Improving Processing Speed of Real-Time Stereo Matching using Heterogenous CPU/GPU Model

A. Al-Marakeby, M Zaki

Abstract: ***This paper presents an improvement of the processing speed of the stereo matching problem. The time required for stereo matching represents a problem for many real time applications such as robot navigation , self-driving vehicles and object tracking. In this work, a real-time stereo matching system is proposed that utilizes the parallelism of Graphics Processing Unit (GPU). An area based stereo matching system is used to generate the disparity map. Four different sequential and parallel computational models are used to analyze the time consumed by the stereo matching. The models are: 1) Sequential CPU, 2) Parallel multi-core CPU, 3) Parallel GPU and 4) Parallel heterogenous CPU/GPU. The dense disparity image is calculated, and the time is highly reduced using the heterogenous CPU/GPU model, while maintaining the same accuracy of other models. A static partitioning of CPU and GPU workload is properly designed based on time analysis. Different cost functions are used to measure the correspondence and to generate the disparity map. A sliding window is used to calculate the cost functions efficiently. A speed of more than 100 frames per second(f/s) is achieved using parallel heterogenous CPU/GPU for 640 x 480 image resolution and a disparity range equals 50.***

Keywords : ***parallel computing, stereo matching, GPU, heterogenous computing, multi-core processors, sliding window.***

I. INTRODUCTION

Depth extraction is essential component in many computer vision applications such as robotics vision, Unmanned Aerial Vehicle UAV, and autonomous vehicles. Depth can be determined by means of LIDAR (Light Detection and Ranging), stereo matching , or RGB-D (Depth Sensor) cameras. In many cases a fusion of multiple sensor and techniques are used to get better performance. Stereo matching is a time consuming and heavy process, but it has the advantage of high resolution compared with other depth extraction sensors and techniques[2][4]. A lot of research, techniques and solutions have been suggested to reduce the time consumed in stereo matching and make it suitable for real time systems. Multi core architectures, FPGA , and GPUs are used to accelerate the processing time for stereo matching[1][7][11].

Revised Manuscript Received on March 30, 2020.
* Correspondence Author

A. Al-Marakeby*, Systems and Computers Engineering Dept. , Faculty of Engineering , Al-Azhar University, Cairo, Egypt.
Email: a.marakeby@azhar.edu.eg
M. Zaki, Systems and Computers Engineering Dept. , Faculty of Engineering , Al-Azhar University, Cairo, Egypt.
Email: azhar@enu.eg.

Using GPUs (Graphics Processing Unit) is a good solution due to the high parallelism nature of these platforms. They are very efficient for simple independent repeated tasks. The stereo matching process has the parallelism nature, while the calculation in some part of the image doesn't depend on other parts and hence, they can be calculated simultaneously. There are many stereo matching techniques such local, semi-global, and global matching. In local methods constraints are used on a small number of pixels near the pixel of interest. In global methods constraints are used on a scan line or on the whole image[10]. Some techniques produce dense disparity maps, other methods produce sparse disparity maps. Sparse disparity maps methods limit the correspondence search to reliable features in the images such as corners, so it is faster than generating dense disparity maps [14]. However, here more focus has been paid for the speed of the stereo matching and achieving real time performance. Different methods of area based local block matching are used, with changing several parameters such as kernel size and disparity range. Different computational models and platforms are used to achieve real time and fast stereo matching. The first model is the normal sequential CPU platform. The second model uses multi-core processor where the matching algorithm is divided into several threads running simultaneously on different cores to reduce the processing time. The third model utilizes the large numbers of cores found in GPUs to run the matching algorithm very fast. Finally, the heterogenous CPU/GPU model divides the load between the parallel multicore CPU and the parallel GPU system. Many details of the systems implementations are discussed later in the following sections. This paper is organized as follows: section 2 discuss the related work. Section 3 introduces the different stereo matching approaches , and illustrates the approach used in this work. Section 4 illustrates the parallel models using multi-core CPU and GPU. Section 5 gives more details on the heterogenous CPU/GPU model. Section 6 gives the results and finally the conclusion is given in section 7.

II. RELATED WORK

The availability of parallel platforms has encouraged researchers to move from working with traditional sequential algorithms to the parallel algorithms specially when working with computationally heavy problems. Stereo matching algorithm needs high computational power and using parallel platforms is the best choice specially when working with real time systems.

FPGA , GPU , DSP , and multicore-CPUs are widely used to accelerate the time taken by different stereo matching algorithms. Chang and Maruyama have developed a real-time stereo vision system using multi-block matching on GPU[1]. They used gray images and they calculated the pixel matching cost using Normalized Cross-Correlation (NCC). They used GTX1080 Ti NVIDIA GPU and achieved a speed of 72 fps or 1444x960 frame sizes. Jin, Seunghun, et al, have designed and Implemented FPGA based real-time stereo vision system [4]. They used fully pipelined stereo vision system providing a dense disparity image with sub-pixel accuracy. Their system is designed and coded using VHDL and implemented using a Virtex-4 XC4VLX200-10 FPGA from Xilinx. They tested the system using both of 30 fps and 60 fps with a resolution of 640 x 480 using census matching method. Frank, and Thomas have developed a real-time Semi Global Matching Stereo (SGM) vision for road scenes[11]. They realized the cost calculation using a 11x11 Census transform and using Hamming distance for resolution of 1242 x 375 at a disparity range D = 160 with a frame rate of 199 Fps. Hernandez and , Daniel, et al have designed real-time stereo estimation via Semi-Global Matching on the GPU[2]. Their design runs on a Tegra X1 at 42 fps for image resolution of 640×480, using 128 disparity levels, and using 4 path directions for the SGM method. Pauwels, Karl, et al made a comparison between FPGA and GPU devices for the computation of phase-based optical flow, stereo, and local image features[7]. Based on their analysis, they provided suggestions to real-time system designers for selecting the most suitable technology, and for optimizing system development on this platform, for a number of diverse applications. . More work for parallel stereo matching can be found[5][15][16].

III. STEREO MATCHING APPROACHES

In stereo matching, we have two images obtained from different cameras and the problem is to obtain the correspondence between the different pixels in the two images as shown in Fig.1.

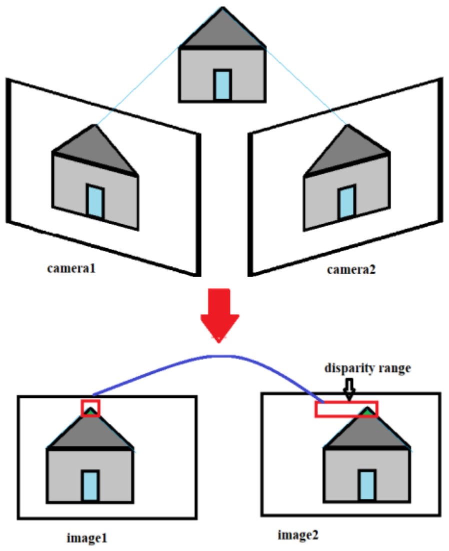

Fig.1 stereo matching and disparity range

Stereo matching can be divided according to cost aggregation into three approaches: Local approaches, semi global approaches and global approaches. According to matching cost computation, stereo matching can be classified into : block matching , Gradient based optimization or feature based matching. In block matching , a maximum match score or minimum error is searched for over a small region , using variants of robust rank metrics or cross-correlation. In Gradient based optimization it is required to minimize a cost function such as SSD*(summation of squared differences)* , SAD *(summation of absolute differences)* ,Hamming of Census ,or NSSD *(Normalized SSD)* over a small region[8][12]. Feature based methods limit the correspondence search to only reliable features in the images. This approach is reliable and fast but generates sparse (i.e not dense) disparity map. The local methods searches for the correspondence for the pixel of interest in a small windows. This technique is faster than global methods but it is not robust for many of variations and occlusions. In the global methods , the disparities of all pixels are decided based on the mutual effect of all pixels. Thus, the global methods achieve lower error rates, but need longer computation time [1][6]. Semi-global matching (SGM) approximates the global methods by solving a one-dimensional minimization problem on several independent paths in the frame[2][9][13]. In this work, the local methods are used to avoid the computational complexity of global and semi-global methods. Different cost functions are used to measure the similarities between pixels in left and right images. SSD , SAD , and NSSD are used and the time is measured for these functions for different platforms and different parameters. Equations 1 to 3 give the mathematical expressions of SSD, normalized SSD , and SAD respectively.

$$C_{SSD} = \sum_{u,v}(I_1(u,v) - (I_2(u+d,v))^2 \qquad (1)$$

$$C_{NSSD} = \sum_{u,v}\left[\frac{I_1(u,v)-\overline{I_1}}{\sqrt{\sum_{u,v}(I_1(u,v)-\overline{I_1})}} - \frac{I_2(u,v)-\overline{I_2}}{\sqrt{\sum_{u,v}(I_2(u,v)-\overline{I_2})}}\right]^2 \qquad (2)$$

$$C_{SAD} = \sum_{u,v}|I_1(u,v) - I_2(u+d,v)| \qquad (3)$$

Where I_1, and I_2 are pixel intensities of image1 and image2 . $\overline{I_1}$ and $\overline{I_2}$ are the mean of the patch intensities of image1 and image2 respectively. Direct implementation of stereo matching algorithm has a computation problem where the same calculations are repeated for adjacent pixels. The calculation of cost function and finding best matches can be accelerated using sliding window approach. Assuming that, the kernel size is *SxS* disparity of pixel (*i,j*) is C(*i,j*) , the calculation time can be highly reduced utilizing the value calculated at pixel (*i-1,j*). Fig.2 shows the calculation of cost function for two adjacent pixels using sliding window. This approach is pointed out in Algorithm 1.

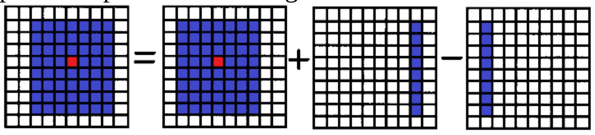

Fig.2 Sliding window fast calculation

Algorithm 1

```
Sliding window Cost function
INPUT: Stereo images (WxH),
 Kernel size (SxS), disparity range D.
OUTPUT: Disparity map. C(x,y,d)
Loop1 : x from 0 to W-D-1
  Loop2 : y from 0 to H-D-1
      Loop3: d from 0 to D-1
        if(x==0 and y==0)
            calculate C(x,y,d)
        if(x>0 and y==0)
             C(x,y,d)=C(x-1,y,d) + new Col. - Old Col.
        if(x==0 and y>0)
            C(x,y,d)=C(x,y-1,d) + new Row. - Old Row.
        if(x>0 and y>0)
                C(x,y,d)=C(x-1,y-1,d) + new Col. - Old Col.
                +new Row -Old Row.
      End  Loop3
    find best match
  End  Loop2
End  Loop1
 Generate disparity map
```

IV. PARALLEL CPU/GPU MODELS

The traditional sequential algorithms became a waste of time and resources, with the appearance of many parallel platforms such as GPU, FPGA , and multi-core processors. When we have a heavy computational problem like stereo matching, the parallel computation is very important. GPUs are parallel devices containing hundreds of processing units called streaming multiprocessors (SMs) [2]. Operations are executed as vector instructions and are highly pipelined in order to accelerate computations and processing. GPUs can be used to solve the complete problem or use it to solve a part of the problem. In the first case, the frames are copied to the memory of the GPU and the whole process is executed using the parallel cores in the GPU. In the second case, there is a workload division between the CPU and the GPU. In this heterogenous computing model the stereo matching process is divided into two parts or tasks executed in parallel: CPU task, and GPU task. The CPU task also can be divided into several parallel threads to utilize the multi-core processors found in all or most advanced processors. Hence we have our models as follows:

1) ***Sequential CPU model***: All instructions and processing are executed in sequential manner and there is no any parallelization. Certainly, It is slow and waste time and resources and can't achieve real time computation of stereo matching.
2) ***Parallel multi-core CPU model***: is a multi-threading model which divides the processing task into several parallel threads running simultaneously on the different cores of CPU. This model reduces the computation time.
3) ***Parallel GPU model***: instead of running few parallel threads , we can run hundreds of parallel threads in this model. This reduces the time of stereo matching more and more, and the frame processing rate arrives the real time utilizing the tremendous number of cores found in GPU.
4) ***Heterogenous CPU/GPU model***: instead of leaving the CPU idle during the GPU processing period, the CPU can work also in completing the stereo matching process. The workload is divided between the CPU and GPU, then it is divided again internally between the different cores exist in each one. This process increases the speed but many issues concerning the partitioning and memory management must be solved and designed efficiently.

All these models are tested, and the processing times are analysed and compared. The fourth model with more details is discussed in the next section.

V. PROPOSED HETEROGENOUS CPU/GPU

There are two hardware systems for the heterogenous CPU/GPU platform. The first system where the CPU and GPU are in different chips and each one has its own memory. In the second system , CPU and GPU exist in a single chip and the memory are available to the both. Working with each system affects the design and the used technique from the partitioning point of view. Also, choosing course-grained or fine-grained methods depends on the available hardware device model. Fig.3 shows how stereo matching processing can be divided into two parts running in parallel on CPU and GPU processors.

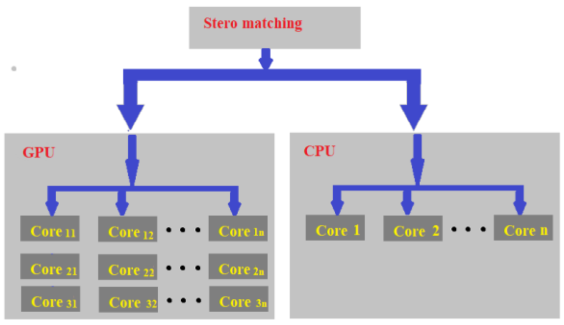

Fig.3 Heterogenous parallel CPU/GPU

The design of the partitioning has many alternatives and design parameters. The partitioning can be static or dynamic as shown in fig.4 .

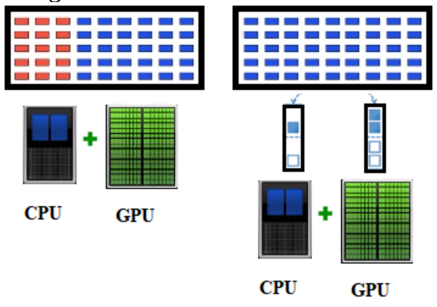

Fig.4 static and dynamic partitioning

The static partitioning is calculated before runtime while dynamic partitioning responds to runtime performance and can be changed at run time. Another issue for partitioning is the data parallelism and task parallelism partitioning. In data parallelism we divide the image into two parts and the GPU runs the complete process for its part of data and the same for the CPU for its data.

In task parallelism some tasks such as cost computation are executed on GPU for the compete image and another tasks are executed on the CPU also for the complete image. In the proposed system, static partitioning and data parallel models are used. The image is divided into two parts (not equal) and CPU and GPU runs in parallel to compute the matching for the assigned data. The available hardware system has separate CPU and GPU processors in different chips with different separated memories. Dynamic partition is inefficient in this case while moving data between memories consumes the time and adds more latency. The same for task parallelism model, where moving data between different memories are required in this model. To avoid this problem , the dynamic partitioning model and the task parallel model are not used. For the best utilization of available resources, a good design of the partition is required to avoid idle states. Run time code profiling is used for each device to get better understand of stereo matching time complexity and requirements. This code profiling, helps dividing the workload properly between CPU and GPU and avoiding the processor idle states. Also this proposed model avoid copying data between device memory and host memory at run time, which consumes more time and reduces the speed and performance.

VI. EXPERIMENTAL SETUP AND RESULTS

The system is implemented using a computer with Nvidia GeForce 940 MX and intel CPU core i5 2.5 GHz processor and 6GB RAM. CUDA (Compute Unified Device Architecture) API is used in developing the software and algorithms of stereo matching. Equations 1,2 and 3 are used to calculate the SSD, normalized SSD and SAD respectively. The kernel size are chosen to be 7x7 , 11x11 , and 15x15. The disparity range are varied between 20 and 100 pixels. Different image sizes are used such as 320 x 240, and 640 x 480. The four models illustrated in section 4 are used to test the system. Tables 1 to 4 shows the results.

Table 1 Sequential CPU

Image Size	Kernel Size	Disparity range	Cost Function	Time
640 x 480	15 x 15	20	SAD	15 ms
640 x 480	15 x 15	50	SAD	19 ms
640 x 480	15 x 15	100	SAD	33 ms
640 x 480	15 x 15	20	SSD	16 ms
640 x 480	15 x 15	50	SSD	19.5 ms
640 x 480	15 x 15	100	SSD	34 ms
640 x 480	15 x 15	20	NSSD	550 ms
640 x 480	15 x 15	50	NSSD	690 ms
640 x 480	15 x 15	100	NSSD	940 ms
640 x 480	11 x 11	20	SSD	13 ms
640 x 480	11 x 11	50	SSD	16 ms
640 x 480	11 x 11	100	SSD	25 ms
640 x 480	7 x 7	20	SAD	11 ms
320 x 240	15 x 15	20	SSD	9 ms
320 x 240	15 x 15	50	SSD	11 ms
320 x 240	15 x 15	100	SSD	14 ms

Table 2 Multi-Core CPU

Image Size	Kernel Size	Disparity range	Cost Function	Time
640 x 480	15 x 15	20	SAD	11 ms
640 x 480	15 x 15	50	SAD	14 ms
640 x 480	15 x 15	100	SAD	26 ms
640 x 480	15 x 15	20	SSD	11 ms
640 x 480	15 x 15	50	SSD	14 ms
640 x 480	15 x 15	100	SSD	27 ms
640 x 480	11 x 11	20	SSD	10 ms
640 x 480	11 x 11	50	SSD	11 ms
640 x 480	11 x 11	100	SSD	20 ms
640 x 480	7 x 7	20	SAD	8 ms
320 x 240	15 x 15	20	SSD	7 ms
320 x 240	15 x 15	50	SSD	8 ms
320 x 240	15 x 15	100	SSD	11 ms

Table 3 Parallel GPU

Image Size	Kernel Size	Disparity range	Algorithm	Time
640 x 480	15 x 15	20	SAD	4 ms
640 x 480	15 x 15	50	SAD	12 ms
640 x 480	15 x 15	100	SAD	19 ms
640 x 480	15 x 15	20	SSD	5 ms
640 x 480	15 x 15	50	SSD	13 ms
640 x 480	15 x 15	100	SSD	19 ms
640 x 480	11 x 11	20	SSD	3.5 ms
640 x 480	11 x 11	50	SSD	10 ms
640 x 480	11 x 11	100	SSD	16 ms
640 x 480	7 x 7	20	SAD	3 ms
320 x 240	15 x 15	20	SSD	2.7 ms
320 x 240	15 x 15	50	SSD	7 ms
320 x 240	15 x 15	100	SSD	13 ms

Table 4 Heterogenous CPU/GPU

Image Size	Kernel Size	Disparity range	Algorithm	Time
640 x 480	15 x 15	20	SAD	3 ms
640 x 480	15 x 15	50	SAD	8 ms
640 x 480	15 x 15	100	SAD	13 ms
640 x 480	15 x 15	20	SSD	4 ms
640 x 480	15 x 15	50	SSD	8.4 ms
640 x 480	15 x 15	100	SSD	14.2 ms
640 x 480	11 x 11	20	SSD	2.8 ms
640 x 480	11 x 11	50	SSD	7 ms
640 x 480	11 x 11	100	SSD	11 ms
640 x 480	7 x 7	20	SAD	2.3 ms
320 x 240	15 x 15	20	SSD	2.7 ms
320 x 240	15 x 15	50	SSD	4.3 ms
320 x 240	15 x 15	100	SSD	7.6 ms

From these results it is found that, sequential CPU model is slow and can't achieve real time performance. In Table.1, it is found that, there is no large difference between SSD and SAD. However, NSSD is heavy and complex and it is difficult to use sliding windows with this cost function. NSSD is not included in parallel tables due its complexity. Moving to parallel multi-core processors a step of performance improvement can be achieved. More improvements can be found when working with the parallel GPU system which contains 384 parallel cores as shown in table.3. The time in table 3 is the calculation time only. There is another small communication time for transferring data from host memory to device memory, which is not dependent on the algorithm parameters. In heterogenous CPU/GPU system, the computation powers of CPU and GPU are utilized, and the fastest speed can be obtained as shown in table 4. The results here are promising and the proposed model is suitable for real time applications such as self-driving cars and robot navigations

VII. CONCLUSION

Stereo matching is a heavy computational problem needs the utilization of parallel hardware capabilities. Here, heterogenous parallel CPU/GPU model is relied upon as a fast tool that utilizes the available resources efficiently.

This model allowed the computation of the stereo matching in real time with high resolution. Heterogenous CPU/GPU model achieved a performance of more than 100 fps for image size of 640 x 480 which is a fast and suitable for many practical applications. Sliding window technique improves the speed by avoiding redoing the calculation of adjacent pixels. Static partitioning based on code profiling helps designing efficient workload partitioning between CPU and GPU and avoid data communication.

REFERENCES

1. Chang, Qiong, and Tsutomu Maruyama. "Real-time stereo vision system: a multi-block matching on GPU." *IEEE Access* 6: 42030-42046, 2018.
2. Hernandez-Juarez, Daniel, et al. "Embedded real-time stereo estimation via semi-global matching on the GPU." *Procedia Computer Science* 80: 143-153, 2016.
3. Hestness, Joel, Stephen W. Keckler, and David A. Wood. "GPU computing pipeline inefficiencies and optimization opportunities in heterogeneous CPU-GPU processors." 2015 IEEE International Symposium on Workload Characterization. IEEE, 2015.
4. Jin, Seunghun, et al. "FPGA design and implementation of a real-time stereo vision system." *IEEE transactions on circuits and systems for video technology* 20.1: 15-26, 2009.
5. Kelly, Philip, Noel E. O'Connor, and Alan F. Smeaton. "A framework for evaluating stereo-based pedestrian detection techniques." *IEEE Transactions on Circuits and Systems for Video Technology* 18.8 : 1163-1167, 2008.
6. Na, Intae, Sang Hyun Han, and Hong Jeong. "Stereo-based road obstacle detection and tracking." *13th International Conference on Advanced Communication Technology (ICACT2011)*. IEEE, 2011.
7. Pauwels, Karl, et al. "A comparison of FPGA and GPU for real-time phase-based optical flow, stereo, and local image features." *IEEE Transactions on Computers* 61.7 : 999-1012 , 2011.
8. Philip Kelly, Noel E. O'Connor, "A Framework for Evaluating Stereo-Based Pedestrian Detection Techniques," IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY, VOL. 18, NO. 8, AUGUST 2008.
9. Samadi, Masoud, Mohd Fauzi Othman, and Shamsudin HM Amin. "Stereo vision based robots: Fast and robust obstacle detection method." *2013 9th Asian Control Conference (ASCC)*. IEEE, 2013.
10. Sappa, Angel Domingo, et al. "An efficient approach to onboard stereo vision system pose estimation." *IEEE Transactions on Intelligent Transportation Systems* 9.3: 476-490, 2008.
11. Schumacher, Frank, and Thomas Greiner. "Matching cost computation algorithm and high speed FPGA architecture for high quality real-time semi global matching stereo vision for road scenes." *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2014
12. Shima, Yoshihiro. "Inter-vehicle distance detection based on keypoint matching for stereo images." *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. IEEE, 2017.
13. Sidhu, Harkanwal Singh, et al. "A robust area based disparity estimation technique for stereo vision applications." *2011 International Conference on Image Information Processing*. IEEE, 2011
14. Tippetts, Beau J., et al. "Dense disparity real-time stereo vision algorithm for resource-limited systems." *IEEE Transactions on Circuits and Systems for Video Technology* 21.10 : 1547-1555, 2011.
15. Wang, Yanhua, et al. "Performance optimization for CPU-GPU heterogeneous parallel system." *2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2016
16. Yu, Wei, Tsuhan Chen, and James C. Hoe. "Real time stereo vision using exponential step cost aggregation on GPU." *2009 16th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2009.