Hardware-Efficient Belief Propagation

Chia-Kai Liang Chao-Chung Cheng Yen-Chieh Lai Liang-Gee Chen Homer H. Chen

National Taiwan University, Taipei, Taiwan R.O.C

Abstract

Belief propagation (BP) is an effective algorithm for solving energy minimization problems in computer vision. However, it requires enormous memory, bandwidth, and computation because messages are iteratively passed between nodes in the Markov random field (MRF). In this paper, we propose two methods to address this problem. The first method is a message passing scheme called tile-based belief propagation. The key idea of this method is that a message can be well approximated from other faraway ones. We split the MRF into many tiles and perform BP within each one. To preserve the global optimality, we store the outgoing boundary messages of a tile and use them when performing BP in the neighboring tiles. The tile-based BP only requires 1-5% memory and 0.2-1% bandwidth of the ordinary BP. The second method is an O(L) message construction algorithm for the robust functions commonly used for describing the smoothness terms in the energy function. We find that many variables in constructing a message are repetitive; thus these variables can be calculated once and reused many times. The proposed algorithms are suitable for parallel implementations. We design a low-power VLSI circuit for disparity estimation that can construct 440M messages per second and generate high quality disparity maps in near real-time. We also implement the proposed algorithms on a GPU, which can calculate messages 4 times faster than the sequential O (L) method.

1. Introduction

Many low-level vision problems can be formulated in the Markov random field (MRF) framework as an energy minimization problem [14]. Suppose that the MRF is represented by a graph $G = (V, D)$, where V is the set of all nodes and D is the set of all edges. The problem is to assign a label l_p to each node p such that the following energy function is minimized:

$$\sum_{p\in V} E_p(l_p) + \sum_{(p,q)\in D} E_s(l_p, l_q), \tag{1}$$

where E_p is an unary data term and E_s is a pairwise smoothness term. Several efficient global optimization methods such as graph cuts [2] and (loopy) belief propagation (BP) [7] have been proposed. They can efficiently find better solutions than local or greedy methods.

However, these global optimization methods have to store all variables in the MRF and require iterative operations, resulting in enormous memory, bandwidth, and computational costs. For example, when we perform BP, we have to store all data terms and messages, totally $O(NL)$ elements, in the main memory, where N is the number of nodes and L is the size of the label set. In each iteration of BP we have to access $O(NL)$ data. For the disparity estimation of a 720p video (1280×720, 30 fps), assuming 100 disparity values and 20 iterations per frame, we have to store 429 MB data and access 1 terabyte data per second (assuming all data are 8-bit). This memory cost is too expensive for most embedded systems, and the bandwidth requirement is unreachable for most existing hardware platforms including desktops.

Therefore, if we want to perform BP on low-power consumer electronics, such as digital cameras, digital TV's, and mobile phones, we must reduce its bandwidth, memory, and computational costs. The reduction of bandwidth cost is also important for high-end platforms because it is the major performance bound [3].

In this paper, we propose two methods to address these issues. The first one is a new message passing scheme called tile-based belief propagation. The key idea of this method is that a message can be well approximated from other faraway messages. We therefore split the MRF into many tiles and perform BP within each tile. To preserve the global optimality, we store the outgoing boundary messages of a tile and use them when performing BP in the neighboring tiles. The tile-based BP only requires 1-5% memory and 0.2-1% bandwidth of the ordinary BP.

The second method is $O(L)$ an message construction algorithm for the robust functions commonly used to describe the smoothness terms in the energy function. We show that many variables in constructing a message are repetitive; therefore they can be calculated once and reused many times.

The proposed methods are suitable for efficient hardware implementation. As an illustration, we use them to design a low-power VLSI circuit for disparity estimation. It can generate VGA-sized disparity maps in near real-time. To our knowledge, this is the first dedicated disparity estimation hardware based on a global optimization method that generates results better than those by the local methods.

We also present a simple GPU implementation of the proposed algorithms to demonstrate that these algorithms applicable for other hardware architectures. While we only focus on the stereoscopic disparity estimation application, it should be noted that the concepts and algorithms presented here can be applied to other low-level vision problems as well. The results of our algorithms on the Middlebury benchmarks are provided in the supplemental material[1].

2. Related Work

In this section we review the most relevant software and hardware methods for accelerating the global optimization algorithms and for building real-time disparity estimation systems.

Felzenszwalb and Huttenlocher were the first to address the computational issues of BP for computer vision applications [6]. They proposed a hierarchical message passing scheme to speed up the convergence, a so-called min-convolution method based on the distance transform theory to reduce the complexity of message construction from $O(L^2)$ to $O(L)$, and a bipartite graph to reduce the memory size by half. Tappen and Freeman showed that a sequential asynchronous message update scheme (dubbed BP-M in [14]) is more efficient than a synchronous one [15].

Many existing BP software programs combine these approaches together. In each level of the hierarchical BP, BP-M is used and each message is computed using min-convolution. However, the min-convolution method is a strictly sequential operation that cannot be performed in parallel. The BP-M does not change the memory and bandwidth costs.

Yu et al. showed that the messages are smooth and compressible [22]. Their method reduces the memory and bandwidth costs of BP to 12.5%[2], but the memory size for the data cost, which is unchanged, is still too large for embedded systems. Also the sequential compression and decompression operations would slow the system.

Real-time disparity estimation is a critical component for robot navigation, gaze correction, free-viewpoint image synthesis, etc. Since the global optimization methods are relatively expensive, most approaches use either local cost aggregation [5, 8, 21] or scanline-based optimization methods [4, 9]. Generally, their performances are inferior to the global optimization methods. Wang et al. used the results of the local stereo estimation to reduce the disparity range for optimization [19]. However, the irregular label set and the additional pre-processing steps are both impeditive to hardware efficiency.

Recently a few implementations of the global optimization on GPU have been developed [18, 20]. While they usually map the original algorithms to the GPU, we show that by modifying the algorithm, a better parallelism can be achieved with lower memory and bandwidth costs. Finally, a straightforward method is to split the image into small blocks and perform global optimization in each block independently [16]. However, this method does not guarantee the global optimality and usually generates poor results.

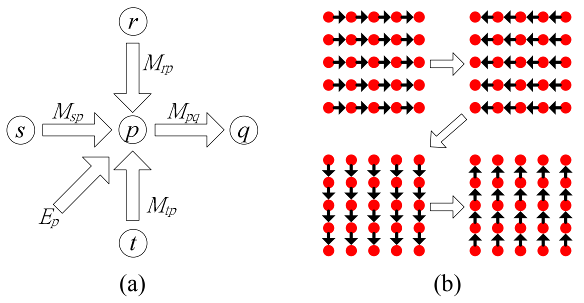


Figure 1. (a) The message construction in belief propagation. (b) The message passing scheme of BP-M for one iteration.

3. Cost Analysis of Belief Propagation

Before describing the proposed algorithms, we review the operations in the max-product BP [7]. In BP, we iteratively send the messages between neighboring nodes to propagate the information. Consider a subset of MRF shown in Figure 1 (a), a message M_{pq} is sent from node p to its neighboring node q, where M_{pq} is an L-D vector and each entity $M_{pq}(l_q)$ is defined as

$$M_{pq}(l_q) = \min_l \left\{ E_s(l, l_q) + H_{pq}(l) \right\}, \tag{2}$$

where $H_{pq}(l)$ is the sum of the data term and the incoming messages from other nodes,

$$H_{pq}(l) = E_p(l) + M_{rp}(l) + M_{sp}(l) + M_{tp}(l). \tag{3}$$

Given enough iterations, the optimal label l^* for each node is chosen independently according to

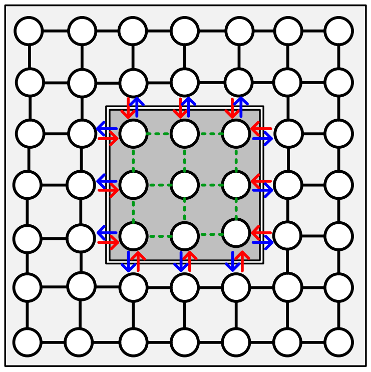
$$l_p^* = \arg\min_l (E_p(l) + M_{qp}(l) + M_{rp}(l) + M_{sp}(l) + M_{tp}(l)). \tag{4}$$

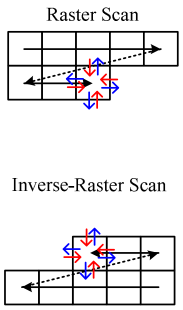
One can calculate Equation (2) by first building L hypotheses and then picking the smallest one from them. The arithmetic complexity of this method is $O(L^2)$, so the direct implementation is very expensive.

For asynchronous message update schemes, the order of the messages has to be specified. The order used in the popular BP-M method is shown in Figure 1 (b). It contains four phases in each iteration, and in each phase it updates all the messages in the same direction. Each constructed message is immediately used in building the next message.

[1] A reference implementation compatible to the Middlebury MRF library is available at http://mpac.ee.ntu.edu.tw/~chiakai/hebp/.

[2] In [22] the data are assumed to be stored at floating point precision. If the data precision is 8-bit, only 30-50% compression rate can be achieved.


(a)



(b)

Figure 2. (a) The MRF is split into two sets. One set contains the nodes in the tile (the dark grey region) and other one contains all other nodes (the light grey region). (b) We interleave the raster and inverse-raster scans to remove the biasing problem.

We can see that according to Equations (2) and (3), in each phase BP-M loads $3NL$ messages and NL data terms, and then outputs NL messages. Therefore, in each iteration it needs to access $4(3+1+1)NL = 20NL$ data from the main memory. Because the messages orthogonal to the current passing direction and data terms are only used to build one message, it is difficult to cache them in the local memory for reuse. All data terms and messages, totally $5NL$ variables, must reside in the main memory till the algorithm stops.

4. Tile-Based Belief Propagation

In this section, we present a new message passing scheme dubbed tile-based belief propagation. It has performance similar to the BP-M does but requires much less memory and bandwidth.

4.1. Basic Concept

We can easily see that in Equation (2), M_{pq} is uniquely determined from E_p and the three incoming messages. Therefore, given four incoming messages and the data term, all outgoing messages can be perfectly reconstructed. Therefore, one can remove half of the messages without affecting the message passing. This is the basic idea used in the bipartite graph [6].

We exploit this idea one step further. Without loss of generality, we spit the nodes in MRF into two sets, as shown in Figure 2 (a). One set S_1 contains the nodes in a 3-by-3 tile (the grey square in Figure 2 (a)) and the other set S_2 contains all other nodes.

When we perform BP in S_2, without knowing all data terms and messages in S_1 (green dotted edges in Figure 2), we only need the messages coming from S_1 to drive the propagation (blue arrows in Figure 2 (a)). All the messages in the tile are irrelevant to the message passing operations outside the tile because they are never used in evaluating

```
1   Arrange the tiles {$S_1$, $S_2$, …, $S_N$} in the raster scan order.
2   For $t$ = 1,…,$T_o$
  2.1 For $i$ = 1,…,$N$
        2.1.1 Load boundary messages from the neighboring tiles.
        2.1.2 Calculate the data terms in $S_i$.
        2.1.3 Perform messages passing in $S_i$ for $T_i$ iterations.
        2.1.4 Save the boundary messages.
  2.2 For $i$ = $N$,…,1
        2.2.1 Load boundary messages from the neighboring tiles.
        2.2.2 Calculate the data terms in $S_i$.
        2.2.3 Perform messages passing in $S_i$ for $T_i$ iterations.
        2.2.4 If ($t$ = $T_o$) Calculate the optimal labels for $S_i$.
        2.2.5 Else Save the boundary messages.
```

Figure 3. The pseudo code of the tile-based belief propagation.

Equation (2) when p and q are in S_2.

Similarly, when we perform belief propagation in S_1, beside the messages coming from S_2 (red inward arrows in Figure 2 (a)), all the messages and data terms outside the tile are unimportant at all. As long as the coming messages carry the reliable information about the outside world, we can use BP to calculate the optimal solution for this tile. Also, as long as the messages toward the outside world (blue outward arrows in Figure 2 (a)) carry the correct information about the tile, we can perform BP outside the tile to obtain the optimal solution.

In other words, we can treat either S_1 or S_2 as a *filter box* of the messages. The messages input to the box are non-linearly filtered by performing BP to generate the output messages. When the filter boxes behave well, the operation would converge. Unlike the bipartite graph method that perfectly reconstructs all messages [6], here we assume that the messages in the subset of the MRF can be approximated from the messages entering the subset. Given enough computations, the approximation quality is good enough to drive the propagation to converge. This assumption is experimentally verified in Section 4.3.

4.2. Algorithm

We follow the concept and propose the tile-based belief propagation. The pseudo code is given in Figure 3. The MRF is first spilt into many non-overlapping $B \times B$ tiles {S_1, S_2,…,S_N}; these tiles are processed in a raster scan order. When the data terms are not pre-computed, we first calculate the data terms for the nodes in the tile S_i. Then we load the messages coming from other tiles, which are called *boundary messages*. We use the data terms, the smoothness terms, and the boundary messages to construct the messages in S_i. Any ordinary BP algorithm can be applied here, but we find that BP-M gives the best results.

After the messages passing is performed for a number of iterations, the messages are stable now. We then generate the outgoing boundary messages and store them to the memory. All messages in the tiles are thrown away.

The raster scan order would introduce a bias to the

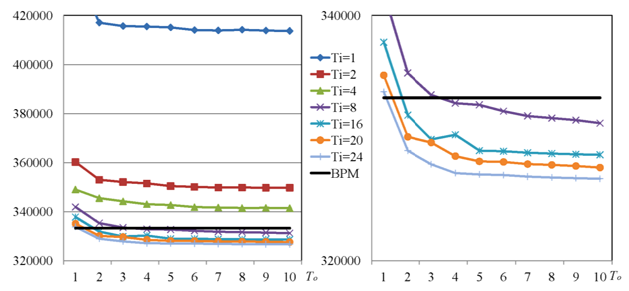


Figure 4. The T_o-energy curves of Tsukuba for B=16. The black line is the final energy value of BP-M after 80 iterations.

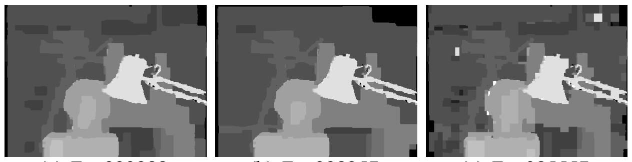


Figure 5. The disparity maps and energy values by (a) BP-M, (b) tile-based BP with B=16, and (c) block-based BP in [16]. The E's are the corresponding final energy values.

message construction. It is because for each tile, the boundary messages from the top and left tiles are always more recent than those from the right and bottom tiles. Note that this bias problem is common in asynchronous message update schemes. We resolve this problem by processing the tiles in an inverse-raster scan after the raster scan is completed (Figure 2 (b) and step 2.2 in Figure 3). At the final iteration ($t = T_o$), we calculate the labels for each tile in the inverse raster scan pass (step 2.2.4 in Figure 3).

The proposed tile-based BP can greatly save the memory, and bandwidth. However, to justify the savings, we have to first evaluate its performance. In the following we experimentally demonstrate that the convergence speed and quality of the tile-based BP are similar to the traditional BP methods for disparity estimation. More results on the Middlebury benchmarks, including image segmentation, denoising, etc, are provided in the supplemental material.

4.3. Performance Analysis

The performance of the tile-based BP depends on the number of outer iterations (T_o), the tile size (B), and the number of inner iterations in the message passing steps (T_i in 2.1.3 and 2.2.3 in Figure 3). We first let T_o be infinite and find the optimal T_i for each B. Because BP is less sensitive to the choice of the smoothness terms than other global optimization methods [14], we use simple truncated linear or truncated quadratic functions as the smoothness terms in our experiments.

One particular case, the stereo estimation of *Tsukuba* is shown in Figure 4, where B is 16. We can see that the algorithm always converges after a few outer iterations regardless of the number of the inner iterations. When T_i=1, the final energy value is much higher than that of BP-M. It is because the messages in a tile are not well approximated. However, as T_i increases, the approximations of the messages become more accurate and the final energy decreases. Our algorithm outperforms BP-M after $T_i > 8$. It is because by resetting the messages in tiles, we can remove the messages fixed at wrong statuses. The resulting disparity maps are shown in Figure 5, where the results from BP-M and the tile-based BP are similar. On the contrary, the method in [16] splits the image into independent blocks and generates a poor result.

	Tsukuba	*Teddy*	*Cones*	*Venus*	Avg.
BP-M 80 iterations	333288	2227224	2466383	802006	
Tile-based BP B=16, T_i=16	328257	2233962	2477568	814453	
	98.5%	100.3%	100.5%	101.6%	100.2%
Tile-based BP B=32, T_i=28	330647	2230923	2478763	815300	
	99.2%	100.2%	100.5%	101.7%	100.4%
Tile-based BP B=64, T_i=56	341915	2238193	2483135	817561	
	102.6%	100.5%	100.7%	101.9%	101.4%
Hierarchical BP	326735	2245899	2481939	823181	
	98.0%	100.8%	100.6%	102.6%	100.5%
Hierarchical BP + Tile-based BP	319467	2217444	2459244	808140	
	95.9%	99.6%	99.7%	100.8%	99.0%

Table 1. The final energy values obtained by BP-M, tile-based BP, hierarchical BP, and hierarchical BP with tile-based BP. The datasets are provided in [12], [13].

We perform similar experiments to B=32, 64 and find the required T_i are 28 and 56, respectively. This means the number of required inner iterations grows linearly with B. We also observe that the algorithm converges very fast for all B's. Setting T_o = 3 is sufficient for most applications.

The final energy values with different settings are listed in Table 1. We can see that the changes due to the use of the tile-based BP are usually smaller than 1%. We also implement the hierarchical belief propagation with 5 scales. If we apply the tile-based BP in each level, the changes in the energy are very small. In fact, the choices of the dampening factor and the normalization methods affect the convergence speed and quality more seriously than the use of tile-based BP.

In summary, the performance of tile-based BP is similar to that of BP-M. The complexity of the tile-based BP depends on T_i and T_o, and its execution time is almost identical to the BP-M with $2T_iT_o$ iterations in the software implementation. However, in hardware implementation, the bandwidth consumption, which is the major performance bound, only depends on T_o. The T_i iterations in each tile can be performed efficiently without accessing the main memory, as we discuss below.

4.4. Cost Analysis

The tile-based BP only stores the boundary messages in the main memory. All the variables required for processing a tile are stored in the local cache or local memory. Therefore, the requirement of the main memory C_M is

B	L	T_o	T_{BPM}	Relative Memory Cost	Relative Bandwidth Cost
16	16	3	20	5%	0.47%
32	16	3	20	2.5%	0.28%
64	16	3	20	1.25%	0.19%
16	32	3	20	5%	0.42%
16	64	3	20	5%	0.40%
16	16	2	20	5%	0.31%
16	16	4	20	5%	0.63%
16	16	3	10	5%	0.94%
16	16	3	40	5%	0.23%
16	16	3	80	5%	0.12%

Table 2. The relative memory cost and the relative bandwidth cost of the tile-based BP to the BP-M with different parameter settings.

$$\left(N/B^2\right)4BL = \left(4NL/B\right). \tag{5}$$

The ratio of C_M to the memory cost of the traditional BP is

$$C_M/5NL = 0.8/B. \tag{6}$$

The ratios for different B's are listed in Table 2. We can see the memory cost of the tile-based BP is only most 5% of that of BP-M when B=16, and 1.25% when B=64. This cost is even much smaller than several local stereo estimation algorithms which require all data terms (NL elements) for adaptive cost aggregation.

The message passing step needs to store $5B^2L$ variables, including the data terms and the messages in a tile, in the local memory. For small B's, the storage of this size is affordable or readily available in most GPUs, DSPs, and other VLSI architectures. On the contrary, the data reuse is difficult in BP-M so local caches are useless.

The reduction of the memory cost is also beneficial to the software implementation. One example is given in Figure 6. While the peak memory of the original BP-M is 603 MB, that of the tile-based BP is only 25 MB. The results of the two methods are similar, and so are the processing times, even though the tile-based BP has to calculate the data terms twice.

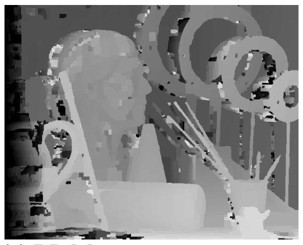
Regarding the bandwidth cost, the tile-based BP loads $4B$ L-D boundary messages and $2B^2$ pixels to calculate the data terms, (see Section 6.1), and stores $4B$ L-D boundary messages for each tile. The total bandwidth cost C_B is

$$\left(N/B^2\right)\left(4BL + 2B^2 + 4BL\right)T_o = \left(8NLB^{-1} + 2N\right)T_o, \tag{7}$$

and the ratio to the bandwidth cost of BP-M is

$$\frac{C_B}{20NLT_{BPM}} = \frac{\left(0.4B^{-1} + 0.1L^{-1}\right)T_o}{T_{BPM}}, \tag{8}$$

where T_{BPM} is the iteration number in BP-M. The ratios for different parameter settings are listed in Table 2. We can see the tile-based BP saves more bandwidth as B or L increases and as T_o decreases. Typically, the bandwidth cost of the tile-based BP is less than 1% of that of the BP-M.


(a) BP-M
T_{BPM} = 64, E = 5580968
Peak Memory: 603 MB

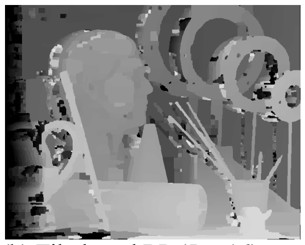

(b) Tile-based BP (B = 16)
T_i = 16, T_o = 2, E = 5544806
Peak Memory: 27 MB.

Figure 6. The disparity maps of the dataset *Art* using (a) BP-M (b) tile-based BP (N = 695×555, L = 113). The execution times are both 5 minutes. All variables are stored as 16-bit integers. This dataset is provided in [10].

5. Parallel Message Construction

It has been shown that when the smoothness term in the MRF is linear or quadratic, the messages can be quickly constructed using the min-convolution method [6]. However, the min-convolution is a strictly sequential process which cannot be performed in parallel. That is, the operation cycle is always $O(L)$ even when there are additional computational resources.

5.1. Algorithm

We find that the *robust functions* are used as the smoothness term (and data term) to model the outliers in many MRF-based computer vision problems [1]. For example, in almost all top-performance disparity estimation algorithms (all top 10 methods at the time of submission) listed in the Middlebury website, the robust functions are chosen to be the smoothness terms.

An important property of the robust functions is that while they increase monotonically in the beginning, they eventually stop growing after reaching a threshold. Therefore, a smoothness term can be represented as

$$E_s(l_p, l_q) = \begin{cases} c \cdot f(\Delta l) & ,\Delta l < T \\ K & ,\text{otherwise} \end{cases}, \tag{9}$$

where f is a monotonic function, c is the weighting factor, Δl represents $|l_p - l_q|$, T is the stop threshold, and K is the stop value. Note that K is no smaller than $cf(T)$. The constant c can be spatially variant without affecting the analysis.

In most applications, T is much smaller than L. Therefore, a $H(l)$ in Equation (2) uses $f(\Delta l)$ to generate $2T$+1 hypotheses (the subscript is omitted for simplicity). All other hypotheses generated by $H(l)$ are $H(l)$+K. From another point of view, in calculating the entity $M_{pq}(l_p)$, only at most $2T$+1 hypotheses are generated using f.

Therefore, when the smoothness term is a robust function, Equation (2) can be re-formulated as

$$M_{pq}(l_q) = \min(R_{q,1}, R_{q,2}), \tag{10}$$

where R_1 and R_2 are two different sets of hypotheses:

$$R_{q,1} = \{H(l_i) + cf(|l_i - l_q|) : |l_i - l_q| \le T\}, \tag{11}$$

$$R_{q,2} = \{H(l_i) + K : |l_i - l_q| > T\}. \tag{12}$$

We show that $\min(R_{q,2})$ is independent of q. Therefore, it can be calculated once and applied to all q's. Let R_2 denote the set of all $H(l) + K$, then we have

$$\min(R_2) = \min(H(l)) + K \le \min(R_{q,2}). \tag{13}$$

Due to the definition of the robust function in Equation (9), when $\mathrm{argmin}(H(l)) = l_i$ and $|l_i - l_q| < T$, $H(l_i) + cf|l_i - l_q| \le H(l_i)+K$. As a result, Equation (10) is equivalent to

$$\begin{aligned} M_{pq}(l_q) &= \min(R_{q,1}, R_2) \\ &= \min(R_{q,1}, \min(H(l)) + K). \end{aligned} \tag{14}$$

Therefore, one can find the minimum of R_2 and use it to Equation (10).

In summary, in the proposed algorithm, we first find the minimum of H's after they are calculated. Then we find the minimum of $R_{q,1}$ for each l_q, and compare it with the minimum of H's plus K to obtain the final message.

5.2. Cost Analysis and Parallel Implementation

The proposed method requires $O(L)$ operations to calculate $\min(R_2) + K$, $O(TL)$ operations to find $\min(R_{q,1})$ for all l_q's, and $O(L)$ operations to calculate the final message values. Therefore, the overall complexity is $O(L + L + TL) = O(TL)$. Because $T << L$ in many applications including disparity estimation, the complexity becomes $O(L)$.

While the complexity is similar to the min-convolution method, our method can be performed in parallel. As a result, it is more suitable for hardware implementation than the min-convolution method. The parallel implementation for $L = 10$ and $T = 2$ is illustrated in Figure 7. Each colored square is a hypothesis. The squares at the bottom row are $H(l)$'s computed using Equation (3). The squares in each other row are $H(l) + cf(\Delta l)$ with a specific Δl. We find the minimum of each column and compare it with the minimum of $H(l)$ to obtain the final message.

All the operations in Figure 7 can be performed in parallel. When each column is processed independently, $\min(H(l))$ can be computed simultaneously. The exact degree of parallelism and the total operation cycles depend on the hardware architecture. At the maximal parallelism, all additions are performed simultaneously and each MIN operation in Figure 7 is performed by a tree-structure architecture. This method requires $(3 + 2T)L$ adders and $(2TL + L - 1)$ comparators. Compared with the size of the local memory, they are infinitesimally small.

Finally, note that the hypotheses and other temporal variables do not have to be explicitly stored in registers. This means all calculations can be wired together and the message can be calculated in just one cycle. This approach is used in our VLSI implementation presented in Section 6.1.

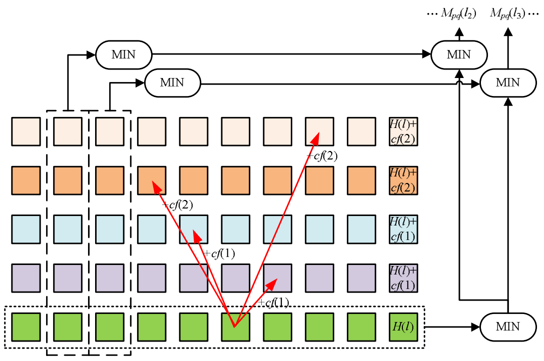


Figure 7. The parallel implementation of the proposed fast message construction algorithm. In this case, $L = 10$, $T = 2$, and f can be any monotonic function.

6. Implementations

We have implemented the proposed algorithms on two different hardware systems, a dedicated VLSI chip and a GPU program.

6.1. Very Large Scale Integration (VLSI) Circuit

We use the UMC 90 nm technology to develop our chip. In our design, the tile size is 16 and the maximal number of disparity values is 64, which is larger than most existing GPU, VLSI, and FPGA implementations [5]. The implementation is exact to the reference software and hence given the identical parameters, the results are the same to those in Section 4.3.

The chip performance and spec are listed in Table 3, and the floorplan of the synthesized circuit is shown in Figure 8. The chip only consumes 445.4 mW when working at 185 MHz. It consists of 2.5M gates (about 10M transistors) and the die size is 25 mm^2. Because our design consumes very little power, it is suitable for embed systems.

The circuit generates messages at a constant rate, 440M messages per second. The speed of the disparity estimation and the quality depend on the parameter settings. If we set $(T_i, T_o) = (5, 1)$, we can process VGA-sized image at 27 fps. Better results can be obtained by increasing the iteration numbers. Compared with the GPU implementation in [20], our chip is 6-40 times faster in terms of the number of disparity estimations per second. Note that this performance is only possible when the tile-based BP is applied. If we use the traditional BP-M scheme, the chip must work no less than 1 GHz to meet the bandwidth requirement.

In the following we highlight two important high-level optimizations that can also be applied to other platforms. The fine-grained optimizations in the design shall be described elsewhere. First, we use the tile-wise pipeline similar to that in the video codec chip to improve the chip

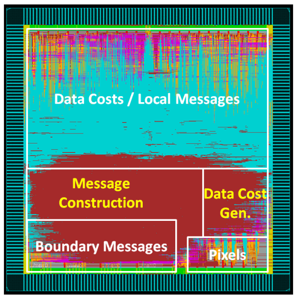


Figure 8. The floorplan of the developed VLSI circuit.

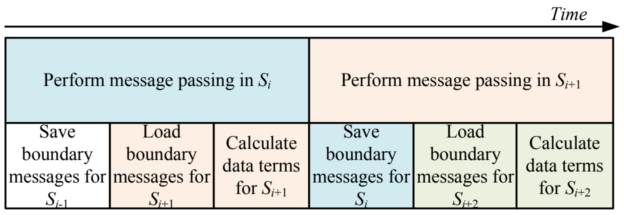

Time
Perform message passing in S_i
Perform message passing in S_{i+1}
Save boundary messages for S_{i-1}
Load boundary messages for S_{i+1}
Calculate data terms for S_{i+1}
Save boundary messages for S_i
Load boundary messages for S_{i+2}
Calculate data terms for S_{i+2}

Figure 9. The two-stage pipeline used in our VLSI circuit.

utilization and reduce latency [3]. As shown in Figure 9, in our two-stage pipeline, we perform message passing of a tile in one stage, and the saving of the boundary messages in another one. In the second stage, we also load the boundary messages, and calculate the data terms for the next tile. In this way all processors are maximally utilized. While the data terms are pre-computed in software, calculating them on-the-fly is more efficient in hardware at the cost of B^2L additional buffers. Also, this method cannot be applied to the scanline-based methods [4, 9] since it would require WL buffers (W is the image width).

Second, we reuse the pixels of different tiles to reduce the bandwidth cost. In calculating the data terms for disparity estimation for one tile, the direct method requires B^2 pixels of the left image, and $B(B+L)$ pixels of the right image. However, for the successive tile, BL pixels of the right image are already loaded at the current tile, and thus we only have to load B^2 new pixels. In this way, BL bandwidth is saved per tile. This method was used in block-based motion estimation in video codec [17].

6.2. Graphics Processing Unit (GPU) Program

To show that the proposed methods can be applied to other platforms, we use the CUDA technique of the Nvidia cooperation [11] to implement a disparity estimation system on GPU. CUDA provides a C language environment to write programs for execution on GPU. In the CUDA programming model, the threads that share one local buffer are arranged in a single *block*. The number of blocks and threads can be much larger than the available resources of the GPU; the GPU automatically switches the threads to hide the latency of memory access and maximize the performance.

Since the message passing operation is sequential within

Technology	UMC 90 nm		
Operating voltage	1.0 V		
Power consumption	445.4mW (at 185MHz)		
Chip area	5 x 5 (mm^2)		
Gate count	Processing core	633K	
	Memory	1.88M	
	Total	2.5M	
Disparity range (L)	64		
Tile size (B)	16		
Processing Speed (at 185 MHz)			
Image resolution	(T_i, T_o)	Fps	DE/s
640×480	(5, 1)	27	530.8M
640×480	(8, 2)	10	199.5M
320×240	(16, 3)	14	66.3M
Number of constructed messages per second			440M

Table 3. The summary of the developed disparity estimation chip. DE/s denotes the number disparity estimation per second.

Tsukuba 384×288, L=16, T=2	Min-conv. CPU	Min-conv. GPU	Proposed GPU
Calculating data terms	17.72	0.03	
Memory transfer	0.00	27.02	
One BP-M iteration	173.94	102.31	19.46
Generate the depth map	10.52	0.03	
Total (5 iterations)	897.94	538.63	124.38
Speedup factor	*1.00*	*1.67*	***7.21***

Table 4. The execution time (ms) for *Tsukuba*.

Cones 450×375, L=60, T=7	Min-conv. CPU	Min-conv. GPU	Proposed GPU
Calculating data terms	17.72	0.03	
Memory transfer	0.00	29.70	
One BP-M iteration	771.43	528.74	112.94
Generate the depth map	44.32	0.03	
Total (5 iterations)	3919.19	2673.46	594.46
Speedup factor	*1.00*	*1.47*	***6.59***

Table 5. The execution time (ms) for *Cones*.

each row/column, we arrange the computations for each row/column in a separate block. For the strictly sequential min-convolution method, each block contains only one thread. On the contrary, our method can be efficiently performed by activating L threads in each block. Each thread first calculates H's using Equation (3) and then the hypotheses in $R_{q,1}$. Since T is small, each thread sequentially calculates $\min(R_{q,1})$ and then all threads jointly calculate $\min(R_2)$. Finally, each thread generates the message entity $M_{pq}(l_q)$ using Equation (14). The time complexity of this approach is $O(T+\log L)$, much smaller than $O(L)$ of the min-convolution method.

Tables 4 and 5 list the execution times of the disparity estimation for Tsukuba and Cones on a 3.0 GHz CPU with 2 GB main memory and an Nvidia GeForce 8800GTS GPU (512 MB video memory). We can see that when the available bandwidth and computational resource are fixed, the proposed method is about four times faster than the min-convolution method because the proposed method activates more threads.

Our proof-of-concept implementation is not optimized yet and hence it is slightly slower than a highly optimized one [20]. A better performance can be achieved by putting the data to the cache-enabled texture memory and using a better memory packing method (in CUDA, only the data stored in a specific texture format are cached). However, the performance gain is not affect by these optimizations.

Performing the tile-based BP together with the parallel message construction in CUDA requires additional effort. Due to the limited local memory, the maximal tile size B is only 12 when L=16. The computation power is not fully utilized with this setting and the bandwidth reduction is overwhelmed by the frequent task switches. Therefore, our current implementation is only slightly faster than BP-M. Nevertheless, the memory reduction of the tile-based BP makes it possible to process large images with a large disparity range such as the case in Figure 6.

7. Conclusion and Future Work

In this paper we have presented two methods to efficiently perform belief propagation on hardware. By properly modifying the message passing scheme, we have shown that by propagating the boundary messages, the BP can be performed within a tile without losing the global optimality. This approach reduces the memory and bandwidth costs by orders of magnitude.

We have proposed a new parallel message construction method for the robust functions. It can efficiently calculate a message in one cycle on hardware and exploit the computational resource of the modern GPU.

The proposed methods allow us to design a VLSI circuit for disparity estimation that consumes less than 1 watt and generates high quality disparity maps in near real-time. We have also developed a simple GPU program which is 4 times faster than the min-convolution method.

Because the modified BP generates globally optimal results and uses simple operations and regular data access, it is more suitable for hardware implementation than many local cost aggregation methods that require irregular and branching operations and many scanline-based methods that require sequential operations and a large line buffer.

We employ simple data and smoothness terms in our stereo estimation system to enable the real-time performance. Nevertheless, because many start-of-the-art off-line methods use BP to minimize the defined complex energy functions, our algorithms can be used to reduce their memory, bandwidth, and computational costs as well.

We have experimentally shown that the tile-based BP quickly converges to a result close to that of the original BP. However, a formal proof is desirable. We find that by resetting the messages, a result with a lower energy can be obtained. Although our initial goal is not improving the performance of BP, this property is worth more study.

Acknowledgement

This project is supported in part by NSC in grant 96-2628-E-002-005-MY2 and Himax Technologies Inc. in grant 96SB20.

References

[1] M. Black, G. Sapiro, D. H. Marimont, and D. Heeger. Robust anisotropic diffusion. *IEEE Trans. Image Processing*, 1998.

[2] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. PAMI*, 2001.

[3] T.-C. Chen, S.-Y. Chien, Y.-W. Huang, C.-H. Tsai, C.-Y. Chen, T.-W. Chen, and L.-G. Chen. Analysis and architecture design of an HDTV720p 30 frames/s H.264/AVC encoder. *IEEE Trans. CSVT*, 16(6):673-688, 2006.

[4] A. Criminisi, A. Blake, and C. Rother. Efficient dense stereo with occlusions for new view-synthesis by four-state dynamic programming. *IJCV*, 71(1): 89-110, 2007.

[5] J. Díaz, E. Ros, R. Carrillo, and A. Prieto. Real-time system for high-image resolution disparity estimation. *IEEE Trans. Image Processing*, 16(1): 280-285, 2007.

[6] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient belief propagation for early vision. *IJCV*, 70(1): 41-54, 2006.

[7] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael. Learning low-level vision. *IJCV*, 40(1): 25-47, 2000.

[8] M. Gong, R. Yang, L. Wang, and M. Gong. A performance study on different cost aggregation approaches used in real-time stereo matching. *IJCV*, 75(2): 283-296, 2007.

[9] M. Gong and Y.-H. Yang. Near real-time reliable stereo matching using programming graphics hardware. In *Proc. CVPR*, volume 1, pages 924-931, 2005.

[10] H. Hirschmüller and D. Scharstein. Evaluation of cost functions for stereo matching. In *Proc. CVPR*, 2007.

[11] Nvidia Corporation. CUDA: compute unified device architecture programming guide. Technical report, 2008.

[12] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 2002.

[13] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Proc. CVPR,* vol. 1, pages 195-202, 2003.

[14] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. F. Tappen, and C. Rother. A comparative study of energy minimization methods for Markov random fields. *IEEE Trans. PAMI*, 30(6): 1068-1080, 2008.

[15] M. F. Tappen and W. T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters. In *Proc. ICCV*, volume 2, pages 900-906, 2003.

[16] Y.-C. Tseng, N. Chang, and T.-S. Chang. Low memory cost block-based belief propagation for stereo correspondence. In *Proc. ICME*, page 1415-1418, 2007.

[17] J.-C. Tuan, T.-S. Chang, and C.-W. Jen. On the data reuse and memory bandwidth analysis for full-search block- matching VLSI architecture. *IEEE Trans CSVT,* 2(1):61-72, 2002.

[18] V. Vineet and P. J. Narayanan. CUDA cuts: fast graph cuts on the GPU. In *Workshop on Visual Computer Vision on GPU's*, 2008.

[19] L. Wang, H. Jin, and R. Yang. Search space reduction for MRF stereo. In *Proc. ECCV*, 2008.

[20] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nistér. Real-time global stereo matching using hierarchical belief propagation. In *Proc. BMVC*, 2006.

[21] K.-J. Yoon and I.-S. Kweon. Locally adaptive support-weight approach for visual correspondence search. In *Proc. CVPR*, pages 924–931, 2005.

[22] T. Yu, R.-S. Lin, B. Super, and B. Tang. Efficient message representations for belief propagation. In *Proc. ICCV*, 2007.