A 124 Mpixels/s VLSI Design for Histogram-Based Joint Bilateral Filtering

Yu-Cheng Tseng, *Student Member, IEEE*, Po-Hsiung Hsu, and Tian-Sheuan Chang, *Senior Member, IEEE*

Abstract—**This paper presents an efficient and scalable design for histogram-based bilateral filtering (BF) and joint BF (JBF) by memory reduction methods and architecture design techniques to solve the problems of high memory cost, high computational complexity, high bandwidth, and large range table. The presented memory reduction methods exploit the progressive computing characteristics to reduce the memory cost to 0.003%–0.020%, as compared with the original approach. Furthermore, the architecture design techniques adopt range domain parallelism and take advantage of the computing order and the numerical properties to solve the complexity, bandwidth, and range-table problems. The example design with a 90-nm complementary metal–oxide–semiconductor process can deliver the throughput to 124 Mpixels/s with 356-K gate counts and 23-KB on-chip memory.**

Index Terms—**Bilateral filtering (BF), integral histogram (IH), very-large-scale-integration (VLSI) design.**

I. Introduction

BILATERAL filtering (BF) is widely adopted in image and video processing such as denoising, texture editing and relighting tone management stylization, and optical flow estimation due to its texture preserving capabilities during processing [1]. BF, originated in [2], is to smooth an image and is defined as

$$\mathrm{BF}(I_c) = \frac{\sum_{q\in S} f(|c-q|)\, g(|I_c - I_q|)\, I_q}{\sum_{q\in S} f(|c-q|)\, g(|I_c - I_q|)} \tag{1}$$

where c is the target pixel and q is the support pixel surrounding c. The support pixel q is in a square filter window S, and its intensity I_q is in the range domain R from 0 to 255 for the gray level. In this equation, I_q values are accumulated and normalized with the space kernel f and the range kernel g. Both f and g are usually Gaussian functions with the arguments of space distance $|c-q|$ and intensity difference $|I_c - I_q|$, respectively. If q is close to c or I_q is similar to I_c, the impact of q will be raised. On the contrary, q is regarded as an outlier. Because of the additional range kernel g, BF can smooth an image without the overblurring of traditional Gaussian filtering.

Based on BF, Eiseman and Durand [3] and Petschnigg *et al.* [4] developed joint BF (JBF) to blend a pair of flash and no-flash photos into a clear one. JBF is defined as

$$\mathrm{JBF}(J_c) = \frac{\sum_{q\in S} f(|c-q|)\, g(|I_c - I_q|)\, J_q}{\sum_{q\in S} f(|c-q|)\, g(|I_c - I_q|)} \tag{2}$$

where I is a guidance image and J is another source image. Through the range kernel g, the guidance image I could identify and suppress outliers for denoising the source image J. With this characteristic, JBF has been adopted in image denosing [5] and disparity-map fusion [6], [7]. Further extending the applications of JBF, Kopf *et al.* [8] proposed the joint bilateral upsampling that employs high-resolution I to enlarge low-resolution J for various image processing, such as tone mapping, colorization, disparity maps [9], [10], demosaicing [11], and texture synthesis [12]. A variety of JBF is the adaptive support weight (ADSW), i.e., a matching cost aggregation approach, proposed by in [13] for disparity estimation in 3-D image processing. The ADSW employs the space and range kernels to deliver better disparity maps than the traditional box filter with a constant coefficient. The concept of the ADSW is further advanced in the disparity estimation algorithms in [14]–[16] and is also adopted by the developing Motion Pictures Expert Group standard, i.e., 3-D video coding [17].

As previously mentioned, both BF and JBF are effective for various applications, but their operation speed is severely limited by their nonlinear calculations. A brute-force calculation would need the computational complexity of $O(|S|^2)$ for each pixel, where $|S|^2$ is the filter window size. To speed up the calculation, several approaches have been proposed in [22]–[32]. The state-of-the-art approaches proposed in [22] and [23] can achieve constant-time complexity. Thus, this paper will adopt Porikli's constant-time approach.

Porikli's approach consists of two steps, i.e., histogram calculation for obtaining the histogram of a filter window S and 1-D convolution for convoluting the histogram with a range kernel. Since the histogram calculation occupies most of the complexity, Porikli applied the integral histogram (IH) approach [24] that accumulates histogram for each pixel by traversing the whole image and then extracts the histogram of the target filter window by adding and subtracting the accumulated histograms at four corners. The accumulated histogram is called the IH that still needs high memory cost and high bandwidth. For the example of the 1920 × 1080p resolution (i.e., HD1080p), the IH approach needs the memory cost of 829 MB and the bandwidth of 106 Gb/frame, as shown later in Table V. In addition, Porikli's approach still suffers from the high compu-

Manuscript received June 23, 2010; revised February 14, 2011, May 17, 2011, and May 19, 2011; accepted May 20, 2011. Date of publication June 09, 2011; date of current version October 19, 2011. This work was supported by the National Science Council, Taiwan, under Grant NSC-99-2221-E-009-0189. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mario S. Pattichis.

Y.-C. Tseng and T.-S. Chang are with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: tyucheng@dragons.ee.nctu.edu.tw; tschang@dragons.ee.nctu.edu.tw).

P.-H. Hsu is with MediaTek, Inc., Hsinchu 30078, Taiwan (e-mail: sadeibear@gmail.com).

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/TIP.2011.2159226

TABLE I
COMPARISON OF COMPUTATIONAL COMPLEXITY AND MEMORY COST IN THE RELATED WORK

Approach			Computational Complexity (per pixel)		Memory Cost (per frame)
	Brute-Force		All	$O(\|S\|^2)$	0
Support Pixel First	Basic		LUT Construction	$O(\|R\|)$	$4MN$
			2-D Conv. by FFT	$O(\|S\| \log \|S\|)$	
	Durand [21]	Piecewise-linear Subsampling	LUT Construction	$O(\|R\|/s_r)$	$4MN/s_s^2$
			2-D Conv. by FFT	$O(\|S\|/s_s^2 \log(\|S\|/s_s^2))$	
	Yang [22]	Piecewise-linear	LUT Construction	$O(\|R\|/s_r)$	$4MN$
			2-D Conv. by Approx. Gaussian	$O(1)$	
	Paris [25]	Bilateral Grid	LUT Construction	$O(\|R\|/s_r)$	$MN\|R\|/(s_r s_s^2)$
			3-D Conv. by FFT	$O(\|S\|\|R\|/(s_r s_s^2) \log(\|S\|\|R\|/(s_r s_s^2)))$	
Target Pixel First	Pham [29]	Separable	1-D Aggre. for Col.	$O(\|S\|)$	0
			1-D Aggre. for Row	$O(\|S\|)$	
	Basic	Histogram	Histogram Calculation	$O(\|R\|\|S\|^2)$	0
			1-D Conv.	$O(\|R\|)$	
	Huang [30]	Extended Histogram	Histogram Calculation	$O(\|R\|\|S\|)$	$\|S+E\|^2\|R\|$
			1-D Conv.	$O(\|R\|)$	
	Weiss [31]	Distributed Histogram	Histogram Calculation	$O(\|R\| \log \|S\|)$	$\|S+E\|^2\|R\|$
			1-D Conv.	$O(\|R\|)$	
	Porikli [23]	Integral Histogram	Histogram Calculation	$O(\|R\|/s_r)$	$MN\|R\|/s_r$
			1-D Conv.	$O(\|R\|/s_r)$	

M: frame height; N: frame width; $|S|$: filter window width; $|R|$: intensity range; s_s: quantization factor for S; s_r: quantization factor for R; E: extension pixel count

tational complexity of 2262 million operations for processing an HD1080p image, even if it has been accelerated by the IH approach. Moreover, the 1-D convolution needs a large range table with 256 items for the range kernel. Due to above problems, it is hard to achieve a real-time performance and thus demands very-large-scale-integration (VLSI) hardware acceleration. However, the previous VLSI implementations could not achieve high throughput for the large filtering window [18], [21] or achieve high throughput only for the small filtering window [19], [20] because they adopted the brute-force calculation of BF, instead of the acceleration approaches.

To solve aforementioned problems, this paper adopts Porikli's acceleration approach and proposes a VLSI design with memory reduction methods and architecture design techniques. The contributions of this paper are as follows: First, for the high memory cost, this paper takes advantage of the progressive computing order in the histogram calculation to reduce the memory cost from a frame to a scan line, which is 0.003%–0.020% of the original approach. Second, for the high computational complexity, this paper maximizes the parallelism of Porikli's approach in the range domain R to decrease its complexity to 0.15%. Third, for the high bandwidth, this paper exploits the timing relationship of IHs to reduce the bandwidth to 32%–36%. Fourth, for the large range table, this paper employs the symmetry and truncation properties of the range kernel to reduce the size and the number of the range table. Finally, with the aforementioned methods, this paper presents an efficient and scalable architecture. An example implementation can achieve 124 Mpixels/s with the 23-KB memory and 365-K gate counts.

The rest of this paper is organized as follows: Section II reviews the previous acceleration approaches for BF and JBF, and Section III focuses on the IH approach and points out its design challenges. Then, Section IV elaborates three memory reduction methods, and Section V describes the proposed architecture design techniques and an architecture design with all the previously mentioned methods. Finally, Section VI presents the result of an implementation example, and Section VII concludes this paper.

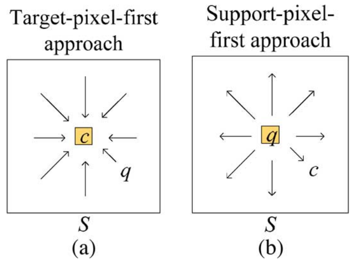


Fig. 1. Classification of acceleration approaches.

II. RELATED WORK

Various acceleration approaches for BF have been proposed and can be classified into two categories, i.e., the target-pixel-first approach and the support-pixel-first approach, according to their computational characteristics, as illustrated in Fig. 1. The target-pixel-first approach is an aggregation process that focuses on a target pixel c and accumulates its support pixels q. On the other hand, the support-pixel-first approach is a diffusion process that regards a support pixel q as a center to diffuse for its target pixels c. With the classification, the previous approaches are reviewed here, and their computational complexity and memory cost are compared in Table I.

A. Support-Pixel-First Approaches

The main idea of the support-pixel-first approaches is to convert the original nonlinear convolution to linear convolution so that the linear convolution can be accelerated by existing algorithms, such as the fast Fourier transform. To convert (1) to linear convolution, terms $g(|I_c - I_q|)I_q$ and $g(|I_c - I_q|)$ are precalculated and stored in memory devices as lookup tables

(LUT). Hence, the approaches consist of two steps, i.e., LUT construction and linear convolution. For the implementation issues, the former needs a large storage, and the later needs an efficient computation.

Durand and Dorsey [25] are the first ones to propose the support-pixel-first approach for acceleration. This approach contains two schemes, i.e., the piecewise-linear scheme and the subsampling scheme, to quantize R by factor s_r and S by factor s_s^2, respectively. Hence, the memory cost and the computational complexity can be reduced by the same factors. Based on the piecewise-linear scheme, Yang *et al.* [22] adopted a constant-time approximate Gaussian filtering for the linear convolution to achieve real-time processing by the graphical-processing-unit (GPU)-based programming.

Paris and Durand [26], [27] indicates that the piecewise-linear scheme would suffer from poor approximation on the texture's discontinuity since it cannot exactly interpolate dense results. To address that, the bilateral grid scheme is proposed to perform a 3-D convolution on $S \times R$, instead of the typical 2-D convolution only on S. However, its memory cost and computational complexity are scaled on dimension R. Following the bilateral grid scheme, Chen *et al.* [28] implemented it by the GPU-based programming to achieve real-time processing. In addition, Adams *et al.* [29] adopts the Gaussian KD tree to improve its speed.

To sum up, the support-pixel-first approaches can convert BF and JBF to linear convolution but suffer from high memory cost for LUTs. Unfortunately, the size of LUTs should be frame–scale–magnitude since their algorithms iteratively performs on the whole frame.

B. Target-Pixel-First Approaches

The main idea of the target-pixel-first approaches is to aggregate the support pixels with kernels, which need the computational complexity of $O(|S|^2)$. To accelerate it, Pham and van Vliet [30] proposed the separable BF that directly changes the original 2-D aggregation to two-step 1-D aggregation for columns and a row. Thus, it can reduce the computational complexity to $O(|S|)$, but it suffers from the axis-aligned artifact.

On the other hand, the histogram-based approaches could reduce computation without significant quality degradation. In the approaches, the space kernel f is simplified to a box filter with constant coefficient, so that (1) is rewritten as

$$\mathrm{BF}(I_c) = \frac{\sum_{q\in S} g\left(|I_c - I_q|\right) I_q}{\sum_{q\in S} g\left(|I_c - I_q|\right)} = \frac{\sum_{b\in R} g\left(|I_c - b|\right) \mathrm{hc}_c(b) b}{\sum_{b\in R} g\left(|I_c - b|\right) \mathrm{hc}_c(b)}. \quad (3)$$

Before convoluting each support pixel q with the range kernel g, the support pixels in the filter window S are classified into the pixel count histogram hc_c, whose subscript refers to the target pixel c. Fig. 2 shows the concept of the classification. According the support pixel's intensity I_q, the corresponding bin b is accumulated. For the exact result of the gray level, the number of bins N_b is set as 256. After classifying all support pixels, the histogram bin value $\mathrm{hc}_c(b)$ can refer to the number of support pixels with intensity b in S. Then, (3) can be finally calculated by 1-D convolution in the range domain R, instead of the original space

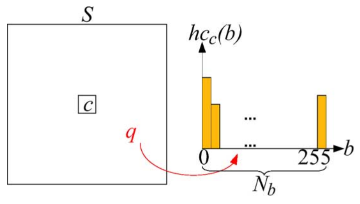


Fig. 2. Concept of histogram-based approaches.

domain S. In summary, the histogram-based approaches include two parts, i.e., histogram calculation and 1-D convolution. The key point of the histogram-based approaches is that the convolution can be decreased from the larger $|S|^2$ to $|R|$. However, the major computational complexity is $O(|R||S|^2)$ in the histogram calculation that demands other acceleration techniques.

To speed up the histogram calculation, Huang [31] proposed the extended histogram approach that calculates multiple target pixels' histograms and shares their partial histograms in runtime. Its computational complexity can be reduced to $O(|R||S|)$, but it spends extra memory cost. Based on the extended histogram approach, Weiss [32] proposed the distributed histogram approach that reassembles the histogram calculation of each row and reduces computational complexity to $O(|R|\log|S|)$. Furthermore, Porikli [23], [24] proposed the IH approach to decrease computational complexity to $O(|R|/s_r)$, which is independent of the filter window size. In addition, factor s_r quantizes the support pixel's intensity. The IH approach can be faster than the brute-force approach when $|R|/s_r$ is smaller than $|S|^2$. That implies that this approach is suitable to be applied when BF has a large filter window size. Based on the IH approach, Ju and Kang [33] modified (3) to

$$\mathrm{JBF}(J_c) = \frac{\sum_{q\in S} g\left(|I_c - I_q|\right) J_q}{\sum_{q\in S} g\left(|I_c - I_q|\right)} = \frac{\sum_{b\in R} g\left(|I_c - b|\right) \mathrm{hi}_c(b)}{\sum_{b\in R} g\left(|I_c - b|\right) \mathrm{hc}_c(b)} \quad (4)$$

to further support JBF. Different from (3), the histogram in the numerator is the pixel intensity histogram hi_c that accumulates the pixel intensity for each bin, instead of the pixel count in hc_c.

In summary, the IH approach is the state of the art in target-pixel-first approaches, but its memory cost is frame–scale–magnitude, like the support-pixel-first approaches. However, as mentioned above, the memory cost of the support-pixel-first approach is hard to be reduced due to its iterative computing, instead of the progressive computing in the IH approach. Thus, this paper focuses on the IH approach.

III. Analysis of the IH Approach

Here, we introduce the IH approach in detail and then analyze the design challenges of JBF, which can be applied to BF as well.

A. IH Approach

Table II presents the computational flow and analysis of the IH approach for JBF to calculate a 1-pixel result, which consists of the integration, extraction, kernel calculation, and convolution processes, in which the first two are for the histogram calculation step and the latter two are for the 1-D convolution step.

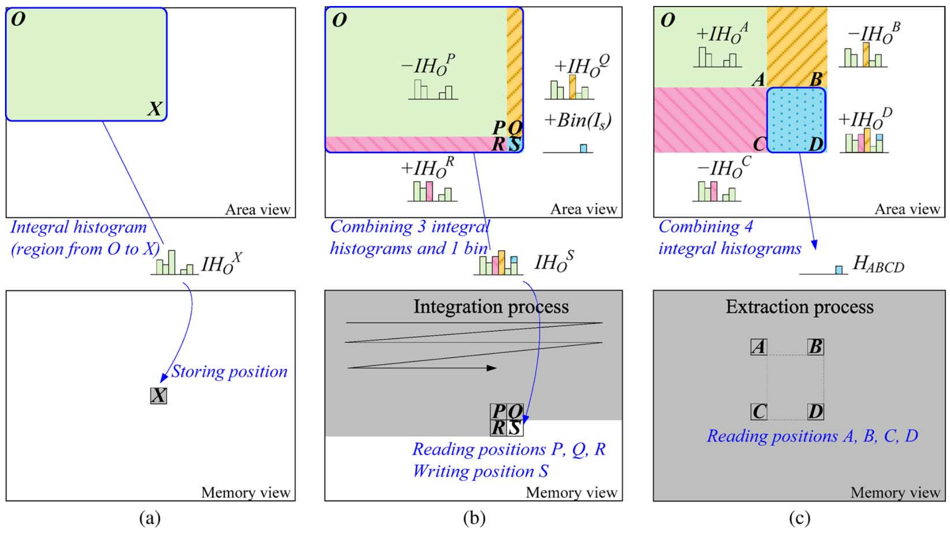


Fig. 3. Concept of the IH approach: (a) Representation of IH_O^X for the region from O to X in area view and memory view; (b) integration process performed by raster-scan order to compute IH_O^S; (c) extraction process performed to extract histogram H_{ABCD} of rectangle ABCD.

TABLE II
COMPUTATIONAL FLOW AND COMPLEXITY ANALYSIS FOR EACH PIXEL IN THE IH APPROACH FOR JBF

Process	Complexity (operation)	BW for IH (data)	BW for pixel (data)
Integration process:			
Pixel count histogram hc_c			
Loop b=0 to N_b-1			
$IHc_O^S(b)=IHc_O^Q(b)+IHc_O^R(b)-IHc_O^P(b)$	ADD: $3N_b$	$4N_b$	
$IHc_O^S(I_S)$ += 1	ADD: 1		
Pixel intensity histogram hi_c			
Loop b=0 to N_b-1			
$IHi_O^S(b)=IHi_O^Q(b)+IHi_O^R(b)-IHi_O^P(b)$	ADD: $3N_b$	$4N_b$	
$IHi_O^S(I_S)$ += J_s	ADD: 1		2 pixels
Extraction process:			
Pixel count histogram hc_c			
Loop $b=0$ to N_b-1			
$hc_c(b) = IHc_O^D(b)+IHc_O^A(b)-IHc_O^B(b)- IHc_O^C(b)$	ADD: $3N_b$	$4N_b$	
Pixel intensity histogram hi_c			
Loop $b=0$ to N_b-1			
$hi_c(b) = IHi_O^D(b)+IHi_O^A(b)-IHi_O^B(b)- IHi_O^C(b)$	ADD: $3N_b$	$4N_b$	
Kernel calculation process:			
Loop b=0 to N_b-1			
$G(b) = g(\|I_c-b\|)$	ADD, LUT: N_b		1 pixel
Convolution process:			
Nu=0, De=0			
Loop b=0 to N_b-1			
De += $G(b)$ x $hc_c(b)$	MUL, ADD: N_b		
Nu += $G(b)$ x $hi_c(b)$	MUL, ADD: N_b		
Result = Nu / De	DIV: 1		1 pixel
Total	**$17N_b$+3**	**$16N_b$**	**4 pixels**

For ease of explanation, we use the area view to show how this approach operates and the memory view to show the memory usage, as illustrated in Fig. 3(a). In the area view, IH_O^X is a histogram of the rectangular area stretched from pixels O to X. Thus, the addition and the subtraction of the IH can be regarded as area merging and cutting, respectively. In the memory view, the data of IH_O^X are stored at X, and the gray region represents occupied memory usage. With these representations, Fig. 3(b) and (c) illustrate the integration and extraction processes.

First, the integration process progressively calculates the IH of each pixel using

$$\mathrm{IH}_O^S = \mathrm{IH}_O^Q + \mathrm{IH}_O^R - \mathrm{IH}_O^P + \mathrm{Bin}(I_S). \tag{5}$$

For the pixel count histogram hc_c and the pixel intensity histogram hi_c, their IHs (i.e., IH_c and IH_i, respectively) are separately computed, as shown in Table II. For hc_c, $\mathrm{Bin}(I_S)$ is 1 for the corresponding bin and 0 for others. On the other hand, for hi_c, this term is J_s for the corresponding bin and also 0 for others. After this process, the IH of each pixel is produced and stored into the memory.

Second, given the IHs, the extraction process can extract histogram hc_c or hi_c of the filter window ABCD centered by the target pixel c using

$$H_{\mathrm{ABCD}} = \mathrm{IH}_O^D + \mathrm{IH}_O^A - \mathrm{IH}_O^B - \mathrm{IH}_O^C. \tag{6}$$

As shown in Fig. 3(c), a histogram with arbitrary filter window size can be obtained by using the IHs of the four corners. With this property, the IH approach can reduce computational complexity to being independent of the filter window size.

Third, the kernel calculation process computes the range kernel by a range table, which includes 256 items for the 256 possible values of $|I_c - b|$. Finally, given the range kernel g and histograms hc_c and hi_c, the convolution process calculates the result of the target pixel c by (4).

B. Design Challenges

Since the complexities listed in Table II are pixelwise, as well as bin-number dependent, they will quickly grow as the resolution and the bin number grow. The detailed design challenges are described below.

1) High Memory Cost for IHs: During the integration process, all the IHs of the whole image are stored in the memory. BF needs a frame–scale–magnitude memory for hc_c, and JBF additionally needs another one for hi_c. Therefore, the total memory cost of JBF is

$$MN \cdot N_b w_b + MN \cdot N_b(w_b + 8) \tag{7}$$

where the former term is for hc_c and the later term is for hi_c. M and N are the frame height and width, respectively, N_b is the number of bin, and w_b is the bit width of a bin. Note that w_b is related to the maximal area of integration, and its value is equal to $\log_2(MN)$. In addition, the bit width of hi_c is more than hc_c by 8 bits because the intensity of a pixel requires 8 bits.

The aforementioned memory cost would be 829.4 MB for the HD1080p resolution (i.e., $N = 1920$, $M = 1080$, $w_b = 21$, and $N_b = 64$). For a VLSI design, these massive data could be configured into off-chip memory [i.e., dynamic random-access memory (RAM)] or on-chip memory (i.e., static RAM). However, as compared with the on-chip memory, the off-chip memory suffers from longer access latency due to its complicated controlling mechanism [34] and from limited bandwidth usage due to being shared by multiple masters. Hence, our strategy for the high memory cost is to reduce the memory requirement and enable data to be stored in the on-chip memory for fast implementation.

2) High Computational Complexity in All Processes: According to the complexity in Table II, generating a 1-pixel result needs $15N_b + 2$ additions, $2N_b$ multiplications, and 1 division. If N_b is 64, the total complexity will be 2262.3 million operations for an HD1080p image. To meet the aforementioned demands, a VLSI design with sufficient parallel operators is necessary.

3) High Bandwidth in Integration and Extraction: In Table II, the bandwidth for the IH requires $16N_b$ for a 1-pixel result, and that will reach 106.168 Gb for an HD1080p image, as shown in Table V. That is because the IHs are frequently accessed. With the strategy for the memory-cost problem, the IHs are stored in the on-chip memory, and its data bus should be increased to address the high bandwidth problem. However, it results in overpartitioned memory and increased area. Thus, a method to reduce the bandwidth is needed.

4) Large Range Table in Kernel Calculation: In the kernel calculation process, a range table with 256 items is needed. However, with the parallel operations for the computational complexity problem, this table should be duplicated. Thus, both the size and the number of the range table results in a large area.

In summary, the IH approach can speed up JBF and BF well but suffers from the previously mentioned design challenges. To address them, a VLSI design with suitable memory reduction and architecture design techniques is necessary.

IV. Proposed Memory Reduction Methods

To solve the high-memory-cost problem, this paper takes advantages of the raster-scan computing order to reduce the memory cost from a frame to a multiple scan line region, named the runtime updating method (RUM). We further reduce the memory cost by slicing each region into stripes, named the stripe-based method (SBM), to avoid frame wide buffer cost. Finally, the origin of each IH stripe is progressively moved with computing. It is advanced to reduce the stripe high buffer to only one-line high buffer, named the sliding origin method (SOM). With these memory methods, the memory cost can be reduced to 0.003%–0.020%. The details of the proposed methods are described below.

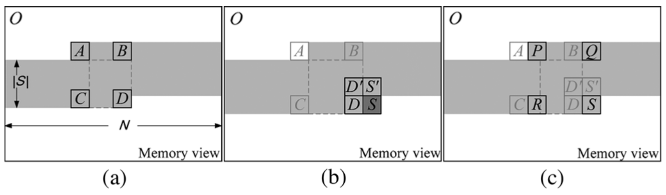


Fig. 4. RUM: (a) Extraction for H_{ABCD}; (b) integration to S; (c) extraction for H_{PQRS}.

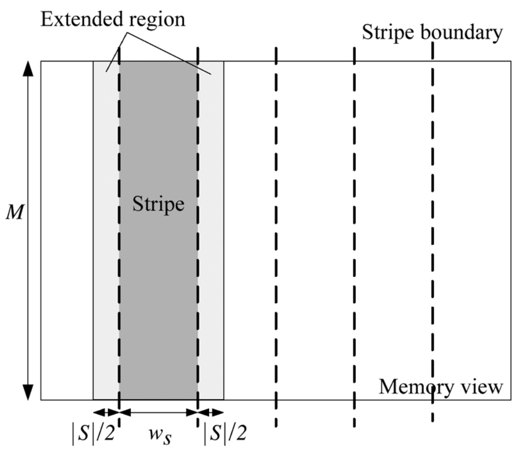


Fig. 5. SBM.

A. RUM

The concept of the RUM is to perform the integration and extraction processes at the same time, instead of two separate iterations in the original flow. Fig. 4 illustrates its memory configuration in the memory view. In Fig. 4(a), the integration process is from pixels O to D. Meanwhile, the extraction process can extract histogram H_{ABCD}. From the data lifetime analysis, all the IHs before pixel A are unnecessary. Hence, only the IHs from pixels A to D have to be stored and require memory space. Thus, the memory cost is

$$|S|N \cdot N_b w_b + |S|N \cdot N_b(w_b + 8) \tag{8}$$

where M in (7) is replaced by the filter window width $|S|$.

Fig. 4(b) and (c) illustrate that the memory is updated when the two processes moves to the next pixel S. In Fig. 4(b), the integration process calculates the new IH_O^S using IH_O^D, $\mathrm{IH}_O^{D'}$, and $\mathrm{IH}_O^{S'}$, and then, the new IH_O^S can overwrite the memory position of the discarded IH_O^A. In Fig. 4(c), the extraction process can extract H_{PQRS}.

With the proposed RUM, the memory cost could be reduced from a full frame to a partial frame. This method can gain considerable reduction since $|S|$ is usually much smaller than M.

B. SBM

The main idea of the SBM is to slice the whole frame into many vertical stripes, and the integration and extraction processes are performed stripe by stripe. Fig. 5 illustrates a whole frame partitioned into stripes. Note that the integration process should be additionally carried out on the extended region, which contains the surrounding support pixels for the

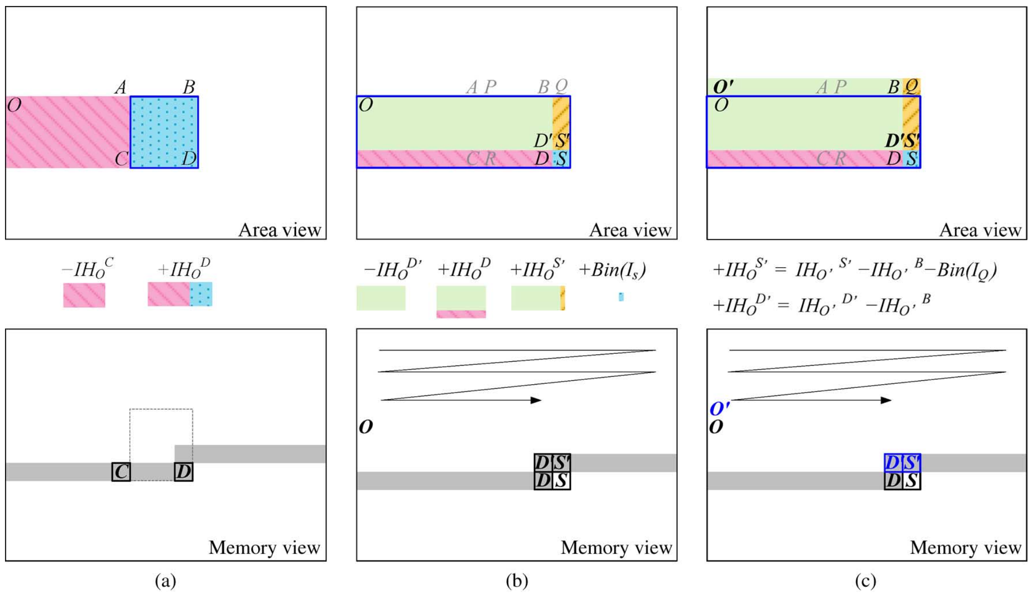

Area view
Memory view
(a)
(b)
(c)
$-IH_O^C$ $+IH_O^D$
$-IH_O^{D'}$ $+IH_O^D$ $+IH_O^{S'}$ $+Bin(I_s)$
$+IH_O^{S'} = IH_{O'}^{S'} - IH_{O'}^B - Bin(I_Q)$
$+IH_O^{D'} = IH_{O'}^{D'} - IH_{O'}^B$

Fig. 6. SOM: (a) Extraction process with sliding origin O; (b) integration process to next pixel S; (c) modified integration process to next pixel S.

target pixels on the stripe boundary. Thus, the total memory cost of the SBM is

$$M\,(|S|+w_s-1)\cdot N_b w_b + M\,(|S|+w_s-1)\cdot N_b(w_b+8) \quad (9)$$

where w_s is the stripe width and w_b is equal to $\log_2[M(|S|+w_s-1)]$. Compared with the original cost in (7), the SBM could reduce significant memory if $(|S|+w_s-1)$ is much smaller than N.

The overhead of the SBM is that the extended regions result in extra computation and bandwidth in the integration process due to repeated performing on these regions. Thinner stripes can reduce memory cost more, but that leads to more overheads. Thus, the selection of w_s is a tradeoff between memory reduction and overheads. That will be discussed in Section VI.

C. SOM

The concept of the SOM is to vertically slide the origin pixel O with the integration and extraction processes to reduce the memory cost from a plane to a line, as shown in Fig. 6. With the sliding origin pixel, the two processes can be simplified, as described below.

For the extraction process in the area view in Fig. 6(a), the original IH_O^A and IH_O^B are zero because the position of O is under A and B, and they cannot form meaningful histogram rectangles. Hence, (6) can be simplified as follows:

$$\begin{aligned} H_{\mathrm{ABCD}} &= \mathrm{IH}_O^D + \mathrm{IH}_O^A - \mathrm{IH}_O^B - \mathrm{IH}_O^C \\ &= \mathrm{IH}_O^D - \mathrm{IH}_O^C. \end{aligned} \quad (10)$$

For the integration process in Fig. 6(b), the new IH_O^S can be computed by

$$\mathrm{IH}_O^S = \mathrm{IH}_O^D + \mathrm{IH}_O^{S'} - \mathrm{IH}_O^{D'} + \mathrm{Bin}(I_S). \quad (11)$$

However, S' and D' are on the previous row of S and D, respectively, and their corresponding origin should be O', as shown in Fig. 6(c), instead of O. Therefore, $\mathrm{IH}_O^{S'}$ and $\mathrm{IH}_O^{D'}$ in (11) should be changed to $\mathrm{IH}_{O'}^{S'}$ and $\mathrm{IH}_{O'}^{D'}$, respectively, by the following derivation, which is corresponding to the area view in Fig. 6(c):

$$\begin{aligned} \mathrm{IH}_O^S &= \mathrm{IH}_O^D + \mathrm{IH}_O^{S'} - \mathrm{IH}_O^{D'} + \mathrm{Bin}(I_S) \\ &= \mathrm{IH}_O^D + \left(\mathrm{IH}_{O'}^{S'} - \mathrm{IH}_{O'}^Q\right) \\ &\quad - \left(\mathrm{IH}_{O'}^{D'} - \mathrm{IH}_{O'}^B\right) + \mathrm{Bin}(I_S) \\ &= \mathrm{IH}_O^D + \left(\mathrm{IH}_{O'}^{S'} - \left(\mathrm{IH}_{O'}^B + \mathrm{Bin}(I_Q)\right)\right) \\ &\quad - \left(\mathrm{IH}_{O'}^{D'} - \mathrm{IH}_{O'}^B\right) + \mathrm{Bin}(I_S) \\ &= \mathrm{IH}_O^D + \mathrm{IH}_{O'}^{S'} - \mathrm{Bin}(I_Q) - \mathrm{IH}_{O'}^{D'} + \mathrm{Bin}(I_S). \end{aligned} \quad (12)$$

With the aforementioned simplification, only the IHs of C, D, S', and D' are associated, and the IHs from D' to D requires memory space, as shown in the memory view in Fig. 6(c). Thus, the total memory cost is

$$N\cdot N_b w_b + N\cdot N_b(w_b+8) \quad (13)$$

where w_b is equal to $\log_2(|S|N)$ since the maximal area of integration is $|S|N$. Compared with the original cost in (7), the height dimension M is eliminated, and w_b is much smaller.

D. Combination

The proposed memory reduction methods could be simply combined as follows. First, the SBM partitions a whole frame into stripes. Then, in each stripe, the RUM and the SOM are performed row by row. This combination can reduce the memory cost to

$$(|S|+w_s-1)\cdot N_b w_b + (|S|+w_s-1)\cdot N_b(w_b+8) \quad (14)$$

where w_b is equal to $\log_2[|S|\,(|S|+w_s-1)]$. Compared with the original cost in (7), M is decreased to 1 due to the RUM and the SOM, and N is decreased to $(|S|+w_s-1)$ due to the SBM. Note that, in this memory cost formulation, N_b and $|S|$

TABLE III
MODIFIED COMPUTATIONAL FLOW AND COMPLEXITY ANALYSIS FOR EACH PIXEL IN THE IH APPROACH FOR JBF

Process	Complexity (operation)	BW for IH (data)	BW for pixel (data)
Integration process:			
<u>Pixel count histogram hc_c</u>			
Loop b=0 to N_b-1			
$IHc_O^S(b)=IHc_O^D(b)+IHc_O^{S'}(b)-IHc_O^{D'}(b)$	ADD: $2N_b$	$4N_b$	
$IHc_O^S(I_S)$ += 1, $IHc_O^S(I_Q)$ -= 1	ADD: 2		
<u>Pixel intensity histogram hi_c</u>			
Loop b=0 to N_b-1			
$IHi_O^S(b)=IHi_O^D(b)+IHi_O^{S'}(b)-IHi_O^{D'}(b)$	ADD: $2N_b$	$4N_b$	
$IHi_O^S(I_S)$ += J_S, $IHi_O^S(I_Q)$ -= J_Q	ADD: 2		4 pixels
Extraction process:			
<u>Pixel count histogram hc_c</u>			
Loop b=0 to N_b-1			
$hc_c(b) = IHc_O^S(b) - IHc_O^R(b)$	ADD: N_b	N_b	
<u>Pixel intensity histogram hi_c</u>			
Loop b=0 to N_b-1			
$hi_c(b) = IHi_O^S(b) - IHi_O^R(b)$	ADD: N_b	N_b	
Kernel calculation process:			
Loop b=0 to N_b-1			
$G(b) = g(\|I_c-b\|)$	ADD, LUT: N_b		1 pixel
Convolution process:			
Nu=0, De=0			
Loop b=0 to N_b-1			
De += $G(b)$ x $hc_c(b)$	MUL, ADD: N_b		
Nu += $G(b)$ x $hi_c(b)$	MUL, ADD: N_b		
Result = Nu / De	DIV: 1		1 pixel
Total	**$11N_b$+5**	**$10N_b$**	**6 pixels**

are related to the application quality, and w_s is related to the hardware performance. The analysis of the parameter selection will be further presented in Section VI.

V. PROPOSED ARCHITECTURE

With the aforementioned memory reduction methods, the computational flow of JBF in Table II is changed to that in Table III. To efficiently implement the architecture, we first propose the R-parallelism method to execute parallel computations in the range domain to meet the required throughput. Then, we take advantages of the timing relationship of the data in the progressive computation to buffer the computed IHs for on-chip bandwidth reduction, named the delay-buffer method. The large range table size due to parallelism is further reduced by exploiting the numerical properties of the Gaussian function. With these memory reduction methods and architecture design techniques, an efficient hardware design is proposed, which can be easily scalable to different performance targets. For ease of explanation, we use an example for the performance target of the HD1080p resolution to present the design. The details of these design techniques are presented below.

A. Overall Architecture

Fig. 7 shows the overall architecture that contains two parts, i.e., interface and core. In this architecture, the image pixels and the IHs are stored at the off- and on-chip memory, respectively. The interface accesses pixels from the off-chip memory through a 64-bit bus, and the core performs the computation of JBF. The bus protocol adopts a simple handshaking mechanism, where the access consists of a request/address phase and a data phase in a pipelined way. For simplification, our core and an off-chip memory are only connected to the bus.

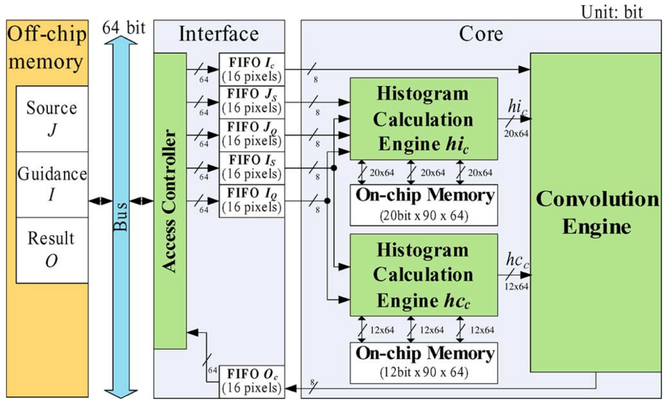


Fig. 7. Proposed architecture of JBF. Our design includes the computing core and the interface connecting to the bus.

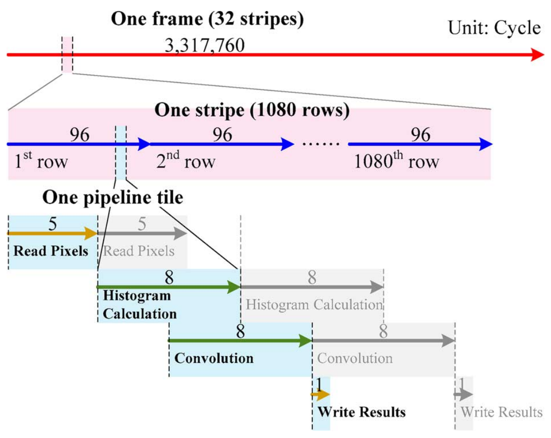


Fig. 8. Schedule of the proposed architecture.

In the interface, the access controller allocates the bus priority to the input and output first-in-first-out (FIFO) buffers by round-robin policy. The size of each buffer is associated with the off-chip bandwidth. Large buffers can support data reuse schemes to reduce the off-chip bandwidth. Because of the sufficient bandwidth in this architecture, we do not apply any data reuse schemes here to have lower buffer cost and set its size as 2×8 pixels, where the value of 8 is to meet the bus width and the value of 2 is to support the ping-pong mechanism for simultaneous reading and writing.

The operations of the architecture are described below with the schedule in Fig. 8, which is hierarchically sliced from a frame to pipeline tiles. The computation of one stripe row requires 90 cycles for the stripe width w_s of 60 and the filter window width $|S|$ of 31. However, the throughput of each pipeline tile is the computation of 8 pixels, and one stripe row needs 12 pipeline tiles to process. Therefore, this architecture takes 96 cycles for one stripe row, and the last six cycles are the bubble cycles. For the process in a pipeline tile, the access controller in the interface first reads pixels from the off-chip memory and stores them into the FIFO buffers. Then, the two histogram calculation engines in the core begin to compute hi$_c$ and hc$_c$, and the convolution engine consecutively produces 8 pixels to the output FIFO buffer. Finally, the interface moves results from the buffer to the off-chip memory.

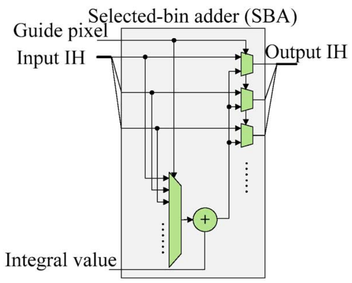


Fig. 9. SBA in the histogram calculation engines.

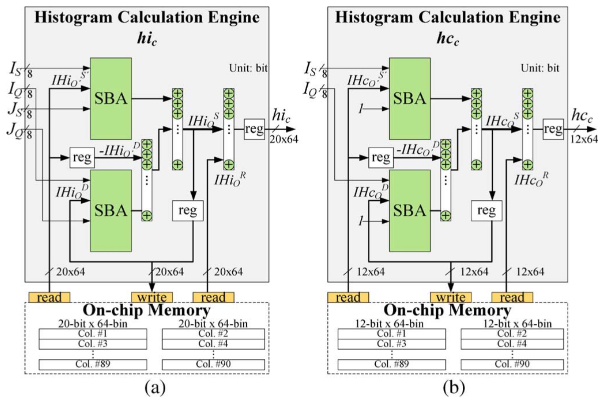

Histogram Calculation Engine hic
Histogram Calculation Engine hcc
SBA
reg
Unit: bit
On-chip Memory
read
write
20x64
12x64

Fig. 10. Proposed architectures of histogram calculation engines hi_c and hc_c.

B. Architecture of Components

In the core, the main components are two histogram calculation engines and one convolution engine for the computation in Table III, which have high computational complexity, as previously mentioned. Thus, the proposed R-parallelism method unrolls all computational loops in the range domain R. The details of this method are described in each engine as follows.

1) Histogram Calculation Engine: The histogram calculation engines perform the integration and extraction processes for hc_c and hi_c, as shown in Table III. With the R-parallelism method, we design their architectures, as shown in Fig. 10, where the selected-bin adder (SBA) is depicted in Fig. 9. These two engines can achieve the throughput of 1 histogram/cycle. Note that the difference of the two engines is that the integral value of SBAs is the source pixel J in the engine hi_c, instead of the constant 1 in the engine hc_c. In addition, all bit widths of data in the engine hi_c are more than those in hc_c by 8 bits (see Fig. 10).

In above architectures, each engine needs to access the five IHs, i.e., $\mathrm{IH}_{O'}^{S'}$, $\mathrm{IH}_{O'}^{D'}$, IH_{O}^{S}, IH_{O}^{D}, and IH_{O}^{R}, from the on-chip memory in one cycle. To reduce the bandwidth problem, we propose the delay-buffer method, which is presented as follows by the data dependence of the associated IHs in two successive cycles. Assume that pixels S, S', D, and D' shown in Fig. 6(d) are located in (x,y), $(x,y-1)$, $(x-1,y)$, and $(x-1,y-1)$ in cycle t, respectively. Hence, their IHs can be notated by

$$\begin{aligned} S^{(t)}&: \mathrm{IH}_O^{(x,y)} & S'^{(t)}&: \mathrm{IH}_O^{(x,y-1)} \\ D^{(t)}&: \mathrm{IH}_O^{(x-1,y)} & D'^{(t)}&: \mathrm{IH}_O^{(x-1,y-1)}. \end{aligned} \quad (15)$$

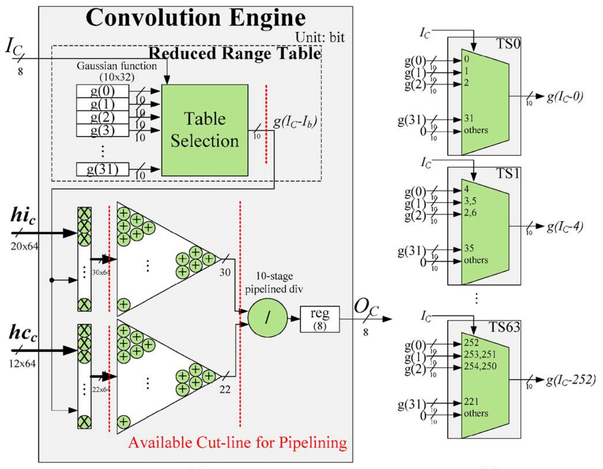


Fig. 11. Proposed architecture of (a) convolution engine and (b) its table selection modules.

For the next cycle $t+1$, their x-coordinates are increased by 1 as follows:

$$\begin{aligned} S^{(t+1)}&: \mathrm{IH}_O^{(x+1,y)} & S'^{(t+1)}&: \mathrm{IH}_O^{(x+1,y-1)} \\ D^{(t+1)}&: \mathrm{IH}_O^{(x,y)} & D'^{(t+1)}&: \mathrm{IH}_O^{(x,y-1)}. \end{aligned} \quad (16)$$

From (15) and (16), we can find that $D^{(t+1)}$ is equal to $S^{(t)}$ and $D'^{(t+1)}$ is equal to $S'^{(t)}$. That means $\mathrm{IH}_{O'}^{D'}$ and IH_{O}^{D} can be obtained by delaying $\mathrm{IH}_{O'}^{S'}$ and IH_{O}^{S} for one cycle, respectively. Therefore, we can use two delay buffers to avoid accessing $\mathrm{IH}_{O'}^{D'}$ and IH_{O}^{D} from the on-chip memory and to reduce the bandwidth from five to three IHs.

2) Convolution Engine: The convolution engine uses histograms hc_c and hi_c to further compute the result pixel by the kernel calculation and convolution processes in Table III. Its architecture is shown in Fig. 11(a). With the proposed R-parallelism method, the convolution process can achieve the throughput of 1 pixel/cycle. Higher throughput can be further attained by adding the registers at the available cutlines for pipelining in the figure, which can enable the operating frequency to be higher.

The R-parallelism method brings high throughput but suffers from large size and large number of range table. First, the range values are converted from fractional to integer ones by

$$G(I_c - I_b) = \mathrm{round}\left(\exp\left(-\frac{|I_c - I_b|}{2\sigma^2 \times 256}\right) \times \mathrm{Scale}\right) \quad (17)$$

where σ^2 is the variance of the range kernel and *Scale* is a multiple to scale the original value ($\leq$1.0) to a larger value. In this design, Scale is set as 1023, and the bit length of the range value is fixed to 10 bits using the unsigned integer representation without the fractional part. The corresponding range table with 256 items should be duplicated to as many as the bin number. For the large size, we take advantages of the symmetry and the truncation property of the Gaussian function to decrease its size from 256 to 32. In addition, to avoid the large number of range

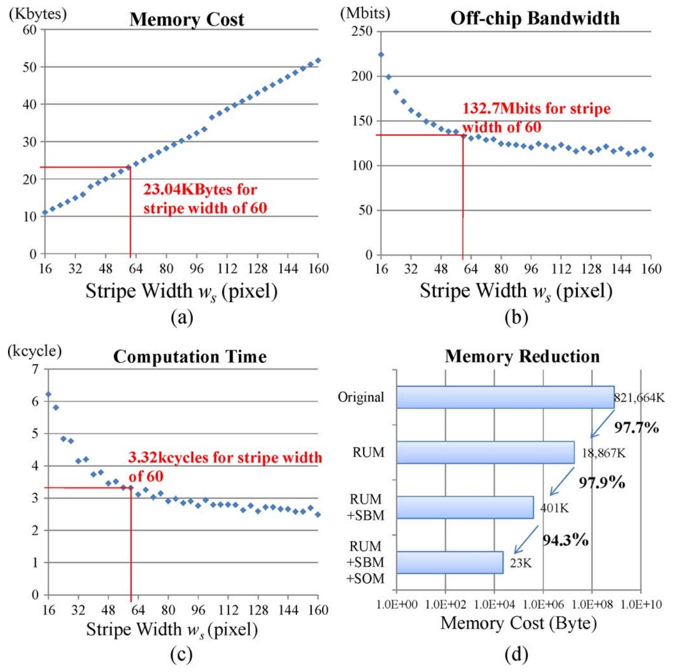


Fig. 12. [(a)–(c)] Hardware performance per frame with different w_s values. (d) Memory reduction with the proposed methods for w_s of 60 $(M = 1080, N = 1920, N_b = 64,$ and $|S| = 31)$.

table, we share one table by the table selection module, as shown in Fig. 11(b), which reduces the number of tables to one. Note that the result of the divisor would directly be in the range of 8 bits because it is used to normalize the sum of pixels with weighting in (4).

Furthermore, the histogram calculation engines and the convolution engine can be serially connected to achieve the throughput of 1 pixel/cycle. More engines can be used to process multiple cascaded pixels simultaneously for higher throughput. The proposed memory reduction methods could be directly extended to support the processing of multiple pixels. In addition, note that, for simpler BF, the histogram calculation engine hi$_c$ and its on-chip memory in the core module, and the two input FIFOs in the interface module could be reduced.

VI. Implementation Result

Here, we first analyze the parameter selection in the proposed memory reduction methods and then demonstrate the result of the implementation example for the HD1080p resolution.

A. Parameter Analysis

As the combined memory cost in (14), there are three parameters, i.e., the filter window width of the space kernel $|S|$, the number of bin N_b, and the stripe width w_s, where the first two are related to the application quality and the last one is related to the target performance. Referring to the quality analysis in [23], we select 31 for $|S|$ and 64 for N_b as an example to illustrate how to determine w_s by considering the hardware performance.

Fig. 12(a)–(c) estimates the hardware performance of JBF with different w_s for the resolution HD1080p. The memory cost is computed with (14) and plotted in Fig. 12(a). The off-chip

TABLE IV
Example Implementation Result of the Proposed Architecture

Technology		UMC 90nm	
Image Size MxN		1920x1080	
Number of Bin N_b		64	
Filter Window Size $\|S\|$x$\|S\|$		31x31	
Stripe Width w_s		60	
Clock Rate (Hz)		100M	200M
Frame Rate (Frame/Sec.)		30	60
Logic Cost Excluding Memories (Equivalent Gate-Count)	Interface	9,578	9,917
	Histogram Cal.	97,766	148,649
	Convolution	168,333	197,351
	Total	276,178	355,917
On-chip Memory (Byte)		23K	23K

TABLE V
Comparison of Hardware Cost per Frame

	Resol.	Complexity (million operation)	Memory Requirement (Kbyte)	Bandwidth for IH (Mbit)	Bandwidth for pixel (Mbit)
Original	VGA	335.1 (100%)	113,050 (100%)	14,470 (100%)	9.8 (100%)
	HD720p	1,005.5 (100%)	353,894 (100%)	45,299 (100%)	29.5 (100%)
	HD1080p	2,262.3 (100%)	829,440 (100%)	106,108 (100%)	66.4 (100%)
Mem. Reduction	VGA	197.0 (59%)	23 (0.020%)	9,083 (63%)	20.3 (206%)
	HD720p	591.1 (59%)	23 (0.007%)	27,250 (60%)	60.8 (206%)
	HD1080p	1,289.7 (57%)	23 (0.003%)	59,454 (56%)	132.7 (200%)
Mem. Reduction + Archi. Design Tech.	VGA	5.1 (0.15%)	23 (0.020%)	5,191 (36%)	20.3 (206%)
	HD720p	1.5 (0.15%)	23 (0.007%)	15,571 (34%)	60.8 (206%)
	HD1080p	3.3 (0.15%)	23 (0.003%)	33,974 (32%)	132.7 (200%)

Number of bins $N_b = 64$; filter window width $|S| = 31$; Stripe width $w_s = 60$; VGA $= 640 \times 480$; HD720p $= 1280 \times 720$; HD1080p $= 1920 \times 1080$

TABLE VI
Previous VLSI Implementations of BF

	[18]	[19]	[20]	[21]	Our Design
Supported Window Size	15x15 BF	3x3 BF-like	5x5 BF	11x11 BF	31x31 BF/JBF
Implementation Method	Xilinx Spartan-3 FPGA	Altera Cyclone-II FPGA	Xilinx Vertex-5 FPGA	TSMC 0.18um Tech. Proc.	UMC 90nm Tech. Proc.
Throughput (pixel/s)	4.8M	124M	41.9M	11M	124M

bandwidth and the computation time are calculated by the following and plotted in Fig. 12(b) and (c), respectively:

$$M(N/w_s)\left(|S| + w_s - 1\right) \cdot 4 \text{ pixels} + M(N/w_s)w_s \cdot 2 \text{ pixels} \tag{18}$$

$$M(N/w_s)\left\lceil\left(|S| + w_s - 1\right)/8\right\rceil \cdot 8 \cdot 1 \text{ cycles} \tag{19}$$

where $M(w_s + |S| - 1)$ is the stripe area with extended regions and N/w_s is the number of stripe in a frame. Note that the computation time is $\lceil(|S| + w_s - 1)/8\rceil \cdot 8$ cycles for one stripe row because one pipelining tile takes 8 cycles. For the bandwidth, the term with 4 pixels is required by the integration process, and the other term with 2 pixels is required by other processes. Since the integration process should additionally perform on the extended regions in Fig. 5, its bandwidth is more than that of the other processes. For the computation time, the proposed architecture takes 1 cycle to produce a 1-pixel result.

TABLE VII
COMPARISON OF DIFFERENT IMPLEMENTATIONS

	Support-Pixel-First				Target-Pixel-First	
	Durand [21]	Chen [27]	Yang [22]	Adams [28]	Porikli [23]	Proposed
Approach	Piecewise-linear Subsampling (s_s=24, s_r=19)	Bilateral Grid (s_s=16, s_r=10)	Piecewise-linear (s_r=32)	Gaussian KD-tree	Integral Histogram (s_r=4)	Integral Histogram (s_r=4)
Implementation	CPU P4 2GHz	GPU Geforce 8800GTX	GPU Geforce 8800GTX	GPU GeForce GTX260	CPU P4 3.2GHz	ASIC
Transistor count (Tech. Process)	55M (130nm)	681M (90nm) [34]	681M (90nm) [34]	1,400M (TSMC 65nm) [35]	55M (130nm)	2.5M (UMC 90nm)
Image Size (Pixel)	10.4M	1.0M	1.0M	10M	1.0M	2.07M
Frame Rate (Frame/sec)	0.16 (high dynamic range)	222	66	0.01-1	3.22	60
Throughput (Pixel/sec)	1.6 M	222M	66M	0.1M-10M	3.22M	124M
Memory (Byte)	-	625K	4M	1G-100M	96M	23K

The selection of w_s is mainly related to the target frame rate. If our target is 30 frames/s, we could select 60 for w_s when the working clock is 100 MHz. Hence, the off-chip bandwidth will be 62.2%, and the memory cost can be reduced to 23 KB, which is 0.003% of the original cost, as shown in Fig. 12(d).

B. Implementation Result

With previously selected parameters, the proposed architecture of JBF has been implemented by Verilog and synthesized under the 90-nm complementary metal–oxide–semiconductor technology process. For verifying our design, all the floating-point computation in the C model is first changed into integer computation. Then, we use the C model to dump the golden data into files using practical sequences and random patterns for gate level simulation. The numerical results of the gate level simulation are verified to be the same as those golden data. Table IV lists the implementation result of the proposed architecture. The hardware design spends equivalent gate counts of less than 300 K and 23-KB on-chip memory to achieve the throughput of HD1080p 30 frames/s at the clock rate of 100 MHz. Moreover, it can process at 200 MHz by pipelining on the available cutlines in the convolution engine and can further achieve the throughput of HD1080p 60 frames/s that is 124 Mpixels/s. In addition, the maximum operating frequencies of the two designs are 180.5 and 284.0 MHz with area cost of 440- and 531-K gates, respectively.

Table V compares the complexity, the memory requirement, and the bandwidths between the proposed methods and the original IH in different resolutions. With the proposed memory reduction and architecture design techniques, the complexity can be reduced to 0.15%, and the memory requirement can be reduced to 0.003%–0.02%. In addition, the bandwidth for the IH (i.e., on-chip bandwidth) can be reduced to 32%–36%, but the bandwidth for the pixel (i.e., off-chip bandwidth) is increased to 20.3–132.7 Mb. Nevertheless, the off-chip bandwidth is affordable by the 64-bit bus processing at 200 MHz. Note that the stripe width w_s is specifically selected for the resolution HD1080p. Thus, it can be reselected by means of the mentioned analysis in Part A to acquire better performance for another resolution.

Table VI compares our proposed hardware design with the previous VLSI implementations. The previous implementations [18], [21] could support large filtering window but low throughput, whereas the implementations in [19] and [20] could reach high throughput for a small filtering window only. Not only can our design achieve high throughput, but it can also support a large filtering window. Table VII compares our design with the other previous GPU and central-processing-unit implementations. Comparing with other design, the proposed architecture could efficiently utilize the hardware cost to achieve high throughput.

VII. CONCLUSION

This paper has presented an efficient scalable implementation for the state-of-the-art IH approach used in JBF and BF applications by the means of well-designed memory reduction methods and architecture design techniques. The memory reduction methods take advantage of the raster-scan processing in the IH approach to reduce the memory cost from the original frame–scale–magnitude to line–scale–magnitude. In the proposed architecture design techniques, the R-parallelism method can increase the processing throughput, and the delay-buffer method can decrease the on-chip bandwidth. In addition, the simplified range table can significantly save the storage of Gaussian values. With these presented methods, an example implementation has shown that it can achieve the processing of HD1080p resolution and 60 frames/s and has spent only 356-K gate counts and 23-KB memory cost. Scalable to other design specifications, it can be easily adapted from the presented architecture. Although the presented methods have been designed for the hardware originally, it can be also applied to other kinds of implementations such as processor-based implementations for computation speedup and memory reduction, which will be future work of this paper.

REFERENCES

[1] S. Paris, P. Kornprobst, J. Tumblin, and F. Durand, “Bilateral filtering: Theory and applications,” *Found. Trends Comput. Graph. Vis.*, vol. 4, no. 1, pp. 1–73, 2009.

[2] C. Tomasi and R. Manduchi, “Bilateral filtering for gray and color images,” in *Proc. IEEE ICCV*, Jan. 1998, pp. 839–846.

[3] E. Eiseman and F. Durand, "Flash photography enhancement via intrinsic relighting," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 673–678, Aug. 2004.

[4] G. Petschnigg, M. Agrawala, H. Hoppe, R. Szeliski, M. Cohen, and K. Toyama, "Digital photography with flash and no-flash image pairs," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 664–672, Aug. 2004.

[5] H. Yu, Y.-L. Zhao, and H. Wang, "Image denoising using trivariate shrinkage filter in the wavelet domain and joint bilateral filter in the spatial domain," *IEEE Trans. Image Process.*, vol. 18, no. 10, pp. 2364–2369, Oct. 2009.

[6] C. Varekamp and B. Barenbrug, "Improved depth propagation for 2D to 3D video conversion using key-frames," in *Proc. IET Eur. Conf. Vis. Media Prod.*, Nov. 2007, pp. 1–7.

[7] C.-C. Cheng, C.-T. Li, P.-S. Huang, T.-K. Lin, Y.-M. Tsai, and L.-G. Chen, "A block-based 2D-to-3D conversion system with bilateral filter," in *Proc. IEEE ICCE*, Jan. 2009, pp. 1–2.

[8] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Trans. Graph.*, vol. 26, no. 3, article 96, pp. 2:1–2:9, Jul. 2007.

[9] Q. Yang, R. Yang, J. Davis, and D. Nister, "Spatial-depth super resolution for range images," in *Proc. IEEE CVPR*, Jun. 2007, pp. 1845–1852.

[10] A. K. Riemens, O. P. Gangwal, B. Barenburg, and R.-P. M. Berretty, "Multi-step joint bilateral depth upsampling," in *Proc. SPIE Vis. Commun. Image Process.*, Jan. 2009, vol. 7257, p. 725 70M.

[11] M.-C. Chuang, Y.-N. Liu, T.-H. Chen, and S.-Y. Chien, "Color filter array demosaicking using joint bilateral filter," in *Proc. IEEE ICME*, Jul. 2009, pp. 125–128.

[12] C. Xiao, Y. Nie, W. Hua, and G. Feng, "Fast multi-scale joint bilateral image and video texture upsampling," *Vis. Comput.*, vol. 26, no. 4, pp. 263–275, Apr. 2010.

[13] 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.

[14] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister, "Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 3, pp. 2347–2354, Mar. 2009.

[15] J. Lu, S. Rogmans, G. Lafruit, and F. Catthoor, "Stream-centric stereo matching and view synthesis: A high-speed approach on GPUs," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 19, no. 11, pp. 1598–1611, Nov. 2009.

[16] N. Y.-C. Chang, T.-H. Tsai, P.-H. Hsu, Y.-C. Chen, and T.-S. Chang, "Algorithm and architecture of disparity estimation with mini-census adaptive support weight," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 20, no. 6, pp. 792–805, Jun. 2010.

[17] O. Stankiewicz, K. Wegner, and M. Wildeboer, A soft-segmentation matching in Depth Estimation Reference Software (DERS) 5.0, ISO/IEC JTC1/SC29/WG11, M17049. Xian, China, Oct. 2009.

[18] C. Charoensak and F. Satter, "FPGA design of a real-time implementation of dynamic range compression for improving television picture," in *Proc. IEEE ICICS*, Dec. 2007, pp. 1–5.

[19] T. Q. Vinh, J. H. Park, Y.-C. Kim, and S. H. Hong, "FPGA implementation of real-time edge-preserving filter for video noise reduction," in *Proc. Int. Conf. Comput. Elect. Eng.*, Dec. 2008, pp. 611–614.

[20] A. Gabiger, M. Kube, and R. Weigel, "A synchronous FPGA design of a bilateral filter for image processing," in *Proc. IEEE IECON*, Nov. 2009, pp. 1990–1995.

[21] S.-K. Han, "An architecture for high-throughput and improved-quality stereo vision processor," M.S. thesis, Dept. Elect. Comput. Eng., Univ. Maryland, College Park, MD, 2010.

[22] Q. Yang, K.-H. Tan, and N. Ahuja, "Real-time O(1) bilateral filtering," in *Proc. IEEE CVPR*, Aug. 2009, pp. 557–564.

[23] F. Porikli, "Constant time O(1) bilateral filtering," in *Proc. IEEE CVPR*, Aug. 2008, pp. 1–8.

[24] F. Porikli, "Integral histogram: A fast way to extract histograms in Cartesian space," in *Proc. IEEE CVPR*, Jun. 2005, vol. 1, pp. 829–836.

[25] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 257–266, Jul. 2002.

[26] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," in *Proc. ECCV*, May 2006, pp. 568–580.

[27] S. Paris and F. Durand, "A fast approximation of the bilateral filter using a signal processing approach," *Int. J. Comput. Vis.*, vol. 81, no. 1, pp. 24–52, Jan. 2009.

[28] J. Chen, S. Paris, and F. Durand, "Real-time edge-aware image processing with the bilateral grid," *ACM Trans. Graph.*, vol. 26, no. 3, article 103, pp. 1–9, Jul. 2007.

[29] A. Adams, N. Gelfand, J. Dolson, and M. Levoy, "Gaussian KD-trees for fast high dimensional filtering," *ACM Trans. Graph.*, vol. 28, no. 3, article 21, pp. 1–12, Aug. 2009.

[30] T. Q. Pham and L. J. van Vliet, "Separable bilateral filtering for fast video processing," in *Proc. IEEE ICME*, Jul. 2005.

[31] T.-S. Huang, *Two-Dimensional Digital Signal Processing II: Transforms and Median Filters*. New York: Spring-Verlag, 1981, pp. 209–211.

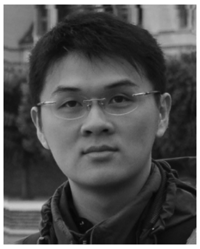
[32] B. Weiss, "Fast median and bilateral filtering," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 519–526, Jul. 2006.

[33] M.-H. Ju and H.-B. Kang, "Constant time stereo matching," in *Proc. Int. Mach. Vis. Image Process. Conf.*, Sep. 2009, pp. 13–17.

[34] Micron Technology, Synchronous DRAM MT48LC2M32B2-1 Meg x 32 x 4 banks [Online]. Available: http://download.um.com/pdf/datasheets/dram/sdram/128MbSDRAMx32.pdf

[35] A. Wong, NVIDIA GeForce 8800 GTX/GTS Tech Report [Online]. Available: http://www.techarp.com/showarticle.aspx?artno=358&pgno=0

[36] A. L. Shimpi and D. Wilson, NVIDIA's 1.4 Billion Transistor GPU: GT200 Arrives as the GeForce GTX 280 & 260 [Online]. Available: http://www.anandtech.com/show/2549



Yu-Cheng Tseng (S'07) received the B.S. degree in electronic engineering in 2006 from the National Chiao Tung University, Hsinchu, Taiwan, where he is currently working toward the Ph.D. degree in the Department of Electronics Engineering.

His current research interests include 3-D video processing and hardware architecture design and implementation.

Po-Hsiung Hsu received the B.S. degree in electrical engineering and computer science undergraduate honors program and the M.S. degree in electrical engineering from the National Chiao Tung University, Hsinchu, Taiwan, in 2008 and 2010, respectively.

He is currently an Integrated Circuit Design Engineer with MediaTek, Inc., Hsinchu. His research interest is image processing.

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

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