Stereo Matching using Multi-Resolution Images on CUDA

Sudhakar Sah
Center for Engr Sciences & Technology-CREST
KPIT Cummins Infosystems Ltd.
Pune, India

Nikhil Jotwani
Center for Engr Sciences & Technology-CREST
KPIT Cummins Infosystems Ltd.
Pune, India

ABSTRACT

Stereo matching technique is used to estimate the depth of objects in an image acquired from real time scenes. The basic algorithm is not very complex but is computationally exhaustive and hinders its usage for real time applications. However, this algorithm is highly data parallel and it highly suitable for execution on GPGPU (General-purpose graphical processing units). In this paper, we are proposing the parallel implementation of the fast stereo matching algorithm based on correlation of multi-resolution images using CUDA (Compute Unified Device Architecture). The performance of this implementation is faster than most of the software implementations of this method and comparable with FPGA implementation and few other optimized methods mentioned in the references. This enables the real time usage of stereo matching method. We have also provided performance comparison and results for different methods of stereo matching on CUDA. The paper concludes with analysis of results and the reasons of the performance variations. We have also given qualitative image data for comparison of accuracy of different stereo correspondence methods.

Keywords

Correlation; Multi-Resolution images; CUDA; Stereo matching.

1. INTRODUCTION

There is a continuous need for increase in processing power to reduce the execution time of computationally exhaustive algorithms. Thus, many chip manufacturers are coming up with different architectures for processors to match the challenge posed by the applications. GPUs (Graphical Processing Units) were mainly used for gaming and imaging applications and not for the general-purpose computations. NVIDIA had come up with a new technology called CUDA (Compute Unified Device Architecture) by adding simple C like APIs along with architectural changes to support these APIs. These APIs resulted in easy programming on these GPUs for general-purpose computation. The current available GPUs have around 512 cores (NVIDIA Tesla architecture) which can be used for high performance on data parallel algorithms.

The stereo matching algorithm is used to find the depth of objects from the reference point in an image taken from real scene. Major applications of stereo matching are in the field of robotics, 3D scene reconstruction, 3D television. The distance of particular object is determined by computing the disparity map. Correlation based stereo matching is one of the simplest methods for getting disparity map but it is computationally exhaustive and takes a lot of time to produce results. Hence, in its original form correlation based method cannot be used for real time operations. Correlation based method is highly data parallel and suitable candidate for CUDA implementation to get speed up. Additionally, an implementation using multi-resolution images helps to reduce the time of execution further. When this method is implemented on GPGPU, it achieves real time performance. This paper talks about implementation of different stereo matching methods using CUDA and comparison of the performance data on various images.

2. RELATED WORK

All accelerating the stereo correspondence method is mainstream research in this area and many papers have been published in stereo matching algorithm describing the methods to reduce execution time and using available parallelism. K.Sunil Kumar and U.B. Desai [1] have developed an approach to integrate different modules in stereo matching, which include feature extraction, matching, and interpolation. Because this method is computationally expensive, they have developed a multi resolution approach to solve the problem. Changming Sun [2] has implemented a method of reducing the computation time of the algorithm by modifying the formulae for calculating the variance and co-variance. Also, he has used the concept of multi-resolution images for improving the accuracy of the disparity map. This method also helps to reduce the computational time. Ke Zhu, Matthias Butenuth and Pablo D’Angelo [4] have discussed the implementation of local and global dense stereo matching method in their paper. They have explained the tradeoff between the accuracy and execution time on the GPU. They have also discussed different parallelization strategies for improving the performance. John Congote, Javier Barandiaran, InigoBarandiaran and Oscar Ruiz [5] have proposed a different implementation of stereo matching using dynamic programming to calculate the dense depth maps using CUDA architecture and achieved real time performance on GPU. They have compared the timing analysis of their implementation against CPU implementations and explained the scalability property by testing their implementation on different GPUs. Andreas Geiger, Martin Roser and Raquel Urtasun [6] have explained a novel approach to stereo matching of high resolution images. They form a group of supportive points and these points are robustly matched for finding the disparities. This allows the exploitation of disparity space and yields accurate dense reconstruction. Yong Zhao, Gabriel Taubin [7] have used progressive multi resolution pipeline, which includes background modeling and dense matching with adaptive windows. Their approach is mainly used for applications where moving objects are of interest. They have achieved 60 frames per second (fps) on an 800*600 video. Yoshikatsukimura [10] has developed a driving support system for safety purpose. Images from a camera on the vehicle are used to capture image and stereo matching algorithm is used to provide the distance information. He uses the multi resolution images and provides the distance information. Jian Sun, Yin Li, Sing Bing Kang and Heung-yeung Shum [11] have proposed a method to deal with occlusion in dense two frames stereo. They have incorporated the visibility constraint in an energy minimization framework resulting in a stereo model that treats both the left and the right image equally. They use an iterative method to determine the minimum value of the energy using belief propagation. Sven Forstmann, Yutaka Kanou, Jun Ohya, Sven Thuering and Alfred Schmitt [12] have implemented stereo matching using dynamic programming to achieve real time performance. They have used coarse to fine scheme and the MMX extensions in the hardware to increase the speed. SirajSabihuddin, JaminIslam and W. James MacLean [13] have provided a hardware implementation to the stereo matching problem. They have used a pipelined architecture i.e. FPGA, which gives almost 200 fps (frames per second) performance. RatheeshKalarot and John Morris [14] have compared the performance of stereo matching algorithm on GPU and FPGA. They have shown that performance of stereo matching on FPGA is very good. However after a certain disparity value, the FPGA implementation does not work.

Our paper is to showcase the performance achieved by using multi resolution method and to compare the performance with other similar implementations. The fps achieved by this paper is very high and it is comparable with some of the hardware implementations available in the literature. Our implementation is software implementation and hence it is not restricted to any disparity range and is flexible.

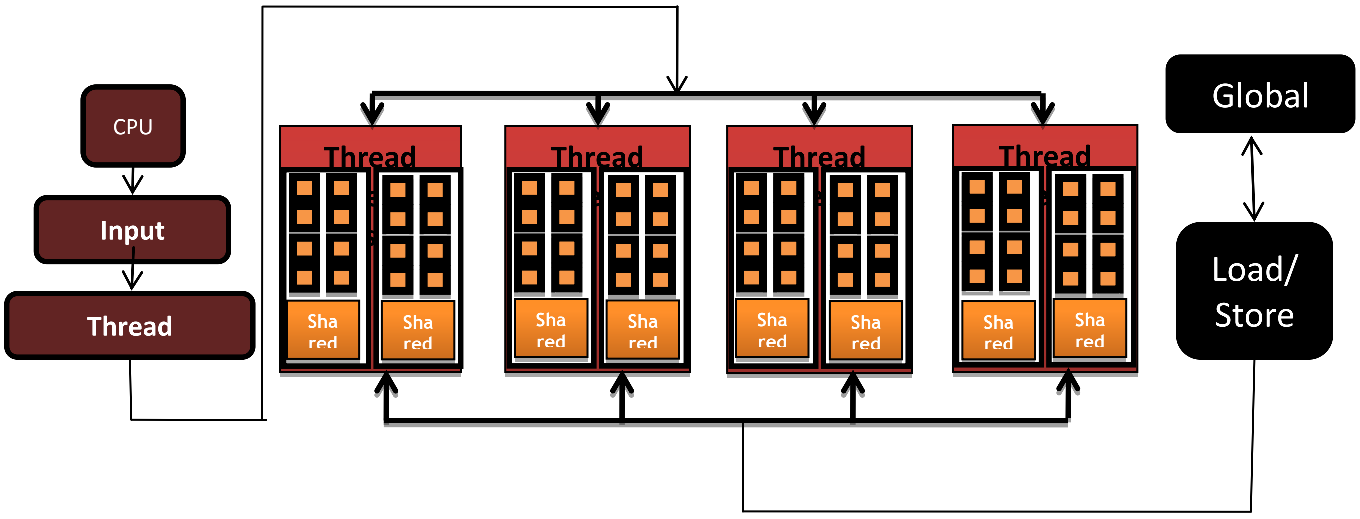


Fig 1: GPGPU Architecture

3. NVIDIA CUDA

GPUswere earlier used to get high quality graphics in gaming applications. In 2007, NVIDIA introduced a new architecture called CUDA (Compute Unified Device Architecture) to enable GPU usage for general-purpose computation. CUDA provided top level APIs to make program development for GPGPUs easier in order to utilize massive parallelization powers of GPU.

Figure 1 shows the hardware architecture of NVIDIA GPGPU [8]. The architecture consists of array of multiprocessor called symmetric multiprocessor (SM), each having its own shared memory and stream processors (SP). NVIDIA GTX 480 has 15 SM and each SM contains 32 SPs. The code is dispatched from CPU to the thread execution manager, which schedules these threads to cores. Hence, user is freed from the burden of writing code and scheduling it for load balancing. Each core can run multiple threads at the same time and hence can produce exceptional speed up required for high performance computation. Every SM can access large chunk of memory called global memory. Global memory size is huge but its performance is low as compared to shared memory. Hence, the memory management affects the GPU implementation throughput heavily. An application written using CUDA can be seen as a host program which runs on CPU in addition to a 'kernel' which is run by multiple threads at the same time on different cores of GPGPU. Threads are executed in groups, which are called as 'blocks'. Grid is the collection of blocks to be executed on the device. Multiple threads execute the kernel code at the same time. CUDA architecture provides different types of memories like global, shared, texture and constant. Performance of the CUDA code largely depends on how well the architecture of the GPU hasbeen exploited. Some of these techniques are discussed in the implementation section.

4. STEREO MATCHING

In this paper, we have described stereo matching using correlation. The disparity map generated as output of the algorithm enables to find distance of the object from the reference plane. There are two images viz. left and right image of the same environment taken from different camera angle. For every pixel in the left image we try to find corresponding pixel in the right image. The displacement of the pixel position in the right image from the pixel position in the left image gives the disparity for that particular pixel. Disparity map is a set of disparity values of each point in left image from right image.

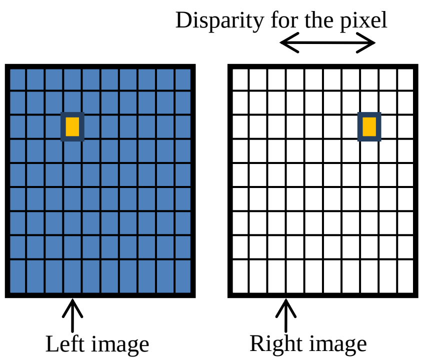


Fig 2: Pixel position in left and right image

As shown in the Figure 2, we are considering the pixel highlighted in the left image and trying to find the pixel in the right image, which has the maximum correlation with pixel under consideration. The corresponding matching pixel in right image is highlighted in Figure 2. Thus, distance between the pixels in the left and right image gives the disparity of particular pixel of left image. Let 'L(i,j)' represent any pixel in the left image and 'R(i,j)' represent any pixel in right image. Let Co(i,j) represent the correlation value, Cov(i,j) be the covariance, Var(i,j) be the variance, La be the mean value for the left image and Ra be the mean value for the right image, then we can find correlation by[2].

$$Co_{(i,j)(L,R)} = \frac{Cov(i,j)(L,R)}{Var(i,j)(L)*Var(i,j)(R)} \quad (1)$$

Where, $Cov_{(i,j)(L,R)}=$

$$\sum_{m=i-Wl}^{i+Wl}\sum_{n=j-Ww}^{j+Ww}(L(m,n)-La)*(R(m+d,n)-Ra) \quad (2)$$

$$Var^2_{(i,j)(L)}=\sum_{m=i-Wl}^{i+Wl}\sum_{n=j-Ww}^{j+Ww}(L(m,n)-La)^2 \quad (3)$$

$$Var^2_{(i,j)(R)}=\sum_{m=i-Wl}^{i+Wl}\sum_{n=j-Ww}^{j+Ww}(R(m+d,n)-Ra)^2 \quad (4)$$

The variance and co-variance takes a lot of time to compute as per equation (2), (3) and (4). Hence, it needs modifications (optimization) to reduce the computational time. The modified form of covariance and variance [2] is shown by equation (5) and (6) respectively.

$$Cov_{(i,j)(L,R)}=\sum_{m=i-Wl}^{i+Wl}\sum_{n=j-Ww}^{j+Ww}L(m,n)*R(m+d,n) - (2*Wl+1)*(2*Ww+1)L_a*R_a \quad (5)$$

$$Var^2_{(i,j)(L)}=\sum_{m=i-Wl}^{i+Wl}\sum_{n=j-Ww}^{j+Ww}(L(m,n))^2 - (2*Wl+1)*(2*Ww+1)L_a \quad (6)$$

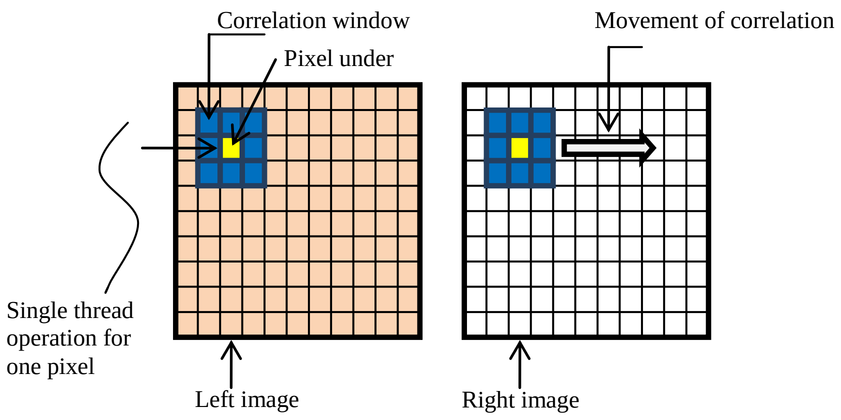


Fig 3: Correlation by single thread for each pixel

An exhaustive search is performed on the right image until the maximum value of correlation is not found for the particular pixel. However, this search is limited to the maximum value of disparity set. The correlation can be performed by using kernels of different sizes viz. 3*3, 5*5, 7*7. This algorithm is highly data parallel. The correlation of each pixel can be computed with a single thread and hence all the pixels can be computed in parallel. In addition, computation of the disparity for each pixel can be performed in parallel.

5. STEREO MATCHINGUSING CORRELATION

In this section, we will describe different parallelization strategies and architectural optimization applied to the correlation method on CUDA to improve the performance.

5.1 Global Memory Implementation

In the first method, we load both the left and right images in the global memory. A correlation window is chosen for e.g. 3*3 and parallel threads compute correlation of each pixel in left image with multiple pixels in right image. Each thread fetches values from the left image in the correlation window and performs correlation with the corresponding pixels from the right image. The values of correlation are calculated for one pixel in the left image by shifting the correlation window in the right image. Figure 4 explains the movement of the correlation window for right image. The correlation is calculated using equations (1)-(6). The amount of shift of the window in the right image is determined by fixing the maximum allowed disparity in advance.

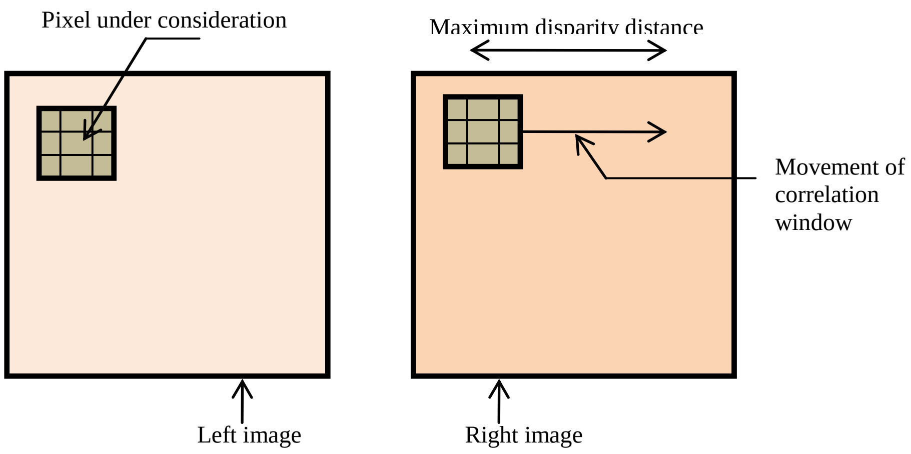

Pixel under consideration
Maximum disparity distance
Movement of correlation window
Left image
Right image

Fig 4: Movement of correlation window for the pixel under consideration

Each thread stores the maximum value of correlation computed by comparing it with the previously calculated values and stores the distance at which the maximum value is obtained. The value of the distance gives us the disparity map for the image. Disparity map is used to determine the distance of the object from the reference plane by using other information related to camera positions. This implementation is simple and results in high speed up compared to the CPU implementation but global memory access is very slow and hence overall speed up is not up to the mark.

5.2 Shared Memory Implementation

In the second method, we store the left image in the shared memory and the right image is fetched from the global memory. The advantage of the second method over the first is that, in the left image there is memory access repetition. Fetching values from shared memory is very fast and results in reduction in execution time. Right image cannot be stored in shared memory because number of pixels to be fetched from the right image exceeds the maximum storage capacity of shared memory. Thus, the values to be computed from the right image would have to be fetched from the shared memory of other blocks (non-coalesced memory access) which would take more time to fetch. The execution of correlation is the same as that explained in the previous implementation. Each thread is used to compute the correlation values of single pixel and to calculate its disparity value. A speed up of almost 8 to 10 times is obtained by utilization of the shared memory for storing the left image.

5.3 Shared Memory and Texture Fetch Implementation

In this approach of implementing stereo matching algorithm using correlation, we can make use of texture memory available in the GPGPU. Fetching data from texture memory is advantageous when there is a fixed pattern of data access. This implementation makes use of 1D texture for storing the image. Here, we store the left image in the shared memory and the right image in the texture memory. We make use of 1D texture fetch to get the values. The reason for using 1D fetch is because of the need of the algorithm. The values to be used for correlation from the right image for a pixel in the left image are along the row in which the pixel lies. The method to find the correlation is similar to that explained in previous section. Each thread is used to calculate the disparity of a single pixel in the left image. Thus, all the pixels are computed in parallel. The difference is just that the values are fetched from shared memory and texture memory for left and right image respectively. The performance of this implementation is low as compared to the previousimplementation where the right image is stored in the global memory. This is because the values to be fetched are not located within the same block (non-coalesced memory access). Thus, we have to fetch it from other blocks, which cause the delay in fetching it.

6. DYNAMIC PROGRAMMING METHOD

Dynamic programming method for stereo matching falls under semi global method as it considers the cost of neighboring pixels for cost computation of current pixel. Dynamic programming is NP hard problem.

Consider, Il – left image and Ir – right image

6.1 Cost computation and aggregation

For each pixel of one scan line, cost matrix Ds is with size WxDmax is created where W is width of image and Dmax is maximum allowed disparity.

$$Ds(x,d) = SAD(x,d) + \min(Є + Ds(x-1, d-1), Ds(x-1,d), Є + Ds(x,d+1)) \quad (7)$$

$$\text{Where, } SAD(x,d) = abs(Il(x,y) - Ir(x+d,y)) \quad (8)$$

and y - scan line

6.2 Minimum path computation

Once Ds is created for each scan line, it is traversed from end and min cost path is followed. Based on the min path, disparity for each pixel in a particular scan line is computed. This is explained with the help of following pseudo code.In this process, the path is stored in one variable which can be used for disparity map computation. We are not discussing the CUDA implementation of dynamic programming in detail because it is out of the scope of this paper. This implementation was running at 28 fps.

```
m = width and n - Dmax

while( m != 1 && n != 1)
{
       Min = min(Up, Upleft, Left)
       If min == Up
        N --
        If min == Left
        m--
        if(min == Upleft)
        n—
        m--
}
```

7. MULTI RESOLUTION METHOD

The methods discussed in earlier sections provided good speed up and dynamic programming method provided better results, still, the results are not meeting the real time criteria. Hence, we make use of multi-resolution [10] images along with correlation for obtaining more accurate results and better speed up. In this method, initially we reduce the size of the image to half.

This reduction of image size as shown in Figure 5 is a complete data parallel method. In this, we fetch three pixels from the image and take their mean. These mean values are then stored. Figure 6 explains the computation of reduction.

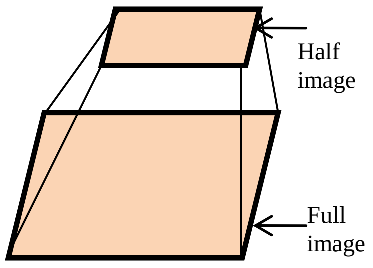


Fig 5: Reduced size of image

Numbers of threads released are equal to the number of pixels in the reduced image. Then we perform correlation on the reduced image. As the image size is reduced to half, execution time also reduces considerably. The correlation technique used is similar to that explained in the previous sections. The left image is stored in the shared memory and the values from the right image are fetched from the global memory (Best possible combination based on performance measurement). Using equations (5) and (6) for finding the variance and co-variance, time taken reduces further (optimization to reduce the computation).At the end of this computation, the disparity map of half size image is obtained. After getting this value, maximum disparity value is determined from disparity map of half size image. This can be done using reduction method in CUDA. Reduction method calculates local maxima of small CUDA blocks. At the end, global maximum is obtained by using local maxima obtained by previous step. The disparity map obtained is interpolated to the size of the input images. The interpolation is calculated by the equation (7). This process is data parallel. Initially, the values in the disparity map obtained from the reduced images are placed over a matrix of double its size.

To calculate interpolation on CUDA, we release threads equal to the product of height of the input image and half the width. Now, we fill in the gaps width-wise first. Therefore, each thread computes value for each unfilled pixel. Interpolation is calculated using equation (9). Once the rows are filled, then the columns are filled following similar manner. Let us call the input image is at level 0 and image with half size at level 1 and so on. p(i,j)L is the position of a pixel at level L and d(p) be the disparity of that particular pixel.

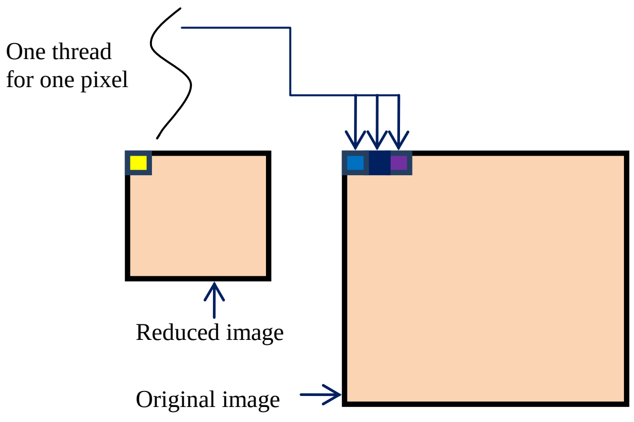


Fig 6: Computation of one pixel value of reduced image by single thread

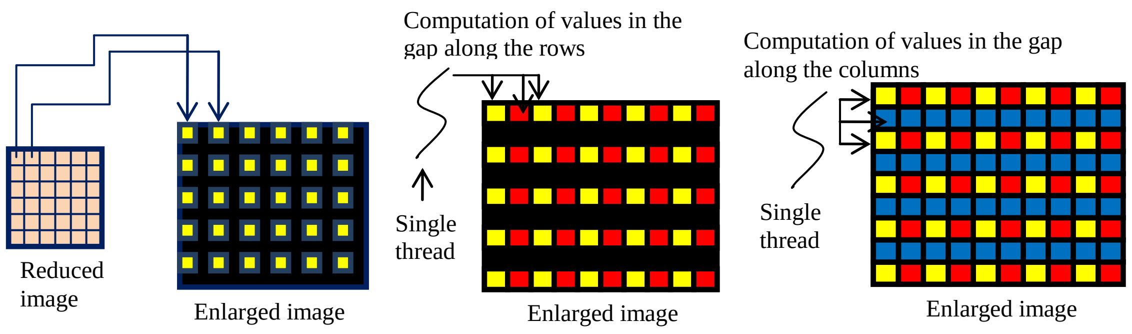


Fig 7: Interpolation of reduced image to original size

Afterwards, we make use of the interpolated disparity map (d(p)L) along with the disparity map obtained for the reduced image and find the cost function. The cost function makes use of the correlation values obtained from the left and right input images. Let Ci,j be the cost function and Coi,j be the correlation values. Let DmL be the maximum value of disparity obtained at level L. then the cost function [3] can be given by equation (10).

Where d(i,j)L is the disparity value of level L and d(i,j)L-1 is the disparity value of level L-1. By multiplying the cost function with the interpolated disparity map, we obtain the actual disparities for the input images.

$$d_{(p)L} = d_{(p)L-1} + \frac{p(L-1) - p(i,j)(L)}{p(i+1,j)(L-1) - p(i,j)(L)} *(d(p_{(i+1,j)}) - d(p_{(i,j)})) \quad (9)$$

$$C_{i,j} = Co_{i,j} + \frac{|d(i,j)L - d(i,j)L-1|}{DmL} \quad (10)$$

This method of implementation takes very less execution time and gives better and accurate results. A speed up of approximately 8 times as compared to the shared memory implementation is obtained.

8. RESULTS DISCUSSION

This section will discuss the performance of all of the methods discussed so far and some of the fast implementations available in literature. We have used NVIDIA GTX 480 on Intel dual core system for all these experiments. VGA image is used as input (from Middlebury website [9]) and performance is measured by varying the maximum disparity range. The block size chosen is 16x16 and the size of the correlation window is set as 7x7. Execution time is directly proportional to the maximum allowed disparity because, as the disparity range increases, number of pixels to be considered for correlation of a single pixel increases. When global memory is used, the pixels are fetched from global memory. As the global memory access is slow, performance is not very good. In the second method, shared memory access is used which gives 10 times better performance compared to the global memory implementation. Table 1 lists fps achieved using these techniques. Texture fetch in general is fast. However, the values to be fetched from the texture memory are not limited to the block size (non-coalesced memory access). Hence, texture fetch is also not improving the performance much as shown in Table 1. The first three methods do not meet the real time performance criteria. Dynamic programming is a different and fast approach to stereo matching as mentioned in previous section. However, it gives almost real time performance, which is around 30 fps. The multi-resolution method is the most efficient method giving more accurate results and real time performance. Multi-resolution method gives 3 times better performance compared to the dynamic programming method and 6 times better compared to correlation method using shared memory.

Table 2 gives details of the performance comparison of different methods available in literature. The multi-resolution implementation gives real time performance and proves to be faster than the state of art implementations present today. The progressive multi resolution implementation also gives real time performance. The progressive multi resolution method is used only for foreground objects and achieves 60 fps at 250 disparities as shown in Table 2. In our multi-resolution implementation, we find the disparity of the entire input image. Even though it processes the entire image, it gives real time performance (maximum 120 fps for maximum disparity 60 and 66 fps for Dmax - 150). Few other methods involving hardware implementation on FPGA is also included in Table 2. These methods perform better than the CUDA implementation proposed by us but lack in flexibility provided by software implementation. Also one of the papers suggests that FPGA implementation for disparity more than 256 is not feasible [14].

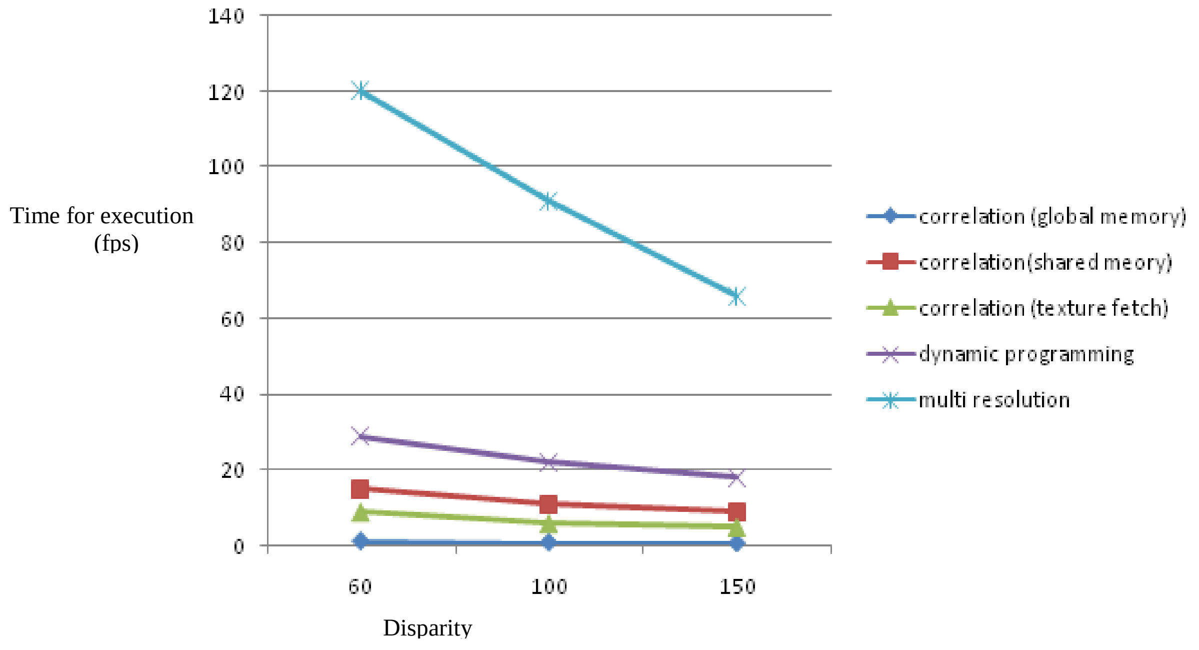


Fig 8: Comparative graph of different techniques

Table 1.Performance comparison of different methods

Method	Image size	Disparity Range	Time (ms)	FPS
Correlation (Global Memory)	640x480	60 100 150	800 1100 1400	1.25 0.9 0.71
Correlation (Shared Memory)	640x480	60 100 150	70 95 120	15 11 9
Correlation (Texture fetch)	640x480	60 100 150	120 165 200	9 6 5
Dynamic Programming	640x480	60 100 150	35 47 55	29 22 18
Multi resolution	640x480	60 100 150	11 14 18	120 91 66

The algorithm was tested on images available in the Middlebury database [9]. Figure 10 shows the results using different approaches of stereo matching. Figure 10 (c) and 10 (d) shows the output of multi-resolution and simple correlation method where, multi-resolution accuracy is more compared to simple correlation. Figure 10(f) shows the result of the dynamic programming method, which takes care of occluded, objects as well. Figure 10(e) is the result obtained by SAD implementation, which is less accurate as compared to the dynamic programming method.

We have tested our implementation on sample images from Middlebury website [9].We have obtained high speed up as compared to other implementations. We have obtained more accurate results as compared to the methods mentioned in Table 1. When the disparity is low we have very high performance. However, as the value of disparity goes on increasing the performance degrades. However, it still provides the performance for real time applications.

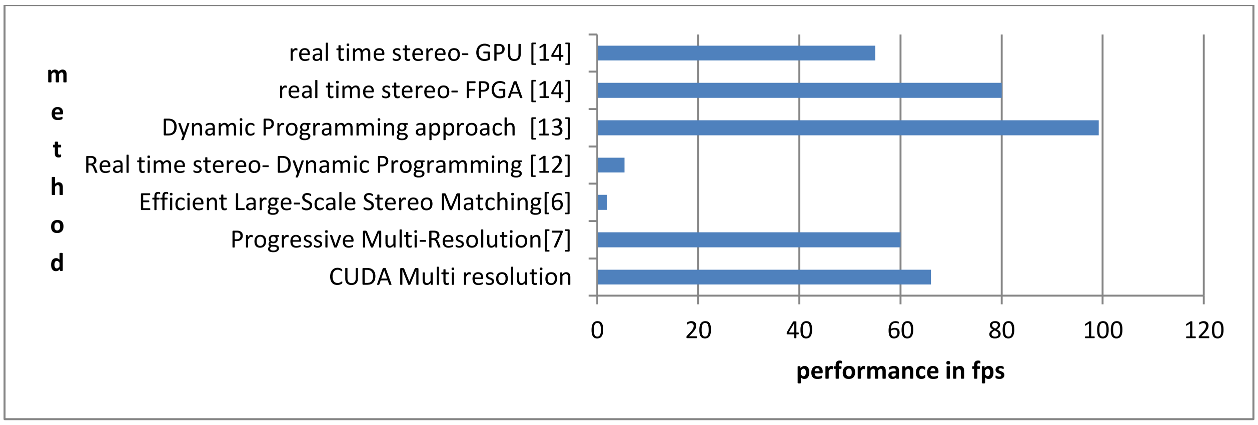


Fig 9: Performance Comparison of different techniques mentioned in Table 2

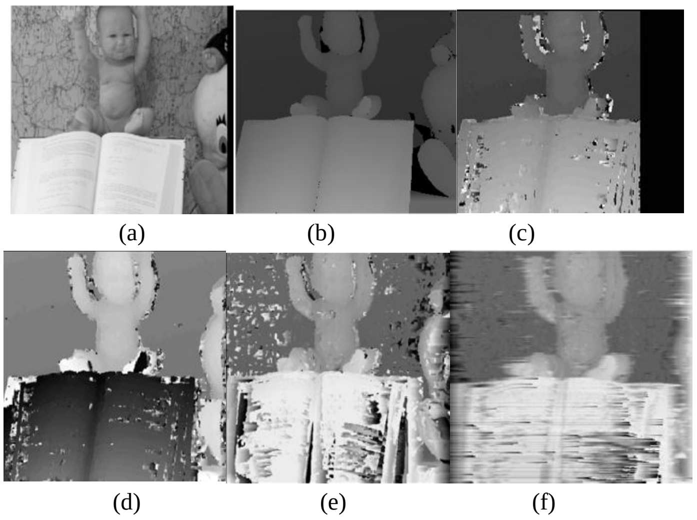


Fig 10: (a) Input left image (b) ground truth (c) multi resolution (d) simple correlation(e) SAD (f) Dynamic programming

9. CONCLUSION

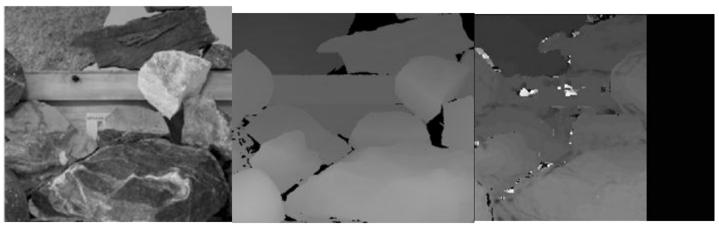
Different implementations of stereo matching algorithm on GPGPU have been discussed in this paper. Correlation based method for stereo matching is simple and data parallel. This paper gives comparative analysis of different implementation of correlation method on CUDA and the impact of these methods on performance is discussed. Performance of global memory implementation is slower than shared memory implementation by a factor of 10. Texture fetch did not result in faster execution as opposed to the expectation because of the reasons explained in implementation section. Finally, to achieve high performance and better accuracy, multi-resolution method is used. This implementation resulted in very high speed up (6 times compared to shared memory implementation) and also provides better accuracy over the traditional correlation methods. The performance analysis has been done for these methods and it is found that the method of stereo matching using multi-resolution images gives better accuracy as compared to others and also gives real time performance. In addition, multi resolution method on CUDA outperforms most of the implementations available in literature. The reason behind this speedup is the optimization provided by multi resolution concept and the GPUs massive parallel processing power. The performance of the multi resolution method can be compared to the implementations on GPUs and FPGA as mentioned in the literature. However, in the FPGA implementation the algorithm cannot be implemented beyond a particular value of disparity. Therefore, we can say that the GPU's massive processing power can provide dramatic change in data parallel algorithm's performance.

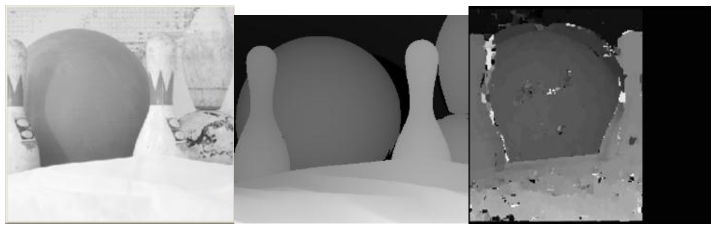
10. ACKNOWLEDGMENT

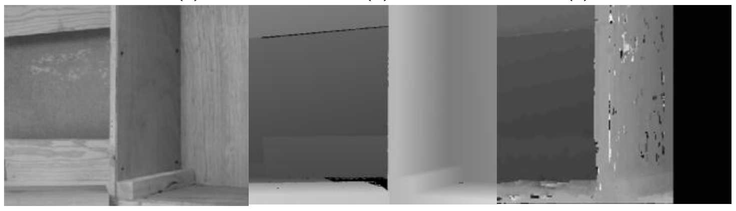
Our sincere thanks to the CREST team in KPIT, Pune, India, and LG-PRI team in South Korea, for their support to improve the quality of this paper. This project was partially funded by LG Inc., South Korea.

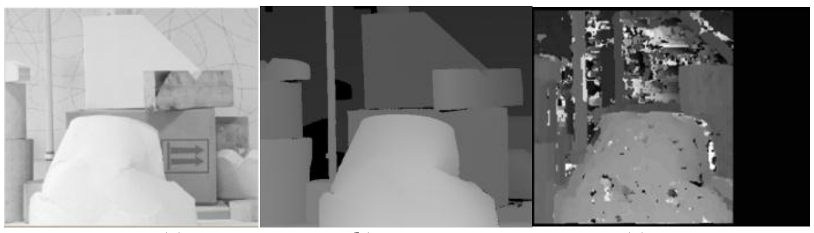
Table 2.Performance Comparison

Method	**Image Size**	**Disparity**	**Performance (fps)**
CUDA Multi resolution	640x480	150	66
Progressive Multi-Resolution Adaptive Windows [7]	800x600	250	60
Efficient Large-Scale Stereo Matching[6]	1382x512	100	2
Fast Stereo Matching Method [2]	512x512	60	0.25
Multi resolution scheme for stereo correspondence using correlation techniques [3]	320x240	60	0.2
Real time stereo by Dynamic Programming [12]	640x480	128	5.38
Dynamic Programming approach to high frame rate stereo correspondence. A pipelined architecture on FPGA [13]	640x480 (pipe) 640x480 (ppipe)	128 128	99.20 49.24
Comparison of FPGA and GPU implementation for real time stereo vision [14]	640x480	128	GPU – 55 FPGA- 80


(a) (b) (c)


(a) (b) (c)


(a) (b) (c)



(a) (b) (c)

Fig 10: a) Input left image b) Ground truth c) Disparity map by multi resolution algorithm

11. REFERENCES

[1] Sunil Kumar, K., Desai, B., U. 1994."New Algorithm for 3D Surface Description and Binocular Stereo Using Integration",journal of Franklin Institute, Volume 331, Issue 5, September 1994, Pages 531–554.

[2] Sun, Changming.1997.Fast Stereo Matching Method.In Digital Image Computing: Techniques and Applications.

[3] Satorre,Rossana.,Compan, Patricia., Botia, Antonio., Rizo, Ramon.Multi Resolution Scheme for Stereo Correspondence using Correlation Techniques. University of Alicante.

[4] Zhu, Ke.,Butenuth, Matthias., D'Angelo, Pablo.2010. Comparison of Dense Stereo Using CUDA.ECCV, Workshop `Computer Vision on GPUs´.

[5] Congote, John.,Barandiaran, Javier., Barandiaran, Inigo., Ruiz, Oscar.2009. Real Time Dense Stereo Matching with Dynamic Programming in CUDA.CEIG'09, San Sebastián.

[6] Geiger, Andreas.,Roser, Martin., Urtasun, Raquel.2010. Efficient Large Scale Stereo Matching. In ProceedingACCV'10 Proceedings of the 10th Asian conference on Computer vision - Volume Part I, Pages 25-38.

[7] Zhao, Yong.,Taubin, Gabriel."Real Time High Definition Stereo on GPGPU using Progressive Multi Resolution Adaptive Windows", Journal Image and Vision Computing, Volume 29 Issue 6, May, 2011, Pages 420-432.

[8] NVIDIA Corporation., 2008. NVIDIA CUDA Programming Guide. Technical Report. California. USA.

[9] Middlebury Stereo Website (05.2010). (http://vision.middlebury.edu/stereo/).

[10] Kimura, Yoshikatsu., A Stereo Matching Method using Multi Resolution Images. R&D Review of Toyota CRDL Vol. 36 NO.1 (2001.3).

[11] Sun, Jian., Li, Yin., Bing Kang, Sing., Shum, Heung-Yeung. 2005. Symmetric Stereo Matching for Occlusion Handling in proceedings CVPR '05 Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02, Pages 399 – 406.

[12] Forstmann, Sven.,Kanou, Yutuka., Ohya, Jun., Thuering, Sven., Schmitt, Alfred.2004. Real Time Stereo by Dynamic Programming, in Computer Vision and Pattern Recognition Workshop, 2004.CVPRW '04.

[13] Sabihuddin, Siraj., Islam, Jamin., MacLean, James. W. 2008.Dynamic Programming Approach to High Frame Rate Stereo Correspondence: APipelined Architecture Implemented on a Field Programmable Gate Array, in Electrical and Computer Engineering,CCECE'08.

[14] Kalraot, Ratheesh., Morris, John.2010. Comparison of FPGA and GPU Implementation of Real Time Stereo Vision, in Computer Vision and Pattern Recognition Workshops (CVPRW), IEEE Computer Society.