to blur object boundaries and remove small structures [8]. Global methods are more accurate than local methods, but their computational complexity is typically much higher than the computational complexity of local methods, and therefore, it makes them unsuitable for real-time application.

An exception is the Semi global matching method [6], which combines several one dimensional optimization from all directions. The computational complexity is $O(width \times height \times disparityRange)$ which results in efficient computations and enough accuracy to compare with global methods. Furthermore, the SGM algorithm has a regular structure that allows a GPU implementation for this project.

Compute Unified Device Architecture (CUDA) initiative by Nvidia Corporation. This architecture offers flexible programming with minimal extensions to the common C programming language, as this can be proved by many related papers or articles [3, 7, 15, 1]. And we have also implemented it, even though our approach did not give the best performance.

6.2 Conclusion

In this project, we have presented a Semi-global matching algorithm which can be efficiently mapped to GPU hardware. It means that it is possible to implement SGM including pixelwise matching with SAD on CUDA. We analysed and compared the results both from CPU serial and GPU parallel. Whilst this analysis provides a good guide on performance of CUDA.

For SGM, we reduced the computational complexity of original SGM for the pixelwise mutual information to SAD as well as limit the direction to only one path for testing. The speed of SGM has benefited from CUDA while achieving an acceptable depth map result. For the term of 'acceptable result', it indicated that our solution currently still lack of quality, further developments are required to maximum quality and performance.

Overall, we concluded that CUDA offers a flexibility and higher abstraction from the graphics processing unit (GPU) hardware. It is important to note that this is an indication only, since for the timing of execution, we are only timing the memory transfer and computation for SGM, all other factors did not considered. In practice, there exist certain factors such as costs and power usage. And we have also deliberately not included any pre- or post-processing and compared the basic SGM algorithms only.

6.3 Future Work

6.3.1 Disparity Refinement

The resulting disparity image can still contains certain kinds of errors. For instance, some outliers which are completely wrong disparities due to low texture, reflections, noise, and so forth. They usually show up as small patches of disparity that is very different to the surrounding disparities, that is, 'Peaks'. Therefore, these are the area of invalid values that need to be recovered.

The matching cost calculation step represents the most important part of our implementation. Both the matching selection and sub-pixel interpolation is based on the information generated in this step. So if we can use a matching metrics or by performing a filtering of the input image to enhance the features in order to help differentiate between the different regions. This will provide a good disparity refinement that immunity to noise.

The selected matching metric is Census transform [18], this is suggested by Prof. Reinhard Klette and Ralf Haeusler. Evaluation papers have shown that it represents one of the best metrics for matching correlation in difficult conditions [9]. In the future, this should be applying into our SGM.

6.3.2 CUDA Optimizations

As we know, there are many different approaches to implement on the GPU. The strategy employed in our project is just another follow-up on existing idea and re-implementing SGM on CUDA to see any potential benefits can be made from it. So it certainly not guaranteed to be a fast speed approach. There are many guidelines from Nvidia website or forum for optimizing CUDA.

For implementing the pipeline using the CUDA interface we had to take into account parallelism possibilities, two levels of parallelism is presented in the CUDA interface, a coarse level without inter-thread communication and a fine level where threads can share data. Our implementation cannot use the level of shared memory yet. To improve performance we definitely require this. One example of stereo Imaging with CUDA has successfully implemented using shared memory, some critical guidelines are shown below [17]:

- Avoid obscenely redundant computation – Many computations preformed for one pixel can also be used by neighbors
- Keep global memory Coalesced

- Minimize global memory reads/writes
- Exploit texture hardware (especially for sub-pixel disparity computations)
- Create enough threads and thread blocks to keep the processors busy.

Another issue for our implementation is where to store the image, we stored in a single one-dimensional array. In comparison with the paper, it used texturing from a CUDA array which provides the following advantages:

- No coalescence requirements and fast, cached access
- Boundary clamping
- Bilinear interpolation (for sub-pixel disparity measurements)
- Changing image source data type is trivial and does not require re-optimization of the algorithm

The example in the paper highlights the power of shared memory and data parallel programming. It allows a significant amount of data sharing between threads and thus a dramatic improvement in efficiency. Rather than each thread computing all the individual SAD values within a kernel the threads cooperate to compute pieces of the SAD for each other.

Since the approach in the paper is highly optimized, it will be a great guideline for us to follow in the future.

6.3.3 Optimize SGM on Multiple Paths

Literature notes the SGM algorithm requires a minimum of 8 directions to be used for optimization, while 16 would be recommended for maximum quality. For our implementation, it is only one path from left border to the other end (right border). So the number of errors is quite significant high, but the execution time is small due to only one path. In order to reduce the computational cost of the algorithm while performing a good quality result, some papers [3, 4] used 4 directions.

The author evaluated and noted that the best option is to choose 8 directions for good quality, but instead of choosing 8 directions, they used 4 directions. The reason is because their results shown an increase of the number of errors to 14% from 12.8%, but the execution time is significant dropped, for a real-time systems the significant gain in execution time compensates this small increase in the number of errors. So in the future, we should continuous our SGM with more paths such as 4 directions, or more to discover if it matches the findings of the paper as mentioned.

6.3.4 More Stereo Vision Algorithms

For this project, we simply implemented SGM only. It will be a great idea for analysing more stereo vision algorithms on CUDA. In this paper [15], the authors briefly tested out five different stereo algorithms:

1. Symmetric Dynamic Programming Stereo (SDPS)
2. Semi global matching (SGM)
3. Blocking matching (BM)
4. Belief propagation (BP, OpenCv)
5. Semi global block matching (SGBM, OpenCv)

In the future, if it is possible, our research area should not only about SGM, we should consider all other algorithms.

Bibliography

[1] Choi, Y., " CUDA implementation of belief propagation for stereo vision ", IEEE Conf. Intelligent Transportation Systems, pp. 1402 - 1407

[2] Guan, S., Klette, R., "Belief-propagation on edge images for stereo analysis of image sequences", In Proc. Robot Vision, LNCS, pages 291–302, Springer, 2008

[3] Haller, I.; Nedevschi, S., "GPU optimization of the SGM stereo algorithm", 2010 IEEEConf. Intelligent Computer Communication and Processing (ICCP), pp. 197 - 202

[4] Hermann, S., Klette, R., Destefanis, E., "Inclusion of a Second-Order Prior into Semi-global matching", 3rd Pacific Rim Symposium on Advances in Image and Video Technology, Lecture Note, vol. 5414, pp. 633-644, 2009.

[5] Hirschmüller, H., Scharstein, D., "Evaluation of cost functions for stereo matching", IEEE Conf. Computer Vision Pattern Recognition (CVPR), pp. 1-8, 2007.

[6] Hirschmüller, H., "Stereo processing by semi-global matching and mutual information", IEEE Trans. Pattern Analysis Machine Intelligence: 328–341, 2008.

[7] Ernst, I., Hirschmüller, H., "Mutual Information based Semi-Global stereo matching on the GPU", 4th ISVC08, 1-3 December 2008

[8] Hirschmüller, H., Innocent, P. R., Garibaldi, J., "Real-time correlation-based stereo vision with reduced border errors", Int. J. Computer Vision, 47: 229–246, 2002.

[9] Hirschmüller, H., Scharstein, D., "Evaluation of Stereo Matching Costs on Images with Radiometric Difference", IEEE Tran. Pattern Analysis Machine Intelligence: 1582-1599, 2009.

[10] Maximus Technology, *http://www.nvidia.com/object/maximus.html*.

[11] Middlebury Stereo Website: *http://vision.middlebury.edu/stereo/*. Last visit: 28 October 2012.

[12] NVIDIA: Fermi Architecture for High-Performance Computing, *http://www.nvidia.com/object/fermi-architecture.html*.

[13] NVIDIA's Next Generation CUDA Compute Architecture, `http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf`.

[14] NVIDIA Tesla C2074 companion processor calculate results exponentially faster, *http://www.nvidia.com/docs/IO/43395/NV-DS-Tesla-C2075.pdf*.

[15] Kalarot, R., Morris, J., Berry, D., Dunning, J., "Analysis of Real-Time Stereo Vision Algorithms On GPU", Control Vision Ltd, Auckland, NZ

[16] Rosenberg, I.D., Davidson, P.L., Muller, C.M.R., Han, J.Y., "Real-time stereo vision using semi-global matching on programmable graphics hardware", Int. Conf. Computer Vision, SIGGRAPH. (2006)

[17] Stam, J., "Stereo Imaging with CUDA", Nvidia Resource website *http://openvidia.sourceforge.net/index.php/OpenVIDIA*.

[18] Zabih, R., Woodfill, J., "Non-parametric local transforms for computing visual correspondence.", Computer Vision – ECCV'94 (1994): 151-158.