Disparity Map Generation On Graphic Processor Unit

Shrilesh Kale - 3438873, Sarthak Bapat - 3441556

Institue for Parallel and Distributed Systems, IPVS, University of Stuttgart
st169725@stud.uni-stuttgart.de, st169660@stud.uni-stuttgart.de

Abstract. The objective of this project is to implement stereo matching algorithm using GPU programming. Disparity is the difference of x coordinate between two corresponding points in the reference image and the target image. Once we get the disparity value of each pixel in the reference image, we could store all these values into another image with grey scaled format which is also called disparity map. Matching cost computation is the method used to get the differences of the pixel values in two images. Commonly, SSD (sum of squared intensity differences) and SAD (sum of absolute intensity differences) algorithms are used to find out disparity value of pixel.

Keywords: Disparity Map · Computer Vision · Stereo Matching · SSD · SAD · GPU

1 Introduction

Stereo matching is a wide research topic in computer vision. It is a subject to estimate disparity information between two or more images obtained from slightly different viewpoints. Depth estimation (Disparity calculation), as the name says, is the task that measures the distance between two objects. Distance information could be found out using stereo vision. Stereo vision method is the intelligent one that uses a pair of cameras to capture the images synchronously and then calculates the distance of the object apart from the cameras through the differences between the images shot by the two cameras separately.

In the implementation part of disparity calculation, the GPU kernel function called matching cost computation compares the differences of the two images using sum of squared intensity differences (SSD) algorithm and then generates the depth information for each pixel and result is then written in memory buffer.

2 Algorithm

Generally, the disparity map algorithms are classified into local and the global approaches. A local approach uses area based or a window based approach. This is because for a given pixel the disparity depends upon the intensity values within a predefined support window. As a result, the local approach algorithms have a low computational complexity as well as a short running time. On the contrary, the global approach views disparity assignment as a problem of minimizing a global energy function over the entire disparity range. This makes the global approach computationally expensive although it gives a good result[5].

There are a few algorithms that could be used to calculate the disparity (depth) of the two stereo images. SD (Squared Differences), SAD (Sum of Absolute Differences), SSD (Sum of Squared Differences), NCC (Normalized Cross Correlation), SGBM (Semi-Global Block Matching) to name a few are being used in stereo matching disparity map calculation. All the algorithms have their own advantages and disadvantages. For instance, the SAD algorithm is fast but the quality of disparity map generated is low because of the noise at the object boundaries and in the texture less regions. SGBM operates on the principle of minimizing a global energy function related to the disparity map (disparity of each pixel)[5].

The objective is to design and implement an efficient algorithm on GPU to generate the disparity map (depth) in real time or near real time. We select the SSD (Sum of Squared Differences) algorithm as it can be efficiently paralleled to execute on the GPU platform. The basic idea of the SSD algorithm is to sum the squared values of the corresponding pixel differences of the corresponding pixel blocks of the left and the right images. This is a local approach algorithm, thus is computationally light and fast to execute. The inputs to the application are two grey scaled images of the same dimensions obtained from a stereo camera. Result is a disparity map calculated from the x-coordinate difference between the two images. Following are the steps involved in the SSD algorithm[4].

1. Calculate the SSD (Sum of Squared Differences)[4] for one pixel using a fixed window in the valid range if the disparity.
2. Find the minimum SSD for a specific disparity.
3. Set the output images's pixel value with the disparity which has the minimum SSD.
4. Repeat the steps for all the pixels in the image.

3 Implementation

3.1 Implementation on CPU

The above discussed SSD algorithm was implemented on the CPU. The computation was done in order to compare the performance of the algorithm on different underlying hardware architectures. Due to the architectural differences it is not possible to have parallelism on CPU unlike the GPU. The inputs are two grey scaled stereo-matched images with the same dimensions(nxn). The image could have different width and height but for the simplicity we assume both as n. The disparity range is upto l. The window size is represented by m. The final disparity selected as per the algorithm is represented by 'Disparity'. The algorithm for the CPU implementation would be as follows[4].

```
for $1 \leq row, col \leq n$
  for $1 \leq d \leq l$
      for $1 \leq x, y \leq m$
          SSD $\equiv SSD + (abs(L[row+y][col+x] - R[row+y-d][col+x]))^2$;
      end for
      if SSDmin $\geq SSDthen$
          SSDmin $\equiv SSD$;
          Disparity $\equiv d$;
      end if
  end for
  D[row][col] := Disparity;
end for
```

As can be seen from the algorithm, the CPU implementation of disparity map calculation takes a good amount of time due to the nesting of the loops over a large n. The time complexity of this algorithm on CPU is O($\mathrm{n}^2.l.m^2$).

3.2 CPU Implementation Results

We now look at the results for the implementation of our algorithm on the CPU. All the results were captured with the window size of 11 in the algorithm. The quality of the images obtained on the CPU is similar to the GPU but the run time of the algorithm is much more on the CPU. We will discuss regarding the performance in a later section.

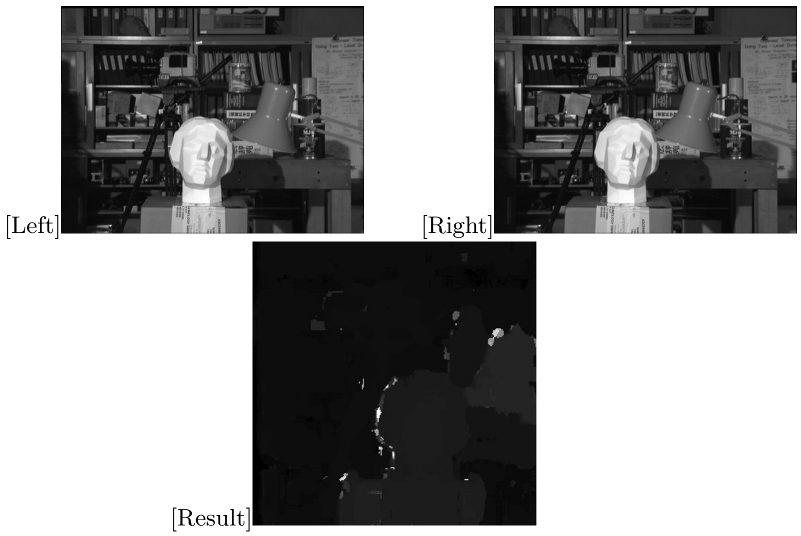


Fig. 1. CPU implementation for image Scene

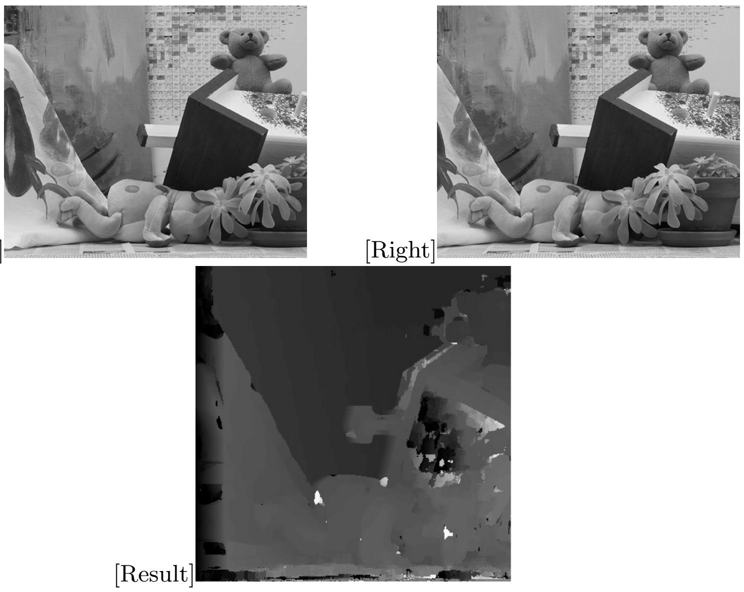


Fig. 2. CPU implementation for image Teddy

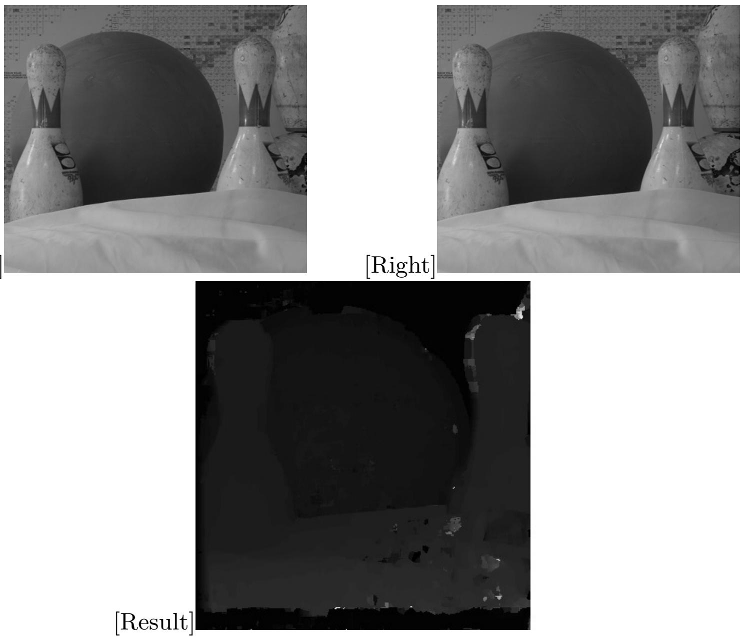


Fig. 3. CPU implementation for image Bowling

In the following section we will see the implementation of our algorithm on GPU and compare the results.

3.3 Implementation on GPU

The algorithm used to compute disparity map on GPU is identical to CPU. Since GPU provides parallel processing, the computation speed on GPU is higher than CPU.Hence Implementation on GPU should provide faster result than implementation on CPU as GPU can process the pixels in parallel fashion.

The implemented algorithm sum of squared intensity difference (SSD)[4] on GPU make use of small window (search range) to cover Left-Image and Right-Image and selects all the pixels in the area covered by the small window. The pixels in the area covered by Left-Image are subtracted from the pixels in the area covered by Right-Image, and the sum of the absolute values of the differences of all pixels is calculated and finally result is squared off to get the final pixel differences. Move the window of right image and perform the above operation until all the pixels of right image gets covered by window. Algorithm then finds minimum SSD for a specific disparity and set the resultant image's pixel value with the disparity which has the minimum SSD. Finally resultant image's memory buffer is written back to host(CPU) to see the final image.

3.4 GPU Implementation Results

The implementation on the GPU is expected to be very fast as compared with the CPU. The factor for the speedup is available in the performance section later. Here, we look at the disparity map result images generated on the GPU platform. The results were captured by setting the window size of 5 and 11 in the algorithm to observe the effect of this parameter on the quality of the image generated. First, we show the grey scaled input left and the right images. They are followed by the corresponding disparity map images for window size 11 and 5. We observe the disparity map for the three input datasets.

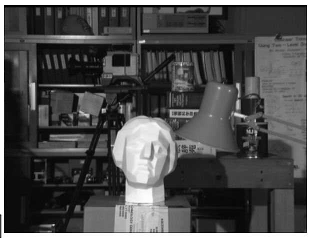
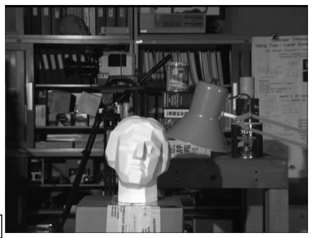
[Left]

[Right]


Fig. 4. Input Images Scene

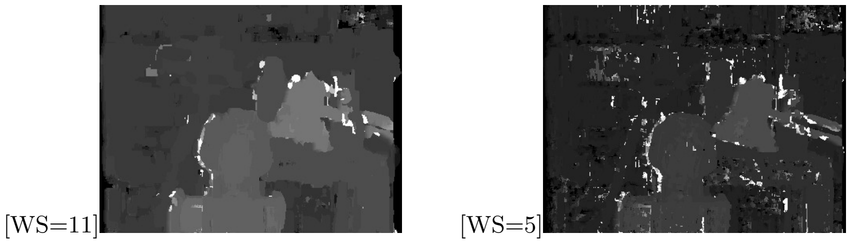


Fig. 5. GPU Result Images Scene

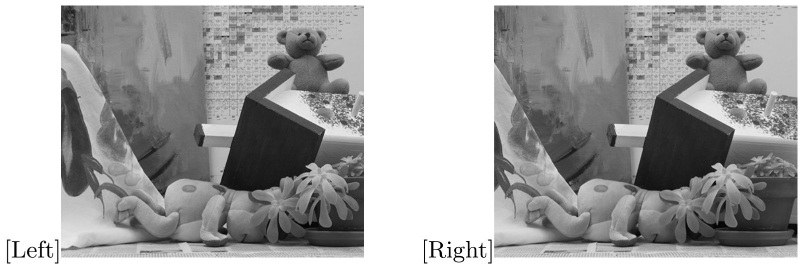


Fig. 6. Input Images Teddy

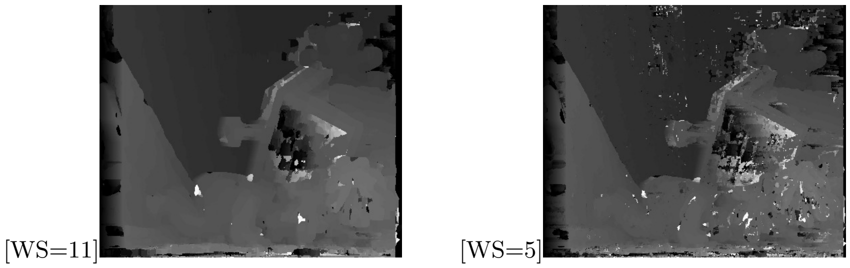


Fig. 7. GPU Result Images Teddy

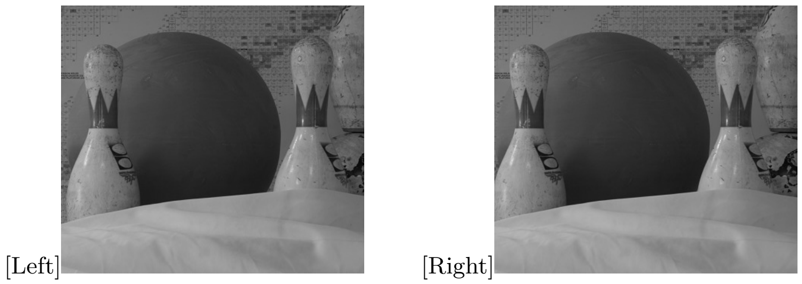


Fig. 8. Input Images Bowling

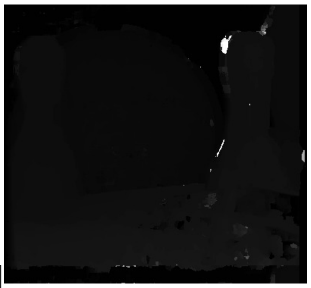
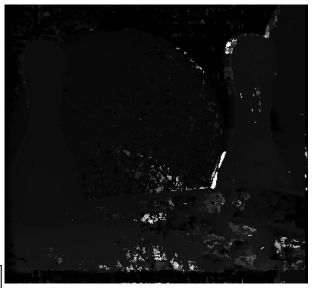
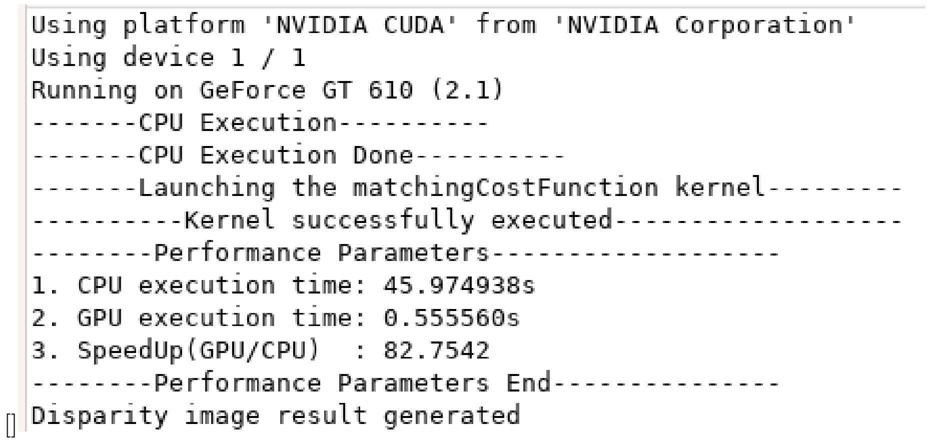
[WS=11]  [WS=5] 

Fig. 9. GPU Result Images Bowling

4 Performance Comparison between CPU and GPU

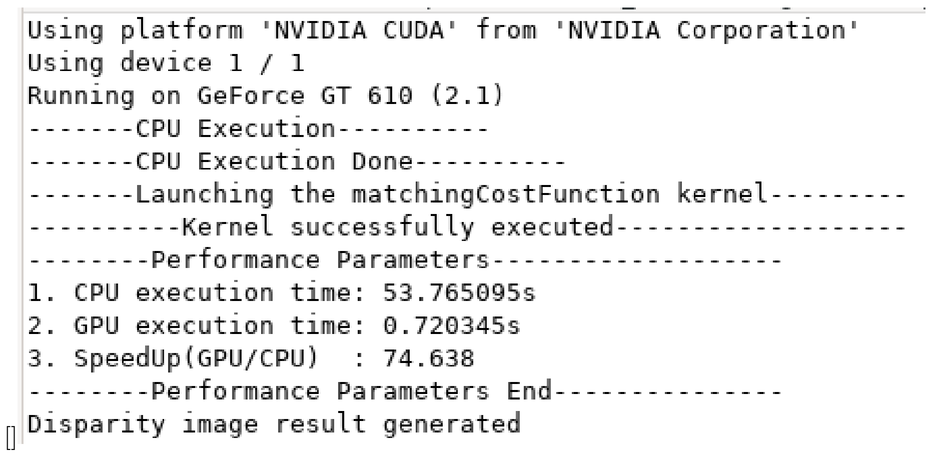
Execution time is an important parameter to compare the performance of an algorithm on different architectures. For our disparity map algorithm, we have comparison of the execution time needed for the algorithm on the CPU and GPU. The following images show the performance and speedup on GPU compared with CPU.



```
Using platform 'NVIDIA CUDA' from 'NVIDIA Corporation'
Using device 1 / 1
Running on GeForce GT 610 (2.1)
-------CPU Execution----------
-------CPU Execution Done----------
-------Launching the matchingCostFunction kernel---------
----------Kernel successfully executed-------------------
--------Performance Parameters-------------------
1. CPU execution time: 45.974938s
2. GPU execution time: 0.555560s
3. SpeedUp(GPU/CPU)  : 82.7542
--------Performance Parameters End---------------
Disparity image result generated
```

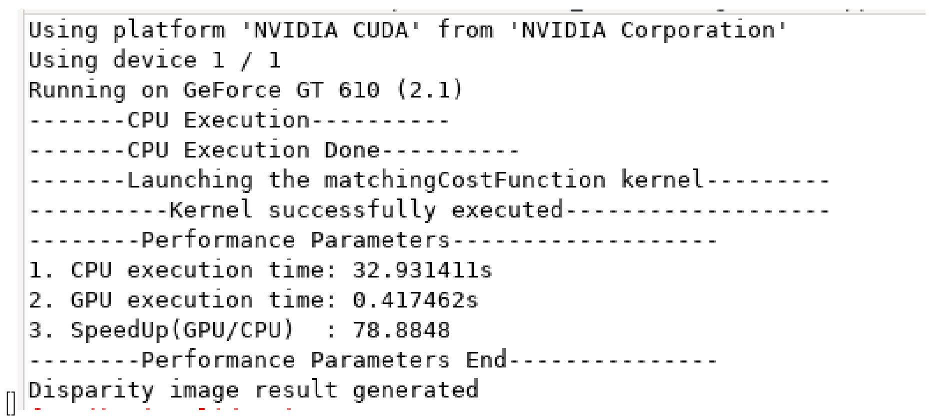
Fig. 10. Performance Parameters for Bowling Image

We can see that there is a massive speedup of 82.75 observed on GPU. This is the result of the parallelism that is available on the GPU unlike the CPU. As a result, the loops involved in the CPU code are executed all parallel on the GPU, thus giving a huge speedup for the repetitive tasks. Similarly, we have the speedups shown below for other results.



```
Using platform 'NVIDIA CUDA' from 'NVIDIA Corporation'
Using device 1 / 1
Running on GeForce GT 610 (2.1)
-------CPU Execution----------
-------CPU Execution Done----------
-------Launching the matchingCostFunction kernel---------
----------Kernel successfully executed-------------------
--------Performance Parameters-------------------
1. CPU execution time: 53.765095s
2. GPU execution time: 0.720345s
3. SpeedUp(GPU/CPU)  : 74.638
--------Performance Parameters End---------------
Disparity image result generated
```

Fig. 11. Performance Parameters for Teddy Image



```
Using platform 'NVIDIA CUDA' from 'NVIDIA Corporation'
Using device 1 / 1
Running on GeForce GT 610 (2.1)
-------CPU Execution----------
-------CPU Execution Done----------
-------Launching the matchingCostFunction kernel---------
----------Kernel successfully executed-------------------
--------Performance Parameters-------------------
1. CPU execution time: 32.931411s
2. GPU execution time: 0.417462s
3. SpeedUp(GPU/CPU)  : 78.8848
--------Performance Parameters End---------------
Disparity image result generated
```

Fig. 12. Performance Parameters for Scene Image

5 Project Contribution and Planning

We as a group of 2 people have actively participated in all the tasks involved in project execution. Firstly we did research about all the stereo matching algorithms that are available to execute on CPU. Unfortunately, we did not get any CPU implementation so we had to do everything on our own. But we were successful in figuring out right algorithm for GPU implementation. We drawn the flow chart on piece of paper and tried to figure out code complexity and proposed the optimal programming solution.

Finally we started our programming task together with private git-lab repository initialization. At each point of time, We committed our code regular basis with meaningful commit message which helped us to rectify our mistakes in the code. Once implementation was ready, we debugged and tested the code with real time data set.

6 Discussion and Conclusion

The grey levels represent the actual depth of the objects in an image, more greyness means the object is near the stereo-camera whereas less greyness represents more depth of an object[Fig.5].

The window size must be large enough to include enough intensity variation for reliable matching, but small enough to avoid the effects of distortion. If Window size is too small, the disparity map will be more noisy resulting in greater white pixels. As the Window size increases, the view becomes smoother and the blackness in the image goes up reducing the grey area.

References

1. Oscar Alvarado-Nava and Eduardo Rodriguez Martinez,"Parallel Implementation in a GPU of the Calculation of Disparity Maps for Computer Vision - July 2018"
2. Sang Hwa Lee and Siddharth Sharma, "Real-Time Disparity Estimation Algorithm for Stereo Camera Systems- August 2011"
3. Manuel Dominguez-Morales, Angel Jimenez-Fernandez,Rafael Paz-Vicente,Alejandro Linares-Barranco, "Stereo Matching: From the Basis to Neuromorphic Engineering-July 2012"
4. Ying Zhang - Msc Thesis, "3D Information Extraction Based on GPU - October 2010"
5. Rostam Affendi Hamzah and Haidi Ibrahim, "Literature Survey on Stereo Vision Disparity Map Algorithms"
6. https://www.programmersought.com/article/93105038292/
7. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3658728/
8. Gasim Mammadov, OpenCL Introduction