Algorithm and Architecture of Disparity Estimation With Mini-Census Adaptive Support Weight

Nelson Yen-Chung Chang, Tsung-Hsien Tsai, Bo-Hsiung Hsu, Yi-Chun Chen, and Tian-Sheuan Chang, *Senior Member, IEEE*

***Abstract*—High-performance real-time stereo vision system is crucial to various stereo vision applications, such as robotics, autonomous vehicles, multiview video coding, freeview TV, and 3-D video conferencing. In this paper, we proposed a high-performance hardware-friendly disparity estimation algorithm called mini-census adaptive support weight (MCADSW) and also proposed its corresponding real-time very large scale integration (VLSI) architecture. To make the proposed MCADSW algorithm hardware-friendly, we proposed simplification techniques such as using mini-census, removing proximity weight, using YUV color representation, using Manhattan color distance, and using scaled-and-truncate weight approximation. After applied these simplifications, the MCADSW algorithm was not only hardware-friendly, but was also 1.63 times faster. In the corresponding real-time VLSI architecture, we proposed partial column reuse and access reduction with expanded window to significantly reduce the bandwidth requirement. The proposed architecture was implemented using United Microelectronics Corporation (UMC) 90 nm complementary metal-oxide-semiconductor technology and can achieve a disparity estimation frame rate of 42 frames/s for common intermediate format size images when clocked at 95 MHz. The synthesized gate-count and memory size is 563k and 21.3 kB, respectively.**

***Index Terms*—Application-specific integrated circuits (ASIC), computer vision, digital circuits, digital integrated circuits, very large scale integration (VLSI).**

I. Introduction

STEREO VISION is an important early vision tool that has been widely adopted by applications such as multiview video coding, freeview TV, 3-D video conferencing, intelligent surveillance, autonomous vehicles, and mobile robots. Stereo vision finds the depth in a scene based on the stereo image pair of the scene. The depth of a pixel is inversely proportional to the disparity of this pixel. The disparity of a pixel is the distance between this pixel and its corresponding pixel in the other image when both images are overlapped. The process of finding the corresponding pixel is often referred as disparity estimation or stereo matching. The resulting disparity of each pixel in an image forms a disparity image or disparity map. For more detail on stereo vision, please refer to [1].

The applications that adopt stereo vision often require high-performance and real-time processing speed. The performance is usually evaluated by the error rate of a disparity map when compared to a ground truth disparity map [2]. Lower error rate implies higher performance. Complex disparity estimation algorithms usually achieve much better performance than simple algorithms. However, simple algorithms are usually much faster than complex algorithms. As a result, most real-time applications have adopted simple algorithms to trade the performance for speed. For applications that cannot accept trading performance for speed, complex algorithms have been adopted and implemented using powerful computation devices such as digital signal processors (DSPs), graphic processing units (GPUs), and dedicated hardware. However, the computation power of DSPs is not high enough to enable complex disparity estimation algorithms to support real-time processing. The GPUs can support real-time disparity estimation, but is too expensive in terms of cost and power consumption for embedded real-time vision applications, such as mobile robotics and intelligent autonomous vehicles. The dedicated hardware approach that uses field programmable gate array (FPGAs)/application-specific integrated circuits (ASICs) can provide high-computation power with relatively less expensive hardware cost. This makes the dedicated hardware approach suitable for implementing complex disparity estimation algorithms for real-time applications. However, complex disparity estimation algorithms are often not hardware-friendly and are bandwidth hungry.

Being aware of the need for high-performance real-time disparity estimation and the abovementioned problems, this paper proposes a hardware-friendly high-performance mini-census adaptive support weight (MCADSW) disparity estimation algorithm and its corresponding real-time very large scale integration (VLSI) hardware architecture.

The proposed MCADSW, which was based on the adaptive support weight (ADSW) [3], adopted the mini-census transform to improve the performance and robustness to radiometric distortion. In addition, we also proposed hardware-friendly

Manuscript received November 23, 2008; revised April 30, 2009 and September 20, 2009. Date of publication March 15, 2010; date of current version June 3, 2010. This research was sponsored by the National Science Committee, Taiwan, under Grant NSC-96-2220-E-009-038. This paper was recommended by Associate Editor C. N. Taylor.

N. Y.-C. Chang is with the Division of Intelligent Robotics Technology, Mechanical and Systems Research Laboratories, Industrial Technology Research Institute, Hsinchu 31040, Taiwan (e-mail: ycchang.twins@ gmail.com).

T.-H. Tsai is with MediaTek, Inc., Hsinchu 30078, Taiwan (e-mail: roy.tsai@gmail.com).

B.-H. Hsu, Y.-C. Chen, and T.-S. Chang are with the Institute of Electronics, National Chiao-Tung University (NCTU), Hsinchu 300, Taiwan (e-mail: sadeibear@gmail.com; leftivan.ee93@nctu.edu.tw; tschang@twins.ee.nctu.edu.tw).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCSVT.2010.2045814

simplifications to make the MCADSW more hardware-friendly. These simplifications includes using YUV color representation instead of $L^*a^*b^*$ color representation, using Manhattan color distance instead of Euclidian color distance, eliminating proximity weight, using scale-and-truncate weight approximation. These simplifications also reduced the execution time of the software implementation by 61.3%.

The proposed architecture of the MCADSW addressed the bandwidth requirement issue by applying the proposed partial column reuse (PCR) and access reduction with expanded window (AREW) techniques. These techniques reduce the bandwidth requirement by two orders compared to a direct implementation without data reuse when processing common intermediate format (CIF) stereo image pairs at 30 frames/s. The proposed architecture computes 42 CIF size disparity maps with 64 disparity levels per second. The equivalent gate-count and total memory size was 562k and 21.3 kB, respectively.

There are three main contributions in this paper. First, a hardware-friendly high-performance algorithm and the corresponding simplification techniques are proposed. Second, the bottleneck that limits the disparity estimation speed, which is the large bandwidth requirement issue, is solved by applying the proposed bandwidth reduction techniques. Both the presented simplification and bandwidth reduction techniques can be extended and applied to other weighted aggregation-based vision algorithms, such as outlier rejection [4] and segment support [5] algorithms. Third, a VLSI architecture of the proposed MCADSW that enables real-time disparity estimation is presented. This architecture can also be extended to support other algorithms based on ADSW algorithm.

The rest of this paper is organized as follows. Section II reviews related work on disparity estimation algorithms and real-time implementations. Section III describes the proposed MCADSW algorithm. Section IV presents the bandwidth reduction techniques. Section V presents the architecture of the MCADSW. Section VI presents the implementation result. Finally, Section VII concludes this paper.

II. Related Works

A. Stereo Matching Algorithm

Disparity estimation algorithms can be categorized into local and global approaches [2]. Local approach determines the disparity of a pixel based on the support window similarity. The local approach usually has low-computation complexity and storage requirement, and has been frequently adopted by real-time implementations [6]–[16]. Global approach determines the disparity of all the pixels in an image as a whole by optimizing a global energy function. However, the optimization is usually complex and extremely computation intensive. Hence, we will focus on local approaches.

Early works on local approach studied the impact of different similarity measures [17], [18]. Their work pointed out that census, rank [19], and mutual information [20] achieved better disparity estimation performance and are more robust to radiometric distortion. Later, [21] investigated the performance of using different color representation. Recently, [22] investigated the performance and speed jointly of different similarity measure and color representation combinations. The result showed that census-based combination achieved better performance, but also takes more time to compute.

Another important research topic that has been studied is the support window size. The simple fixed size rectangular window adopted in early local approaches suffered from incorrect disparity estimation in depth-discontinuity, textureless, and repeating pattern regions. To remedy this, [23], [24] proposed variable window size algorithms. Later, [25] also proposed a variable window size algorithm that adaptively adjusted the window size based on a reliability measure. The variable window size could effectively improve the disparity estimation performance in textureless and repeating pattern regions, but not in depth-discontinuity regions. Being aware of this, [26], [27] proposed shiftable windows algorithms to improve the performance. Kang *et al.* [28] combined both the concept of variable window size and shiftable window together. However, the qualitative result of their work still showed great room for improvement.

The reason for not being able to completely improve the performance in the depth-discontinuity region is because the assumption of having the same disparity in a window does not hold in depth-discontinuity and slanted surface regions. Understanding this, Veksler [29] proposed a compact window class method which could model nonrectangular support windows. Although their result showed significant performance improvement compared to previous algorithms, the performance near the boundary region was still inferior to complex global approaches. Yoon *et al.* proposed an adaptive support weight (ADSW) [3] algorithm that assigned different weights to the pixels in a support window based on the proximity and color distances to the center pixel. As a result, the ADSW could achieve the effect of using a support window of arbitrary size and shape. As a result, the performance of ADSW was comparable to some of the complex global algorithms. Later, segmentation-based support methods were also proposed [4], [5]. The outlier rejection [4] used a binary weight based on the segmentation region instead of the weight used in the ADSW. Tombari *et al.*'s segment support algorithm [5] assigned fixed weight of one to the pixels in the same segment as the center pixel is in, and assigned weight to the pixels outside the center pixel's segment according to the color similarity between the outside pixel and the center pixel. Recently, [30] conducted a detailed comparison on the performance and processing speed of local algorithms. Their result showed that the segment support has the highest performance but was two times slower than the ADSW. The performance of the ADSW was only slightly inferior to the segment support algorithm.

B. Real-Time Implementations

Real-time stereo matching implementations can be categorized into general purpose processor solutions, DSP solutions, GPU solutions, and dedicated hardware solutions.

The general purpose processor solutions rely on the great computation power in state-of-the-art processors to accommodate the high-computation complexity of stereo matching algorithms. Early works [8], [31] tried to implement real-time stereo matching on general purpose processors, however they could only achieve nonvideo rate performance due to limited computing power at their times. As the processor technology advances, [32]–[34] implemented real-time stereo matching algorithms on general purpose processors. They managed to achieve real-time processing, but the performance of their disparity map was not very high because of using simple local algorithms. Although simple local algorithms have been adopted by most general purpose solutions, Forstmann *et al.* [35] proposed a real-time implementation of the less complex global algorithm, the dynamic programming, on general purpose processors. Their performance is higher than most of the previous local algorithms, but their real-time processing speed is limited to images smaller than VGA.

The DSPs have better processing speed on signal processing algorithms because of the single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) architectures than general purpose processors. In addition, they are often less expensive and less power consuming than the state-of-the-art general purpose processors. Hence, DSP solutions are more favorable in embedded stereo vision applications. Konolige's small vision system [8] is one of the most famous early real-time DSP solutions. Recently, [16] also proposed a real-time DSP implementation with jigsaw matching templates. Although the DSP solutions may have more computation power than general purpose processors, the data word alignment and bandwidth issue often limit their capability. As a result, the DSP solutions are usually limited to local algorithms and could not provide high-performance stereo matching result in real-time.

Another powerful solution is the GPU solutions. The GPUs have extremely high-memory bandwidth that ranges from 6.5 GB/s to 128 GB/s and can have up to 256 stream processors. With so much hardware resource, the GPU solutions [36]–[39] could implement high-performance complex stereo matching algorithms. However, GPUs are too expensive and power consuming to be used in embedded vision applications currently.

The dedicated hardware solutions can also provide great computation power while allowing the computation resource to be optimized for utilization by designing the architecture in a customized way. This enables the dedicated hardware solutions to be more cost efficient than the GPU solutions. The dedicated hardware includes both FPGA/programmable logic device (PLD) and ASIC. Faugeras *et al.*'s PeRLe-1 board [40] and Nishihara's PRISM-3 based stereo system [41] were two of the earliest dedicated hardware solutions. Later, other early dedicated hardware solutions [7], [9], [10], [25] have also been proposed. Among these works, [9] and [10] were two of the first real-time implementation adopting the census matching. However, these early solutions only implemented simple local algorithms. Consequently, their performance was not high. Being aware of the performance limitation of local algorithms, hardware architectures have been proposed for dynamic programming [42] and hierarchical belief propagation (HBP) [43], [44] algorithms. Their performance was very high since they were based on high-performance global methods. However, their hardware cost was also very high compared to other dedicated hardware implementations. Recently, Tsai *et al.* [45] studied data reuse techniques in aggregation-based algorithms to reduce the internal storage size, computation resource, and bandwidth requirement.

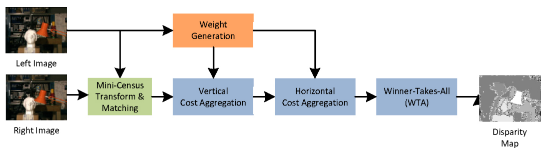

Left Image
Weight Generation
Mini-Census Transform & Matching
Vertical Cost Aggregation
Horizontal Cost Aggregation
Winner-Takes-All (WTA)
Right Image
Disparity Map

Fig. 1. Overall flow of the proposed mini-census ADSW algorithm.

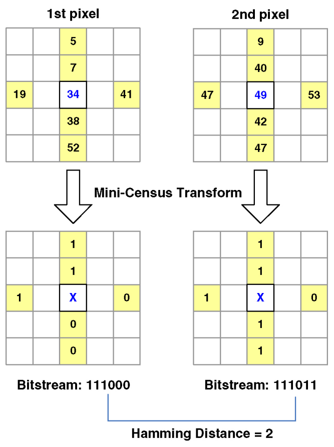


Fig. 2. Census transform and matching.

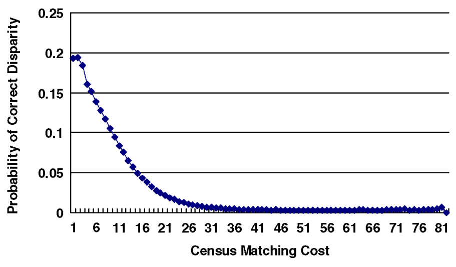
III. Proposed Mini-Census ADSW Algorithm

A. *Algorithm Overview*

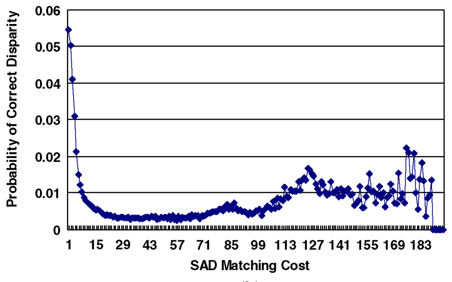
Fig. 1 shows the overall flow of the proposed mini-census adaptive support weight (MCADSW) algorithm. The MCADSW algorithm consists of four major steps. First, the mini-census transform and matching step performs mini-census transform on the left and right images and computes the initial matching cost of each pixel. The second step is the weight generation which generates the weight coefficients needed in the cost aggregation step. Once both the initial matching costs and weight coefficients are available, the matching cost will be aggregated through a two-pass cost aggregation step. Finally, the best disparity can be obtained by finding the disparity with the minimum aggregated matching cost through a winner-takes-all (WTA) method.

B. *Mini-Census Transform and Matching*

The mini-census transform, a modified and simplified version of census transform [19], compares the luminance of the 6 pixels within a support window with the center pixel. The 6-pixel template is shown in Fig. 2. If a pixel's luminance is larger than the center pixel's luminance, it is given the label 0, otherwise the label 1. After the comparison of the 6 pixels, a binary bitstream is obtained which characterizes the luminance



(a)



(b)

Fig. 3. Probability of correct disparity versus different matching cost in *Teddy*. (a) Census matching cost. (b) SAD matching cost.

relation between the center pixel and its surrounding 6 pixels. With the mini-census bitstream, we can represent each pixel using only 6 bits.

The mini-census matching cost between 2 pixels is defined as the hamming distance between the mini-census bitstreams. We would refer the mini-census matching cost as the census cost hereon for brevity, which is defined as

$$E_{i,d} = H(b_{L,i}, b_{R,i,d}) \tag{1}$$

where $E_{i,d}$ is the census cost of pixel i at disparity d; $b_{L,i}$ is the bitstream of pixel i in the left image and $b_{R,i,d}$ is the bitstream of pixel i at disparity d in the right image; $H()$ is the hamming distance function.

Fig. 2 also illustrates an example of the mini-census transform and census cost. After the transform, the mini-census bitstreams of the 2 pixels in the figure are 111 000 and 111 011, respectively. The hamming distance between the bitstreams is 2; hence, the census cost is 2.

Since the bitstream represents relative information, the census cost is therefore much less sensitive to brightness bias and exposure gain. In addition, the census cost better reflects the probability of finding the correct match when compared to the sum of absolute difference. Fig. 3 illustrates the probability of finding a correct disparity given a matching cost for the Middleburry stereo image pair *Teddy*. Fig. 3(a) illustrates the 9×9 census cost case and Fig. 3(b) illustrates the 15×15 sum of absolute differences (SAD) case. The probability decreased as the census cost increased. However, the probability did not always decrease as the SAD cost increase, especially when the cost is large. This implies sometimes the SAD cost of a correct disparity may be larger than other disparities. Similar result could also be observed in other stereo image pairs. The main reason for this is because the relative information encoded in the census resulted in better matching performance in occluded

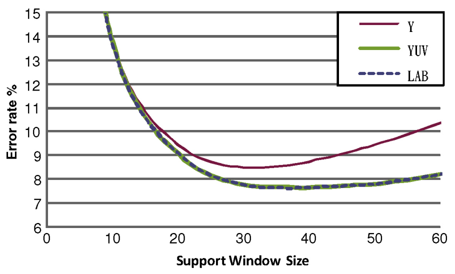


Fig. 4. Disparity estimation error rate of using different color spaces.

region than SAD did. Despite the benefits of the census cost, census cost is very sensitive to image noise and hence should be used with care.

C. Weight Generation

The adaptive weight generation was based on the proximity and color distances in the original ADSW algorithm. However, we removed the proximity weight based on our observation that it mainly benefits the performance when the support window is larger than 19×19. We have compared the average disparity error rate of MCADSW with and without the proximity weight. The average disparity error rate was averaged over the overall error rate of *Tsukuba*, *Venus*, *Teddy*, and *Cones* stereo image pairs from the Middlebury stereo vision evaluation website [2]. Note that we have decided to use the overall error rate instead of the nonocclusion error rate because in most applications, such as robotics, freeview TV, and 3-D modeling, prefer overall error rate over nonocclusion error rate. This is due to the fact these applications do not just rely the depth at nonocclusion, the depth at depth discontinuities is also critical to the performance of these applications. From our experiment result, the difference of the disparity estimation error rate between using and not using the proximity weight was less than 3% for window size of 31×31. Therefore, we decided to trade the very small performance loss for the computation complexity reduction.

The color weight was originally defined as a Gaussian function of the color distance between a pixel i in the support window and the center pixel c of the window. The color weight $w_{i,c}$ of pixel i with respect to pixel c is

$$w_{i,c} = \exp\left(-\frac{\Delta C_{i,c}}{\gamma_c}\right) \tag{2}$$

where $\Delta C_{i,c}$ is the color distance between pixel i and c; γ_c is a tuning constant. This weight allows the pixel with color similar to the center pixel to have more influence on the final matching cost. Note that since we use two-pass aggregation, the weight for vertical and horizontal aggregations are generated separately. The vertical weight is generated with respect to the center pixel of each column in the support window, whereas the horizontal weight is generated from the center row with respect to the center pixel of the support window.

To reduce computation complexity and make the algorithm more hardware-friendly, we proposed three simplifications to the weight generation.

First, we adopted the YUV color representation instead of the $L^*a^*b^*$ color representation, which was originally

TABLE I
PERFORMANCE COMPARISON BETWEEN USING EUCLIDEAN AND MANHATTAN COLOR DISTANCES

Method	Average Error Rate (%)	Error Rate (%)				*Tsukuba* Execution Time (seconds)
		Tsukuba	*Venus*	*Teddy*	*Cones*	
Euclidean	7.47	3.47	0.91	14.3	11.2	4.75
Manhattan	6.94	3.08	0.59	14.0	10.1	3.12

adopted by the ADSW, in our MCADSW. Using YUV color representation allowed us to use hardware-friendly positive integer numbers instead of complex hardware-unfriendly signed floating-point numbers during the weight generation. Fig. 4 shows the average error rate of MCADSW using different color representations. The error rate was averaged over the overall error rates of the four stereo image pairs from the Middleburry stereo vision evaluation website. Note that the error tolerance used to evaluate the error rate in this paper is one disparity level unless otherwise specified. From Fig. 4 we noticed that the error rate difference between using YUV and $L^*a^*b^*$ color representations was insignificant since both error curves were almost overlapped. The error rate in the case of using Y-only color representations was significantly larger than using $L^*a^*b^*$. Therefore, we have decided to adopt YUV color representation in MCADSW.

Second, we used Manhattan color distance instead of Euclidean color distance to eliminate the three square and one square root computations that was necessary for computing Euclidean distance. Table I lists the disparity estimation error rate and execution time on a Pentium IV 2.8 GHz machine when Euclidean and Manhattan distances were used in MCADSW. Interestingly, the result using Manhattan color distance slightly outperformed using Euclidean color distance. One possible explanation is that YUV color representation is not perceptually uniform and linear like $L^*a^*b^*$ color representation is. Therefore, Euclidean color distance could no longer reflect the actual color distance. After using Manhattan color distance, the execution time of the MCADSW was reduced by 34.3%.

Third, we proposed a scale-and-truncate approximation of the color weight function. The scale-and-truncate approximation approximated the exponential function by scaling it up by 64 then truncate it to leave only one nonzero most significant bit (MSB). The reason for scaling up by 64 is because the error rate stopped decreasing after the scale factor exceeds 64. The reason for preserving only one nonzero MSB is because the error rate difference between preserving one bit and two bits was less than 0.5%. With only one bit in the color weight being one, the multiplication between the color weight and the initial cost can be implemented with only one simple shift operation. The curve of the final scale-and-truncated weight is shown in Fig. 5. After applying the proposed approximation, the execution time could be reduced by 41.0%.

D. Vertical and Horizontal Cost Aggregation

The final matching cost was aggregated from the weighted cost within the support window using a two-pass approach proposed by Wang *et al.* [39]. The two pass approach first

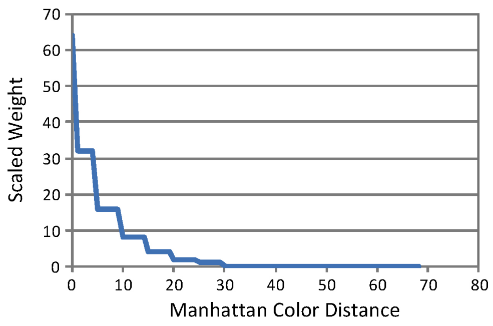


Fig. 5. Curve of the scaled-and-truncated weight function.

aggregates vertically to give a vertical aggregated cost of each column, then the vertical aggregated costs are aggregated horizontally together to give the final matching cost. The vertical aggregated cost $E_{v,\text{col},d}$ of a column "col" within the support window at disparity d can be defined as

$$E_{v,\text{col},d} = \sum_{i \in \text{col}} E_{i,d} w_{v,i,c} \tag{3}$$

where $E_{i,d}$ is the initial census cost of the pixel i in the column, $w_{v,i,c}$ is the vertical weight of pixel i with respect to the center pixel c of the column. The horizontal aggregated cost $E_{h,\text{row},d}$ at disparity d, which is also the final matching cost $E_{\text{final},d}$, is defined as

$$E_{\text{final},d} = E_{h,\text{row},d} = \sum_{\text{col} \in W} E_{v,\text{col},d} w_{h,j,c} \tag{4}$$

where $E_{v,\text{col},d}$ is the vertical aggregated cost of the column "col" within the aggregation window W; $w_{h,\text{col},c}$ is the horizontal weight of column col's center pixel j with respect to the center pixel c of the window.

The two-pass approach reduces computation complexity when compared to the direct approach. If the window size is $(r+1) \times (r+1)$ and the disparity range is D, the complexity of the two-pass approach is *O(2rD)*, whereas the complexity of the direct approach is $O(r^2 D)$. In addition to the computation complexity reduction, the two-pass approach also reduces the internal bandwidth of the hardware design. However, [39] have reported observable quality drop of the disparity map after applying the two-pass approach to the original ADSW. Our own experimental result on the ADSW showed approximately less than 3% average error rate increase after applying the two-pass approach.

E. Performance Evaluation

Table II lists the error rate and desktop PC execution time of the proposed MCADSW and other state-of-the-art algorithms. The error rate evaluates the performance of a disparity

TABLE II
PERFORMANCE COMPARISON OF THE MCADSW AND OTHER ALGORITHMS

Method	Error Rate %					PC Spec. (Brand, processor, clock rate)	*Tsukuba* Exec. Time (seconds)
	Tsukuba	*Venus*	*Teddy*	*Cones*	Average		
SSD + MF [2]	7.07	5.16	24.8	19.8	14.2	Intel CoreDuo 2.99 GHz	0.64
EffectiveAggr[46]	2.11	4.75	15.2	12.6	8.7	Intel CoreDuo 2.14 GHz	0.20
RealDP [35]	2.85	6.42	N.A.	N.A.	N.A.	AMD AthlonXP 2800+	0.02
RealTimeBP[43]	3.40	1.90	13.2	11.6	7.5	Intel Pentium IV 3.0G	3.39
RealTimeGPU [39]	4.22	2.98	14.4	13.7	8.8	Intel Pentium IV 3.0G	4.91
CoopRegion [48]	1.13	0.18	9.03	7.80	4.5	Intel Pentium M 1.6G	∼20.00
Original ADSW [3]	1.85	1.19	13.3	9.79	6.5	AMD AthlonXP 2700+	∼60.00
Our ADSW	4.18	3.41	20.6	16.0	11.1	Intel Pentium IV 2.8 GHz	95.65
MCADSW	2.80	0.64	13.70	10.1	6.8	Intel Pentium IV 2.8 GHz	1.84

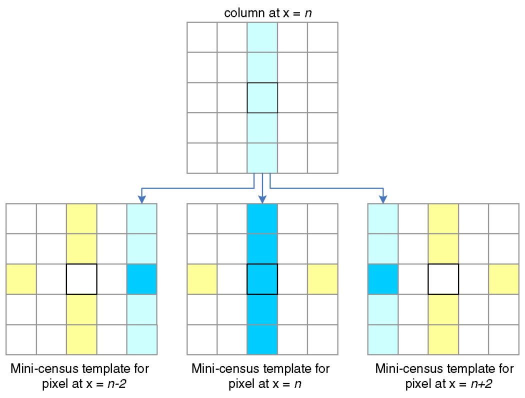

column at x = *n*
Mini-census template for pixel at x = *n-2*
Mini-census template for pixel at x = *n*
Mini-census template for pixel at x = *n+2*

Fig. 6. Partial column reuse in mini-census transform.

estimation algorithm and is independent of which computing platform an algorithm is being implemented. The execution time evaluates the computing complexity of an algorithm based on desktop PC-based implementations. Although the execution time of an algorithm depends on the clock rate of the target processor and code optimization, comparing the execution time of different algorithms gives a rough figure of their computation complexity. The error rate and the execution time of other algorithms in the table were acquired from their published works. For ADSW, we have included the execution time provided from their published work as well as the execution time acquired from us running their software. The error rate and execution time we gathered is labeled as "Our ADSW." For algorithms that did not provide the execution time of the *Tsukuba* stereo image pair, such as RealTimeBP and RealTimeGPU, we estimated their execution time based on their best disparity estimation speed, which is usually represented in terms of *million disparity estimation per second* (MDE/s). For instance, RealTimeGPU reported a disparity estimation speed of 0.36 MDE/s on an Intel Pentium IV 3 GHz machine in their work; by dividing the number of disparity estimations needed in the *Tsukuba* stereo image pair, which is $384 \times 288 \times 16 = 1.77$ MDE, by the disparity estimation speed, we get an estimated execution time of 4.91 s.

The average error rate of the proposed MCADSW was comparable to RealTimeBP and was slightly inferior to the original ADSW. When compared to the state-of-the-art CoopRegion [48] method, the average error rate of the MCADSW was 2.3% higher. However, the execution speed of the MCADSW

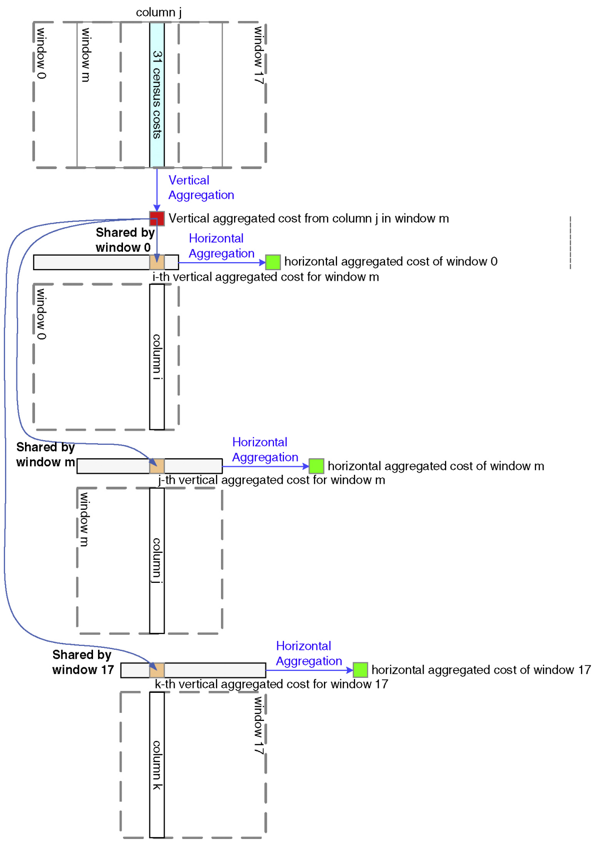

column j
window 0
window m
31 census costs
window 17
Vertical Aggregation
Vertical aggregated cost from column j in window m
Shared by window 0
Horizontal Aggregation
horizontal aggregated cost of window 0
i-th vertical aggregated cost for window m
window 0
column i
Shared by window m
Horizontal Aggregation
horizontal aggregated cost of window m
j-th vertical aggregated cost for window m
window m
column j
Shared by window 17
Horizontal Aggregation
horizontal aggregated cost of window 17
k-th vertical aggregated cost for window 17
window 17
column k

Fig. 7. Partial column reuse in cost aggregation.

was at least 10 times faster than CoopRegion, 30 times faster than the original ADSW, and near two times faster than the RealTimeBP. This implies the MCADSW is likely to have a much lower computation complexity than the compared high-performance disparity estimation algorithms. Only SSF + MF, EffectiveAggr, and RealDP were faster than the MCADSW. However, SSF + MF and, EffectiveAggr had significantly higher average error rate than the MCADSW as well.

IV. BANDWIDTH REDUCTION TECHNIQUES FOR MCADSW ARCHITECTURE

Reducing bandwidth requirement is important because available bandwidth is limited. We proposed PCR and AREW techniques to reduce the bandwidth requirement.

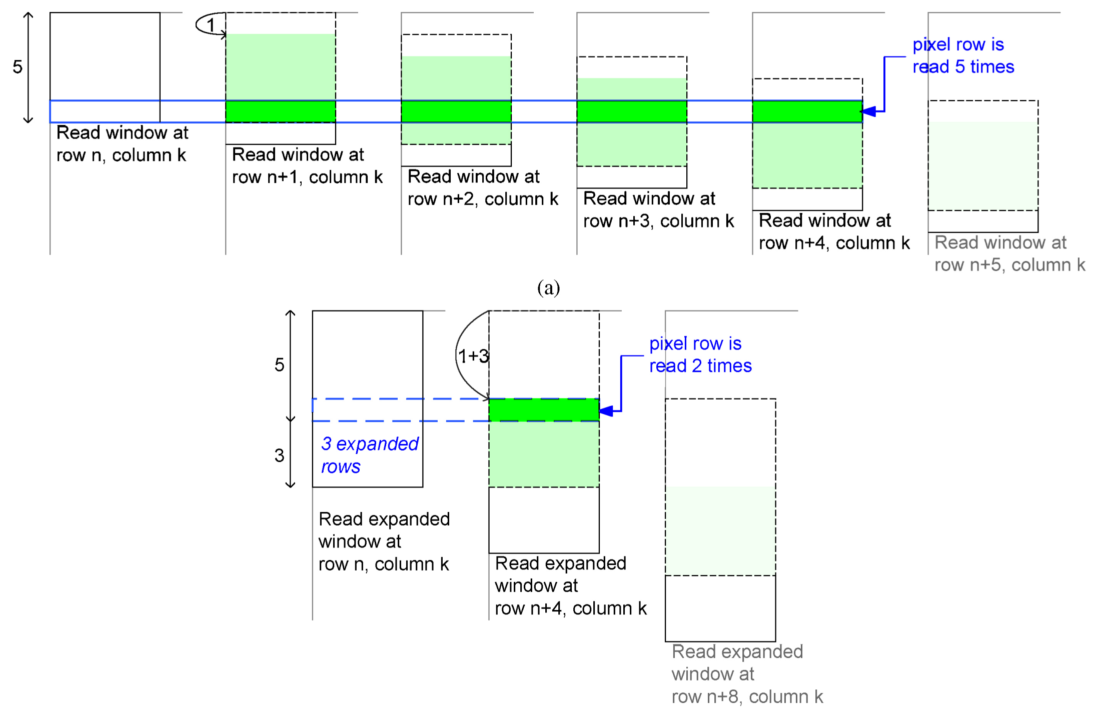

5
1
pixel row is read 5 times
Read window at row n, column k
Read window at row n+1, column k
Read window at row n+2, column k
Read window at row n+3, column k
Read window at row n+4, column k
Read window at row n+5, column k
(a)
5
3
1+3
pixel row is read 2 times
3 expanded rows
Read expanded window at row n, column k
Read expanded window at row n+4, column k
Read expanded window at row n+8, column k
(b)

Fig. 8. Example of access count reduction with expanded window. (a) Without expanded rows. (b) With three expanded rows.

A. Partial Column Reuse

The PCR reuses the data in each column to reduce the memory bandwidth and computation requirements. A column is usually a part of multiple horizontally overlapped windows. Therefore, the data of each column can be shared by the computation of the final result for these windows. The data could be original pixel data or temporary intermediate results. By storing these data, memory accesses and computations can be reduced. As a result, each column is only read and computed once.

Fig. 6 illustrates how the PCR is applied to the mini-census generation. The pixels in column $x = n$ contribute to the generation of three mini-censuses. Fig. 7 illustrates how a vertical aggregated cost is shared by 18 horizontally overlapped aggregation windows. This reduced the read count of a pixel column from 18 to 1 for these 18 windows. Since PCR can reuse computation as well, PCR may also be applied to other types of implementations, such as processor-based, DSP-based, GPU-based, and FPGA-based, to reduce computation requirement.

B. Access Reduction With Expanded Window

The AREW reduces the bandwidth requirement by deliberately expanding the size of the read window. The expanded window reduces the read count of a pixel by reducing the number of overlapping windows containing this pixel. We will explain this by using an example of vertically expanded window shown in Fig. 8. Note that we have ignored considering horizontal overlap for the sake of clarity. In this example, the original window size is 5×5 pixels and the number of vertical expanded row is 3.

Fig. 8(a) illustrates how windows are overlapped vertically without expanding rows. The first window is located at row n and column k. When the window changes the row position at the end of a horizontal scan, the new window would be vertically overlapped with the old window. As a result, the second window is located at row $n + 1$ and column k. The position of the first window is shown by the box with dashed line. The overlapped region is marked by darker color. Since we only buffer the pixel data within the read window due to cost consideration, the vertically overlapped region must be re-accessed. Consequently, the access count of a pixel is determined by the number of overlapping window containing this pixel. The maximal access count of a pixel in this case is five as shown in the figure.

Fig. 8(b) illustrates the case with expanded window. With the expanded rows, the vertically enlarged window would result in farther vertical jump distance when a row change happens. As a result, the second window is located at $n + 4$. The maximal access count of each pixel is only 2, which is much smaller than in the case without expanded window. Horizontal expansion also reduces the access count in the same way as the vertical expansion. If we could enlarge the window to the size of the image, the read count of each pixel would be only one. However, expanding the window would also require larger internal storage size and more hardware resource. Therefore, the number of expanded row and column should be carefully selected. In our case, the number of expanded row and the number of expanded column are both 17. The AREW is applied to the mini-census generation and weight generation.

The bandwidth requirement to external frame memory can be estimated based on the read count of each pixel.

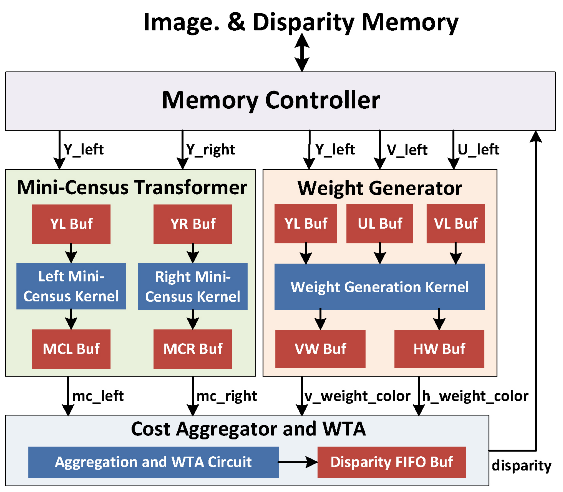


Fig. 9. Block diagram of the MCADSW.

The read count of a pixel is determined by the number of times it is overlapped by mini-census transform windows and aggregation windows. In a direct implementation without any data reuse, a pixel is overlapped by 3 mini-census transform windows in the horizontal direction, 4 mini-census transform windows in the vertical direction, 31 aggregation windows in the horizontal direction and 31 aggregation windows in the vertical direction. The read data width of a pixel in the mini-census transform is one byte, whereas the read data width of a pixel in the aggregation is three bytes. As a result, the total bandwidth requirement for a CIF size base image at 30 frames/s is about $((7 \times 31 \times 31) \times 1\,\text{byte} + 31 \times 31 \times 3\,\text{byte}) \times (352 \times 288) \times 30$ frames/s $= 27.22$ GB/s. neglecting the boundary case. If we assume the pixel data read for the weight generation already included the pixel data read for the mini-census transform, the total bandwidth can be reduced to $(31 \times 31 \times 3\,\text{byte}) \times (352 \times 288) \times 30$ frames/s = 8.17 GB/s. After applying the PCR and AREW bandwidth reduction techniques, the average read count of a pixel can be reduced to 5.17 times. The bandwidth requirement can therefore be reduced to $5.17 \times 3\,\text{byte} \times (352 \times 288) \times 30$ frames/s = 44.99 MB/s. The proposed bandwidth reduction can also be applied to other aggregation-based stereo matching architectures to reduce their bandwidth requirement.

V. Proposed Real-Time Architecture for MCADSW

A. Architecture Overview

Fig. 9 shows the architecture of the MCADSW. The architecture consists of a memory controller, mini-census transformer, weight generator, and a cost aggregator and WTA module. The details of each module are explained in the following subsections.

B. Mini-Census Transformer

Fig. 10 shows the architecture of one of the two (left and right) identical mini-census transform units. Each unit consists of an input buffer, a mini-census kernel, and a mini-census buffer. The input buffer stores the input image data read from the external image. The input data are first packed into data words and stored into the word buffers. Once the data are ready for mini-census transform, the data in the input buffer are read into the mini-census kernel. The center pixel is stored in the register and compared with its surrounding 6 pixels in the mini-census template. The comparison result, the 6-bit mini-census bitstream, is then written into the mini-census buffer.

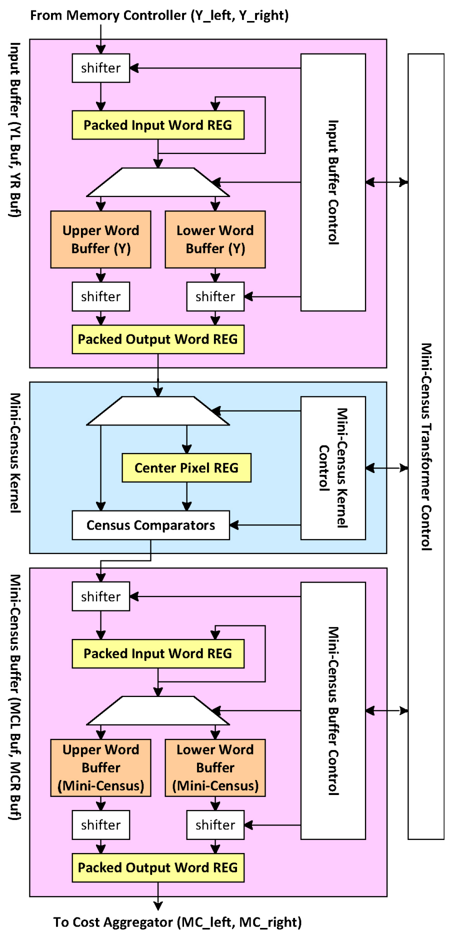

From Memory Controller (Y_left, Y_right)
Input Buffer (YL Buf, YR Buf)
shifter
Packed Input Word REG
Upper Word Buffer (Y)
Lower Word Buffer (Y)
shifter
shifter
Packed Output Word REG
Input Buffer Control
Mini-Census Kernel
Center Pixel REG
Census Comparators
Mini-Census Kernel Control
Mini-Census Buffer (MCL Buf, MCR Buf)
shifter
Packed Input Word REG
Upper Word Buffer (Mini-Census)
Lower Word Buffer (Mini-Census)
shifter
shifter
Packed Output Word REG
Mini-Census Buffer Control
Mini-Census Transformer Control
To Cost Aggregator (MC_left, MC_right)

Fig. 10. Architecture of the mini-census transformer.

C. Weight Generator

Fig. 11 shows the architecture of the weight generator. The architecture is similar to the mini-census transformer. However, there are three sets of input buffer because the weight generation needs all three color components. The mini-census kernel is replaced by a weight generation kernel which reads the input pixels column by column to generate the vertical weight. The color distances between the center pixel and others are computed in the Manhattan color distance computer. Once the color distance is available, it is used to look up the corresponding weight from the weight table. During the column by column read from the input buffer, the center pixel of each column is also stored in the horizontal row buffer. After the vertical weight is generated, the horizontal weight is generated from the pixel data in the horizontal row buffer. This avoids reading the input buffer again during the horizontal weight generation. The vertical weight buffer

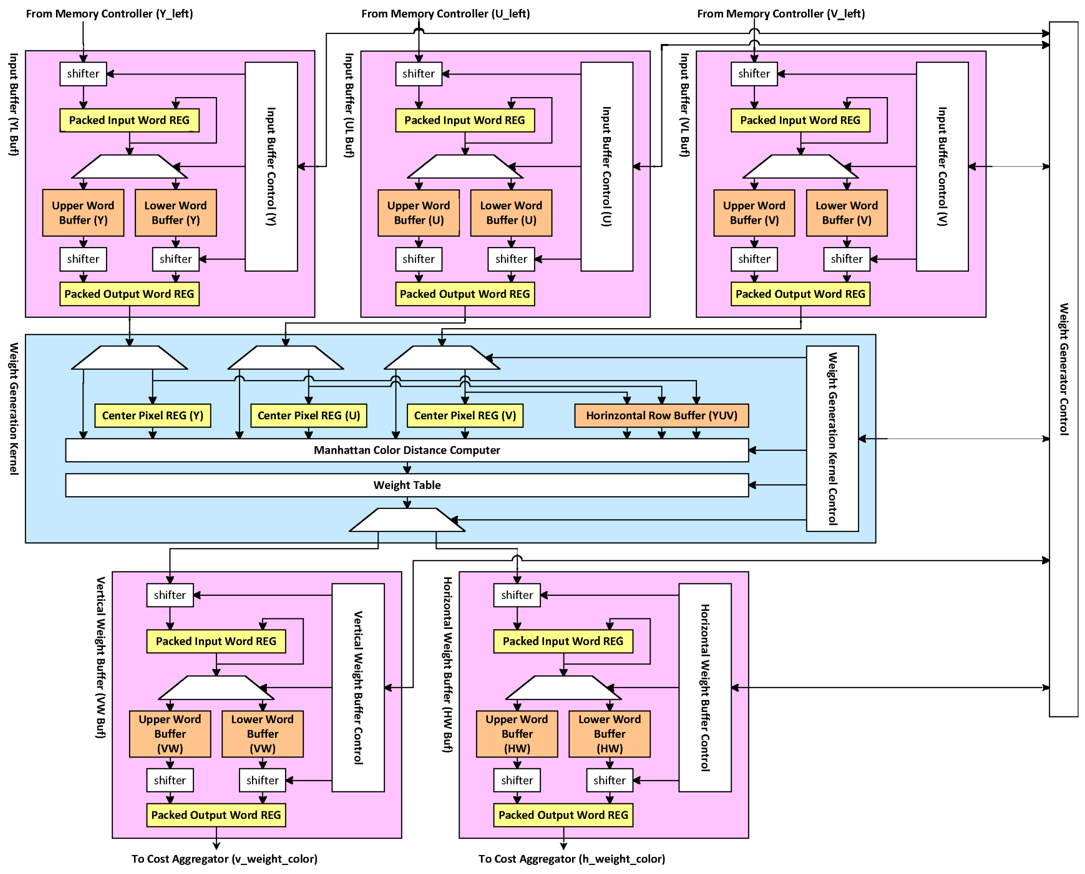

From Memory Controller (Y_left)
From Memory Controller (U_left)
From Memory Controller (V_left)
Input Buffer (YL Buf)
Input Buffer (UL Buf)
Input Buffer (VL Buf)
shifter
Packed Input Word REG
Upper Word Buffer (Y)
Lower Word Buffer (Y)
Packed Output Word REG
Input Buffer Control (Y)
Input Buffer Control (U)
Input Buffer Control (V)
Weight Generation Kernel
Center Pixel REG (Y)
Center Pixel REG (U)
Center Pixel REG (V)
Horizontal Row Buffer (YUV)
Manhattan Color Distance Computer
Weight Table
Weight Generation Kernel Control
Weight Generator Control
Vertical Weight Buffer (VW Buf)
Horizontal Weight Buffer (HW Buf)
Vertical Weight Buffer Control
Horizontal Weight Buffer Control
To Cost Aggregator (v_weight_color)
To Cost Aggregator (h_weight_color)

Fig. 11. Architecture of the weight generator.

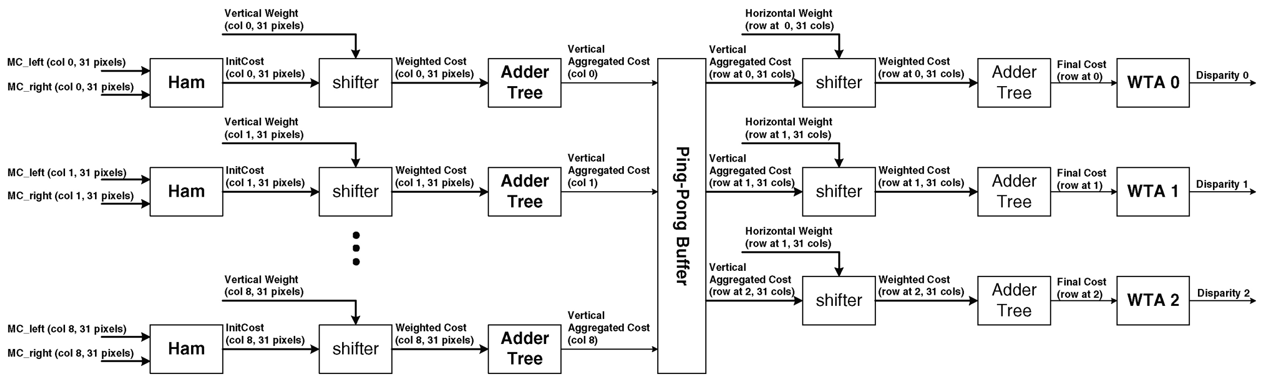

Vertical Weight (col 0, 31 pixels)
MC_left (col 0, 31 pixels)
MC_right (col 0, 31 pixels)
Ham
InitCost (col 0, 31 pixels)
shifter
Weighted Cost (col 0, 31 pixels)
Adder Tree
Vertical Aggregated Cost (col 0)
Ping-Pong Buffer
Horizontal Weight (row at 0, 31 cols)
Vertical Aggregated Cost (row at 0, 31 cols)
Weighted Cost (row at 0, 31 cols)
Final Cost (row at 0)
WTA 0
Disparity 0
WTA 1
Disparity 1
WTA 2
Disparity 2
Vertical Weight (col 8, 31 pixels)
MC_left (col 8, 31 pixels)
MC_right (col 8, 31 pixels)

Fig. 12. Architecture of the cost aggregator and WTA.

and horizontal weight buffer store the resulting vertical and horizontal weight.

D. Cost Aggregator and WTA

Fig. 12 shows the architecture of the cost aggregator and WTA. The mini-census cost computation and vertical cost aggregation are shown to the left of the ping-pong buffer, whereas the horizontal cost aggregation and WTA are shown to the right. The census costs between the left and right mini-census bitstreams are computed by the hamming distance computation unit. To increase processing speed, each unit computes the census cost of all the pixels within a column in parallel. After the census costs are available, they are multiplied by the vertical weights and summed together to give the vertical aggregated cost. The vertical aggregated costs are stored in the ping-pong buffer. The ping-pong buffer is scheduled as shown in Fig. 13 to ensure that the aggregation can be performed continuously without any pause. The ping-pong buffer outputs 33 vertical aggregated costs to the three shifters used for horizontal aggregation. We have applied the PCR technique so that the first 31 costs in the 33 costs are sent to the first shifter, the second 31 costs are sent to the second shifter, and the third 31 costs are sent to the third shifter. This reduces the output bandwidth requirement of the ping-pong buffer. Once the final matching costs are available, they are compared with the current minimal final costs in the WTA modules. After the cost of all the disparities are compared, the disparity with the minimal cost is the final output disparity.

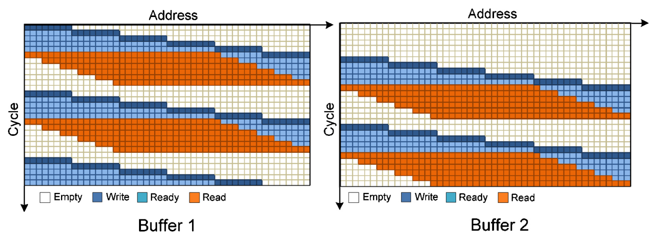


Fig. 13. Schedule of the ping-pong buffer.

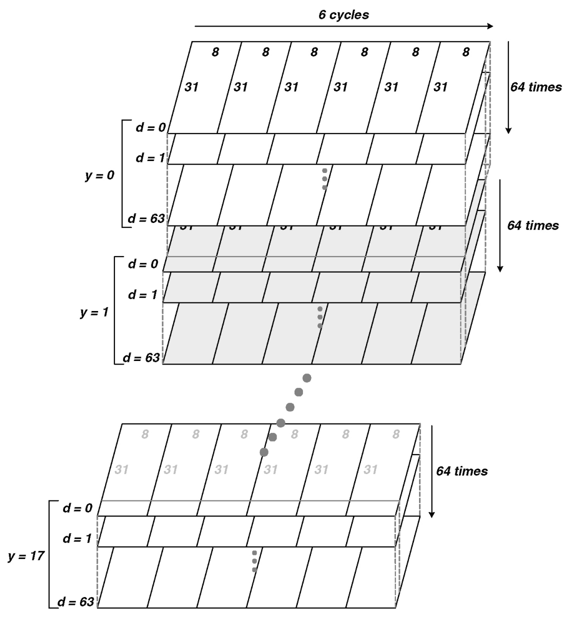


Fig. 14. Processing schedule of the cost aggregator.

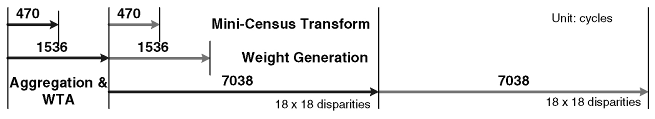


Fig. 15. Schedule of the MCADSW architecture.

Fig. 14 illustrates the processing order of the cost aggregator. The cost aggregator is capable of processing eight columns of 31 pixels simultaneously. The initial pipeline delay is 7 cycles. After the initial pipeline delay, it takes 6 cycles to process all 31×48 pixels to give 18 final matching costs. These 18 matching costs are of the same disparity. After the final matching costs of a disparity are computed and compared, the final matching costs of the next disparity are computed and compared. Once the final disparity of an 18-pixel row is determined, the cost aggregator and WTA start processing the next 18-pixel row. For example, the disparity of $y = 0$ is estimated first, then the disparity of $y = 1$ is estimated. After $(7 + 6 \times 64) \times 18 = 7038$ cycles, the disparity of an 18×18 block are determined.

E. Memory Controller

The memory controller interfaces to external memory and arbitrates the external memory access requests from the mini-census transformer, weight generator, and cost aggregator and WTA. The data port width between the image memory and the memory interface is 32-bit. The arbitration is a hybrid of round-robin and fixed priority strategy. The depth first-in-first-out always has the highest priority due to the high penalty of suspending of the cost aggregator and WTA. The priority of the mini-census transformer and weight generator are determined by round-robin.

TABLE III
CORE CHARACTERISTICS OF THE PROPOSED MCADSW

Technology	UMC 90 nm
Max Clock Rate	95 MHz
Equivalent Gate-Count (Excluding Memories)	562 642
Memory Size	21.3 kB
Image Size	352×288 (CIF)
Disparity Range	64
Maximal Frame Rate	42 frames/s @ 95 MHz

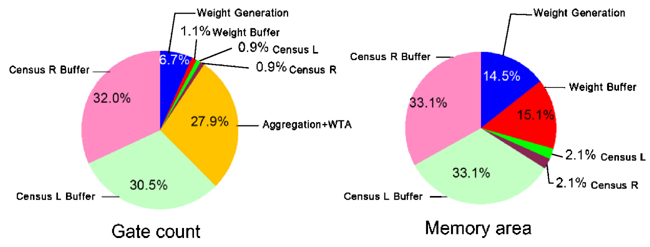


Fig. 16. Percentage of the combinational gate count and memory area.

F. Scheduling

Fig. 15 illustrates the scheduling of the MCADSW architecture. The cost aggregator starts processing data after all the 31×48 mini-censuses and weights are available. This takes 470 cycles to prepare the censuses and 1536 cycles to prepare the weights. Since the cost aggregator and WTA take 7038 cycles to finish, the mini-census transform and weight generation of the next 18×18 block can be performed at the same time. Based on this scheduling, it takes approximately 2.25 million cycles to complete the disparity estimation of all the 320 18×18 blocks in a CIF sized stereo image pair.

VI. IMPLEMENTATION RESULT

A. Gate Count and Memory Size

The proposed MCADSW architecture was synthesized using United Microelectronics Corporation (UMC) 90 nm standard cells. Table III lists the core characteristics of the synthesized design. The total equivalent gate-count is about 563k excluding the memories and the maximum operation frequency is 95 MHz. The equivalent gate-count and the memory area distributions are shown in Fig. 16. The total gate-count was dominated by the cost aggregator and WTA, census left buffer, and census right buffer. This was due to the high-computation resource requirement and complex demultiplexing circuits. The memory area was dominated by the weight generation, weight buffer, census left buffer, and census right buffer.

TABLE IV
SPEED COMPARISON OF DIFFERENT IMPLEMENTATIONS

Design	Implementation	Image Size	Disparity Range	frames/s	MDE/s
Proposed MCADSW	**UMC 90 nm Std. Cell**	**352 × 288**	**64**	**42**	**272.5**
TrellisDP [41]	Xilinx Virtex II Pro-100	320 × 240	128	30	294
HBP [43]	Xilinx Virtex II Pro-100 × 2	320 × 240	32	30	73.7
EffectAggr [45]	Intel Core 2 Duo 2.14 GHz	463 × 370	75	1.67	18.9
RealDP [34]	AMD AthlonXP 2800	384 × 288	100	18.9	209
CBiased [36]	Nvidia Geforce 7900	512 × 512	96	24	605
SepLaplacian [37]	Nvidia Geforce 7900	256 × 256	96	87	547
RealTimeBP [42]	Nvidia Geforce 7900	320 × 240	16	16	19.6
RealTimeGPU [38]	ATI Radeon 9800	320 × 240	32	22	53.0
ReliableGPU [35]	ATI Radeon 9800	N.A.	N.A.	16.6	N.A.
GradientGuided [46]	ATI Radeon 9800XT	512 × 384	40	14.7	117

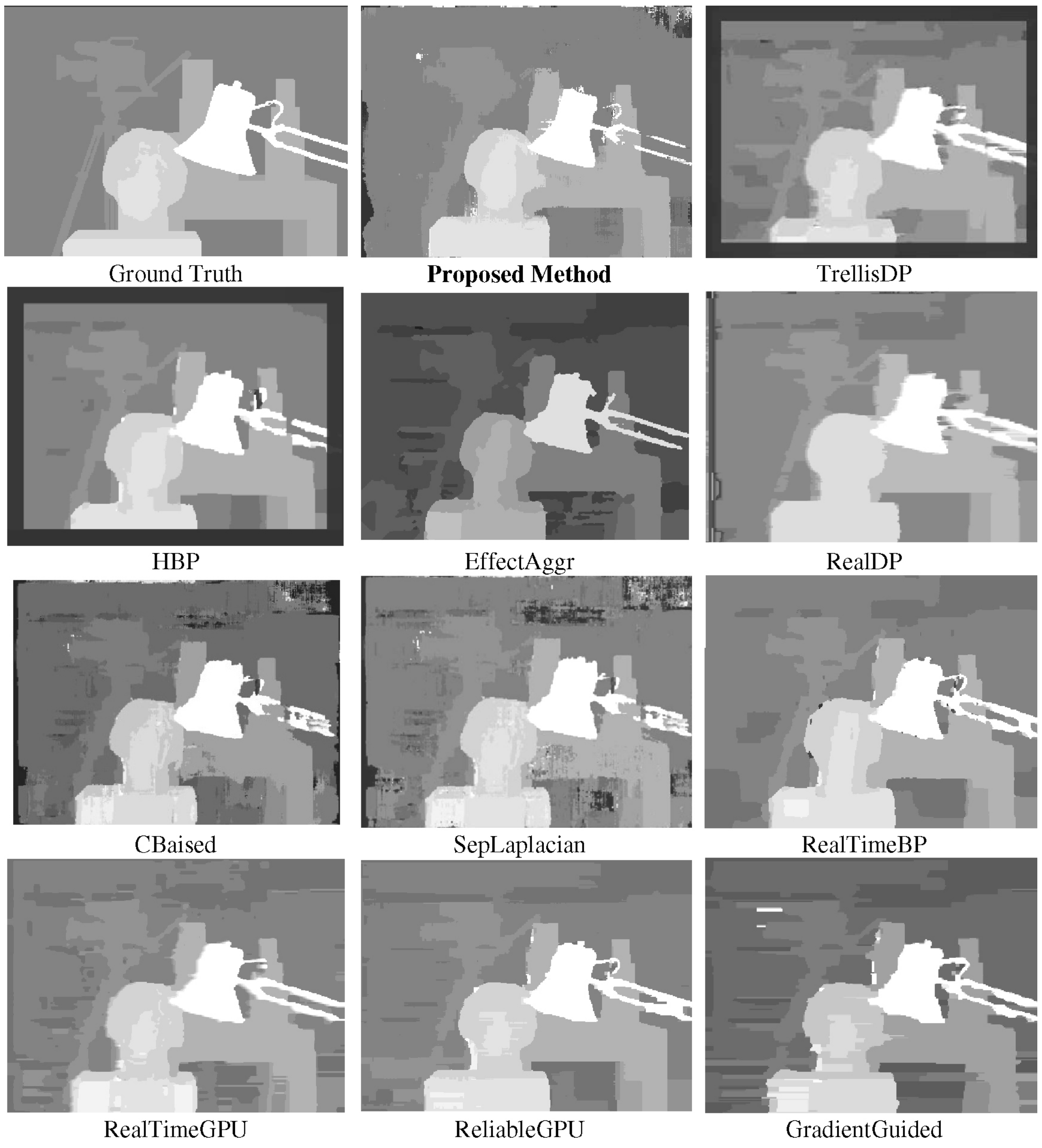


Fig. 17. Disparity map of different implementations.

TABLE V
PERFORMANCE COMPARISON OF DIFFERENT IMPLEMENTATIONS

Design	*Tsukuba*	*Venus*	*Teddy*	*Cones*	*Saw Tooth*	*Map*
Proposed	**2.80**	**0.64**	**13.7**	**10.1**	**2.11**	**3.21**
TrellisDP [41]	2.63	3.44	N.A.	N.A.	1.88	0.91
HBP [43]	2.85	1.92	N.A.	N.A.	6.25	6.45
EffectAggr [45]	2.11	4.75	15.2	12.6	N.A.	N.A.
RealDP [34]	2.85	6.42	N.A.	N.A.	6.25	6.45
CBiased [36]	4.77	10.2	N.A.	N.A.	0.82	0.65
SepLaplacian [37]	13.0	19.4	N.A.	N.A.	N.A.	N.A.
RealTimeBP [42]	3.40	1.90	13.2	11.6	N.A.	N.A.
RealTimeGPU [38]	4.22	2.98	14.4	13.7	N.A.	N.A.
ReliableGPU [35]	1.36	1.09	N.A.	N.A.	2.35	0.55
GradientGuided [46]	2.48	3.91	N.A.	N.A.	1.63	0.73

B. Performance Comparison

Tables IV and V compare the disparity estimation speed and performance of the MCADSW architecture with other existing high-performance real-time implementations quantitatively. In Table V, the performance was evaluated using Middleburry's stereo image pairs and their evaluation method [2]. The error rate is the overall error rate with the tolerance of one disparity level. In addition to the quantitative performance evaluation, we also included the disparity maps generated by different implementations in Fig. 17 for qualitative performance comparison.

From the speed perspective, the TrellisDP, CBiased, and SepLaplacian outperformed the MCADSW architecture in terms of the MDE/s. The TrellisDP was faster because it adopted a fully parallel systolic array architecture with 128 processing elements (PEs). Their systolic architecture maximized the utilization of the PEs to achieve a processing speed of 294 MDE/s. However, the systolic architecture would require very high bandwidth and large storage to keep the 128 PEs working without idling. The CBiased and SepLaplacian were at least two times faster than the MCADSW because they were implemented using high-performance programmable GPUs. These high-performance GPU had extremely high bandwidth and computation hardware resource available. For instance, Nvidia's Geforce 7800GTX GPU had 256 MB of GDDR3 dynamic random access memory (DRAM) clocked at 600 MHz, with a data port of 256-bit, the maximum available peak bandwidth can reach up to 38.4 GB/s [49]. Together with an operating clock of 430 MHz and 8 vertex and 16-pixel shaders, it is reasonable for GPU-based implementation to achieve such high-processing speeds. In contrast, the MCADSW architecture required much lower clock rate, less bandwidth, and smaller silicon area. This makes the MCADSW more applicable to embedded vision applications. As to the disparity estimation performance, only the TrellisDP was comparable to the MCADSW in terms of average error rate. The average error rate excluding *Teddy* and *Cones* stereo image pairs was 2.19% and 2.21% for the MCADSW and TrellisDP, respectively. The performances of the Cbiased and SepLaplacian were both inferior. This can also be observed in the disparity maps shown in Fig. 17. The disparity map of the MCADSW had more accurate depth-discontinuity than others in most regions except in the camera region. The reason for the camera region being blurry was probably due to the removal of the proximity weight. The MCADSW provided high-disparity estimation speed and performance that was comparable to other high-performance real-time implementations. In addition, the MCADSW required less bandwidth and would therefore be more suitable for embedded vision systems.

VII. CONCLUSION

This paper presented a hardware-friendly high-performance disparity estimation algorithm, the MCADSW, and its corresponding architecture for real-time stereo matching. The proposed hardware-friendly simplifications not only made the MCADSW more hardware-friendly, but also reduced the execution time of the MCADSW algorithm by 61.3%. In the design of the MCADSW architecture, we proposed the PCR and AREW techniques to significantly reduce bandwidth requirement. The proposed architecture was synthesized using UMC 90 nm standard cells. At the operation frequency of 95 MHz, the proposed architecture can achieve 42 frames/s of CIF size disparity map with 64 disparity levels. The equivalent gate-count and total memory size are 562k and 21.3 kB, respectively.

REFERENCES

[1] M. Brown, D. Burschka, and G. Hager, "Advances in computational stereo," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 8, pp. 993–1008, Aug. 2003.

[2] D. Scharstein and R. Szeliski, "A Taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. J. Comput. Vision*, vol. 47, nos. 1–3, pp. 7–42, Apr. 2002.

[3] K.-J. Yoon and I.-S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650–656, Apr. 2006.

[4] M. Gerrits and P. Bekaert, "Local stereo matching with segmentation-based outlier rejection," in *Proc. 3rd Canadian Conf. Comput. Robot Vision*, Jun. 2006, p. 66.

[5] F. Tombari, S. Mattoccia, and L. Di Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," in *Lecture Notes in Computer Science*, vol. 4872, Berlin, Germany: Springer, Dec. 2007, pp. 427–438.

[6] T. Kanade, M. Okutomi, and T. Nakahara, "A multiple-baseline stereo method," in *Proc. Defense Advanced Research Projects Agency (DARPA) Image Understanding Workshop*, 1992, pp. 409–426.

[7] S. Kimura, T. Kanade, H. Kano, A. Yoshida, E. Kawamura, and K. Oda, "CMU video-rate stereo machine," in *Proc. Mobile Mapping Symp.*, 1995, pp. 9–12.

[8] K. Konolige, "Small vision systems: Hardware and implementation," in *Proc. 8th Int. Symp. Robotics Res.*, Oct. 1997, pp. 203–212.

[9] J. Woodfill and B. Von Herzen, "Real-time stereo vision on the PARTS reconfigurable computer," in *Proc. IEEE Workshop FPGAs Custom Comput. Mach.*, 1997, pp. 240–250.

[10] P. Corke and P. Dunn, "Real-time stereopsis using FPGAs," in *Proc. IEEE Region Ten Annu. Conf. (TENCON): Speech Image Tech. Comput. Telecommun.*, Apr. 1997, pp. 235–238.

[11] S. Kimura, T. Shinbo, H. Yamaguchi, E. Kawamura, and K. Nakano, "A convolver-based real-time stereo machine (SAZAN)," in *Proc. Conf. Comput. Vision Pattern Recognit.*, vol. 1. 1999, pp. 457–463.

[12] M. Hariyama, N. Yokoyama, M. Kameyama, and Y. Kobayashi, "FPGA implementation of a stereo matching processor based on window-parallel-and-pixel-parallel architecture," in *Proc. 48th Midwest Symp. Circuit Syst.*, vol. 2. Aug. 2005, pp. 1219–1222.

[13] M. Gong and Y.-H. Yang, "Near real-time reliable stereo matching using programmable graphics hardware," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, vol. 1. Jun. 2005, pp. 924–931.

[14] M. Hariyama, H. Sasaki, and M. Kameyama, "Architecture of a stereo matching VLSI processor based on hierarchically parallel memory access," *Inst. Electron., Informat. Commun. Eng. (IEICE) Trans. Info. Syst.*, vol. E88-D, no. 7, pp. 1486–1491, 2005.

[15] D. Masrani and W. MacLean, "A real-time large disparity range stereo-system using FPGAs," in *Proc. IEEE Conf. Comput. Vision Syst.*, Jan. 2006, p. 13.

[16] N. Chang, T.-M. Lin, T.-H. Tsai, Y.-C. Tseng, "Real-time DSP implementation on local stereo matching," in *Proc. IEEE Conf. Multimedia Expo*, Jul. 2007, pp. 2090–2093.

[17] J. Banks and P. Corke, "Quantitative evaluation of matching methods and validity measures for stereo vision," *Int. J. Robot. Res.*, vol. 20, no. 7, pp. 512–532, Jul. 2001.

[18] H. Hirschmuller and D. Scharstein, "Evaluation of cost functions for stereo matching," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, Jun. 2007, pp. 1–8.

[19] R. Zabih and J. Woodfill, "Non-parametric local transforms for computing visual correspondence," in *Proc. 3rd Eur. Conf. Comput. Vision*, 1994, pp. 150–158.

[20] H. Hirschmuller, "Accurate and efficient stereo processing by semi-global matching and mutual information," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, vol. 2. Jun. 2005, pp. 807–814.

[21] M. Bleyer, S. Chambon, U. Poppe, and M. Gelautz, "Evaluation of different methods for using color information in global stereo matching approaches," in *Proc. Congr. Int. Soc. Photogrammetry Remote Sensing*, Jul. 2008, pp. 63–68.

[22] N. Chang, Y.-C. Tseng, and T.-S. Chang, "Analysis of color space and similarity measure impact on stereo block matching," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, Dec. 2008, pp. 926–929.

[23] M. Okutomi and T. Kanade, "A locally adaptive window for signal matching," *Int. J. Comput. Vision*, vol. 7, pp. 143–162, Jan. 1992.

[24] T. Kanade and M. Okutomi, "A stereo matching algorithm with an adaptive window: Theory and experiment," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 16, no. 9, pp. 920–930, Sep. 1994.

[25] M. Hariyama, T. Takeuchi, and M. Kameyama, "Reliable stereo matching for highly-safe intelligent vehicles and its VLSI implementation," in *Proc. IEEE Intell. Vehicles Symp.*, Oct. 2000, pp. 128–133.

[26] R. Arnold, "Automated stereo perception," Artif. Intell. Laboratory, Stanford Univ., Stanford, CA, Tech. Rep. AIM-351, 1983.

[27] A. F. Bobick and S. S. Intille, "Large occlusion stereo," *Int. J. Comput. Vision*, vol. 33, no. 3, pp. 181–200, Sep. 1999.

[28] S.-B. Kang, R. Szeliski, and J. Chai, "Handling occlusions in dense multi-view Stereo," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, vol. 1. 2001, pp. 103–110.

[29] O. Veksler, "Stereo matching by compact windows via minimum ratio cycle," in *Proc. IEEE Int. Conf. Comput. Vision*, vol. 1. Jul. 2001, pp. 540–547.

[30] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Classification and evaluation of cost aggregation methods for stereo correspondence," in *Proc. IEEE Int. Conf. Comput. Vision Pattern Recognit.*, Jun. 2008, pp. 1–8.

[31] *Digiclops Product Datasheets*, Point Grey Research Inc., Richmond, BC, Canada, 1997 [Online]. Available: http://www.ptgrey.com/products/digiclops/Digiclops.pdf

[32] *Triclops SDK*, Point Grey Research Inc., Richmond, BC, Canada, 2001 [Online]. Available: http://www.ptgrey.com/products/triclopsSDK/index.asp

[33] H. Hirschmuller, "Improvements in real-time correlation-based stereo vision," in *Proc. IEEE Workshop Stereo Multi-Baseline Vision*, Dec. 2001, pp. 141–148.

[34] H. Hirschmuller, P. R. Innocent, and J. Garibaldi, "Real-time correlation-based stereo vision with reduced border errors," *Int. J. Comput. Vision*, vol. 47, nos. 1–3, pp. 229–246, Nov. 2004.

[35] S. Forstmann, Y. Kanou, O. Jun, S. Thuering, and A. Schmitt, "Real-time stereo by using dynamic programming," in *Proc. Comput. Vision Pattern Recognit. Workshop Real-Time 3-D Sensor Their Use*, Jun. 2004, p. 29.

[36] G. Minglun and Y. Yee-Hong, "Near real-time reliable stereo matching using programmable graphics hardware," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, vol. 1. Jun. 2005, pp. 924–931.

[37] L. Jiangbo, G. Lafruit, and F. Catthoor, "Fast variable center-biased windowing for high-speed stereo on programmable graphics hardware," in *Proc. IEEE Int. Conf. Image Process.*, vol. 6. Oct. 2007, pp. 568–571.

[38] L. Jiangbo, S. Rogmans, G. Lafruit, and F. Catthoor, "Real-time stereo correspondence using a truncated separable Laplacian kernel approximation on graphics hardware," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2007, pp. 1946–1949.

[39] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, "High-quality real-time stereo using adaptive cost aggregation and dynamic programming," in *Proc. 3rd Int. Symp. 3-D Data Process., Vis., Transmission*, Jun. 2006, pp. 798–805.

[40] O. Faugeras, B. Hotz, H. Matthieu, T. Vieville, Z. Zhang, P. Fua, E. Theron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy, "Real time correlation-based stereo: Algorithm, implementations and applications," Institut National de Recherche en Informatique et Automatique (INRIA), Tech. Rep. 2013, 1993.

[41] H. K. Nishihara, "Real-time stereo and motion-based figure-ground discrimination and tracking using LOG sign-Correlation," in *Proc. 27th Asilomar Conf. Signals, Syst., Comput.*, 1993, pp. 95–100.

[42] S. Park, H. Jeong, "Real-time stereo vision FPGA chip with low-error rate," in *Proc. Int. Conf. Multimedia Ubiquitous Eng.*, Apr. 2007, pp. 751–756.

[43] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister, "Real-time global stereo matching using hierarchical belief propagation," in *Proc. Brit. Mach. Vision Conf.*, 2006, pp. 989–998.

[44] S. Park, C. Chen, and H. Jeong, "VLSI architecture for MRF-based stereo matching," *Lecture Notes in Computer Science*, vol. 4599, Berlin, Germany: Springer, Aug. 2007, pp. 55–64.

[45] T.-H. Tsai, N. Chang, and T.-S. Chang, "Data reuse analysis of local stereo matching," in *Proc. Int. Symp. Circuits Syst.*, May 2008, pp. 812–815.

[46] F. Tombari, S. Mattoccia, L. Di Stefano, and E. Addimanda, "Near real-time stereo based on effective cost aggregation," in *Proc. Int. Conf. Comput. Vision Pattern Recognit.*, 2008, pp. 1–4.

[47] M. Gong and R. Yang, "Image-gradient-guided real-time stereo on graphics hardware," in *Proc. 5th Int. Conf. 3-D Digital Imaging Modeling*, 2005, pp. 548–555.

[48] Z. Wang and Z. Zheng, "A region-based stereo matching algorithm using cooperative optimization," in *Proc. IEEE Conf. Comput. Vision Pattern Recognit.*, 2008, pp. 1–8.

[49] S. Wattson. (2005 June 22). "NVIDIAs GeForce 7800 GTX graphics processor," in *The Tech Report* (Online). Available: http://techreport.com/articles.x/8466/5

Nelson Yen-Chung Chang (S'06) received the B.S. degree in electrical engineering from National Tsing-Hua University, Hsinchu, Taiwan, in 2000, and the M.S. and Ph.D. degrees in electronic engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 2002 and 2009, respectively.

He is currently an Engineer with the Division of Intelligent Robotics Technology, Mechanical and Systems Research Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan. His current research interests include real-time vision system, robot vision, advanced robotic embedded systems, and vision algorithms.

Tsung-Hsien Tsai (S'08) received the B.S. degree in electrical engineering from National Tsing-Hua University, Hsinchu, Taiwan, in 2006, and the M.S. degree in electronic engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 2008.

He is currently an Engineer with the Wireless Communication Technology/Chip Design Division, MediaTek, Hsinchu, Taiwan. His current research interests includes WiMAX and system intregration.

Bo-Hsiung Hsu was born in Taiwan, in 1985. He received the B.S. degree in electrical engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 2008, where he is currently pursuing the M.S. degree.

His current research interests include high-resolution depth estimation for stereo, image processing, and very large scale integration design.

Yi-Chun Chen was born in Taiwan, in 1986. He received the B.S. degree in electrical engineering from National Chiao-Tung University, Hsinchu, Taiwan, in 2008, where he is currently pursuing the M.S. degree.

His current research interests include 2-D video to 3-D video conversion for stereo, image processing, and very large scale integration design.

Tian-Sheuan Chang (S'93–M'06–SM'07) received the B.S., M.S., and Ph.D. degrees in electronic engineering from National Chiao-Tung University (NCTU), Hsinchu, Taiwan, in 1993, 1995, and 1999, respectively.

He is currently an Associate Professor with the Department of Electronics Engineering, NCTU. From 2000 to 2004, he was a Deputy Manager with Global Unichip Corporation, Hsinchu, Taiwan. His current research interests include (silicon) intellectual property (IP) and system-on-a-chip design, very large scale integration signal processing, and computer architecture.