Analysis of Real-Time Stereo Vision Algorithms On GPU

Ratheesh Kalarot*†, John Morris*, David Berry† and James Dunning†
*Department of Computer Science, University of Auckland, New Zealand
†Control Vision, Auckland, New Zealand

***Abstract*—Inferring 3D depth map of scene using passive stereo vision has got a lot of attention and focus of research for past few decades. Despite availability of dozens of stereo correspondence algorithms , choice of commercially viable real-time stereo solutions are limited and their trade-offs are less studied in the literature. This partially because of the ill-posed nature of stereo reconstruction problem and heavy computation requirements of a practically usable stereo configuration. With the new developments in general purpose graphic computing such as *Compute Unified Device Architecture* (CUDA) on graphic processing units (GPU) has become a commercially viable and scalable platform to support various stereo algorithms to run real-time. In this paper we analyse 5 potential real-time stereo algorithms , implemented on GPU using CUDA and compare their performance.**

I. Introduction

Passive stereo vision is an attractive solution for finding real-time dense depth information of a scene for various reasons . Unlike active depth inferring solutions such as LIDAR ,*Microsoft-Kinect* , stereo vision has no actives beams send and hence more acceptable to applications were objects are sensitive and range limited to such active beams. It is also much cheaper compared its compared to its counterparts such as *Velodyne* sensors. Despite these advantages , stereo vision systems has not yet impressed as industrial solution to real-time depth sensing applications. There exist a trade-off existing among cost , fastness and accuracy on stereo solutions.In recent times, with the availability of low cost , high capable GPUs and efficient exploitation of parallelism in stereo algorithms, is a notable improvement in all the tree factors .

In this paper, we analyse real-time adapted implementation 5 stereo algorithms on GPU. Details of these algorithms are given in section III.

A. Related work

Many GPU based stereo implementations are reported in literature. Earlier GPU based stereo implementations were realised by indirectly using the specialised functional units (*e.g.* vertex shader and pixel shader) due to the lack of genera purpose computing platforms [1], [2].

Simple correlation based block matching and dynamic programming based stereo algorithms are easier to get implemented on GPU because of their relative easiness of mapping to the hardware and linear relations to computing resource requirement with regard to number of disparity levels. We had implemented Symetric dynamic programming based stereo system targeting high resolution stereo [**?**], [3] and their multi resolution variations [4].Dynamic programming stereo for low stereo configuration were also reported in literature [5], [6], [7].

Many SGM implementations on GPU using are also reported in literature. A full implementation SGM using mutual information based cost is done by Ernst and Hirschmuller [8]. SGM with BT based cost is reported by Gibson *et al.* [9]. many other adaptations of SGM are also reported [10], [11].

Due to non-linear memory storage requirements and inherently slow memory access patterns , Belief propagation and graph cut based stereo algorithms are less preferred for real-time stereo implementations. Yet CUDA based implementations are recently , widely reported [12], [13], [14]

Despite all these implementations , choice for high definition , high fidelity stereo(Mega pixels images) with large disparity (¿100)desired by many applications are very limited.There has been little attention analysing such options , helping some one to choose from.In this paper , we put together the available options and compare and contrast their strengths and weaknesses.

II. Compute Unified Device Architecture (CUDA)

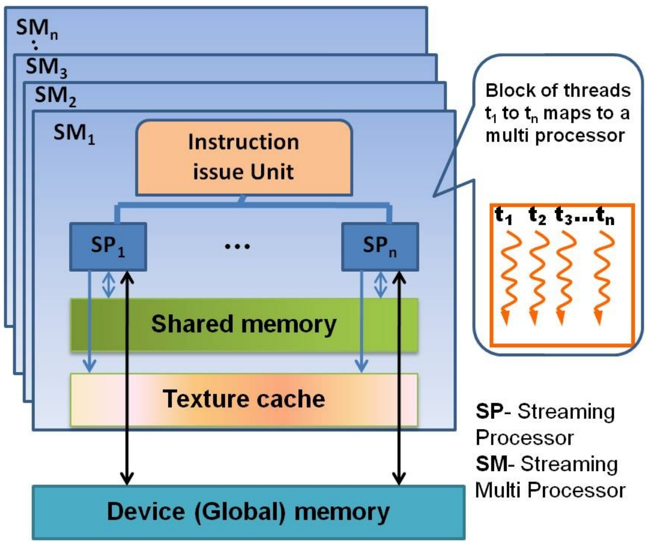


Fig. 1. The CUDA architecture.

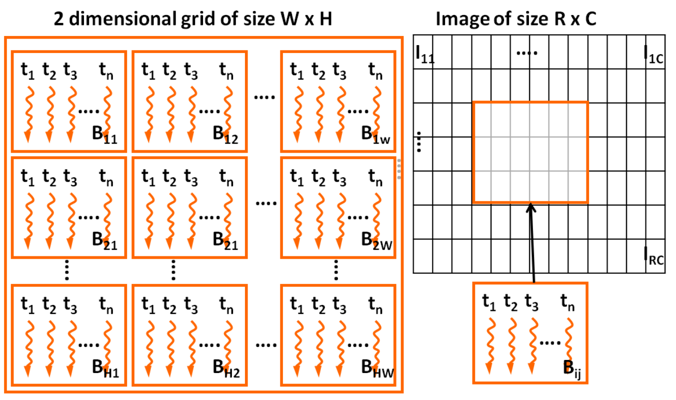


Fig. 2. Typical mapping of block matching algorithms on GPU.Threads blocks are allocated as a 2D grid and each thread block computes disparity for 2D block of the image

Compute Unified Device Architecture (CUDA) [15] is a unified general purpose parallel computing GPU architecture, an initiative by Nvidia Corporation.This architecture that offers a flexible programming with minimal extensions to the common high-level C programming language. A CUDA enabled GPU (see Figure 1) consists of many Streaming Multiprocessors ($\mathrm{SM}_1..\mathrm{SM}_n$), each multiprocessor being a set of Streaming Processors ($\mathrm{SP}_1..\mathrm{SP}_n$) that share a fast access local memory referred to as the shared memory. All the multiprocessors can also access a global memory called the device memory. A multiprocessor has a common instruction issue unit and a very light weight thread scheduling mechanism to switch threads quickly when needed, eg during an I/O wait.A number of threads $t_1..t_n$ can be assigned to a streaming processor to form a thread block . The block maps physically to this processor although the number of threads in a block can exceed the number of the streaming processors. In such a case the threads are split into so-called warps, such that only one warp is active at a time. Threads in an active warp will execute the instructions in a parallel step locked manner, but branches in the code are executed sequentially. Threads agreeing on the branch are executed in parallel while others wait until their corresponding branch is chosen for execution.
Figure 2 shows an example of stereo computation mapping to GPU. A block of threads containing thread $t_1..t_n$ is allocated to process a 2D block of image.
In this study , we chose a commercial high end graphics card *GeForce GTX 480* - widely used in desk-top PCs, for our analysis. Throughput results on a low end mobile graphics card *GeForce GT 540M*, is also for comparison purpose.Figure 3 shows specifications of both cards.

III. Stereo Algorithms Overview

In this section we give a brief description of 5 stereo implementations that we analyse. All the algorithms are fully realised on GPU using CUDA.First two implementations-symmetric dynamic programming stereo(SDPS) and , semi global matching(SGM) are done ourself and will be discussed in detail. Other three implementations - block matching (BM),belief propagation (BP) and belief propagation constant space (BP_CS) are taken as is from OpenCv (see [16]) , the most popular open source library for computer vision. We also include semi global block matching (SGBM) a block matching based semi global stereo algorithm , also available in OpenCv, as a candidate real-time algorithm on CPU , to give perspective.

Features	GeForce GTX 480	GeForce GT 540M
CUDA Cores	480	96
Processor Clock	1401 MHz	1344 MHz
Memory Clock	1848 MHz	900 MHz
Memory Size/Type	1.5 GB GDDR5	1 GB DDR3
Memory Bandwidth	177.4 GB/sec	28.8 GB/sec

Fig. 3. Properties of graphics cards used in this study.

A. *Symmetric dynamic programming stereo (SDPS)*

A detailed description of SDPS and its GPU realisation can be found elsewhere (see[17], [3]. SDPS essentially is a dynamic programming based stereo algorithm find a low cost path of stereo matching along a scan line. Its key feature is that, rather than directly match the left image to the right image or vice versa, the SDPS reconstructs a Cyclopæan image˜that would be seen by a virtual camera midway between the optical centres of the two real cameras and matches it simultaneously to both the stereo images. A careful exploitation of this feature will allow half the number of disparities of a pixel in scan line to be calculated in parallel , a highly desirable fine grain parallelism for GPUs.
As a typical dynamic programming algorithm, the SDPS accumulates costs for potentially optimal paths through the disparity search space (DSS) for a single Cyclopæan scan-line.The matching score is optimised only along a single direction . Both the storage needed and constraints on implementations are highlighted by outlined below basic computations for a simplified SDPS that matches the Cyclopæan image signals (grey values or colours) directly to the left and right image signals. For each Cyclopæan scan-line of the length of w pixels, a potentially optimal predecessor for each DSS point is stored in the predecessor array of the size $O(w\Delta)$, where Δ is the range of disparities.
Let x and d denote an x-coordinate and an x-disparity for a point in a scan-line-wise DSS. Let $D_{x,d}$ be a signal dissimilarity score for the corresponding binocularly visible pixels at positions $x + \frac{d}{2}$ and $x - \frac{d}{2}$ in the relevant scan-lines across the left and right stereo images, respectively. Provided that a single continuous surface is reconstructed, each DSS point (x, d) can have one of the three visibility states: $s \in \{\mathrm{B}, \mathrm{M_L}, \mathrm{M_R}\}$ where B stands for a binocularly visible point and $\mathrm{M_L}$ and $\mathrm{M_R}$ indicate a point that is visible only monocularly, on the left or right image, respectively. Let D_o be a constant cost associated with a partially occluded or unmatched surface point, this cost also serving as a smoothing constraint along the scan-line.

Potentially optimal costs, $C_{x,d,s}$, accumulated by the SDPS and predecessors, $\pi_{x,d,s} \equiv (x', d', s')$, for each DSS position (x, d, s) are as follows:

$$C_{x,d,\mathrm{B}} = D_{x,d} + \min\{C_{x-1,d-1,\mathrm{ML}}, C_{x-2,d,\mathrm{B}}, C_{x-2,d,\mathrm{MR}})\} \tag{1}$$

$$\pi_{x,d,\mathrm{B}} = \arg\min\{C_{x-1,d-1,\mathrm{M_L}}, C_{x-2,d,\mathrm{B}}, C_{x-2,d,\mathrm{M_R}}\} \tag{2}$$

$$C_{x,d,\mathrm{M_L}} = D_{\mathrm{o}} + \min\{C_{x-1,d-1,\mathrm{M_L}}, C_{x-2,d,\mathrm{B}}, C_{x-2,d,\mathrm{M_R}}\} \tag{3}$$

$$\pi_{x,d,\mathrm{M_L}} = \arg\min\{C_{x-1,d-1,\mathrm{M_L}}, C_{x-2,d,\mathrm{B}}, C_{x-2,d,\mathrm{M_R}}\} \tag{4}$$

$$C_{x,d,\mathrm{M_R}} = D_{\mathrm{o}} + \min\{C_{x-1,d+1,\mathrm{B}}, C_{x-1,d+1,\mathrm{M_R}}\} \tag{5}$$

$$\pi_{x,d,\mathrm{M_R}} = \arg\min\{C_{x-1,d+1,\mathrm{B}}, C_{x-1,d+1,\mathrm{M_R}}\} \tag{6}$$

Visibility constraints on transitions between the single-surface DSS points considerably reduce the data needed to be stored in the predecessor array [17] differentiating it from other approaches.It also helps reduce the number of costly global memory access in our implementation. Each goal disparity profile is reconstructed by back-tracking of a sequence of transitions only between the potentially optimal visibility states. In this version , we have used 32 bit pixel depth (compared our 8 bit implementation in [**?**]) . This in allows high depth (eg 24 bit RGB, 16 bit gray etc) to b processed natively without any performance penalty and .Disparities are also stored in 32 bit , removing any practical limits on number of disparity levels that can be preserved.In this experiment constant occlusion cost D_{o} was ste to 20.

B. Semi global matching (SGM)

Original Semi global matching (SGM) algorithm proposed by Hirschmuller [18]uses a mutual information based pixel matching cost and proceeds with aggregation of matching cost as in multiple directions , similar to dynamic programming approach explained above. The details can be found else where. It is essentially minimising the following energy

$$E(d) = \sum_{p} (C(p, D_p)) + \sum_{q\varepsilon N_p} P_1 T[|D_p - D_q| = 1] + \sum_{q\varepsilon N_p} P_2 T[|D_p - D_q| > 1] \tag{7}$$

where D is the number of disparity levels , C the cost functions and Np is the neighbourhood of point p in al directions. Function T returns the weight of penalty cost$P_1$and P_2 evaluating respective conditions.$D_p$and D_q represents the disparity of current evaluating point p and neighbouring point q.

Unlike traditional dynamic programming . SGM requires finding minimum cost along a column of all possible disparity levels . This reduction is a major parallel computing hurdle of the SGM . We use parallel radix sorting to engage maximum computing resource .

In original implementation the cost function C is based on mutual information but we prefer a simple Sum of absolute difference (SAD) cost function for performance , in our implementation. We also limit the number of scan line directions to 4 as the diagonal directions cause a lot of non-coalesced memory accesses patterns in GPU. Left to right consistency check for occlusion is detection, proposed in original version is also discarded.

As in our SDPS implementation , SGM is also implemented in 32 bit allowing potential hight pixel depth data matching. Algorithm parameters P_1 and P_2 are kept 5 and 10 respectively.

C. Block matching (BM)

Block Matching stereo method is a local dense stereo matching method [19] where a where cost of matching found by correlation of small in window in left image over the suspected matching place in the right image. Disparity value of a pixel in left image is decided by the winner take all (WTA) approach by with disparity with least cost is assigned declared as the disparity of the pixel. In a parallel processing view this the ideal computation structure for fine grain parallelism where cost for different possible disparities for each individual pixel can be evaluated in parallel . Fig. 2 shows the mapping of block matching algorithm to GPU in this OpenCv implementation. Each thread block is operating on a small block of the image pixels (typically 128x21). The cost for each disparity is computed sequentially in a thread . This mapping allows save some computation in window cost calculating by employing a moving column of cost along the scan line. Cost used in this particular implementation is sum of squared difference (SSD). It also implements some preprocessing (sobel operation) and post processing (speckle filtering) steps . Main disadvantage of any such local method is that it perform badly at texture less regions . To avoid this , one need to keep a high window size, on the cost of blurring disparity map and increased computation time. . In our analysis window size is kept 3 while rest of the parameters were left untouched.

D. Belief propagation (BP, OpenCv)

A detailed description of belief propagation algorithm can be found elsewhere [20]. Belief propagation for correspondence is best explained as labelling problem assigning disparity labels to each pixel to minimise an energy function. Let the set of pixels in image P need to be mapped to a set of disparities L the energy function is

$$E(f) = \sum\limits^{p\varepsilon P} D_p(f_p) + \sum\limits^{(p,q)\varepsilon A} V(f_p - f_q)) \tag{8}$$

First term is the data cost or dissimilarity cost ($Data\_Cost$) of each pixel and the second term is the discontinuity cost($Disc\_Cost$). This can be performed hierarchically to reduce computation over image size. In OpenCv implementation data cost is calculated as a weighted intensity difference which is thresholded to a max value.

$$Data\_Cost = data\_weight \cdot \min(|I_2 - I_1|, max\_data\_term) \tag{9}$$

The discontinuity term or smoothing term is calculated a thresholded discontinuity of single jump in disparity value between neighboring pixels.

$$Disc\_Cost = \min(disc\_single\_jump \cdot |f_1 - f_2|, max\_disc\_term) \tag{10}$$

While the belief propagation algorithm algorithm has high level of parallelism passing messages independent , it needs a lot of memory read and writes storage for passing messages . The storage space required for the above implementation is given below.

$$MessageStorage = width\_step \cdot height \cdot ndisp \cdot 4 \cdot (1 + 0.25) \tag{11}$$

$$DataCostStorage = width\_step \cdot height \cdot ndisp \cdot (1 + 0.25 + \cdots + \frac{1}{4^{levels}}) \tag{12}$$

E. Belief propagation- constant space (BP_CS , OpenCv)

As discussed in the earlier section belief propagation algorithm proves to be space greedy. As the order of memory requirement is multiple of size of the image and number of disparity , scalability of OpenCv BP is very limited. For a minimum stereo setting of 512*512 image with 64 disparities , the memory required will be in beyond the capacity of commercial graphic cards. In belief propagation constant space (OpenCV BP CS) implementation [21] , storage requirement is reduced and is made independent of number of disparities by trading amount computation [yang 2010].BP CS recomputes data term at each level it also applies the course to fine computation in disparity levels along with the spatial hierarchical computation done in normal BP.It also results in reduced accuracy around depth discontinuity.

F. Semi global matching (SGBM ,OpenCv)

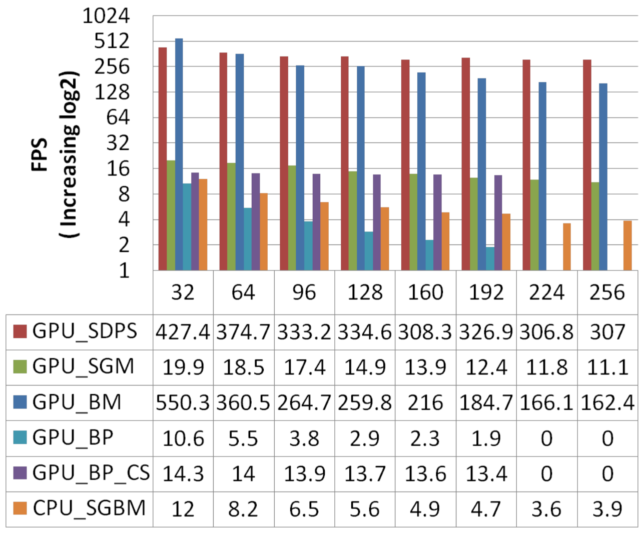
For comparison sake, we also include close to real-time performing algorithm on CPU.OpenCv has an efficient implementation of semi global matching algorithm which they call semi global block matching (SGBM) . On lower stereo settings SGBM can perform near real-time on CPU.In this particular implementation of SGM ,the matching cost is derived by correlation of small blocks rather than individual pixels.Then number of passes of is limited to 5 rather than traditional 8 pass algorithm. Default configuration of this implementation has few filtering and post processing steps which we have avoided for a fair comparison.The window size is also kept low to reduce computation.

IV. Performance analysis and comparison

In this section we analyse and compare performance of the above algorithms on GPU. Performance results of SGBM on CPU is also given for contrasting the results. The characteristic of each algorithm and their effective mapping on GPU is also discussed.

A. Speed up

Figure 4 shows frame per second (fps) achieved for different disparity levels for different algorithms on a low image resolution of 512x512 pixels. It is clear the GPU_SDPS ans GPU_BM has very high frame rate compared to other algorithms as expected (please note that the frame rate is shown in exponential scale). This partially because of the low number of computation requirements in both algorithm and really efficient mapping to CPU resources. in GPU_BM matching Figure 2 each *Thread Block is matched a small 2D grid in image, hence number of threads in each block are independent of number of of disparity levels. In SDPS each Thread Block is mapped a scan line in an image and number of threads in a block is half the number of disparity levels. It is interesting to note that , for higher number of disparities , SDPS performs better because of the efficient use of the number of threads available in each multiprocessor.*

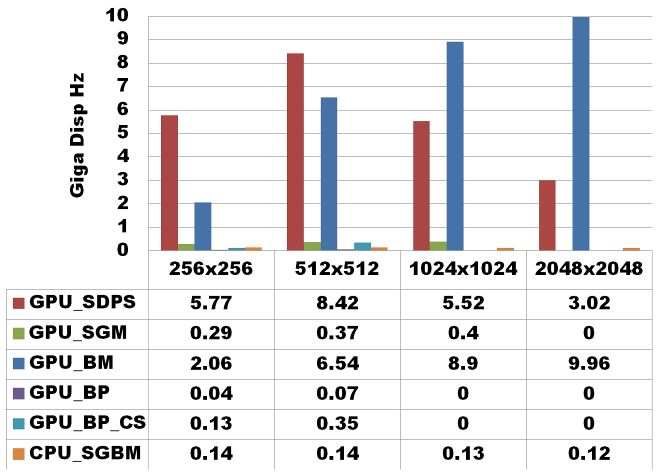


	32	64	96	128	160	192	224	256
🟥 GPU_SDPS	427.4	374.7	333.2	334.6	308.3	326.9	306.8	307
🟩 GPU_SGM	19.9	18.5	17.4	14.9	13.9	12.4	11.8	11.1
🟦 GPU_BM	550.3	360.5	264.7	259.8	216	184.7	166.1	162.4
🟦 GPU_BP	10.6	5.5	3.8	2.9	2.3	1.9	0	0
🟪 GPU_BP_CS	14.3	14	13.9	13.7	13.6	13.4	0	0
🟧 CPU_SGBM	12	8.2	6.5	5.6	4.9	4.7	3.6	3.9

Fig. 4. Fps Vs number of number of disparity levels (on X axis) for 512x512 resolution image running on GeForce GTX 480 GPU

B. Throughput

It is interesting to note the actual throughput in terms number of disparities processed in time of various algorithms . We had proposed measure of $GigaDispHz$ [?] which simply measures the number of calculation performed by each algorithm in second in terms of evaluation a billion disparity options. It should be noted that , for a high definition real-time stereo system , through put should be above $GigaDispHz$. Figure 5 shows the $GigaDispHz$ of various algorithms on a low stereo configuration 512x512 image size for various disparity levels. SDPS and OPENCVBM has leaner dependency on the number of disparities levels. On other hand belief propagation and other algorithms fail to scale because of the exponential growth of computation requirement with leaner increase in number of disparity levels. SDPS achieves its best throughput on 256 disparity levels as a multiprocessor gets 128 threads to actively utilise all resources hiding memory fetch latencies.



	256x256	512x512	1024x1024	2048x2048
GPU_SDPS	5.77	8.42	5.52	3.02
GPU_SGM	0.29	0.37	0.4	0
GPU_BM	2.06	6.54	8.9	9.96
GPU_BP	0.04	0.07	0	0
GPU_BP_CS	0.13	0.35	0	0
CPU_SGBM	0.14	0.14	0.13	0.12

Fig. 5. GigDispHz vs number of disparity levels.

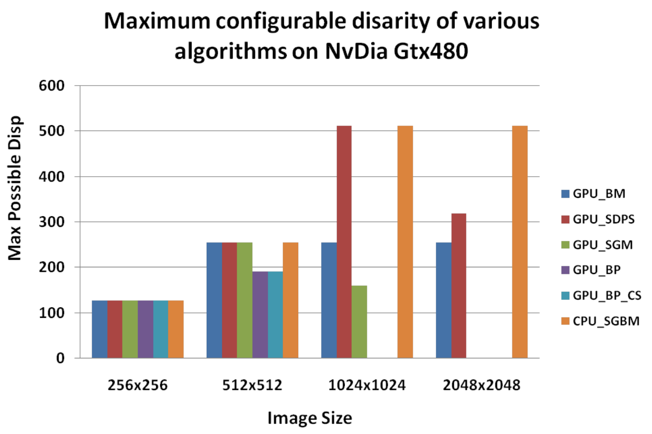
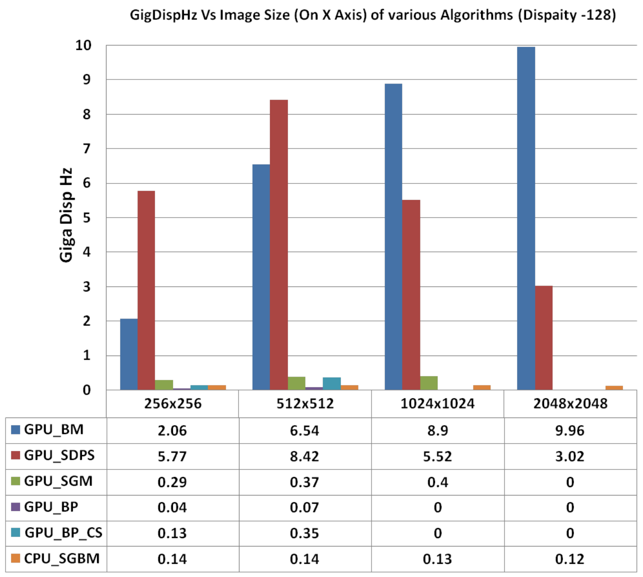


Fig. 7. GigDispHz vs number of disparity levels.

Figure 6 shows throughput of various algorithm on different image size for a fixed disparity range of 128. For a standard high definition stereo configuration of 1024x1024 image size , BM, SDPS and SGM achieves real-time performance. On 2048x2048 resolution , BM reaches its maximum throughput while SDPS performance is reduced because of its poor thread occupancy. This is to be attributed the increased usage of shared memory for a lengthier scan-line.



	256x256	512x512	1024x1024	2048x2048
GPU_BM	2.06	6.54	8.9	9.96
GPU_SDPS	5.77	8.42	5.52	3.02
GPU_SGM	0.29	0.37	0.4	0
GPU_BP	0.04	0.07	0	0
GPU_BP_CS	0.13	0.35	0	0
CPU_SGBM	0.14	0.14	0.13	0.12

Fig. 6. GigDispHz vs image size.

C. Scalability and configuration limits

Figure 7 shows

D. Accuracy

Real-time stereo algorithms trades matching accuracy for speeding benefits. We have tried to keep the parameters of these algorithm to their default values and avoid any kind of post processing , whenever possible We have provided the Middlebury [19] images as well as on our own high resolution synthetic data set FATBOY [22]. Figure 9 shows the root mean square error and mismatch (bad pixels) percentage for various data sets on each algorithm. However we feel that one default configuration for all data set is not indicative error measure of the algorithms, as the practical applications will adapt the parameters to their conditions and each the section of algorithm will depend on the specific condition. In general SDPS has a clear advantage on accuracy over other algorithms. SGM can perform better than SDPS with more saline paths and addition of right to left scan line consistency. Belief propagation also look good on many data conditions but its inability to scale will make it a poor choice. Figure 8 shows the colour coded results of on these data set along with few real world results (CITR1, CITR2).

V. Conclusion

We have analysed and compared candidates for possible real-time algorithms on GPU. While the comparison provide a notion of real-time possibilities of different algorithms , it is unfair to compare the accuracy of each of them naively. There are tedious trade-offs and variants for each algorithm exhibiting different performance indices. What we have tried to do here is compare the computational performance of real-time focussed implementation of algorithms , avoiding any post processing and pre processing steps rather than qualitative judgement

SDPS , BM and SGM emerges to be candidate for high-resolution stereo because of their scalability. BM has a poor matching performance on texture less images. SDPS suffers from streaking artefact. A full version of SGM can reach comparatively better matching accuracy for high performance penalty. As shown in our tables, real-time SGM falls in the similar class of accuracy of top real-time algorithms.

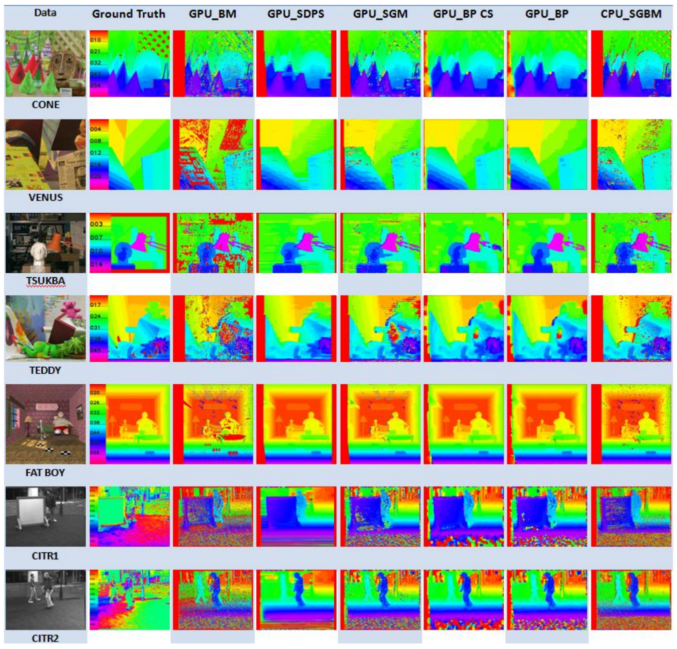
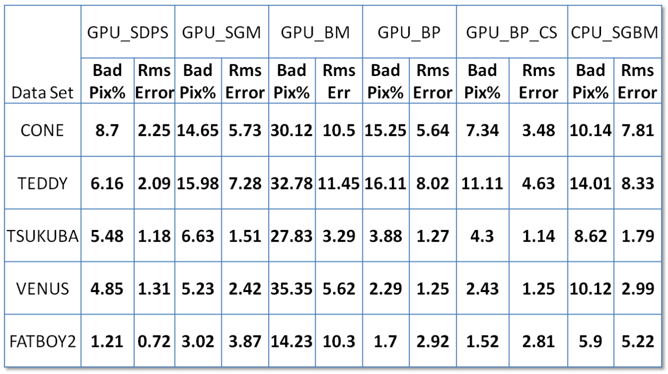


Fig. 8. GigDispHz vs number of disparity levels.



	GPU_SDPS		GPU_SGM		GPU_BM		GPU_BP		GPU_BP_CS		CPU_SGBM	
Data Set	Bad Pix%	Rms Error	Bad Pix%	Rms Error	Bad Pix%	Rms Err	Bad Pix%	Rms Error	Bad Pix%	Rms Error	Bad Pix%	Rms Error
CONE	**8.7**	**2.25**	**14.65**	**5.73**	**30.12**	**10.5**	**15.25**	**5.64**	**7.34**	**3.48**	**10.14**	**7.81**
TEDDY	**6.16**	**2.09**	**15.98**	**7.28**	**32.78**	**11.45**	**16.11**	**8.02**	**11.11**	**4.63**	**14.01**	**8.33**
TSUKUBA	**5.48**	**1.18**	**6.63**	**1.51**	**27.83**	**3.29**	**3.88**	**1.27**	**4.3**	**1.14**	**8.62**	**1.79**
VENUS	**4.85**	**1.31**	**5.23**	**2.42**	**35.35**	**5.62**	**2.29**	**1.25**	**2.43**	**1.25**	**10.12**	**2.99**
FATBOY2	**1.21**	**0.72**	**3.02**	**3.87**	**14.23**	**10.3**	**1.7**	**2.92**	**1.52**	**2.81**	**5.9**	**5.22**

Fig. 9. Percentage of bad pixels for a threshold of 2 and root mean square error for different data sets

References

[1] Q. X. Yang, L. Wang, and R. G. Yang, *"Real-time global stereo matching using hierarchical belief propagation," in* Proc. British Machine Vision Conf. (BMVC 2006)*, 2006, p. III:989.*

[2] J. Woetzel and R. Koch, *"Multi-camera real-time depth estimation with discontinuity handling on PC graphics hardware," in* Proc. IAPR Int. Conf. on Pattern Recognition (ICPR 2004)*, vol. 1, 2004, pp. 741–744.*

[3] R. Kalarot and J. Morris, *"Comparison of FPGA and GPU implementations of real-time stereo vision," in* Proc. 6th IEEE Workshop on Embedded Computer Vision*, 2010, pp. 9–15.*

[4] R. Kalarot, J. Morris, and G. Gimel'farb, *"Performance analysis of multi-resolution symmetric dynamic programming stereo on GPU," in* 25th International Conference Image and Vision Computing New Zealand (IVCNZ) 2010*, 2010.*

[5] L. Wang, M. Liao, M. L. Gong, R. G. Yang, and D. Nister, *"High-quality real-time stereo using adaptive cost aggregation and dynamic programming," in* Proc. Int. Symposium on 3D Data Processing, Visualization and Transmission*, 2006, pp. 798–805.*

[6] M. L. Gong and Y. H. Yang, *"Real-time stereo matching using orthogonal reliability-based dynamic programming,"* IEEE Trans. on Image Processing*, vol. 16, no. 3, pp. 879–884, 2007.*

[7] S. Park and H. Jeong, *"Real-time stereo vision fpga chip with low error rate," in* Proc. Int. Conf. on Multimedia and Ubiquitous Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 751–756.*

[8] I. Ernst and H. Hirschmuller, *"Mutual information based semi-global stereo matching on the GPU," in* Advances in Visual Computing*, 2008, pp. I: 228–239.*

[9] J. Gibson and O. Marques, *"Stereo depth with a unified architecture GPU," in* Computer Vision on GPU*, 2008, pp. 1–6.*

[10] M. Humenberger, T. Engelke, and W. Kubinger, *"A census-based stereo vision algorithm using modified semi-global matching and plane fitting to improve matching quality," in* Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*, june 2010, pp. 77 –84.*

[11] A. Woodward, D. Berry, and J. Dunning, *"Real-time stereo vision on the vision server framework for robot guidance," in* 25th International Conference Image and Vision Computing New Zealand (IVCNZ) 2010*, 2010.*

[12] V. Vineet and P. Narayanan, *"CUDA cuts: Fast graph cuts on the GPU," in* Proc. IEEE CS Conf. Computer Vision and Pattern Recognition (CVPR 2008)*, 2008, pp. 1–8.*

[13] S. Grauer-Gray and C. Kambhamettu, *"Hierarchical belief propagation to reduce search space using CUDA for stereo and motion estimation," in* WACV. *IEEE Computer Society, 2009, pp. 1–8.*

[14] *Open source BSD license. (2011) OpenCv documentation -GPU-accelerated Computer Vision. [Online]. Available: http://opencv.itseez.com/modules/gpu/doc/gpu.html*

[15] *Nvidia Corporation. (2011) Compute Unified Device Architecture(CUDA). [Online]. Available: http://en.wikipedia.org/wiki/CUDA*

[16] *Open source BSD license. (2011) OpenCv-Open Source Computer Vision Library. [Online]. Available: http://opencv.willowgarage.com/wiki/*

[17] G. Gimel'farb, *"Probabilistic regularisation and symmetry in binocular dynamic programming stereo,"* Pattern Recognition Letters*, vol. 23, no. 4, pp. 431–442, 2002.*

[18] H. Hirschmüller, *"Accurate and efficient stereo processing by semi-global matching and mutual information," in* CVPR (2)*, 2005, pp. 807–814.*

[19] D. Scharstein and R. Szeliski, *"A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,"* Int. Journal of Computer Vision*, vol. 47, pp. 7–42, 2002.*

[20] P. F. Felzenszwalb and D. P. Huttenlocher, *"Efficient belief propagation for early vision,"* International Journal of Computer Vision*, vol. 70, no. 1, pp. 41–54, Oct. 2006. [Online]. Available: http://dx.doi.org/10.1007/s11263-006-7899-4*

[21] Q. Yang, L. W. 0002, and N. Ahuja, *"A constant-space belief propagation algorithm for stereo matching," in* CVPR*, 2010, pp. 1458–1465.*

[22] R. Kalarot and J. Morris, *"Dataset for symmetric dynamic programming stereo evaluation," 2010. [Online]. Available: http://www.cs.auckland.ac.nz/ ratheesh/SDS/sds.html*