EURASIP Journal on Image and Video Processing
a SpringerOpen Journal

RESEARCH **Open Access**

Real-time stereo matching architecture based on 2D MRF model: a memory-efficient systolic array

Sungchan Park*, Chao Chen, Hong Jeong and Sang Hyun Han

Abstract

There is a growing need in computer vision applications for stereopsis, requiring not only accurate distance but also fast and compact physical implementation. Global energy minimization techniques provide remarkably precise results. But they suffer from huge computational complexity. One of the main challenges is to parallelize the iterative computation, solving the memory access problem between the big external memory and the massive processors. Remarkable memory saving can be obtained with our memory reduction scheme, and our new architecture is a systolic array. If we expand it into N's multiple chips in a cascaded manner, we can cope with various ranges of image resolutions. We have realized it using the FPGA technology. Our architecture records 19 times smaller memory than the global minimization technique, which is a principal step toward real-time chip implementation of the various iterative image processing algorithms with tiny and distributed memory resources like optical flow, image restoration, etc.

Keywords: Real-time, VLSI, belief propagation, memory resource, stereo matching

1 Introduction

The stereo matching problem is to find the corresponding points in a pair of images portraying the same scene. The underlying principle is that two cameras separated by a baseline capture slightly dissimilar views of the same scene. Finding the corresponding pairs is known to be the most challenging step in the binocular stereo problem.

As shown in Table 1, the conventional methods can be categorized into the local and global methods [1]. The unit, million disparity estimations per second (MDE/s), is the product of the number of pixels, disparity levels, and frame-rate and therefore, stands for the overall computational speed. Note that the global methods have the low throughput due to their small number of processors.

The local method, typically window correlation and dynamic programming (DP) methods, examines subimages only to obtain local minima as solutions. Inherently, this method needs relatively small operations and memory, making it the popular approach in real-time DSP systems [2,3] and parallel VLSI chips [4-7]. The local method can be easily realized in the massive parallel structure as shown in Table 1. Nevertheless, there are many situations where this method may fail: the occlusion, uniform texture, ambiguity of the low texture, etc. Even further, the window method tends to yield blurred results around the object boundary.

In contrast, the global method, typically graph cut [8,9] and BP [10-12], deals with whole images, resulting in the global minima, analogously to the approximated global minimum principle. This approach has the advantage of low error rate but tends to need huge computational loads and memory resources. Recently, some researchers realized BP using PC aided by specialized parallel processors on GPU graphic card [13]. As described in Table 1, the so-called real-time BP can yield reasonable results only for the small throughput (MDE/s). Unfortunately, the specialized GPU relies upon high speed clocks and a small number of processors, which cannot be regarded as fully parallel architecture. Thus, it has the throughput limitation. Nevertheless, this system is successfully used in the real-time computer vision area [14]. There is no full parallel system that has fast computational power (MDE/s) for the high resolution images or the fast frame rates. Further, there is no genuine compact hardware

* Correspondence: mrzoo@postech.ac.kr
Department of Electrical Engineering, Pohang University of Science and Technology, Pohang, 790-784, South Korea

Table 1 Comparison of several real-time stereo systems

System	Style	MDE/s	Processor, no. PE	Clock speed
Adaptive window [5]	Local	819	ASIC, 512	200 MHz
DP chip [19]	Local	295	FPGA, 128	50 MHz
Real-time DP [20]	Semi-Global	205	MMX, 8	2.2 GHz
Real-time BP [13]	Global	19.7	GPU, 26	670 MHz

dedicated to the global stereo matching in real time. Most of the existing systems are impractical in terms of size, power requirement, and expense and are not suitable for compact applications like robot vision.

If a massive parallel architecture is realized as shown in Figure 1 then the computational time may be reduced drastically. However, this global matching architecture is not workable simply because of the enormous data bus bandwidth between the processors and the big external memory resource. In an effort to avoid this bottleneck, the memories must be evenly distributed throughout the processors so that each processor may access its own memory unhindered by the others. This distributed approach also raises problems when the number of processors is excessively large and the memory size is too big, making the VLSI implementation a formidable task. Therefore, we need to use distributed internal memories of small size, which can be easily accessed by many processors simultaneously.

Consider the one chip solution with a systolic array and efficient memory configuration. To avoid the huge memory, we tried to implement the BP on the FPGA by reducing the memory size [15], which is similar to the hierarchical iteration sequence [16]. In this paper, we use IF scheme [16] for our architecture and make it 2 times smaller than IF considering the message propagation direction, as we will call “Fast belief propagation (FBP)”. Based on this method, we built a full parallel architecture that is efficient in memory usage as well as equivalent to the original belief propagation (BP) method in terms of accuracy.

For a real-time application with small and compact hardware, GPU- and CPU-based system is not good due to their bulky size. We used this architecture to build a stereo vision chip and observed the expected performance–realtime and small memory for high precision depth images.

The remainder of this paper is organized as follows. Section 2 explains the background of the belief propagation. Section 3 defines a layer structure and explains an FBP sequence. A new iteration filter algorithm considering iteration directions is described in Section 4. For a VLSI realization, Section 5 suggests a parallel architecture and its memory complexity. Experiments are presented in Section 6. Section 7 draws conclusions on our newly developed architecture.

2 Review of belief propagation

The basic concept of belief propagation (BP) is to find iteratively the maximum a posteriori (MAP) solution on a 2-D Markov random field (MRF). All the parameters and variables are defined on the 2-D graph Figure 2 (we use the notation from [10]). P: a set of nodes on 2-D MRF, which in fact corresponds to pixels on an image. D: a set of hidden states stored in the nodes. $\boldsymbol{p} \in P$: a node that is located on the coordinate $\boldsymbol{p} = (p_0, p_1)$. $d_{\boldsymbol{p}} \in D$: a hidden state at $\boldsymbol{p}$. g^l, g^r: left and right images of N_0 by N_1 size. Also, N_E denotes the edge set and therefore, $(\boldsymbol{p}, \boldsymbol{q}) \in N_E$ for an edge between two nodes $\boldsymbol{p}$ and $\boldsymbol{q}$.

With the help of these notations, the pairwise MRF energy model can be defined as determining the estimate $\hat{d}$, given an energy function $E(\cdot)$:

$$\hat{d} = \arg\min_{d} E(d), \tag{1}$$

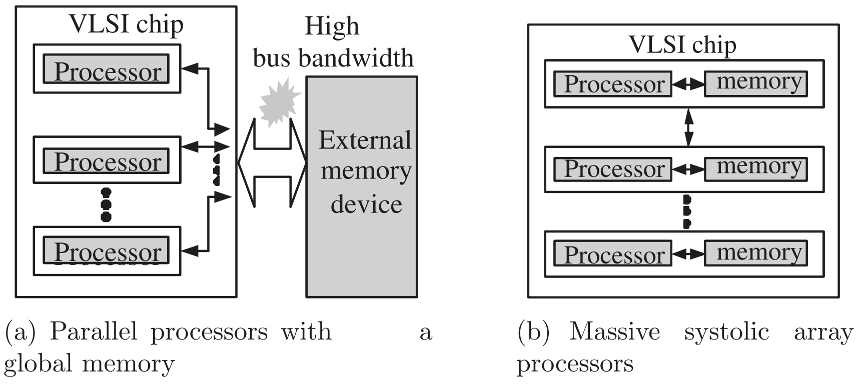


Figure 1 Alternative architectures for parallel algorithms. (**a**) Parallel processors with a global memory. (**b**) Massive systolic array processors.

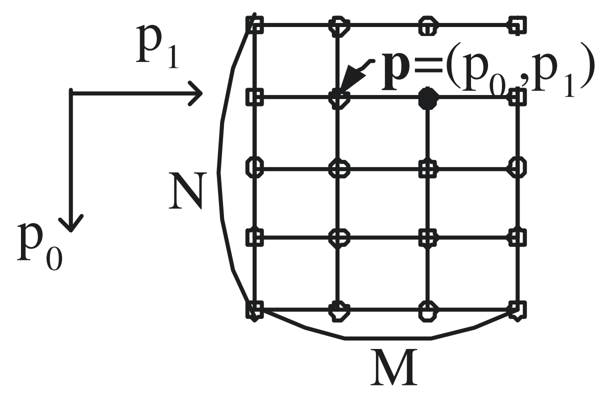


Figure 2 A 2-D regular graph which corresponds to a 2-D image.

$$E(d) = \sum_{(p,q)\in N_E}^{d} (d_p, d_q) + \sum_{p\in P} D_p(d_p). \quad (2)$$

$D(d_p)$ is the *data cost* for the node p having the state d_p. Similarly, $V(d_p, d_q)$ is the *edge cost* for a pair of neighbor nodes p and q having states d_p and d_q, respectively.

We assume a condition of parallel optics without the loss of generality. Then, stereo matching simply involves finding a point $(p_0, p_1 + d_p)$ in the right image which corresponds to a point (p_0, p_1) in the left image. Thus, the hidden state d_p represents the offset between the corresponding pixels, as is called *disparity*.

At each state d_p, the data cost constrained by the left and right images is defined as

$$D_p(d_p) = \min(C_d|g^r(p_0, p_1 + d_p) - g^l(p_0, p_1)|, K_d), \quad (3)$$

where C_d and K_d are a weighting factor and upper bound of the cost, respectively. This upper bound is useful in making the data cost robust to occlusions and artifacts that may violate the common assumptions that the ambient brightness must be uniform.

Also, the disparity should vary smoothly almost everywhere except at some places like object boundaries. In order to allow this discontinuity, we keep the edge cost $V(d_p, d_q)$ constant whenever the difference becomes larger than the predefined parameter K_d:

$$V(d_p, d_q) = \min(C_v|d_p - d_q|, K_v), \quad (4)$$

where C_v and K_v are similarly defined as the constant.

Finding the state $\hat{d}$ with minimum energy in Equation 1 amounts to the estimation problem with MAP. As is well known, the approximated MAP solution $\hat{d}$ can be estimated using the following BP update [10]:

$$m^l_{pq}(d_q) = \min_{d_p}\left(V(d_p, d_q) + D_p(d_p) + \sum_{r\in N(p)\backslash q}\left(m^{l-1}_{rp}(d_p) - \alpha\right)\right), \quad (5)$$

$$\alpha = \frac{1}{S}\sum_{d_p} m^{l-1}_{rp}(d_p). \quad (6)$$

$N(p)\backslash q$ is the neighbors of node p excluding q, α is the normalization value, and S is the state size. This equation expresses the following mechanism. The message $m^l_{pq}(d_q)$ at node p is updated at time l and then sent to the neighbor node q. After L iterations, the expected $\hat{d}_p$ at each node can be decided with Equation 7.

$$\hat{d}_q = \operatorname*{argmin}_{d_q}\left(D_q(d_q) + \sum_{p\in N(q)} m^L_{pq}(d_q)\right). \quad (7)$$

Let us explain the hierarchical BP in brief. It is based on the iteration scheme in multiple different scale levels. Between the levels, 2 × 2 scale change is considered to aid the coarse-to-fine iteration. According to this scheme, we need to over-sample the message and data costs in the coarse level to obtain the cost for the finer level. In this paper, L^k, $l^k \in [1, L^k]$, $p^k = \left[p^k_0 p^k_1\right]$, m^k, and $D^k_{p^k}$ denote the iteration number, the iteration time index, the node, message, and data cost in the $M/2^k$ by $N/2^k$ hierarchical graph of the scale level $k \in [0, K - 1]$, respectively. Here, K - 1 means the coarsest level. As shown in Figure 3, the data cost at k is calculated from the data cost at k - 1 by the summation over a 2 × 2 block. At the scale level 0, the data cost $D^0_{p^0}(d)$ is equivalent to $D_p(d_p)$ that is calculated from the left and right image pixel:

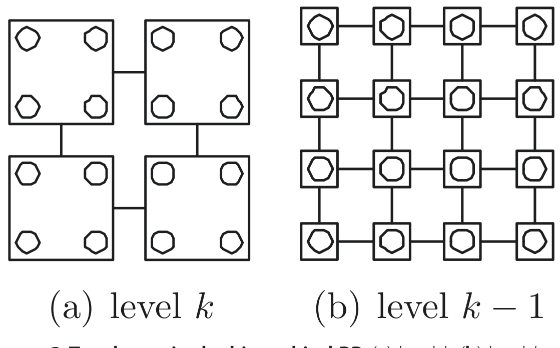


Figure 3 Two layers in the hierarchical BP. (**a**) level *k*; (**b**) level *k* - 1.

$$\begin{aligned} D^k_{\boldsymbol{p}^k}(d) &= \sum_{e_0=0}^{1}\sum_{e_1=0}^{1} D^{k-1}_{\left[2p_0^k+e_0 \quad 2p_1^k+e_1\right]}(d) \\ &= \sum_{e_0=0}^{2^k-1}\sum_{e_1=0}^{2^k-1} D^{0}_{\left[2^k p_0^k+e_0 2^k \quad p_1^k+e_1\right]}(d). \end{aligned} \tag{8}$$

If the memory complexity at each node is B bits, the overall memory size is $\sum_{k=0}^{K-1} B(N/2^k)(M/2^k)$ bits.

3 The proposed fast belief propagation sequence

In this section, we propose our FBP algorithm and architecture that enable us to run the BP on the FPGA with tiny distributed RAMs and show the remarkable memory reduction. It is 2 times smaller than the Iteration Filter's memory reduction scheme [16]. Before entering this section, I recommend for readers to understand the Iteration Filter scheme [16] that is wholly different from the normal iteration sequence and shows the amazing memory reduction effect. We redesign the Iteration Filter algorithm and implement it on the FPGA.

If we consider a separate layer for each iteration, then we can build a stack of layers. In this structure, the iteration can be represented as the upward propagation. Thus, Figure 4 can be redrawn as Figure 5. From this interpretation, we are considering the 2D graph with the iteration as the 3D layer graph (p_0, p_1, l) with the propagation. Let us define message and data cost sets at each node and layer l as:

$$M(\boldsymbol{p}, l) = \left\{ m^l_{\boldsymbol{pq}}(d_q) | d_q \in [0, S-1],\ \boldsymbol{q} \in N(\boldsymbol{p}) \right\}, \tag{9}$$

$$D(\boldsymbol{p}, l) = \left\{ D_{\boldsymbol{pq}}(d_q) | d_q \in [0, S-1] \right\}. \tag{10}$$

From these definitions, we can simplify the message update function in Equation 5 as:

$$M(\boldsymbol{p},l) = f(M(N(\boldsymbol{p}),l-1),\ D(\boldsymbol{p},l-1)), \tag{11}$$

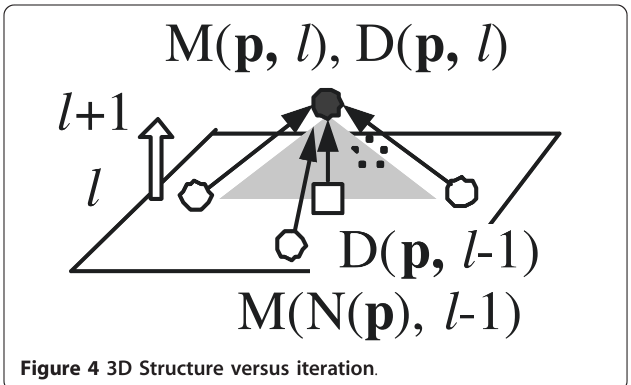


Figure 4 3D Structure versus iteration.

$$D(\boldsymbol{p},l) = D(\boldsymbol{p},l-1), \tag{12}$$

where ($N(\boldsymbol{p})$, l - 1) and $M((N(\boldsymbol{p}), l - 1)) = \{M(\boldsymbol{u}, l - 1)| u \in N(\boldsymbol{p})\}$ represent the neighbor nodes and their message costs in the buffer, respectively.

As an initialization stage, each node $\boldsymbol{p}$ observes the input to obtain the data cost $D(\boldsymbol{p}, 0)$. Afterward, in every iteration l, each node calculates the new message $M(\boldsymbol{p}, l)$ according to the update function $f(\cdot)$ and after then stores it as $M(\boldsymbol{p}, l - 1)$ in the buffer.

Let $\boldsymbol{Q}(l)$ and $M(\boldsymbol{Q}(l))$ denote the set of nodes in lth layer and its message cost set, respectively. Then, $M(\boldsymbol{Q}(l))$ can be updated from $M(\boldsymbol{Q}(l - 1))$ and $D(\boldsymbol{Q}(l - 1))$ in the buffer:

$$M(\boldsymbol{p},l) = f(M(N(\boldsymbol{p}),l-1),D(\boldsymbol{p},l-1])), \tag{13}$$

$$\begin{aligned} &(\boldsymbol{p}, l) \in \boldsymbol{Q}(l),\ (N(\boldsymbol{p}), l-1) \in \boldsymbol{Q}(l-1), \\ &\boldsymbol{Q}(l) = \{(p_0, p_1, l) | p_0 \in [0, N-1], p_1 \in [0, M-1]\}. \end{aligned} \tag{14}$$

Consider a new FBP computing order based on the IF scheme. Note that $Q(p_0 - l, l)$ forms a *linear* array of M nodes on the p_1 axis in the lth layer. If we collect all the layers of $Q(p_0 - l, l)$ in terms of p_0 then $\boldsymbol{Q}(p_0)$ forms a *planar* array of LM nodes:

$$Q(p_0, l) = \{(p_0 - l, p_1, l) | p_1 \in [0, M-1]\}, \tag{15}$$

$$\boldsymbol{Q}(p_0) = \{Q(p_0, l) | l \in [1, L]\}. \tag{16}$$

with the notation $Q(p_0 - l, l)$ and $\boldsymbol{Q}(p_0)$, we can build an efficient computation order. We will call this memory-efficient BP sequence, *FBP*. The cost of $\boldsymbol{Q}(p_0)$ is updated from the buffer of the message M ($\boldsymbol{Q}(p_0 - 1)$), $M(\boldsymbol{Q}(p_0 - 2))$, and data cost $D(\boldsymbol{Q}(p_0 - 1))$ as described in Algorithm 1. As shown in Figure 6, our memory resource consists of local and layer buffers. The layer buffer stores all the layers' costs of $\boldsymbol{Q}(p_0 - 1)$ and $\boldsymbol{Q}(p_0 - 2)$. The local buffer holds only one layer's costs on $Q(p_0, l - 1)$.

Algorithm 1: FBP algorithm

For ℓp_0 in the lth iteration layer profile, each node at (p_0 - l, p_1) and the lth layer can be updated from the node at $N(p_0 - l, p_1)$ and the (l - 1)th layer. Thus, as shown in Figure 7 and Equation 17, the nodes at $Q(p_0, l)$ can be computed from $Q(p_0, l - 1)$, $Q(p_0 - 1, l - 1)$, and $Q(p_0 - 2, l - 1)$.

$$\{Q(p_0-2, l-1), Q(p_0-1, l-1), Q(p_0, l-1)\} \tag{17}$$

$$= \{(N(p_0 - l, p_1), l-1) | p_1 \in [0, M-1]\}. \tag{18}$$

$Q(p_0, l)$ and $Q(p_0, l - 1)$ belong to $\boldsymbol{Q}(p0)$. Hence, given the layer buffer $\boldsymbol{Q}(p0 - 2)$ and $\boldsymbol{Q}(p0 - 1)$ and the local buffer $Q(p_0, l - 1)$, the costs in $Q(p_0, l)$ are updated at

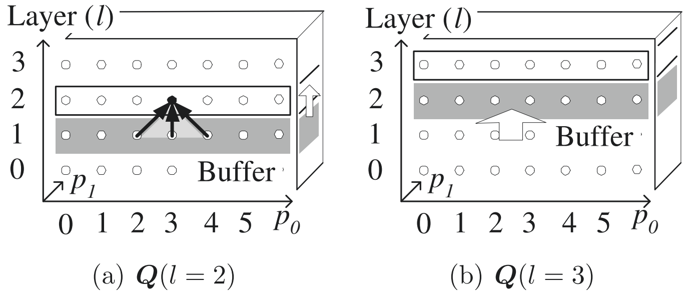


Figure 5 Prior iteration sequences in the 3D layer graph. (**a**) $\boldsymbol{Q}(l = 2)$; (**b**) $\boldsymbol{Q}(l = 3)$.

each layer l recursively, which sequence is described in Figure 6a, b, and 6c. That is, given $M(\boldsymbol{Q}(p_0 - 1))$, $M(\boldsymbol{Q}(p_0 - 2))$, and $D(\boldsymbol{Q}(p_0 - 1))$, we can calculate $M(\boldsymbol{Q}(p_0))$. The new costs in local buffer should be stored in the layer buffer to process the next set $\boldsymbol{Q}(p_0 + 1)$ in the next time. This sequence shifts the layer buffer to the p_0 axis direction. Then, for p_0 from 0 to $N + L - 1$, we can obtain the final iterated message $M(Q(p_0, L))$. For the example, as shown in Figure 6b, and 6c, the location of the buffer is changed from $\boldsymbol{Q}(p_0 = 5)$ to $\boldsymbol{Q}(p_0 = 6)$ by our sequence.

In the hierarchical case, as shown in Figure 6d, we can construct the hierarchical layer structure by considering the hierarchical iterations. At each level, we can follow the FBP sequence at each level only if considering two by two scale changes between levels. Please refer to [16] for the detailed hierarchical memory reduction scheme of IF.

If we use the notation B as BP memory complexity at each node and consider the nodes of L^k by $M/2^k$ size in $\boldsymbol{Q}^k(\cdot)$, we need two layer buffers of the $BL^kM/2^k$ size and one local buffer of $BM/2^k$ size at each level k. Thus, compared with the hierarchical BP, the overall memory size can be reduced from $\sum_{k=0}^{K-1} B(N/2^k)(M/2^k)$bits to $\sum_{k=0}^{K-1} B(2L^k + 1)(M/2^k)$ bits by adopting the iteration filter scheme to our VLSI sequence. This can be shown as follows.

$$\text{Reduction rate} = \frac{\sum_{k=0}^{K-1} B(N/2^k)(M/2^k)}{\sum_{k=0}^{K-1} B(aL^k + 1)(M/2^k)}, \tag{19}$$

$$(a = 2). \tag{20}$$

If we approximately consider the total memory as the 0th level, the reduction rate amounts to $N/(2L^0 + 1)$ times when $2L^0 \ll N$. In summary, the update sequence must be effective whenever N, one of the image size components is big, and L^0, the iteration number, is small.

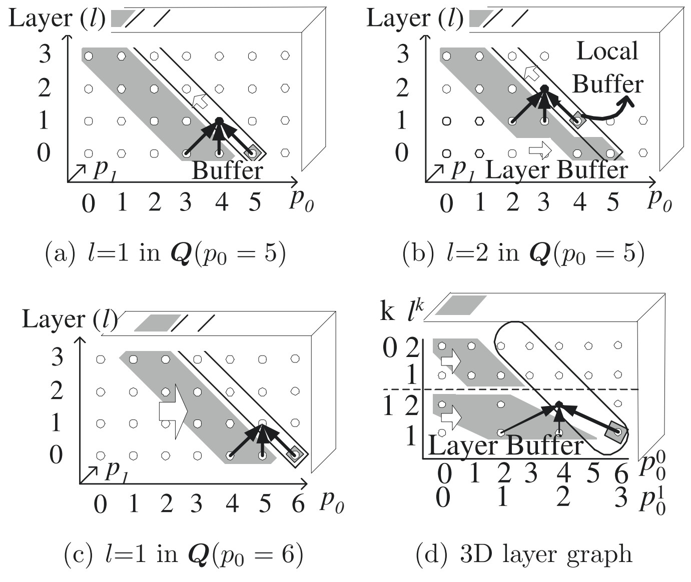


Figure 6 The message update sequences. (**a**) $l = 1$ in $\boldsymbol{Q}(p_0 = 5)$; (**b**) $l = 2$ in $\boldsymbol{Q}(p_0 = 5)$; (**c**) $l = 1$ in $\boldsymbol{Q}(p_0 = 6)$; (**d**) 3D layer graph.

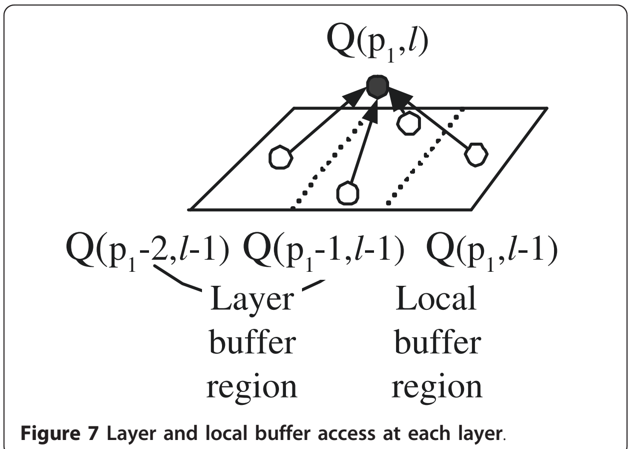


Figure 7 Layer and local buffer access at each layer.

4 New iteration sequence considering the iteration direction

Let us consider the message propagation direction for the further memory reduction. As shown at the definition of $M(\boldsymbol{p}, l)$ in Equation 9, we assumed that the messages of all the directions are stored in the buffer. However, due to the message propagation direction information, we can reduce the memory resource 2 times smaller. Among the neighbor messages $M(N(\boldsymbol{p}), l - 1)$, only $m_{\boldsymbol{rp}}^{l-1}(d_{\boldsymbol{p}})$ for $\boldsymbol{r} \in N(\boldsymbol{p})$ is necessary for updating $M(\boldsymbol{p}, l)$. In Figure 8, let us denote the message propagation direction as $\Delta = \boldsymbol{p} - N(\boldsymbol{p})$. The needed messages for the update are the ones that are propagated from neighboring node $N(\boldsymbol{p})$ to $\boldsymbol{p}$. Except for the message of the direction Δ = [+1 0] that is propagated from local buffer, all the other messages are being loaded from the layer buffer. This is summarized at the access column part of Table 2. But, in the data cost case, as shown in Figure 9, we do not need to consider the propagation direction and simply read $D(Q(p_0 - 1, l - 1))$ in the layer buffer $\boldsymbol{Q}(p_0 - 1)$ for $D(Q(p_0, l))$ because $D(Q(p_0, l))$ is equal to $D(Q(p_0 - 1, l - 1))$ like Equation 12.

Table 2 Number of messages stored at each node in the buffer

	Access(Δ)		Store(Δ)	
	Directions	No.	Directions	No.
$Q(p_0, l - 1)$	[+1 0]	1	[±1 0], [0 ± 1]	4
$Q(p_0 - 1, l - 1)$	[0 ± 1]	2	[-1 0], [0 ± 1]	3
$Q(p_0 - 2, l - 1)$	[-1 0]	1	[-1 0]	1

As explained in the FBP algorithm, at each update time, the location of the buffer is shifted to p_0 axis being updated by the new cost. The newly updated messages and data cost in the local buffer should be stored in the layer buffer for the processing of the next $\boldsymbol{Q}(p_0 + 1)$. Thus, if the messages from all possible directions be saved in the local buffer, then some messages can be transferred to $Q(p_0 - 1, l - 1)$. At the same time, some old costs in $Q(p_0 - 1, l - 1)$ are moved to $Q(p_0 - 2, l - 1)$ in a similar way. With this scheme, the number of propagation directions to be stored at the buffer is described at the store(Δ) part in Table 2.

From the definition in Equations 15 and 16, the number of nodes is LM for both $\boldsymbol{Q}(\boldsymbol{p}_0 - 2)$ and $\boldsymbol{Q}(\boldsymbol{p}_0 - 1)$ and M for $Q(\boldsymbol{p}_0 - (l - 1), l - 1)$. Table 2 shows the required number of messages and data costs at each node. The number of states is S, and the number of bits for the message cost and data cost is B_m and B_D, respectively. Then, by multiplying all the parts, we can calculate the memory size of the buffer as shown in Table 3.

If $B = 4B_mS + B_DS$, then we can obtain as follows:

$$\text{Reduction rate} = \frac{\sum_{k=0}^{K-1} B(N/2^k)(M/2^k)}{\sum_{k=0}^{K-1} B(aL^k + 1)(M/2^k)}, \tag{21}$$

$$(a = 1). \tag{22}$$

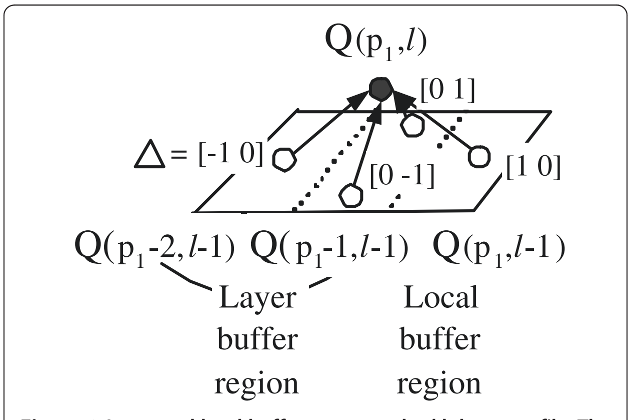


Figure 8 Layer and local buffer access at the *l*th layer profile: The propagation direction of the message is denoted as vector Δ.

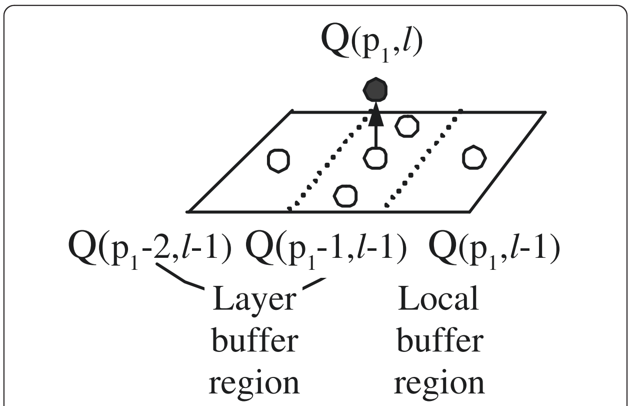


Figure 9 Layer and local buffer access at each layer.

Table 3 FBP buffer size

Buffer	Message	Data cost
Layer buffer $\boldsymbol{Q}(\boldsymbol{p}_0 - 2)$	B_mSML	0
Layer buffer $\boldsymbol{Q}(\boldsymbol{p}_0 - 1)$	$3B_mSML$	B_DSML
Local buffer $Q(\boldsymbol{p}_0 - (l - 1), l - 1)$	$4B_mSM$	B_DSM
Total	$4B_mSM\ (L + 1)$	$B_DSM\ (L + 1)$

If you compare Equations 20 and 22, the value a is changed from two to one. Therefore, due to the propagation direction of BP, we can obtain 2 times smaller memory than the iteration filter [16].

5 Systolic VLSI architecture

Our architecture has four hierarchical levels. This level affects the iteration times. The higher hierarchical levels make iteration times smaller because the message can be converged faster in the coarse level. In our FBP architecture, it makes the memory size much smaller because our memory resource is dependent on iteration times. The HFBP algorithm can be easily realized with a systolic array architecture. As depicted in Figure 10, it consists of identical PE groups with nearest neighbor communication. In our implementation, it has a total of 20 PE groups. The PE group is divided into eight identical PEs as shown in Figure 11. Therefore, it amount to 160 PEs for processing a pair of 160 × 240 images. Figure 12 represents the local and layer buffer assignment for each $PE_{k\ =\ 1,\ldots,7}$ in the PE group. Thus, the $8/2^k$ number of PEs in the group is activated at level k due to the scale-down of the hierarchical structure.

As shown in Figure 11, the PE group consists of two parts. The first part is the data cost module that computes the initial costs using the left and right scan lines of the images. The other group is for updating the message and data cost. The pixel data from the left and right cameras enter into the PE group and each PE computes the data cost and the new message using the old messages from neighboring PEs and its own buffers. Figure 13 shows the data cost module that calculates the hierarchical data costs along the levels 0 to 3. In Figure 13b, the left and right scan lines are first stored in the registers, and then the right scan line registers are shifted by state d to compute $D_p(d_p)$ according to Equation 3. For each state, the data cost $D_p(d)$ at level 0 is obtained by taking the absolute difference of the left and right pixel values. On the other hand, B in Figure 13c is used for computing the higher level data cost $D^k_{\boldsymbol{p}^k}(d)$. For the level k's cost, the previous level k-1 data costs are summed up and then accumulated over 2^k scan lines. This is equivalent to applying the summation of the $2^k \times 2^k$ window for the hierarchical data cost; each data cost is used by the PE at each level. Data costs at each level, computed in the data cost module, are processed and saved in the corresponding PEs and buffers. See Figure 13. As described in Figure 12, the multiplexer (MUX) selects the messages and data costs at each level from which new messages and data costs can be updated and saved at the local buffer. Meanwhile, the old costs in this buffer are shifted into the layer buffer. In the four scale levels, 4-to-1 message multiplexer (MUX) is used.

For S number of states, the time complexity O(S) is needed to update one message at each node by forward, backward, and normalization operations [10]. Normally, it needs 3S steps. As explained in Equation 9, four messages that are propagated to neighbor nodes need to be computed at each node. To compute these messages, our system needs only $6S$ clocks due to the pipeline structure. See Figure 14.

Since $(M/2^k)$ nodes are handled by $(M/2^k)$ processors in parallel on p^k_1 axis, the total required clocks are reduced from $\sum_{k=0}^{K-1} 6S(M/2^k)(N/2^k)$ to $\sum_{k=0}^{K-1} 6SL^k(N/2^k)$. As a whole, each PE calculates the messages in parallel by accessing the local buffer or the layer buffer which is located in the neighboring PEs or PE groups.

6 Experimental results

Our new architecture has been tested by both a simulation and FPGA realization.

6.1 Software simulation

First, we verify our VLSI algorithm using the Middlebury data set with a software simulation. In the previous sections, we presented a new architecture which is

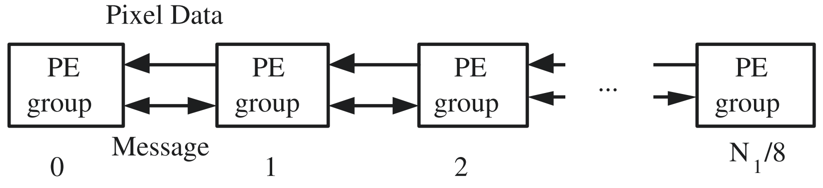


Figure 10 Systolic array architecture of FBP.

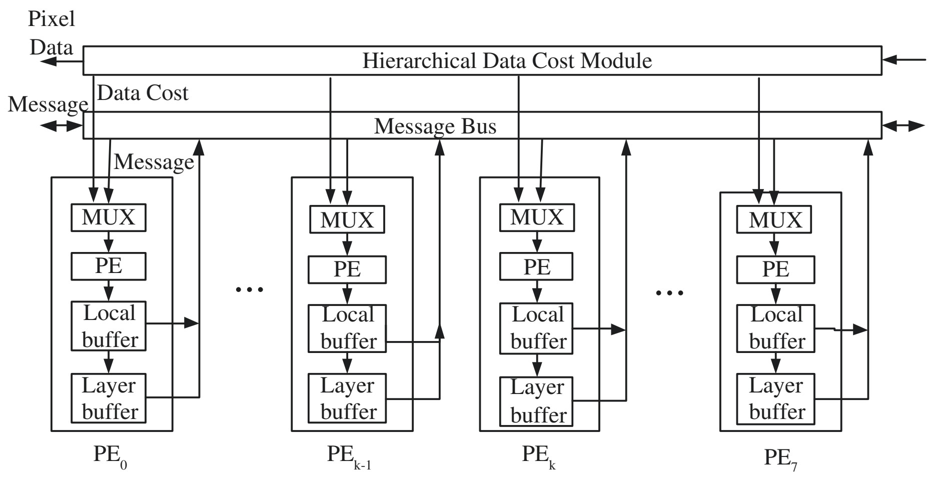


Figure 11 Internal architecture of the PE group.

equivalent to HBP in terms of input-output relationship and which is a systolic array with a small memory space. Hence, it is suitable for VLSI implementation.

The requirement for both memory resource and computation time is only dependent on the layer number L^k. Therefore, it is reasonable to analyze the performance in terms of iterations as well as various images. We specify the accuracy using the following equation.

$$\begin{aligned}\text{error}(\%) &= \frac{100}{N}\sum_{(p_0,p_1)\in Pm}\left(|\hat{d}(p_0,p_1) - d_{True}(p_0,p_1)| > 1\right),\\ N &= \sum_{(p_0,p_1)\in Pm} 1,\end{aligned}$$

where $\hat{d}$ is the estimated disparity, d_{True} is the true disparity, Pm is the area except for the occlusion part, and N is the pixel number in its area. This error means the rate where the disparity error is larger than 1.

For fair comparison, the same parameters are used throughout the experiments: C_v = 28, K_v = 57, C_d = 4, and K_d = 60. Figures 15 and 16 are the results of the Middlebury test images. In Figure 15, four levels are used both for HBP and HFBP. The layer number at each level is assigned as (8, 8, 8, 8) from coarse-to-fine scale levels. With the same iterations, HFBP and HBP show the same lower error results.

Figure 16 shows the relationship between the iteration layers and FBP's average memory reduction rates when compared with HBP, where the same iteration times, (L, L, L, L), are applied for each layer. Due to the hierarchical scheme, the iteration converged around 28 iterations and yielded 0.8% maximum error. The remarkable result, though, is the memory reduction, which is around 32 times. In fact, even less memory is possible for a higher error rate. Thus, this architecture makes the performance scalable between the space and accuracy.

Table 4 compares our FBP FPGA with other real-time systems in terms of error. It is evident that our method shows almost the same error as Real-time BP. Here, real-time BP is also based on the HBP algorithm [10]

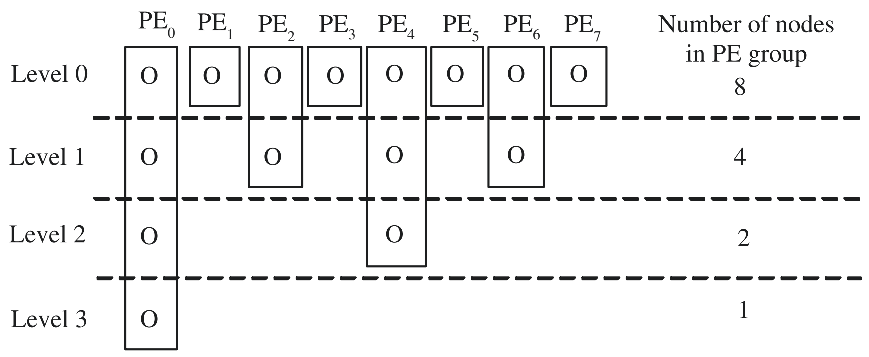


Figure 12 Activated PE and hierarchical buffer assignment at each level in the PE group.

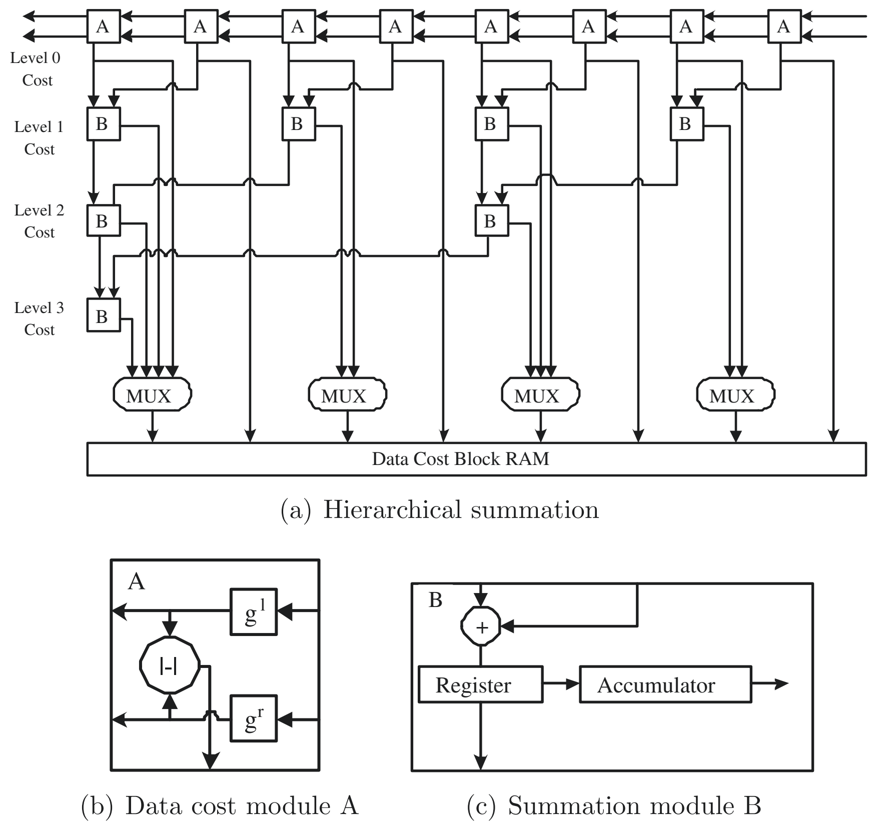


Figure 13 Architecture for hierarchical data cost module. (**a**) Hierarchical summation. (**b**) Data cost module A. (**c**) Summation module B.

and known for the lowest error among real-time systems.

6.2 FPGA implementation

We developed the VHDL code on FPGA as follows using the specs: S = 32, B_m = 7, B_D = 10, (L^3, L^2, L^1, L^0) =(8, 8, 8, 10), 15 frames/sec at 160 × 240 or 160 × 480 image.

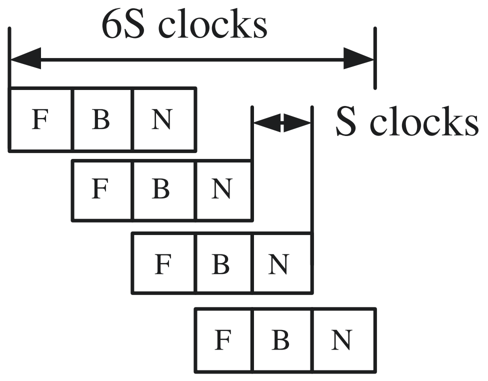


Figure 14 The pipeline message computation sequence with forward (F), backward (B), and normalization (N) operations.

If we use Equation 22, the total buffer size becomes 3.3 Mb, which is 19 times smaller than HBP's 62 Mb. Also, for processing one frame image, the 160 PEs need 0.6 MHz clocks. This speed amounts to 18.8 MHz clocks processing 15 frames in 1 s. In order to achieve maximum 36.8 MDE/s throughput for a 160 × 480 image, only a 18.8 MHz system clock is necessary ideally. Tables 5 and 6 show the computational performance between our new system and other systems. The local matching is effectively implemented as the pipeline and parallel structure since it does not need to access the huge memory size iteratively. GPU is the SIMD processor with a high speed core clock and external memory clock. Even if it is not a full parallel structure, it operates in real time due to the high clock speed and small number of parallel processors. But, our system is the fully parallel and can operate at the much slower 25 MHz clock speed. Furthermore, our system has one chip solution that consumes less memory resources inside the FPGA and can easily be parallelized to multiple chips due to the systolic array architecture. This simple and regular architecture is suitable for VLSI implementation. In addition, the semi-global matching [17] needs two frames' latency times, but our FBP has

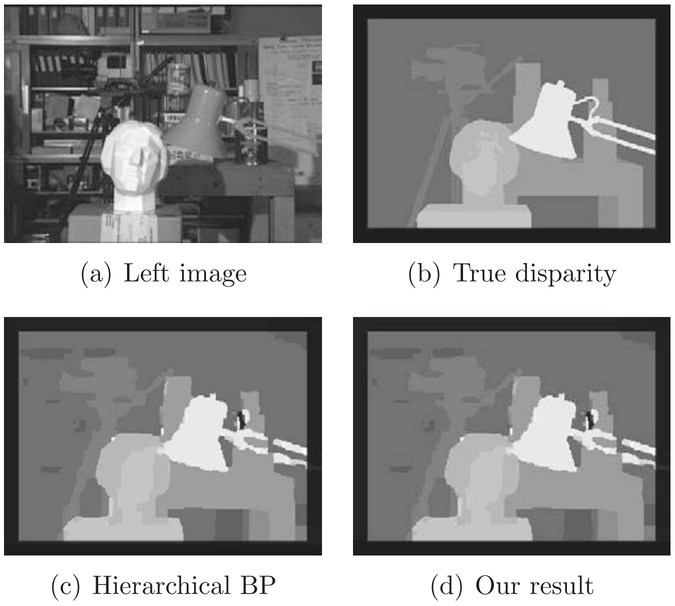


Figure 15 Output comparisons of Tsukuba images at 28 layers. (**a**) *Left* image. (**b**) True disparity. (**c**) Hierarchical BP. (**d**) Our result.

the latency time below one frame due to the processing sequence like the filter.

For a higher resolution solution, we need to increase the computational power. It is possible by simply cascading several chips together in proportion to the image size or increasing the clock speed.

It has been observed that the FPGA, incorporating 160 PEs, operates at a 25 MHz clock rate. For convenience, more specifications are summarized in Table 7. Ideally, to store the local and layer buffers, our necessary memory size is around 3.3 Mb. But, in the real implementation, we used 395 internal block RAMs in FPGA, which amount to 7.1 Mb. Incidentally, assigning each buffer to Block RAMs may result in unused leak memory, that is waste, that can be avoided in full ASICs.

Table 4 Disparity error comparison of several real-time methods (%)

Image	System	Tsukuba	Map	Venus	Sawtooth
Our FBP	Virtex2	1.7	0.5	0.7	0.8
Real-time BP [13]	Geforce 7900	1.5	NA	0.8	NA
Accelerated BP [21]	Virtex2	2.6	0.2	0.8	0.8
Semi-Global matching [17]	Virtex5	4.1	NA	2.7	NA
Trellis DP [19]	Virtex2	2.6	0.9	3.4	1.9
Real-time DP [20]	MMX	2.9	6.5	6.5	6.3
Local matching [22]	Virtex5	9.8	NA	5.3	NA

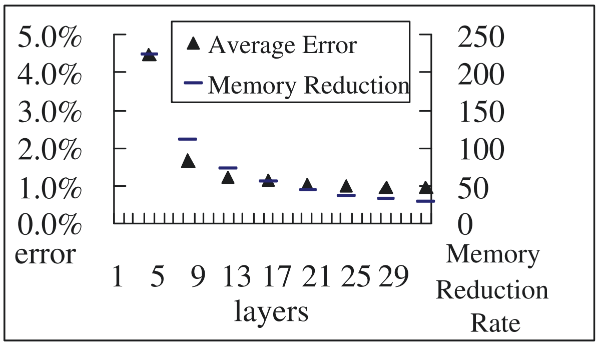


Figure 16 Relation between average error convergence and memory reduction R_m in Middlebury test images.

Table 5 Comparisons of computation time between the real-time systems

Spec	System	Image	Levels	fps
Our FBP, One FPGA	FPGA, Virtex2	160 × 480	32	15
Two FPGAs	FPGA, Virtex2	320 × 480	32	15
Semi-global matching [17]	FPGA, Virtex5	640 × 480	128	103
Local matching [22]	FPGA, Virtex5	640 × 480	64	230
Accelerated BP [21]	FPGA, Virtex2	256 × 240	16	25
Real-time BP [13]	GPU, Geforce 7900	320 × 240	16	16
Real-time DP [20]	CPU, MMX	320 × 240	100	26.7
Trellis DP [19]	FPGA, Virtex2	320 × 240	128	30

Table 6 Comparisons of hardware spec. between the real-time systems

Spec	System	clock	PEs	Int. Mem.	Ext. Mem.
Our FBP, One FPGA	Virtex2	25 MHz	128	3.3 Mb	No
Two FPGAs	Virtex2	25 MHz	256	6.6 Mb	No
Semi-global matching [17]	Virtex5	133 MHz	30	3.3 Mb	Yes
Local matching [22]	Virtex5	93 MHz	64	5.8 Mb	No
Real-time BP [13]	Geforce 7900	670 MHz	26	NA	62 Mb
Accelerated BP[21]	Virtex2	65 MHz	24	2 Mb	9 Mb
Real-time DP [20]	MMX	NA	NA	NA	Yes
Trellis DP [19]	Virtex2	50 MHz	128	Yes	No

Table 7 Additional hardware specifications used in our system

	Spec. (Resource usage percentage)
FPGA	Xilinx Virtex II pro-100
Number of multiplier	0
Number of divider	0
Number of slice flip flops	30,585 (34%)
Number of 4 input LUTs	46,812 (53%)

The new architecture is implemented in FPGA as shown in Figure 17. Here, Figure 17a is a block diagram and Figure 17b is a photo of the actual board. As can be seen, two cameras supply a pair of video streams and two FPGAs perform preprocessing and our FBP algorithm. The disparity map forms a stream from FPGA to a grabber through Camlink cables. From the video RAM on the grabber board, the PC reads the disparity data and converts it to a gray scale image for the observation. Figure 18 shows the typical video output of the FPGA.

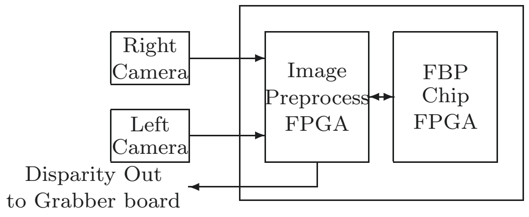
7 Conclusions

In this paper, a new architecture for the global stereo matching algorithm has been presented. The key idea is to rearrange the computation order in BP to obtain a parallel and memory-efficient structure. As the results show, our system spends 19 times less memory than the ordinary BP. The memory space can be negotiated with the iteration number. The architecture is also scalable in terms of image size; the regular structure can be easily expanded by cascading identical modules.

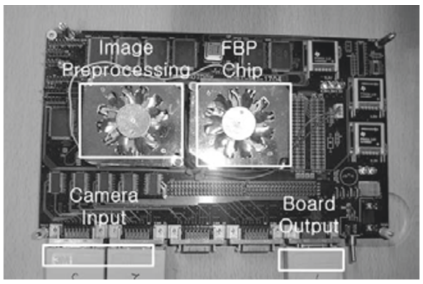
When applied to binocular stereo vision, this architecture shows the ability to process stereo matching in real time. Experimental results confirm that this array architecture easily provides high throughput with low clock speed where small iterations are guaranteed by the hierarchical iteration scheme.

In the future, we plan to realize this architecture with a small and compact ASIC chip. Beyond the programmable chips, we can simply expect a real-time chip with higher resolution and the lowest error rate with huge PE numbers. Unlike the bulky GPU and CPU systems, making the complex stereo matching system with a compact chip may lead to many real-time vision applications.

Furthermore, if we change the message and data cost model, our memory-efficient architecture can be considered to other BP-based motion estimation and image restoration [10]. The combined effort of parallel processing and efficient memory usage makes a chance to implement a compact VLSI chip. Furthermore, more general iterative algorithms can be considered, which communicate only neighbor pixels in the image, such as GBP typical cut [18]. As explained in [16], if we apply the IF scheme to these algorithms, we can reduce their memory resources to a tiny



(a) Overall system



(b) Hardware board

Figure 17 The overall hardware system. (**a**) Overall system. (**b**) Hardware board.

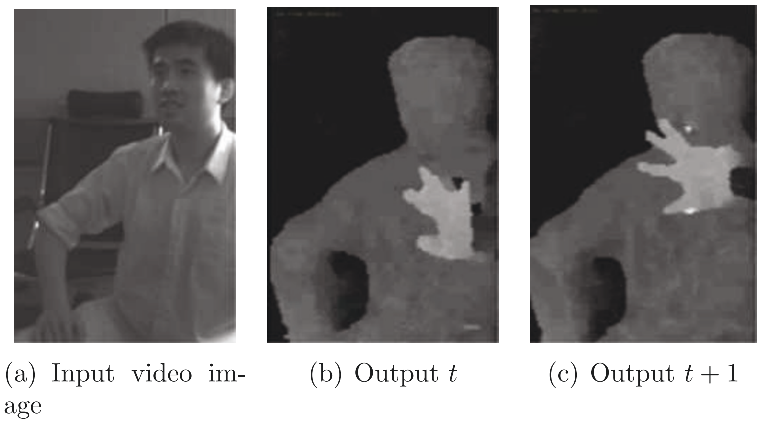


Figure 18 FPGA output for real images. (**a**) Input video image. (**b**) Output *t*. (**c**) Output *t* + 1.

size. Thus, if they have simple update logics for the iteration, then full parallel VLSI architectures may be realizable.

Acknowledgements
This work was supported by the following funds: the Brain Korea 21 project and the Ministry of Knowledge Economy, Korea, under the Core Technology Development for Breakthrough of Robot Vision Research support program supervised by the National IT Industry Promotion Agency.

Competing interests
The authors declare that they have no competing interests.

Received: 24 January 2011 Accepted: 17 August 2011
Published: 17 August 2011

References

1. Scharstein D, Szeliski R: **A taxonomy and evaluation of dense two-frame stereo correspondence algorithms.** *Int J Comput Vision* 2002, **47(1-3)**:7-42.
2. Kanade T, *et al*: **A stereomachine for video-rate dense depth mapping and its newapplications.** *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition* 1996.
3. Konolige K: **Small vision systems: Hardware and implementation.** *Proceedings of Eighth International Symposium Robotics Research* 1997.
4. Corke P, Dunn P: **Real-time stereopsis using fpgas.** *IEEE TEN-CON.Speech and Image Technologies for Computing and Telecommunications* 1997, 235-238.
5. Hariyama M, *et al*: **Architecture of a stereo matching VLSI processor based on hierarchically parallel memory access.** *The 2004 47th Midwest Symposium on Circuits and Systems* 2004, **2**:II245-II247.
6. Kimura S, *et al*: **A convolver-based real-time stereo machine (SAZAN).** *Proceedings of Computer Vision and Pattern Recognition* 1999, **1**:457-463.
7. Woodfill J, Von Herzen B: **Real-time stereo vision on the parts reconfigurable computer.** *IEEE Workshop FPGAs for Custom Computing Machines* 1997, 242-250.
8. Kolmogorov V, Zabih R: **Computing visual correspondence with occlusions using graph cuts.** *ICCV* 2001, **2**:508-515.
9. Xiao J, Shah M: **Motion layer extraction in the presence of occlusion using graph cuts.** *IEEE Trans Pattern Anal Mach Intell* 2005, **27(10)**:1644-1659.
10. Felzenszwalb PF, Huttenlocher DR: **Efficient belief propagation for early vision.** *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2004, **1**:I261-I268.
11. Zheng NN, Sun J, Shum HY: **Stereo matching using belief propagation.** *IEEE Trans Pattern Anal Mach Intell* 2003, **25(7)**:787-800.
12. MacCormick J, Isard M: **Estimating disparity and occlusions in stereo video sequences.** *Asian Conference on Computer Vision (ACCV)* 2006, 32-41.
13. Yang Q, *et al*: **Real-time global stereo matching using hierarchical belief propagation.** *The British Machine Vision Conference* 2006.
14. Mignotte M, Jodoin P-M, St-Amour J-F: **Markovian energy-based computer vision algorithms on graphics hardware.** *ICIAP'05, LNCS* 2005, **3617**:592-603.
15. Park S, Chen C, Jeong H: **VLSI Architecture for MRF Based Stereo Matching.** *7th International Workshop SAMOS* 2007, 55-64.
16. Park S, Jeong H: **Memory-efficient iterative process on a two-dimensional first-order regular graph.** *Opt Lett* 2008, **33(1)**.
17. Banz Christian, *et al*: **Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation.** *International Conference on Embedded Computer Systems (SAMOS)* 2010, 93-101.
18. Shental N, *et al*: **Learning and inferring image segmentations using the GBP typical cut algorithm.** *ICCV* 2003, 1243-1250.
19. Park S, Jeong H: **Real-time stereo vision FPGA chip with low error rate.** *International Conference on Multimedia and Ubiquitous Engineering* 2007, 751-756.
20. Forstmann S, *et al*: **Real-time stereo by using dynamic programming.** *CVPR, Workshop on Real-Time 3D Sensors and Their Use* 2004.
21. Park S, Jeong H: **High-speed parallel very large scale integration architecture for global stereo matching.** *J Electron Imaging* 2008, **17(1)**:010501.
22. Jin Seunghun, *et al*: **FPGA design and implementation of a real-time stereo vision system.** *IEEE Trans Circuits Syst Video Technol* 2010, **20(1)**:15-26.

doi:10.1186/1687-5281-2011-4
Cite this article as: Park *et al.*: Real-time stereo matching architecture based on 2D MRF model: a memory-efficient systolic array. *EURASIP Journal on Image and Video Processing* 2011 **2011**:4.