Stereo Matching using FBP: A Memory Efficient Parallel Architecture

Sungchan Park, Chao Chen and Hong Jeong,
Electronic Department, Pohang University of Science and Technology, Pohang 790-784, South Korea
E-mail: mrzoo, chen9810, hjeong@postech.ac.kr

Abstract—There are growing needs in computer vision applications for stereopsis, requiring not only accurate distance but also fast and compact physical implementation. Presently, the global matching techniques provide remarkably robust and precise results in stereo matching. Unfortunately, they suffer from huge computational and memory complexity. We have observed that changing the order of iterative computation in certain ways may result in remarkable memory saving. For an M by N image with L iterations, the new algorithm can reduce the memory resources from $O(MN)$ to $O(ML)$. When applied to the hierarchical BP, this algorithm records 19 times saving in memory space. The target architecture is systolic array, which consists of identical processors connected with nearest-neighbors only. We have realized this parallel architecture with the Xilinx vertexIIpro-100 FPGA to process the 160 × 240 video images with 25 MHz clock rate and observed 32 disparity levels at 15 frames/s.

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 scene. Finding the corresponding pairs is known to be the most challenging step in the binocular stereo problem.

Conventional methods can be categorized into local and global methods [15]. 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 memories, becoming the popular approach to real-time DSP systems [6], [11] and parallel VLSI chips [2], [16], [1], [5]. The local method can be easily realized in massive parallel structure. Nevertheless, there are many situations where this method may fail; occlusion, uniform texture, ambiguity of low texture, etc. Even further, the popular block matching tends to yield blurred results around the object boundary.

Contrarily, the global method, typically graph cut [10], [17] and BP [7], [8], [12], deals with the whole image, resulting in 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 PCs aided by specialized parallel processors on GPU graphics card [3]. The so-called real-time BP can yield reasonable results only for small throughput (MDE/s). Unfortunately, the specialized GPU relies upon high speed clocks and small number of processors, which cannot be regarded as fully parallel architecture. Nevertheless, this system is successfully used in several real-time computer vision area [9]. There is not yet any genuine system which has fast computational power (MDE/s) for the high resolution images or fast frame rates. Further, there is not yet genuine compact hardware dedicated to global stereo matching in real time. Most of the existing systems are impractical in terms with the size, power requirements, and expense and are not suitable for compact applications like robot vision.

If a massive parallel architecture is realized on a chip, then the computational time may be reduced drastically. However, this simple architecture is not workable simply because of the enormous data bus bandwidth between the processors and the external memory. In an effort to avoid bottleneck, memories must be evenly distributed throughout the processors so that each processor may access its own memory unhindered from others. This distributed approach also raises problems when the number of processors is excessively large and the memory size is too big, making 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.

Let’s consider one chip solution with a systolic array and efficient memory configuration. As an effort to avoid huge memory, we try to seek for a method to change the ordinary iteration scheme possibly by one-pass algorithm, as we will
call "Fast belief propagation (FBP)." Based on this method, we want to build a full parallel architecture which is efficient in memory usage as well as equivalent to the original BP method in terms with accuracy.

As a real-time application which needs small and compact hardware, we used this architecture for building stereo vision chip and observed the expected performance - real-time and small memory for high precision depth images.

2 Review of Belief Propagation

The basic concept of BP is to find iteratively the maximum a posteriori (MAP) solutions on 2-D MRF. The general framework for these problems can be described from now on (we use the notation from [7]). $P$: a set of nodes on 2-D MRF, which in fact correspond to pixels on an image. $D$: a set of hidden states stored in the nodes. $\mathbf{p} \in P$: a node which is located on the coordinate $\mathbf{p} = (p_0, p_1)$. $d_{\mathbf{p}} \in D$: a hidden state at $\mathbf{p}$. $g^l$, $g^r$: left and right images of $N_0$ by $N_1$ size. Also, $N_E$ denotes the edge set and therefore $(\mathbf{p}, \mathbf{q}) \in N_E$ for an edge between two nodes $\mathbf{p}$ and $\mathbf{q}$.

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

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

$$E(d) = \sum_{(\mathbf{p}, \mathbf{q}) \in N_E} V(d_{\mathbf{p}}, d_{\mathbf{q}}) + \sum_{\mathbf{p} \in P} D_p(d_{\mathbf{p}}),$$

$$D_p(d_{\mathbf{p}}) = \min(C_d|g^l(p_0, p_1 + d_{\mathbf{p}}) - g^r(p_0, p_1)|, K_d),$$

$$V(d_{\mathbf{p}}, d_{\mathbf{q}}) = \min(C_v|d_{\mathbf{p}} - d_{\mathbf{q}}|, K_v).$$

Here, $D(d_{\mathbf{p}})$ is the data cost for the node $\mathbf{p}$ having the state $d_{\mathbf{p}}$. Similarly, $V(d_{\mathbf{p}}, d_{\mathbf{q}})$ is the edge cost for a pair of neighbor nodes $\mathbf{p}$ and $\mathbf{q}$ having states $d_{\mathbf{p}}$ and $d_{\mathbf{q}}$, respectively. $C_d$ and $K_d$ are, respectively, a weighting factor and a lower bound. $C_v$ and $K_v$ are similarly defined.

Finding the state $\hat{d}$ with minimum energy in Eq. (1) amounts to the estimation problem with MAP. As is known well, the approximated MAP solution $\tilde{d}$ can be estimated using the following BP updation [7].

$$m_{pq}^l(d_q) = \min_{d_p} \left( V(d_{\mathbf{p}}, d_{\mathbf{q}}) + D_p(d_{\mathbf{p}}) + \sum_{r \in N(p) \setminus q} m_{rp}^{l-1}(d_p) - \alpha \right),$$

$$\alpha = \frac{1}{S} \sum_{d_p} m_{rp}^{l-1}(d_p).$$

Here, $N(p) \setminus q$ is the neighbors of node $p$ excluding $q$, $\alpha$ is the normalization value, and $S$ is the state size. This equation expresses the following mechanism. The message $m_{pq}^l(d_q)$ at node $p$ is updated at time $l$ and then sent to the neighbor node $q$. After $L$ iterations, the expected $d_{\mathbf{p}}$ at each node can be decided with Eq. (7).

$$d_q = \arg\min_{d_q} \left( D_q(d_q) + \sum_{p \in N(q)} m_{pq}^L(d_q) \right).$$

3 The proposed belief propagation sequence

If we consider a separate layer for each iteration, then we can build a stack of layers. The iteration can be represented as the upward propagation as shown in Fig. 1, which is redrawn in Fig. 2. In this interpretation, we are considering the 2D graph with iteration as the 3D layer graph $(p_0, p_1, l)$ with propagation. Let us define message and data cost set at each node and layer $l$ as

$$M(\mathbf{p}, l) = \{m_{pq}^l(d_q) | d_q \in [0, S - 1], q \in N(\mathbf{p})\},$$

$$D(\mathbf{p}, l) = \{D_p(d_{\mathbf{p}}) | d_{\mathbf{p}} \in [0, S - 1]\}.$$  

From these definitions, we can simplify the message update function in Eq. (5), as follows,

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

where $(N(\mathbf{p}), l - 1)$ and $M((N(\mathbf{p}), l - 1)) = \{M(u, l - 1) | u \in N(\mathbf{p})\}$ represents the neighbor nodes and their hidden variable costs in the buffer, respectively.
As an initialization stage, each node $p$ observes the input to obtain the data cost $D(p, 0)$. Afterwards, in every iteration $l$, each node calculates the new message $M(p, l)$ according to the update function $f(\cdot)$ and after then stores it as $M(p, l - 1)$ in the buffer.

Let $Q(l)$ and $M(Q(l))$ denote, respectively, the set of nodes in $l$th layer and its hidden variable cost set. Then, $M(Q(l))$ can be expressed as an updation from $M(Q(l - 1))$ and $D(Q(l - 1))$ in the buffer:

$$M(p, l) = f(M(N(p), l - 1), D(p, l - 1)), \quad (p, l) \in Q(l), \quad (N(p), l - 1) \in Q(l - 1),$$

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

Let’s consider a new computing order with a new set of nodes in BP case using the iteration filter principle [13]. Note that $Q(l) = \{l, l + 1\}$ forms a linear array of $M$ nodes on $p_1$ axis in the 3rd layer. If we collect all the layers of $Q(l) = \{l, l + 1\}$ in terms of $p_0$, then $Q(l)$ forms a planar array of $LM$ nodes.

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

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

Observing $Q(p_0 - l, l)$ and $Q(p_0)$, we can build an efficient computation order. That is, the cost of $Q(p_0)$ is updated from the buffer of the message $M(Q(p_0 - 1))$, $M(Q(p_0 - 2))$, and data cost $D(Q(p_0 - 1))$ as described in algorithm 1. From now on, we call this sequence as FBP sequence.

**Algorithm (FBP sequence)**

Given $M(Q(p_0 - 1))$, $M(Q(p_0 - 2))$, and $D(Q(p_0 - 1))$ in the buffer, we can calculate $M(Q(p_0))$ and $D(Q(p_0))$. Then, for $p_0$ from 0 to $N + L - 1$, we can obtain the final iterated cost $M(Q(p_0 - L, L))$.

According to the normal iteration scheme, as shown in Fig 2(a) and 2(b), $M(Q(l))$ is updated along the scanning direction $l$ with the buffer $Q(l - 1)$. Now, the new interpretation results in Fig 3(a), 3(b), and 3(c). The buffer location has been changed from $Q(l - 1)$ to $Q(p_0 + a, a \in \{-2, -1\}$. Notice that the buffer is shifted towards the scanning direction $p_0$ and that the final cost $M(Q(p_0 - L, L))$ is obtained at each $p_0$. Hence, the buffer length $N$ on $p_0$ axis can be reduced to the layer number $L$.

As shown in Fig 3(b), the buffer consists of the local layer buffers, which stores all the layers’ costs of the previous groups $M(Q(p_0 - 1))$, $M(Q(p_0 - 2))$, and $D(Q(p_0 - 1))$. Among them, the local buffer holds only one layer’s costs in the current group. Thus, the new cost $M(Q(p_0 - n, n))$ can be saved in the local buffer for the next layer processing, while the previous old cost in the local buffer, $M(Q(p_0 - (n - 1), n - 1))$ is transferred to the layer buffer after being used for calculating the new cost.

In fact, this order of computation is equivalent to shifting the layer buffer to the $p_0$ axis direction.

Figure 3. New orders of message updation: (d) is equivalent to (b), where layers are shifted.

### 4 Hierarchical FBP structure

#### 4.1 Review of hierarchical belief propagation

In the hierarchical structure, $2 \times 2$ scale change can be considered between levels to aid coarse-to-fine iteration. According to this scheme, we must over-sample the cost of hidden variable in the coarse level to obtain the cost for finer level [7]. Let’s consider that $L^k$, $k \in [1, L^k]$, $[p_{0}^{k}, p_{1}^{k}, L^k]$, $m^k$, and $D^k$ denote, respectively, the iteration number, the iteration time index, the node index, message, and data cost in $M^k/2^k$ by $N/2^k$ by $L^k$ hierarchical layer graph of scale level $k \in [0, K - 1]$. This is shown in Fig. 4(a). Here, $K - 1$ means the most coarse level.

As shown in Fig 5, the data cost at $k$ is calculated from the data cost at $k - 1$ by summation over a $2 \times 2$ block. As a special case, the data cost $D_{p|^k_d}$ is equivalent to $D_{p|d}$ which is calculated from the left and right image pixel.

$$D_{p|^k_d}(d) = \frac{1}{2} \sum_{e_{0}=0}^{1} D_{[2p_{0}^{k}+e_{0}, 2p_{1}^{k}+e_{1}]}^{k-1}(d)$$

$$= \sum_{e_{0}=0}^{1} D_{[2p_{0}^{k}+e_{0}, 2p_{1}^{k}+e_{1}]}^{0}(d)$$

(16)
For the finer level, the message oversampling is done by
\[ M^k(p^k_0, p^k_1, 0) = M^{k+1}(p^{k+1}_0/2, p^{k+1}_1/2, L^{k+1}) \]
\[ ((p^{k+1}_0/2, p^{k+1}_1/2) = (p^{k+1}_0, p^{k+1}_1)). \]

### 4.2 New hierarchical FBP sequence

If we assume \( l \) as the shift amount to \( p_0^k \) axis direction, the group \( Q^k(p_0^k - 2L_{k+1}^k) \) at \( k \)th level can be determined as follows.

\[ Q^k(p_0^k - l, l^k) = \{(p_0^k - l, p_1^k, l^k) | p_1^k \in [0, M/2^k - 1]\}, \]
\[ Q^k(p_0^k - 2L_{k+1}^k) = \{Q^k(p_0^k - l, l^k) | l^k \in [1, L_k^k]\}, \]
\[ (l = l^k + 2L_{k+1}^k, l_{k+1}^k = L_{k+1}^k + 2L_{k+2}^k, L_{k+1}^k = L_{k-1}^k), \]

where \( L_{k+1}^k \) is the sum of shift amount of the previous scale levels. Here, we should consider the scale change and the cost oversampling between the level \( k \) and \( k + 1 \).

\[ M^k(p_0^k - 2L_{k+1}^k, p_1^k, 0) = M^{k+1}(p^{k+1}_0, p^{k+1}_1 - L_{k+1}^k, L_{k+1}^k + 1). \]

Hence, the shift value \( L_{k+1}^k \) of the coarser level \( k + 1 \) should be reflected as 2 times bigger than the current level \( k \).

As shown in Fig. 4(a), extending the previous iteration order, \( Q^k(p_0^k - 2L_{k+1}^k) \) at each level \( k \) is processed with the buffers \( Q^k(p_0^k - 2L_{k+1}^k + a) \), \( a \in [-2, -1] \) in depth first search manner for oversampling. At each update, the buffer is shifted towards the scanning direction \( p_0^k \) like the previous iteration sequence.

If we use the notation \( B \) in FBP memory complexity part and consider the nodes of \( L_k^k \) by \( M/2^k \) size in \( Q^k(\cdot) \), the overall memory size can be reduced from \( \sum_{k=0}^{K-1} B(N/2^k)(M/2^k) \) bits to \( \sum_{k=0}^{K-1} B(L_k^k + 1)(M/2^k) \) bits. This can be shown as follows.

\[
\text{Reduction rate} = \frac{\sum_{k=0}^{K-1} B(N/2^k)(M/2^k)}{\sum_{k=0}^{K-1} B(L_k^k + 1)(M/2^k)}.
\]

Since \((M/2^k)\) nodes are handled by \((M/2^k)\) processors in parallel on \( p_0^k \) axis, the total required clocks are reduced from \( \sum_{k=0}^{K-1} 6S(M/2^k)(N/2^k) \) to \( \sum_{k=0}^{K-1} 6SL_k^k(N/2^k) \).

### 5 VLSI Architecture

The experimental chip is a four layer HFBP, where the layer number obtained by try and errors seems to be reasonable. Fig. 6 is the systolic array for HFBP, which consists of identical PEs with nearest neighbor communication. This architecture has total 20 PE-groups for a pair of 160 × 240 images.

A PE group is again shown in Fig. 7. This group actually consists of two parts as shown. The first part is for the data cost module which 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.

Fig. 8 represents the local and layer buffer assignment for each PE in PE group. PE group includes total 8 PEs and the 8/2^k number of PEs are used at level k.

As shown in Fig. 7 and 8, the data costs at each level, that are computed in data cost module, are processed and
saved in the corresponding PEs and buffers. The multiplexer (MUX) selects the corresponding messages and data costs at each level as shown in Fig. 8. In 4 scale levels, 4-to-1 message multiplexer (MUX) is enough. Another MUX is needed to choose one of the two cases: the data cost is read either from the layer buffer or from the data cost module.

As a whole, PEs calculates the messages in parallel by accessing the local buffer or the layer buffer messages in the neighboring PE or PE group.

6 Experimental Results

6.1 Software simulation

First, we verify our VLSI algorithm using the Middleburry data set with software simulation. In the previous sections, we presented a new architecture which is equivalent to HBP in terms with input-output relationship and which is a systolic array with small memory space.

The requirement for both memory resource and computation time is wholly dependent on the layer number \( L \). Therefore, it is reasonable to analyze the performance in terms with iterations as well as various images. Also, we specify the accuracy using the following equation.

\[
\text{error}(\%) = \frac{100}{N} \sum_{(p_0, p_1) \in P_m} \left( |\hat{d}(p_0, p_1) - d_{\text{True}}(p_0, p_1)| > 1 \right)
\]

\[
N = \sum_{(p_0, p_1) \in P_m} 1,
\]

where \( \hat{d} \) is the estimated disparity, \( d_{\text{True}} \) is the true disparity, \( P_m \) is the area except for the occlusion part, and \( N \) is the pixel number in its area. This method states that the rate for the disparity error is larger than 1.

For fair comparisons, the same parameters are used throughout the experiments: \( C_v = 28 \), \( K_v = 57 \), \( C_d = 4 \), and \( K_d = 60 \). Fig. 9 and 10 are the results of the Middlebury test images.

In Fig. 9, 4 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, the HFBP and HBP shows the same lower error results, which must be due to the hierarchical structure.

Fig. 10 shows the relation between the iteration layers of HFBP and memory reduction times, where the same iteration times, \((L, L, L, L)\), is applied for each layer. All the same \( N \) is used, which is the average height of Middlebury test images.

Due to the hierarchical scheme, the iteration converged around in 28 iterations and yielded 0.9% error. The remarkable result, though, is the memory reduction, which is around 32 times. In fact, less memory is possible for higher error rate. Overall, this architecture makes the performance scalable in terms with the space and accuracy.

Table 1 compares our chip with other realtime systems in terms with the error. It is evident that the new method shows almost the same error as realtime BP. Here, realtime BPs are also based on HBP algorithm [7] and known for the lowest error among real-time systems.

6.2 FPGA Implementation

We developed the VHDL code for FPGA as follows using the specs: \( S = 32 \), \( B_m = 7 \), \( B_D = 10 \),
Table 1. Disparity error comparison of several real-time methods (%)

<table>
<thead>
<tr>
<th>Image</th>
<th>Tsukuba</th>
<th>Map</th>
<th>Venus</th>
<th>Sawtooth</th>
</tr>
</thead>
<tbody>
<tr>
<td>DP chip [14]</td>
<td>2.63</td>
<td>0.91</td>
<td>3.44</td>
<td>1.88</td>
</tr>
<tr>
<td>Real-time BP [3]</td>
<td>1.5</td>
<td>NA</td>
<td>0.8</td>
<td>NA</td>
</tr>
<tr>
<td>New chip</td>
<td>1.7</td>
<td>0.5</td>
<td>0.7</td>
<td>0.8</td>
</tr>
</tbody>
</table>

$(L_3, L_2, L_1, L_0) = (8, 8, 8, 10)$, 15 frames/sec at $160 \times 240$ or $160 \times 480$ image.

If these parameters are used for the memory and time complexity equations, as explained in 4.2, the total buffer size becomes 3.3 Mbits, which is smaller than HBP's 62 Mbits by 19 times. Also, for processing one frame image, the 160 PEs needs 0.6 M clocks instead of 157 M clocks. This speed amounts 9.4 M clocks for processing 15 frames in one second. In order to achieve 18.4 MDE/s throughput for a $160 \times 240$ image, only 9.4 MHz system clock is needed.

![Image](image1.png)

(a) Input video  (b) Output t  (c) Output t + 1

Figure 11. FPGA chip output for real images.

The computational performance also is shown in Table 2.

Table 2. Comparisons of computation time

<table>
<thead>
<tr>
<th>Spec</th>
<th>Image</th>
<th>Levels/fps</th>
<th>System/ Clock(Hz)</th>
<th>MDE/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our chip</td>
<td>160x240, 160x480</td>
<td>32/15</td>
<td>FPGA/25M</td>
<td>18.4, 36.9</td>
</tr>
<tr>
<td>Real-time DP [4]</td>
<td>320x240</td>
<td>100/26.7</td>
<td>MMX/2.2G</td>
<td>205</td>
</tr>
</tbody>
</table>

It shows the differences between the new and other systems. In our case, with the slow 25 MHz clock speed, we achieved maximum 36.9 MDE/s. Furthermore, our system have one chip solution which consumes less memory resources inside the chip and can easily be parallelized due to systolic array architecture. This architecture is suitable for VLSI implementation, where regular, less space, and local connections are preferred, among others, for easy design and fabrication.

An importance observation remains in terms with image size. According to the new architecture, the size of memory depends upon the image width. What about large image? If we simply increase the width, we will need chips with more internal memories. An alternative solution is cascading several chips together in proportion to the image size. Actually, the new architecture is scalable in terms with performance and image size. The performance is scalable since the level number and the accuracy are mutually disproportional. The architecture is scalable since the chips can be expanded by simply cascading with adjacent chips.

It has been observed that the chip, incorporating 160 PEs, operates at 25 MHz clock rate. For convenience, more specifications are summarized in Table 3. To store the local and layer buffers, whose size is around 3.3 Mbits, we used 395 internal block RAMs in FPGA, which amount to 7.1 Mbits. Incidentally, assigning each buffer to block RAMs may result in unused leak memory, that is waste, that can be avoided in full ASICs.

The new architecture is implemented in VLSI as shown Fig. 12. Here, Fig. 12(a) is a block diagram and 12(b) is

![Image](image2.png)

Figure 12. The overall hardware system

the photo of the actual board. As can be seen, the two cameras supply a pair of video streams and the two FPGAs performs preprocessing and image matching. The disparity map forms a stream from FPGA to a grabber through camlink cable. From the video RAM on the grabber board, PC reads
the disparity data, converts it to a gray scale image for obser-

7 Conclusions

In this paper, a new architecture for finding an approxi-
mated global solution on the 2D MRF has been presented. The key idea is to rearrange the computation order in BP to obtain parallel and memory efficient structure. As the results show, our system spends 19 times less memory than the ordinary BP. As a compromise between accuracy and memory space, the memory space can be negotiated with the iteration number. This architecture is also scalable in terms with the image size; the regular structure can be easily expanded by cascading identical modules.

When applied to the binocular stereo vision, this architecture shows the ability to process the stereo matching in real-time. The experimental results confirm that this array architecture is especially useful for high resolution images with fast frame rates, where small iterations are guaranteed by the hierarchical iteration scheme.

In the future, we plan to realize this architecture with a compact ASIC chip. Beyond the programmable chips, we can expect higher resolution real-time chips with huge PE numbers and fast clock speed. Making the complex stereo matching system with a compact chip may lead to many real-time vision applications.

References


[15] D. Scharstein and R. Szeliski. A taxonomy and eval-
uation of dense two-frame stereo correspondence al-

[16] J. Woodfill and B. Von Herzen. Real-time stereo vi-