FPGA Implementation of a Memory-Efficient Stereo Vision Algorithm Based on 1-D Guided Filtering

Yuki Sanada, Katsuki Ohata, Tetsuro Ogaki, Kento Matsuyama, Takanori Ohira, Satoshi Chikuda, Masaki Igarashi, Tadahiro Kuroda, Masayuki Ikebe, Tetsuya Asai and Masato Motomura

Abstract**—** This paper presents an FPGA implementation of a memory-efficient stereo vision algorithm. Recently, a hardware-oriented stereo vision algorithm using 1-D guided filtering was proposed [1]. Our architecture is based on this algorithm and calculates the depth map in 1-D space, and therefore, the required amount of memory is significantly reduced. To realize high speed processing, we apply a full-pipeline and highly parallel structure. Implemented on FPGA, our design uses an 89 kb memory and achieves a 188 frame per second rate for 384×288 stereo images. The accuracy of the disparity map is not satisfactory; however, it can be improved by a small amount of software processing (8.6 ms) [1]. This result shows that the proposed architecture is highly efficient in terms of the required memory, and its processing speed and accuracy is the same as that of other methods.

Index Terms**—**Stereo Vision, Guided Filter, FPGA

I. INTRODUCTION

STEREO matching is a method for estimating a depth map from a stereo image pair. It is desirable to calculate a depth map in real time so that it can be applied for tracking objects, surveillance, pedestrian detection, and so on. Therefore, many researchers have proposed high speed and high accuracy stereo matching methods, and an FPGA or GPU is often used as an accelerator.

GPU-based methods yield high accuracy disparity information, since they employ a complex algorithm. Recently, several methods that employ an edge-preserving filter, such as a guided filter [2], bilateral filter [3], and domain transform filter [4], have been proposed. The method proposed in [5] provides the best accuracy of all the high-speed methods. It achieves a speed that is close to real-time, and holds the second place in the Middlebury stereo evaluation ranking [6].

FPGA-based methods achieve a very high processing speed for a large image having dense disparity. Correlation-based algorithms (SSD, CENSUS) and dynamic programming are often employed ([7],[8]). The objective of some research studies was to achieve an implementation that requires only a small amount of resources. For example, the method presented in [9] reduces the required memory size to 0.9 Mb. However, in these studies, the error rates that were obtained were high or were not evaluated numerically. Meanwhile, Jin and Maruyama presented a system that is, to the best of our knowledge, state of the art [10]. It maintains an accuracy that places in the middle of the Middlebury stereo evaluation ranking and achieves 199.7 fps for an image size of 1024×768.

These methods are satisfactory in terms of both processing speed and accuracy. However, in terms of their application in mobile devices, there is room for improvement. A small device cannot be equipped with a high-end GPU, since it consumes a large amount of energy and uses a large space. Jin and Maruyama's method, which achieves the best performance in FPGA, requires a memory size of 7 Mb. In cases where only on-chip or limited off-chip memory can be used to create a compact package, 7 Mb is too large a memory size.

In this paper, we propose a memory-efficient stereo vision architecture for implementation in mobile device applications. Our algorithm is based on 1-D guided filtering and calculates the disparity map using 1-D information only. Our designed architecture employs a 1-D guided filter in parallel and applies a full pipeline structure. Therefore, it achieves both a reduction in the required amount of memory and high speed processing. On the other hand, the error rate is quite high, because 2-D information is missing. However, by applying simple 2-D software processing to the 1-D hardware output, the error rate can be reduced. When realized in an Altera Stratix II device, our circuit required 89 kb of memory and achieved a rate of 188 fps at 20 MHz for producing a 384×288 disparity map.

The rest of this paper is organized as follows. In Section II, we introduce our stereo matching algorithm and 2-D-software refinement processing. Then, the details of our designed architecture are described in Section III. In Section IV, the experimental results for FPGA implementation are given. Finally, in Section V we present our conclusions.

II. ALGORITHM AND 2-D SOFTWARE REFINEMENT

In general, a local stereo matching algorithm is divided into four steps: i) Cost calculation: a cost map is calculated for each disparity according to the similarities of the corresponding pixels in the stereo image; ii) Cost aggregation: cost maps are filtered to suppress noise; iii) Disparity computation: disparity for each pixel, usually that with the lowest cost, is selected; iv) Disparity refinement: errors in the depth map are detected and fixed.

Our algorithm is based on that presented in [2], and thus, the processes in steps i), iii), and iv) remain the same. The

Y. Sanada is with the Graduate School of Information Science and Technology, Hokkaido University, Kita 14, Nishi 9, Kita-ku, Sapporo 060–0814, Japan (e-mail: sanada@lalsie.ist.hokudai.ac.jp).

K. Ohata, T. Ogaki and T. Kuroda are with the Faculty of Science and Technology, Keio University Hiyoshi 3-14-1, Kouhokuku, Yokohama 223–8522, Japan.

K. Matsuyama, T. Ohira, S. Chikuda, M Igarashi, M. Ikebe, T. Asai and M. Motomura are with the Graduate School of Information Science and Technology, Hokkaido University, Kita 14, Nishi 9, Kita-ku, Sapporo 060–0814, Japan.

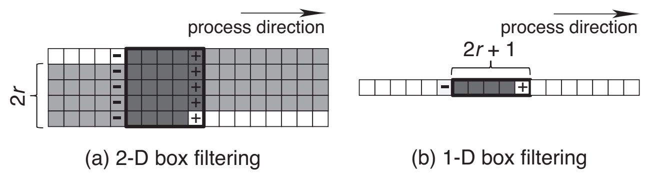

Fig. 1: Comparison of the buffer size. The black frame of pixels are used in the calculation. In a 2-D process, the gray pixels outside the black frame must be conserved.

difference is in the cost aggregation step (ii). We applied gray-scale and one-dimensional filtering to 2-D guided filtering to reduce the required amount of memory. 2-D software processing, which improves a 1-D disparity map, is also a characteristic of our system.

A. 1-D Cost Aggregation

Rheman *et al.* employed a guided filter in cost aggregation [2]. The given input image is a pixel $I(x, y)$; the equations are

$$A(y,x) = \frac{\overline{I(y,x)C(y,x)} - \overline{I(y,x)}\ \overline{C(y,x)}}{\overline{I(y,x)^2} - \overline{I(y,x)}\ \overline{I(y,x)} + \epsilon} \tag{1}$$

$$B(y,x) = \overline{C(y,x)} - A(y,x)\overline{I(y,x)} \tag{2}$$

$$C'(y,x) = \overline{A(y,x)}I(y,x) - \overline{B(y,x)} \tag{3}$$

where $\overline{X(y,x)}$ denotes the mean of X in the window centered at position (y, x), and ϵ denotes a regularizing parameter. This filter smooths the cost C using I, and outputs C'.

In our method, we adopted a 1-D process and gray scale for guided filtering. If the input image is given by a raster scan, calculating $\overline{X}$ in 2-D space requires a large amount of memory. Using a fast calculation algorithm [11], 2-D box filtering requires $2r$ line buffers (Fig. 1(a)). Here, r is a filter radius. On the other hand, calculating $\overline{X}$ in 1-D space reduces the required memory size to $2r + 1$ (Fig. 1(b)). The amount of calculation required using the formulae (1) to (3) is six times greater, and thus, a 1-D method can greatly reduce the required memory size. Gray scale guided filtering is also memory efficient. The amount of information contained in the RGB scale is three times that in the gray scale, and therefore, gray scale calculating reduces the memory size by 1/3. It should be noted that the accuracy of the disparity map does not deteriorate significantly, since the RGB information is already contained in the initial cost.

The 1-D process increases the error rate but it can be refined to some degree by adopting a variable filter size. In our system, since there is no correlation between different rows, the filter radius can be assigned to each row independently. The filter radius is determined by tendency T of texture continuity:

$$T(y,x) = \sum_x |I^f_{r\text{large}}(y,x) - I^f_{r\text{small}}(y,x)| \tag{4}$$

where I^f_r is the gray scale image 1-D box filtered with radius r. If the values of both inputs are close, it is considered that the textures repeat a similar pattern. The filter size of each line processing is selected using the tendency and thresholds. In our experiment, two threshold values, θ_1 and $\theta_2(= 2\theta_1)$, and three approximate radius values,r_s, $r_m(= r_s + r_{\text{step}})$, and $r_m(= r_s + 2r_{\text{step}})$, were determined. We confirmed that the error rate is improved by approximately 0.18%.

B. 2-D Software Refinement

2-D software processing consists of preprocessing, error correction, and applying a median filter.

Before completing the refinement, the error detection and noise reduction is preprocessed. The 1-D hardware output does not consider the vertical connections of disparities, and errors occur as horizontal streaks. In error detection, the vertical gradient of the target pixel (x, y) and $(y \pm 1, x)$ is used for labeling. The labeled pixel contains streak errors or correct edge information. In noise reduction, if the disparity of the upper and lower pixel is equal, the center pixel is corrected to the same value.

Error correction is performed according to the five nearest non-labeled pixels above and below the target pixel. The mode disparity of a 5-pixel set whose colors are closer to that of the target pixel is used for the correction.

2-D refinement is completed with a 5 × 5 median filter for denoising. The details of 2-D software refinement are described in [1]. Figure 2 shows a comparison of the 1-D hardware and 2-D software outputs.

In this study, the entire 2-D refinement algorithm is processed by software, but the preprocessing can be implemented on hardware equipped with a 3-line buffer. On the other hand, the error correction must be executed by software. This requires a vertical direction process, and means that a frame-size memory is necessary for the hardware implementation.

III. FPGA IMPLEMENTATION

The solid blocks of Fig. 3 show the block diagram of our architecture. The depth resolution ($\equiv 2^N$) determines the degree of the parallelism, where the output latency increases as N increases, while the throughput is constant for different N values. Since our circuit is fully pipelined, each pixel of the left and right image ($H \times W$) is input and 1 disparity pixel is output in every clock cycle. Therefore, a latency of about $4 \times W$ clock cycles is required, but the frame rate is determined by the clock frequency f and the image size.

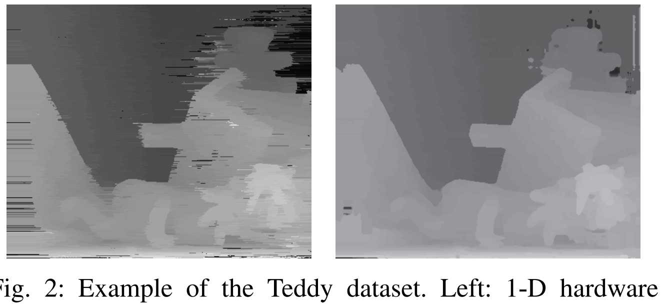

Fig. 2: Example of the Teddy dataset. Left: 1-D hardware output. Right: Processed 2-D software refinement.

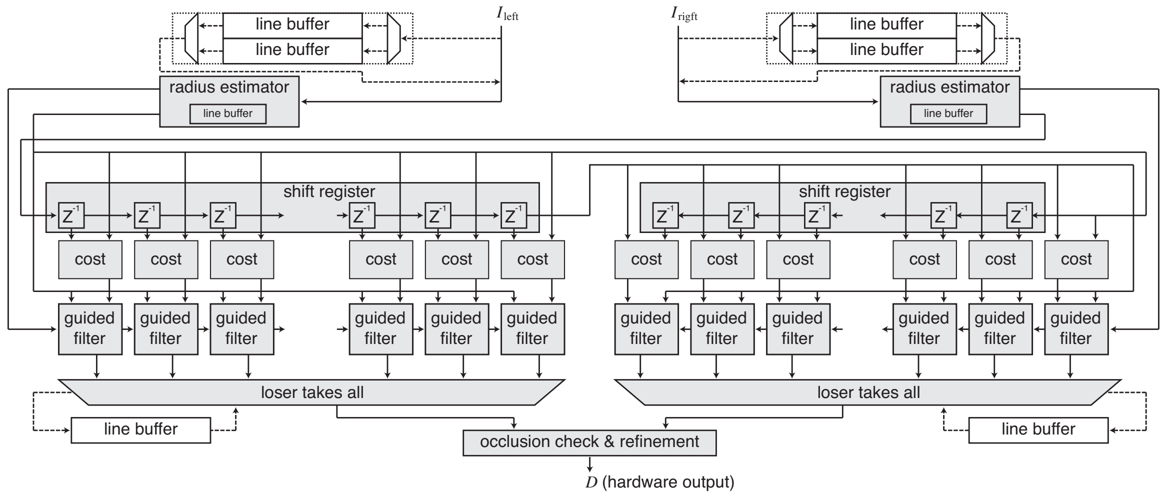

Fig. 3: Block diagram of proposed parallel stereo-matching architecture

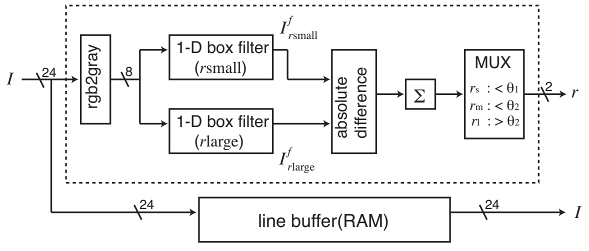

Fig. 4: Block diagram of radius estimator

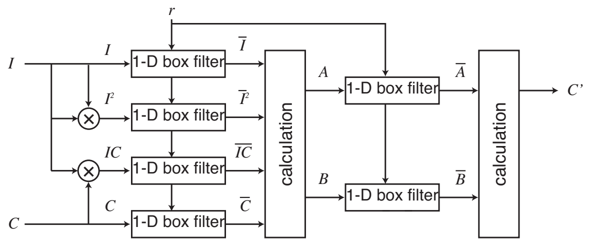

Fig. 5: Block diagram of 1-D-guided filter

The system consists of two radius estimators for left and right sequential images, $2^{N+1}-1$ delay registers (Z^{-1}) for generating binocular disparity, 2^{N+1} cost estimators, 2^{N+1} guided filters, two loser-takes-all circuits to find the minimum costs for the left and right blocks, and an occlusion-check and refinement circuit.

A. Radius Estimator

The given input (sequential) left and right image are first sent to the radius estimator (Fig. 4), which outputs the radius values for the subsequent guided filters. Since the radius estimator employs gray-scale information, an RGB 24 bit pixel is converted to a gray-scale 8 bit pixel. Then, the sum of the absolute difference of $I^f_{r\text{large}}$ and $I^f_{r\text{small}}$ is calculated, and the filter radius is determined by comparing this value with the threshold θ. The bandwidth of the output is sufficient to use 2 bits as a control signal.

The radius values are obtained after reading one line of input images. Therefore, the input image sequences are delayed by the line buffers, as shown in Fig. 4.

B. Cost Estimator

After the radius estimation step, the delayed image sequences arrive as inputs at the shift register (leftmost and rightmost of Z^{-1} in Fig. 3), which gives the pixel values shifted from 0 to 2^N-1 pixels in both the left and right direction. The cost estimator receives both shifted and non-shifted pixel values and accumulates the cost between them in parallel. The disparity between the left and right images is equivalent to the number of shifts. In the cost estimator, the cost map C for each disparity d is calculated using

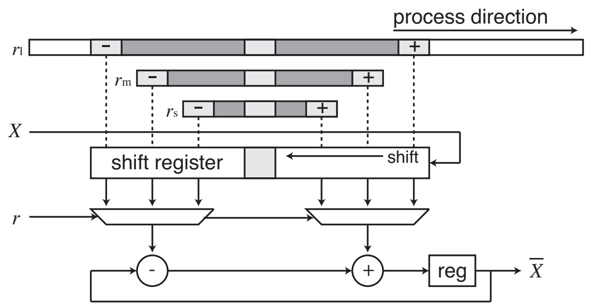

Fig. 6: Block diagram of 1-D box filter

$$\begin{aligned} C_d(y,x) = \min[|I_{\text{left}}(y,x) - I_{\text{right}}(y,x-d)|, \tau_1] \\ +\gamma\min[|\nabla_x I_{\text{left}}(y,x) - \nabla_x I_{\text{right}}(y,x-d)|, \tau_2] \end{aligned} \tag{5}$$

where ∇_x is the gradient of the x direction, α is the parameter to balance the effect of color and gradient, and $\tau_{1,2}$ are threshold values. The color cost employs the RGB channel, and the gradient cost employs gray scale.

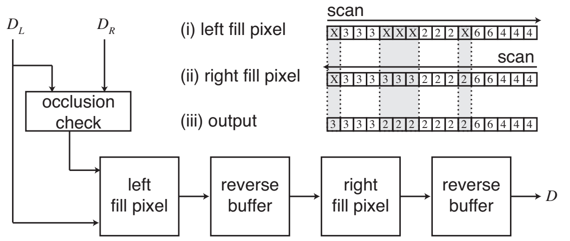

Fig. 7: Block diagram of occlusion check and refinement circuit

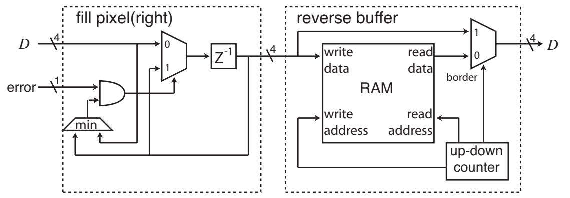

Fig. 8: Fill pixel circuits and reverse buffer

C. 1-D Guided Filter

The calculated cost maps C are then smoothed, using the edge of the gray scale image I, by parallel guided filters. As shown in Fig. 5, our 1-D guided filter circuit consists of six 1-D box filters and associated combinational arithmetic circuits, which calculate formulas (1), (2), and (3). The radius of the 1-D box filter circuits is variable, and can be set at three different values.

The 1-D box filter is based on a fast calculation algorithm [11], and is shown in Fig. 6. A register keeps the summed values of the filter kernel. When the kernel slides to the next pixel, a new summed value is given by (old summed value) + (rightmost pixel value of the target box) - (leftmost pixel value of the box). Therefore, the shift register is required to preserve the leftmost pixel value. The 1-D box filter shares a shift register with a different radius value, because equipping the system with several shift registers is a waste of cost. Note that the center of different filter kernel is matched. In this method, regardless of different radius values, the output timing is constant.

It should be noted that, in practical implementation, one may remove two box filters and one multiplier from the 1-D guided filter, because the values of $\bar{I}$ and $\bar{I}^2$ in Fig. 5 are common over the line.

D. Occlusion Check and Refinement

Among the smoothed cost maps, the minimum values for left and right views are selected by loser-takes-all circuits, and the corresponding disparity values are passed to an occlusion check and refinement circuit.

In the occlusion check, the calculated disparity map is checked using left and right consistency. Disparities that satisfy $D_L(y,x) \neq D_R(y, x - D_L(y,x))$ are detected as an error.

The error is corrected by using the disparity values on the border of the error region. This requires bidirectional scanning, which we realized in the following method (Fig. 7, (i, ii and

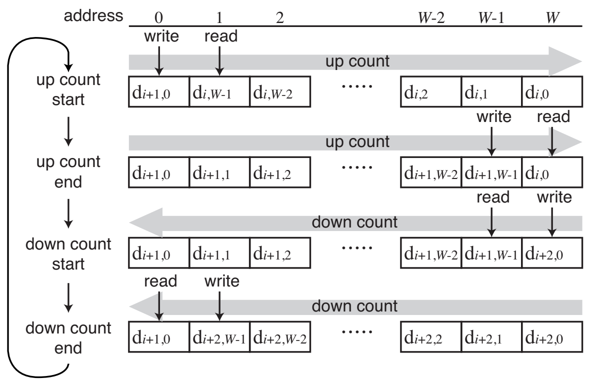
address
0
1
2
W-2
W-1
W
write
read
up count start
up count
$d_{i+1,0}$ $d_{i,W-1}$ $d_{i,W-2}$ ..... $d_{i,2}$ $d_{i,1}$ $d_{i,0}$
write
read
up count end
up count
$d_{i+1,0}$ $d_{i+1,1}$ $d_{i+1,2}$ ..... $d_{i+1,W-2}$ $d_{i+1,W-1}$ $d_{i,0}$
read
write
down count start
down count
$d_{i+1,0}$ $d_{i+1,1}$ $d_{i+1,2}$ ..... $d_{i+1,W-2}$ $d_{i+1,W-1}$ $d_{i+2,0}$
read
write
down count end
down count
$d_{i+1,0}$ $d_{i+2,W-1}$ $d_{i+2,W-2}$ ..... $d_{i+2,2}$ $d_{i+2,1}$ $d_{i+2,0}$

Fig. 9: Reverse buffer operation

iii)): i) The error region is filled by the disparity values on the left border; ii) The flow direction of the disparity sequences is inverted, and the resulting error region is filled by the smallest disparity values among the filled values in i) and the disparity values on the right border; iii) The flow direction is further inverted; the inverted flow represents the refined disparity output. Figure 7 shows a block diagram of processes i) to iii), where the left and right fill pixel blocks represent the disparity filling circuits consisting of combinational circuits, and the reverse buffers invert the signal flows. The advantage of this scheme is that the disparity filling circuits consist of very simple logic elements (Fig. 8). As shown in Fig. 9, the reverse buffer is constructed using a one line-size RAM and one up-down counter only. The read address precedes the write address operation, and both addresses are given by the same up-down counter. In the count-up operation, the input sequence is written from 0 to W addresses sequentially. In the count-down operation, the output sequence is read in the order of addresses from W to 0, and thus, the signal flows are inverted. At the same time, the input sequences are written from W to 0 addresses, sequentially. Returning to the count-up operation, the output sequence is read in the order of addresses 0 from W, and thus, the signal flows are similarly inverted.

After the errors have been corrected, the hardware processing is complete, and the disparity map is transmitted to the CPU.

IV. Experimental Results

A. Hardware Experiments

The proposed system was implemented on a commercial FPGA board (MMS Co., Ltd., Power Medusa MU200-SXII with Altera Stratix II and onboard SRAMs). The system was coded by Verilog HDL, and the RTL model was synthesized by Quartus II. Because the number of logic elements was limited, N was set at 3 (depth resolution is 2^3), and, because of this reduction, the original image was shrunk to 192×144 pixels. Table I summarizes the implementation and performance. The parameters used for cost calculation and aggregation are $\{\tau_1, \tau_2, \gamma\} = \{41, 2, 12\}$, $\{\epsilon, w_{r\text{large}}, w_{r\text{small}}, \theta_1, w_s, w_d\} = \{2, 8, 6, 330, 5, 9\}$.

Figure 10 shows the estimated depth results for three different samples produced by the FPGA. Although the precision

ALUT	Register	Block memory	DSP block 9-bit	Input Res. & Depth	Output Res. & Depth	FPGA CLK
36,969	35,360	32,451	504	192×144 (24-bit rgb)	192×144 (3-bit)	20 MHz

TABLE I: Implementation and performance summary

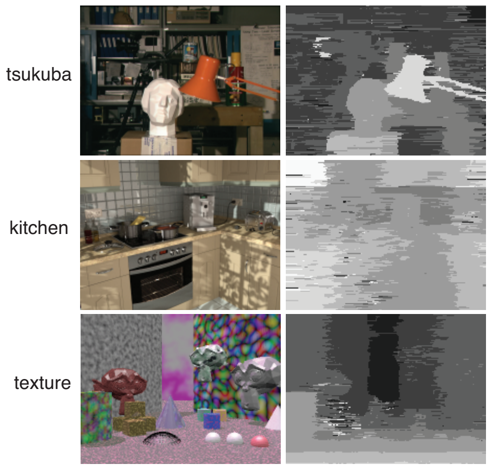

Fig. 10: Experimental results(192×144 pixels, 3-bit depth)

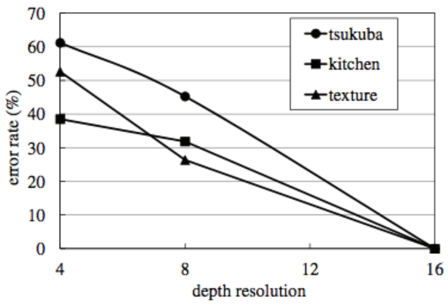

Fig. 11: Error rate vs. implemented depth resolution

of the depth values is degraded because of the reduction in depth resolution, the FPGA could generate an approximate depth map. It should be noted that this is certainly due to the limited number of logic elements in Stratix II, because our RTL results perfectly matched the numerical results at any depth resolution, and the RTL results with the 3-bit depth map matched the FPGA results as well. We also compared the error rates of the original 4-bit depth map and the degraded 2- and 3-bit depth map, as shown in Fig. 11.

These results indicate that the proposed architecture is not nominal, but can be synthesized and operated with acceptable clocks and number/scale of hardware resources. To compare our method with that presented in [10], we compiled and estimated a method using an image size of 1024×768(Table II). When we compiled the method, the depth resolution was reduced to 16 in order to achieve successful compilation. Using the compilation results, we estimated the hardware specification at a depth resolution of 64. Since 52% of RAM is depth resolution-dependent, doubling the depth resolution to 32 increases the total size of RAM to 152% of that of the compiled result. When extending the depth resolution from 32 to 64, doubling the size of RAM again results in an

	[10]	Ours	Estimated
Disparity range	60	16	64
LUTS	122 k	111 k	202 k
RAM	7189 k	89 k	266 k
CLK	318.3 M	20.0 M	40.0 M
FPS	199.7	25.4	25.4

TABLE II: Comparison of the hardware specifications for Jin and Maruyama's method and our method for an image size of 1024×768. Because of lack of resources, our method was only compiled, and not implemented.

enormous FPGA size. Instead, we doubled the CLK of the radius generators and guided filters. Thirty two guided filters are operated twice and generate 64 filtered costs. Thus, the additional circuits shown inside the dotted box in Fig. 3 are needed. They are: (i) four line buffers to hold the left and right lowest cost and its disparity; and (ii) two RGB line buffers for the left and right input image, which have a total size of 130 kb. As a result, the estimated size of RAM for a depth resolution of 64 was 266 kb, which is only 3.7% of that of the system proposed in [10]. Therefore, we can say our method is a low-memory method.

B. Software Experiments

The measurement of the software was conducted on an Intel Core i3 2.53GHz PC, using C++ and four Middlebury stereo pairs. The threshold value was set to 40. The output is shown in Fig. 12. The Middlebury stereo evaluation ranking of our method is listed in Table III. The average processing time was 56.6 ms for a 1024×768 image and 8.6 ms on average for the Middlebury dataset stereo pairs, which are 400×380. When the size of the Middlebury dataset is used, a significant amount of time remains to place the depth-aided application.

V. CONCLUSIONS

This paper presented an FPGA implementation of a memory-efficient stereo vision algorithm. The most significant element of the memory reduction was adapting the 1-D process and gray scale to guided filtering. The proposed architecture employs 1-D guided filters in parallel and a full-pipeline

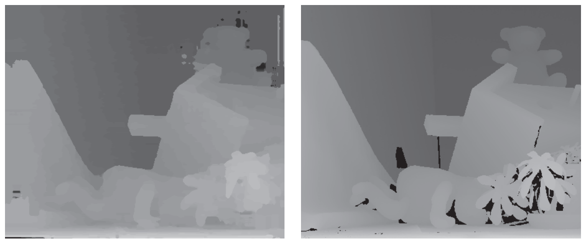

Fig. 12: Left: Results of the Middlebury data set. Right: Ground truth

	Tsukuba			Venus			Teddy			Cones			
Method	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	nonocc	all	disc	average
CostFilter[2]	1.51	1.85	7.61	0.20	0.39	2.42	6.16	11.8	16.0	2.71	8.24	7.66	5.55
Fast[10]	1.38	1.84	7.36	0.30	0.48	2.09	7.41	12.7	17.5	3.44	9.19	9.90	6.13
Ours	2.25	2.81	9.45	1.23	1.76	6.83	4.84	10.5	13.0	3.54	8.90	9.83	6.24
RealTimeBP[12]	1.49	3.40	7.87	0.77	1.90	9.00	8.72	13.2	17.2	4.61	11.6	12.4	7.69
RealTimeDP-Tree[13]	1.49	2.51	6.60	2.37	2.97	13.1	8.11	13.6	15.5	8.12	13.8	16.4	8.71

TABLE III: Middlebury evaluation of previous methods and our method

structure to achieve high speed processing. The results of the FPGA implementation showed that our design performs at a rate of 188 fps and requires only an 89 kb bit memory for a 384×288 image. Moreover, the disparity map, which is improved by a small amount of software processing (8.6 ms), maintained sufficient accuracy. Owing to the use of Middlebury datasets, the average percentage of bad pixels is 6.24%. Our architecture is targeted at applications for mobile devices. The results that we obtained for the speed, accuracy, and cost of the processing are satisfactory.

ACKNOWLEDGMENT

The authors would like to thank Semiconductor Technology Academic Research Center (STARC), Japan, for funding this research project.

REFERENCES

[1] K. Ohata, Y. Sanada, T. Ogaki, K. Matsuyama, T. Ohira, S. Chikuda, M. Igarashi, M. Ikebe, T. Asai, M. Motomura, and T. Kuroda, "Hardware-oriented stereo vision algorithm based on 1-D guided filtering and its FPGA implementation," in *Proc. IEEE International Conference on Electronics, Circuits, and Systems(ICECS)*, Dec. 2013. pp. 169-172.

[2] C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition(CVPR)*, Jun. 2011. pp. 3017-3024.

[3] Q. Yang, "Recursive bilateral filtering," in *Proc. 12th European Conference on Computer Vision(ECCV)*, Oct. 2012. part I, pp. 399-413.

[4] C. Pham and J. Jeon, "Domain transformation-based efficient cost aggregation for local stereo matching," *IEEE Trans. CSVT*, Vol. 23, Issue 7, pp. 1119-1130. Jul. 2013.

[5] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, "On building an accurate stereo matching system on graphics hardware," in *Proc. International Conference on Computer Vision (ICCV) Workshops*, Nov. 2011. pp. 467-474.

[6] D. Scharstein and R. Szeliski, "Middlebury stereo evaluation," `http://vision.middlebury.edu/stereo/`

[7] K. Ambrosch, M. Humenberger, and W. Kubinger, "SAD-based stereo matching using FPGAs," in *Embedded Computer Vision, Advances in Pattern Recognition, Springer*, 2009, pp. 121-138.

[8] S. Jin, J. Cho, X. D. Pham, K. M. Lee, S.-K. Park, M. Kim, and J. W. Jeon, "FPGA design and implementation of a real-time stereo vision system," *IEEE Trans. CSVT*, Vol. 20, Issue 1, pp. 15-26. Jan. 2010.

[9] Y. Shan, Z. Wang, W. Wang, Y. Hao, Y. Wang, K. Tsoi, W. Luk, and H. Yang, "FPGA based memory efficient high resolution stereo vision system for video tolling," in *Proc. International Conference on Field-Programmable Technology (FPT)*, Dec. 2012. pp. 29-32.

[10] M. Jin and T. Maruyama, "A fast and high quality stereo matching algorithm on FPGA," in *Proc. International Conference on Field Programmable Logic and Applications (FPL)*, Aug. 2012. pp. 507-510.

[11] M. J. McDonnell, "Box-filtering techniques," *Computer Graphics and Image Processing*, vol. 17, pp. 65-70, 1981.

[12] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister, "Real-time global stereo matching using hierarchical belief propagation," in *Proc. British Machine Vision Conference(BMVC)*, Sep. 2006. vol. 2. pp. 989-998.

[13] M. Jin and T. Maruyama, "A real-time stereo vision system using a tree-structured dynamic programming on fpga," in *Proc. ACM/SIGDA International Symposium on Field Programmable Gate Arrays(FPGA)*, Feb. 2012. pp. 21-24.