Hardware-Efficient Belief Propagation

Chia-Kai Liang, *Student Member, IEEE*, Chao-Chung Cheng, Yen-Chieh Lai, Liang-Gee Chen, *Fellow, IEEE*, and Homer H. Chen, *Fellow, IEEE*

***Abstract*—Loopy belief propagation (BP) is an effective solution for assigning labels to the nodes of a graphical model such as the Markov random field (MRF), but it requires high memory, bandwidth, and computational costs. Furthermore, the iterative, pixel-wise, and sequential operations of BP make it difficult to parallelize the computation. In this paper, we propose two techniques to address these issues. The first technique is a new message passing scheme named tile-based belief propagation that reduces the memory and bandwidth to a fraction of the ordinary BP algorithms without performance degradation by splitting the MRF into many tiles and only storing the messages across the neighboring tiles. The tile-wise processing also enables data reuse and pipeline, resulting in efficient hardware implementation. The second technique is an *O*(*L*) parallel message construction algorithm that exploits the properties of robust functions for parallelization. We apply these two techniques to a VLSI circuit for stereo matching that generates high-resolution disparity maps in near real-time. We also implement the proposed schemes on GPU which is four-time faster than standard BP on GPU.**

***Index Terms*—Belief propagation, Markov random field, energy minimization, embedded systems, VLSI circuit design, general-purpose computation on GPU (GPGPU).**

I. INTRODUCTION

MANY problems in early vision can be formulated as a label assignment problem, where each node of a graphical model is assigned a label representing a certain local quantity such as disparity or motion vector. Finding the optimal set of labels is an NP-hard energy minimization problem, for which many powerful optimization algorithms, such as graph cuts [8], loopy belief propagation (BP) [14], and tree reweighted message passing [20], have been developed. Among them, the loopy belief propagation has been widely applied to stereo matching [32], image denoising [13], image inpainting [21], and super resolution [14], just to name a few. The success of BP is due to its regularity and simplicity. It uses a simple message update process to iteratively refine the beliefs of labels for each node. A message sent from one node to another is updated according to neighboring messages and local energy functions, using simple arithmetic operations.

However, BP algorithms generally require a great amount of memory for storing the messages, typically on the order of tens to hundreds times larger than the input data. Besides, since each message is processed hundreds of times, the saving/loading of messages consumes considerable bandwidth. Therefore, although BP may work on high-end platforms such as desktops, it cannot be applied to most consumer electronic devices that have limited memory, computational power, and energy. Furthermore, because the messages are sequentially updated and each message is constructed via a sequential procedure, it is difficult to utilize hardware parallelism to accelerate BP.

In this paper, we propose two techniques, tile-based BP and parallel message construction, to address these issues. Tile-based BP splits the MRF into many tiles and only stores the messages across the neighboring tiles. The memory and bandwidth required by this technique is only a fraction of the ordinary BP algorithms. But the quality of the results, as tested by the publicly available Middlebury MRF benchmarks, is comparable to other efficient algorithms.

The parallel message construction technique is based on the observation that many hypotheses used to construct the messages are repetitive; therefore, they only need to be computed once. This observation allows us to reduce the complexity of message construction from $O(L^2)$ to $O(L)$. Moreover, unlike previous sequential algorithms, the proposed algorithm can be easily parallelized.

The proposed techniques can be realized in both hardware and software. A software reference implementation compatible to the Middlebury MRF library is available online [44], while two hardware examples (the first one is a VLSI circuit and the second one a GPU program) are analyzed in this paper.

The rest of paper is organized as follows. In Section II, we review the related work. In Section III, we formulate the energy minimization problem and describe the basic BP algorithm, with an analysis of the performance bottlenecks that have been largely neglected before. In Section IV, we describe tile-based BP and compare it with other optimization algorithms using the Middlebury MRF benchmarks. We describe the parallel message

Manuscript received October 10, 2009, revised April 24, 2010, accepted Oct. 10, 2010. This work was supported in part by NSC under the grant 96-2628-E-002-005-MY2 and by Himax Technologies Inc. An early version of this work was presented in CVPR 2009 [25].

C.-K. Liang was with the Graduate Institute of Communication Engineering, National Taiwan University, Taipei, Taiwan (R.O.C) 10617.

C.-C. Cheng and Y.-C. Lai are with the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei, Taiwan (R.O.C) 10617.

L.-G. Chen is with the Graduate Institute of Electronics Engineering and the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan (R.O.C) 10617.

H. H. Chen is with the Graduate Institute of Communication Engineering, the Graduate Institute of Networking and Multimedia, and the Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan (R.O.C) 10617. E-mail: homer@cc.ee.ntu.edu.tw.

construction algorithm in Section V and two implementation examples of the proposed techniques in Section VI. We discuss the limitations and extensions of the proposed techniques in Section VII and conclude in Section VIII.

II. Related Work

In this section we review the most relevant methods for accelerating belief propagation. We also briefly discuss the methods for accelerating BP in stereo matching.

Felzenszwalb and Huttenlocher were the first to address the computational issues of BP for computer vision applications [13]. They proposed a hierarchical message passing scheme to speed up the convergence, a so-called min-convolution algorithm using distance transform 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. However, the min-convolution method is strictly sequential and cannot be performed in parallel. Tappen and Freeman showed that a sequential asynchronous message update scheme is more efficient than the ordinary synchronous and bipartite graph schemes [34]. This scheme, short name BP-M [33], is efficient, but the bandwidth and memory issues are not considered.

Yu et al. showed that the messages are smooth and compressible and reduced the memory cost to 12.5% [43], but the memory for the data costs is unchanged and still too large for embedded systems. The sequential compression and decompression operations would complicate the implementation. In [22], Kumar and Torr proposed two techniques to reduce the memory cost of the generalized belief propagation (GBP) [41]. While their method shares similar motivations with ours, GBP is too expensive for real-time applications.

Simply splitting the image into small blocks and performing optimization for each block independently as in [35] does not guarantee the global optimality and leads to poor results. Park et al. proposed a VLSI architecture of hierarchical BP [27]. In their architecture, nodes in different layers are properly grouped for efficient data cost generation and message passing. However, their processing element (PE) uses the min-convolution algorithm that is much slower than our algorithm. Furthermore, while their PE only supports linear smoothness cost, the PE using our method can support any robust smoothness cost.

Efficient optimization methods like BP or graph-cuts are intensively used in stereo matching. However, due to the real-time demand in robot navigation, gaze correction, and free-viewpoint image synthesis, most systems use either local cost aggregation [12], [16], [42] or scanline-based optimization methods [11], [17]. Generally, for the same the energy functions, these algorithms are inferior to BP in performance. To accelerate optimization algorithms for stereo matching, a few GPU implementations have been proposed [9], [37], [40]. Unlike these methods that simply map the original algorithms to the high-performance GPU, we analyze the fundamental flows and operations of the algorithm and design a parallelism that requires smaller memory and bandwidth. Wang et al. used the results of local stereo estimation to reduce the size of the subsequent optimization problem [38]. However, the irregular disparity range and additional pre-processing steps are both impeditive to hardware efficiency.

III. Preliminaries of Belief Propagation

In this section we describe the energy minimization problem and the memory, bandwidth, and computational costs of BP.

A. Energy Minimization on Markov Random Field

Without loss of generality, we consider the discrete-labeling problem of a four-connected two-dimensional MRF. We adopt the notation defined in [33] with slight modifications. Specifically, we denote the set of all possible labels by l, the number of nodes (pixels) of the MRF by N, and the size of l by L.

The optimal label assignment corresponds to the maximum a posterior (MAP) solution of the MRF. If we take the negative log of the posteriori distribution as the energy (or cost) function, the optimal label assignment should minimize this function. The energy function E is defined by

$$E = \sum_p d_p(l_p) + \sum_{\{p,q\}\in\mathcal{N}} V_{pq}(l_p, l_q), \tag{1}$$

where the first term is the data energy, which is the sum of per-node data costs. The second term is the smoothness energy, which is the sum of smoothness costs of all neighboring node pairs. For two nodes $p = (x_p, y_p)$ and $q = (x_q, y_q)$, $\{p, q\}$ belongs to the set $\mathcal{N}$ if $|x_p - x_q| + |y_p - y_q| = 1$.

The energy functions are application-dependent. The data cost encodes the consistency between the assigned labels and the observed data, whereas the smoothness cost encodes the prior distribution of the neighboring labels. It has been shown that for general energy functions, finding the globally optimal solution is NP-hard and computationally intractable [8].

B. Loopy Belief Propagation

BP is an efficient technique for statistical inference. It was originally designed to find exact solution for graphical models with no cycle. However, for graphs with loops such as the MRF, BP can still find a strong locally optimal solution [14], [15].

Our focus is on the min-sum/max-product BP designed for finding the MAP solution. For each node p, we iteratively update L-dimensional messages sending from p to its neighbors. Each entity of the message in the t-th iteration is constructed using the messages in the $(t-1)$-th iteration:

$$m^t_{p\to q}(l_q) = \min_{l_p\in L}\left(V_{pq}(l_p, l_q) + H^{t-1}_{p\to q}(l_p)\right), \tag{2}$$

where $H^{t-1}_{p\to q}(l_p)$ is defined as

$$H^{t-1}_{p\to q}(l_p) = d_p(l_p) + \sum_{\{s,p\}\in\mathcal{N}\setminus\{q,p\}} m^{t-1}_{s\to p}(l_p). \tag{3}$$

Specifically, we first combine the messages from p's neighbors except q with the data cost to form an L-dimensional vector $H_{p \to q}$ using (3). Then, we combine $H_{p \to q}$ with various V_{pq}'s to generate hypotheses and pick the smallest one.

Many message-update schemes have been proposed to accelerate the convergence speed. In the BP-M scheme [34], the messages are passed along the rows, then along columns. For each row (column), the message from the first node is passed in the forward direction until the end of the row (column) is reached. Then the messages are passed backward. This scheme is asynchronous because each updated message is immediately used to update the next message.

After T iterations, where T is manually specified, or after the messages become fixed, we compute an L-dimensional belief vector for each node,

$$b_p\left(l_p\right) = d_p\left(l_p\right) + \sum_{\{s,p\} \in \mathcal{N}} m^T_{s \to p}\left(l_p\right), \tag{4}$$

and choose the optimal label of each node independently as the minimal of the belief vector. We find that BP-M is much faster than the synchronous scheme and the bipartite method and more stable than the hierarchical BP [13]. Therefore, we use it as the baseline for performance comparison.

At the first glance, BP seems highly suitable for parallel processing. First, in BP-M, several rows can be processed in parallel. The algorithm has no branching, so its throughput rate is constant. Second, the message update only uses simple additions and comparisons. Even we place hundreds of such processing elements on the chip, the required chip area is still very small. Third, the message construction process has a regular memory access pattern. Therefore, the parallelism does not require a complex memory access controller.

However, the degree of parallelism for ordinary BP is very limited. First, because each message is used to update other messages, they must reside in the main memory before the program terminates. Therefore, we have to store $4NL$ message entities. Consider performing stereo matching on *1080p30* (1920×1080 pixels per frame and 30 frames per second) video. If the disparity range is 300 and all data have 8-bit precision, we need 2.3 gigabytes (GBytes) of memory to store the messages. While it is common to have a few GBytes memory on a desktop computer, the memory of most consumer electronics and mobile devices is typically less than one hundred megabytes [1]. In fact, the data costs alone which are precomputed and kept in the main memory are too expensive for most mobile devices. Even for high-performance graphics cards, the available memory is usually no more than 1 GBytes. Hence, greedy or local algorithms are still commonly adopted for such platforms.

Second, the great number of messages leads to huge bandwidth consumption. We can see from (2) and (3) that generating an outgoing message requires the cost of loading three incoming messages and the data cost (assuming the smoothness costs can be calculated on the fly). To perform message passing over all nodes for T iterations, we have to load $16NLT$ elements from and write $4NLT$ elements to the memory. Therefore, the total bandwidth cost is $20NLT$. Consider again the stereo matching case described above, if we perform 100 iterations for each frame, the required bandwidth would be 33.9 terabytes per second (TBytes/s). Such a bandwidth is obviously unachievable for most existing platforms. Even with moderate N and L, the required bandwidth will soon exceed the desktop's capacity. Thus, we conclude that the bandwidth consumption is the main bottleneck of belief propagation.

To overcome this bottleneck, one may keep the updated messages in the on-chip cache for reuse. However, after a message is updated in BP-M, it will not be fetched within the next few thousand cycles, during which hundreds of new messages are generated. Therefore, the cache would have a very low hit rate. This suggests that all the data must be retrieved from and flushed to the off-chip memory, which requires large bandwidth and introduces long latency. This problem exists in other message-passing schemes as well.

Lastly, while the message construction operation (2) is simple, it becomes expensive when the size of label set is large. In a straightforward implementation, one has to go over all hypotheses to find the minima for each entity, resulting in $O(L^2)$ complexity. If the smoothness costs are either linear or quadratic functions of $|l_p - l_q|$, the complexity can be reduced to $O(L)$ by using the min-convolution method [13]. However, this method is strictly sequential and impossible to parallelize. Consequently, this method unavoidably takes $O(L)$ cycles.

In summary, although the basic operations of BP are simple, the massive messages it has to deal with require high memory and bandwidth. Besides, the sequential pixel-wise message-passing process makes it difficult to cache the data for reuse. Clearly, bandwidth consumption strongly limits the performance of BP. Moreover, the existing message construction method is sequential in nature, which cannot take advantage of modern parallel architectures.

IV. Tile-Based Belief Propagation

In view of the memory and bandwidth issues of BP resulted from the massive amount of messages, we want to design an algorithm that reduces the amount of messages and message traffics without performance degradation. In addition, the algorithm should have the following properties:

- **Regularity**. If the input data partitioned into subsets for the iterative algorithm, the workload of each iteration should be similar, so should the size of each data subset. The memory should be accessed in a fixed pattern.
- **Locality**. If intermediate variables can be reused many times, they should be kept on the on-chip buffer to avoid the expensive, slow off-chip memory access.

A. Basic Concept of the Proposed Tile-Based BP

Let us first consider the process of generating outgoing messages of a node p. According to (2), we need four incoming messages toward p, the data costs of p, and the smoothness cost between p and q. Besides these, we do not need the data of the

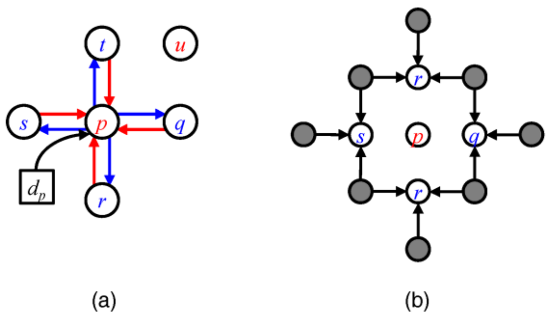


Fig. 1. (a) Message passing in the bipartite graph. To generate the messages from node p to its neighbors q, r, s, and t (outward blue arrows) we only need to access the messages toward (inward red arrows) and the data cost d_p. The messages related to node u are never involved. (b) To generate the messages toward p, we need the messages from the neighbors of the neighbors of p (the shaded circles).

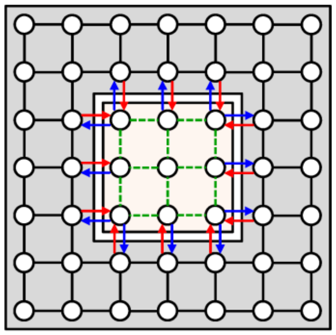


Fig. 2. The 7-by-7 MRF is split into two subsets. One subset N_1 contains the nodes in the tile (the light-shaded region) and the other subset N_2 contains all other nodes (the dark-shaded region).

nodes far away from p, as illustrated in Fig. 1(a). This property is exploited in bipartite graph where the nodes are split into two sets so that every edge connects two nodes of different sets. To generate messages from the first set to the second one, we only need the messages from the second set to the first one. Therefore, only a half of the messages are stored [13].

An interesting question is, what are the data required to generate the messages toward p? The answer is the data costs of p's neighbors, and the messages sent from the neighbors of these neighbors, as shown in Fig. 1(b). Again, we do not have to access the variables outside the group of those nodes. This rule can be easily extended. To generate the messages from the shaded nodes in Fig. 1(b) to p's neighbors, we only need the messages from their neighbors. Therefore, if we have the messages from the boundary of a region, we can sequentially generate the messages inward. After we reach the region center, we can then sequentially generate the outward messages. The only required inputs are the data costs and smoothness costs of this region and the messages of the boundary nodes.

In the discussion above, we consider a small region and a single iteration of message passing. However, this concept can be extended to multiple regions and iterations. For example, we can spit the nodes of MRF into two sets, as shown in Fig. 2. One set N_1 contains the nodes in a 3-by-3 tile and the other set N_2 contains all other nodes. When we perform BP in N_2, without

Algorithm Tile-Based Belief Propagation

```
1  for $t = 1, 2, \ldots, T_o$
2      for $i = 1, 2, \ldots, K$
3          Load or compute the energy functions of $S_i$.
4          Load boundary messages from the neighboring tiles.
5          Perform messages passing in $S_i$ for $T_I$ iterations.
6          Save boundary messages.
7      end for
8      for $i = K, K-1, \ldots, 1$
9          Load or compute the energy functions of $S_i$.
10         Load boundary messages from the neighboring tiles.
11         Perform messages passing in $S_i$ for $T_I$ iterations.
12         if $t = T_o$ then Calculate the optimal labels for $S_i$.
13         else Save boundary messages.
14     end for
15 end for
```

Fig. 3. Pseudo code of the tile-based belief propagation.

knowing messages in N_1 (dotted edges in Fig. 2), we only need the messages coming from N_1 to drive the propagation (outward arrows in Fig. 2). All the messages in the tile are irrelevant to the message passing outside the tile because they are never used in evaluating (2).

Similarly, when we perform BP in N_1, beside the messages coming from N_2 (inward arrows in Fig. 2), other messages outside the tile are unimportant. As long as the incoming messages carry the reliable information about the outside world, we can use BP to calculate the beliefs for the nodes in the tile.

In other words, the only messages that must always reside in the memory are those sending from one subset to another. When we perform message passing in one subset, the messages in the other one can be removed without affecting the operation. Unlike the bipartite graph method that attempts to reconstruct perfect messages [13], here we assume that the messages in the subset 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 IV.C.

B. Algorithm Description

We follow the concept described above and propose the tile-based belief propagation. The pseudo code is given in Fig. 3. We first split the MRF into many non-overlapping $B \times B$ tiles $\{S_1, S_2, \ldots, S_K\}$, ordered in a raster order. We then process these tiles sequentially (lines 2-7 in Fig. 3). First, if the data costs and the smoothness of the nodes in the current tile S_i are precomputed, we load them from the main memory. Otherwise, we load the necessary data to compute them on the fly. For certain applications, the on-line computation can greatly reduce the required memory and bandwidth, as we show in Section IV.D.

Then, we load the messages coming from the neighboring tiles, which are called *boundary messages*. We use the data terms, the smoothness terms, and the boundary messages to perform message passing between the nodes 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 becomes stable. We then generate the outgoing boundary messages and store them to the memory. All messages inside S_i are thrown away.

Processing the tiles in raster scan order would introduce a bias to the algorithm. 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. The bias may cause a strip artifact to the solution (see Figs. 7(d) and 8(d)). We resolve this problem by processing the tiles in an inverse-raster scan order (lines 8-14 in Fig. 3) after the raster scan is completed.

Another method to remove this bias is to use the boundary messages in a synchronous fashion. That is, all generated boundary messages are used for the next iteration. However, we find this method converges much slower than the method we used. This is similar to the case in the pixel-wise message passing where the asynchronous method is faster than the synchronous one [34].

At the final iteration ($t = T_O$), we calculate the labels for each tile (line 12 in Fig. 3). The operations of the tile-based meet the hardware-friendly properties mentioned in the beginning of the section. First, we split the input data into several tiles of identical size. Processing different tiles requires similar operations and costs. Therefore, the algorithm is very regular. We have also tried to split the MRF into irregular patches using the over-segmentation techniques and found the performance is similar to the regular approach. However, the computation varies with the shape and size of each patch. It is also difficult to efficiently record the boundary messages along the arbitrary-shaped boundaries. Also, the segmentation is itself complex and expensive for hardware implementation.

Second, the messages in the tile can be reused for updating each other many times. Because the number of these messages is much smaller than that of totally messages, they can be kept in the small but fast on-chip memory. Also, the data costs can be computed locally for each tile.

C. *Performance*

In tile-based BP, only a fractional number of messages are stored, and hence the memory and bandwidth costs are significantly reduced. However, we must ensure that the results of tile-based BP are comparable to those by other algorithms. Otherwise, one could simply use the cheap local or scan-line-based algorithms for the low-end devices. Moreover, tile-based BP should find good solutions for most problems, not only for a few specific ones.

Generally, the quality of the solutions depends on the intrinsic parameters and the definitions of the energy functions. We first analyze the effect of the parameter settings and compare tile-based BP with BP-M for stereo matching. Then, we compare tile-based BP with other optimization algorithms.

There are three parameters in tile-based BP: the size of the tile ($B \times B$), the number of the inner iterations (T_I), and the number of outer iterations (T_O). Intuitively, using more inner iterations, the reconstructed messages in the tile become more accurate and the outgoing boundary messages become more reliable. Using more outer iterations, we have more chances to refine the wrong

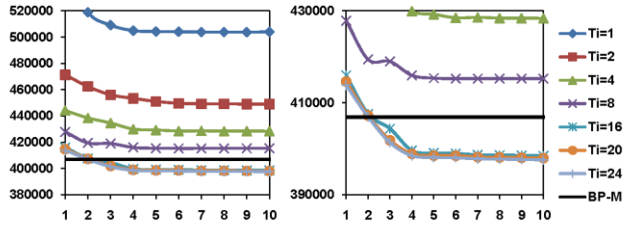


Fig. 4. *Left*: energies of stereo matching on Tsukuba using tile-based BP with different number of inner iterations. *Right*: the close-up of the left chart. The x-axis is the index of the outer iteration and the bold horizontal line is the reference energy obtained using BP-M with 500 iterations. The energy terms are defined according to [33].

TABLE I
THE ENERGIES OF BP-M AND TILE-BASED BP FOR THE STEREO MATCHING BENCHMARKS

	Tsukuba	***Venus***	***Teddy***
BP-M 500 iterations	406936	3099668	1393166
Tile-based BP $(B,T_I) = (16,20)$	396953 **(97.55%)**	3080688 **(99.39%)**	1384749 **(99.40%)**
Tile-based BP $(B,T_I) = (32,28)$	402363 (98.88%)	3106563 (100.22%)	1387006 (99.56%)
Tile-based BP $(B,T_I) = (64,56)$	403448 (99.14%)	3093890 (99.81%)	1385229 (99.43%)

boundary messages and thus obtain better solutions. However, we also hope that tile-based BP works in a way similar to BP-M, which converges to a strong local optimal solution in a reasonable number of iterations.

We perform many experiments to determine these parameters. One representative example of stereo estimation benchmark "Tsukuba" is shown in Fig. 4. We set B=16 and several different T_I's, and compare the resulting energies with the *reference energy* obtained using BP-M after 500 iterations (the black line in Fig. 4).

We can see that as the algorithm iterates, the energy keeps decreasing. Generally, tile-based BP converges after $T_O = 5$. When we only perform message passing once for each tile (i.e., $T_I = 1$), because the messages in the tile are not well reconstructed, the energy is much higher than the reference energy. However, as we increase T_I, the energy goes down and become lower than the reference one after $T_I = 16$. By further increasing T_I, we can achieve a slightly lower energy in this case, but generally there is no significant gain by setting $T_I > 20$. However, when we set T_I to an extremely large value, the energy slightly increases.

We then increase the tile size and, as we expect, because more messages have to be reconstructed, we need more inner iterations. For $B = 32$, the required T_I is 28, and for $B = 64$, the required T_I is 56. Therefore, when the tile size increases quadratically, T_I only increases linearly. The energies obtained by tile-based BP with different tile sizes and inner iteration numbers are listed in Table I. We can see that the energies obtained by tile-based BP are generally lower than those by BP-M except one dataset with one specific parameter setting. We also observe that the energies slightly increase when the tile size increases. It is because when B becomes larger, there are fewer boundary

TABLE II
COMPARISON OF ENERGIES

Application	Binary Image Segmentation			Denoising/Inpainting		Stereo Matching			Photomontage	
Dataset	*Flower*	*Person (Fig. 5)*	*Sponge*	*House (Fig. 6)*	*Penguin*	*Tsukuba (Fig. 7)*	*Venus (Fig. 8)*	*Teddy*	*Family*	*Pano*
ICM [3]	113.85%	131.59%	109.71%	111.26%	132.91%	653.36%	405.11%	234.30%	1620.87%	528.25%
Expansion [8]	100.00%	100.00%	100.00%	102.79%	104.38%	100.44%	102.56%	100.52%	102.26%	101.99%
Swap [8]	100.00%	100.00%	100.00%	100.91%	112.58%	100.82%	103.08%	100.84%	197.10%	104.41%
BP-S [33]	100.09%	100.04%	100.02%	100.25%	101.33%	115.66%	110.20%	106.96%	1966.02%	567.98%
BP-M [34]	100.04%	100.02%	100.00%	100.15%	101.40%	110.20%	101.70%	104.20%	341.45%	188.21%
Tile-based BP	100.04%	100.02%	100.00%	100.12%	101.16%	107.51%	101.07%	103.57%	605.12%	290.02%
TRW-S [20]	100.00%	100.00%	100.00%	100.00%	100.03%	100.02%	100.02%	100.62%	101.09%	115.30%

The energies are measured relative to the maximum lower bound of the energy function (equation (1)*) obtained by TRW-S.*
For ICM, Expansion, Swap, BP-M, BP-S, and TRW-S, each algorithm is performed for at most 500 iterations.
For tile-based BP, the block size is 16, the number of inner iterations is 20, and the number of outer iterations is at most 12.

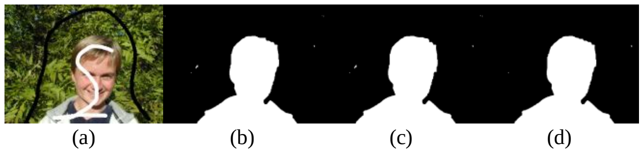


Fig. 5. Binary image segmentation of the benchmark “Person”. (a) The input image and the strokes. (b) Result of BP-M. (c) Result of tile-based BP. (d) Result of TRW-S.

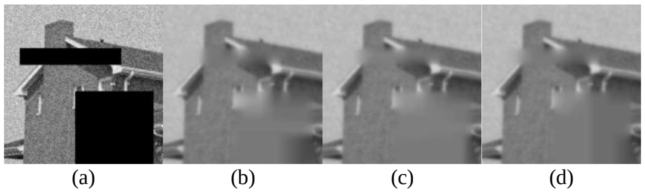


Fig. 6. Image denoising and inpainting of the benchmark “House”. (a) The input image where the black regions are to be inpainted. (b) Result of BP-M. (c) Result of tile-based BP. (d) Result of TRW-S.

messages to guide the message passing. As a result, the reconstructed messages are less accurate.

We further compare tile-based BP with other techniques, including ICM [3], two graph cuts algorithms (Expansion and Swap) [8], two BP algorithms (BP-S and BP-M), and TRW-S [20], using the Middlebury MRF benchmarks [33], and show the results in Table II. The data and smoothness energies are defined as in [33]. We use the online reference implementation to generate the energy functions. Because the performance of tile-based BP is stable under various parameter settings, we only list the results obtained by setting $(B, T_I, T_O) = (16, 20, 12)$.

For interactive binary image segmentation [28], our method performs similar to BP-M and slightly worse than graph cuts algorithms. The results of the benchmark “Person” is shown in Fig. 5. Visually, the results of BP-M and tile-based BP are close to the globally optimal solution obtained by TRW-S.

For the image denoising and inpainting, the label set is the intensities {0, 1, …, 255}. Because the size of the label set is large, the graph cuts algorithms are inferior to the message passing ones. The results for the benchmark “House” is shown in Fig. 6. We can see that visually the result of tile-based BP is very close to the best result of TRW-S.

For stereo matching, tile-based BP always performs better than BP-M. The performance of the graph cuts algorithms depends on the definition of the smoothness energy. They are better than the BP algorithms when the smoothness energy is a truncated linear function (for “Tsukuba” and “Teddy”) and worse when it is a truncated quadratic function (for “Venus”).

The results and ground truths of the benchmarks “Tsukuba” and “Venus” are shown in Figs. 7 and 8, respectively. We also show a few variants of tile-based BP. The block-based BP algorithm described in [35] totally drops the boundary messages, minimizes the energy of each block independently, and converges to a bad local minimum. We can see the resulting blocky artifact in Figs. 7(c) and 8(c).

If we only process the tiles in a raster scan order, as we have discussed, the reliabilities of the boundary messages from different directions are unbalanced, and thus the messages are biased. One result without applying inverse raster scan is shown in Fig. 7(d), where the disparity values at the right side of the statue are overly influenced by the statue. This artifact is common in many scanline-based optimization algorithms. However, right after applying one inverse raster scan, this problem is gone, as shown in Figs. 7(e) and 8(e).

Finally, for interactive digital photomontage [1], our method becomes worse than BP-M, which is also inferior to the graph cuts algorithms. This suggests that the energies defined for this application are intrinsically difficult for the BP algorithms.

In summary, for the applications where BP-M works well, tile-based BP works comparably well. Actually, the parameters of the basic routines in the BP algorithms, such as message damping and normalization, affect the quality of the results more than the tile-based message passing scheme. In this following subsection, we quantitatively analyze the costs of tile-based BP.

D. Cost Analysis

Compared with ordinary BP, tile-based BP significantly reduces the memory and bandwidth consumptions. For a 4-connected MRF with N nodes, BP-M has to store NL precomputed data costs and $4NL$ message entities. Therefore, the total memory cost is $5NL$, $5L$ times larger than the input image.

In tile-based BP, most messages are dropped after the tile is processed, and hence we only have to store the boundary messages and the messages in the current tile. Specifically, the number of the entities for the boundary messages is

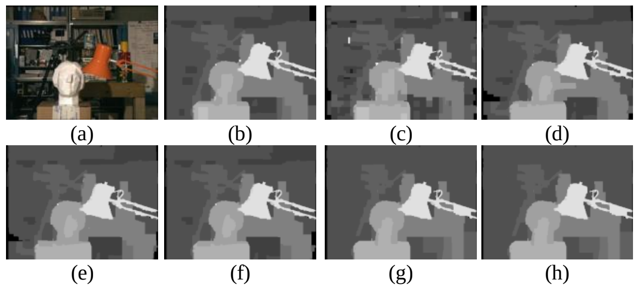


Fig. 7. Stereo matching of “Tsukuba”. (a) The left image. (b) Result of BP-M after 500 iterations. (c) Result of block-based BP. (d) Result of tile-based BP without inverse raster scan. (e) Result of tile-based BP with one outer iteration and (f) 20 outer iterations. (g) Result of TRW-S. (h) Ground truth.

$$\left(NB^{-2}\right)\cdot 4B\cdot L = 4NLB^{-1}, \qquad (5)$$

where the first term is the number of tiles and the second term is the number of outgoing boundary messages for each tile.

The messages in the current tile require a buffer of size $4B^2L$. However, where to put this buffer depends on the chosen platform. In our hardware implementation, we put it on the fast on-chip memory, so it does not occupy the main memory. In our software implementation using C language, we allocate this buffer in the main memory for simplicity.

Because the data costs are repeatedly accessed in the iterative message processing, we can compute the data costs once for each tile. Therefore, the relative computational cost is negligible. For example, calculating all the data costs of the dataset “Venus” in Fig. 8 only takes 0.063 seconds on a Xeon 2.0GHz desktop (using 1 core), but performing one BP-M iteration takes 1.3 seconds. In this way, we reduce the storages of data costs from NL to B^2L, manageable for on-chip memory. The smoothness costs can be handled in a similar way.

The on-the-fly computation of the data costs requires the storage of input data, such as the left and right images in stereo matching. However, the input data are only of order N and typically much smaller than the storage of data costs. For most systems, these input data are also used for visualization or other applications. Therefore, we do not consider this cost here.

The memory costs of tile-based BP with different design choices relative to BP-M are listed in Table III. The relative costs are invariant to the size of the label set (L) and slightly vary with N. In the hardware implementation we only store the boundary messages in the main memory, and thus the costs are dramatically reduced. For B = 16, tile-based BP only requires 5.0% memory of BP-M. For the stereo matching of the VGA-sized data, this leads to less than 5 megabytes main memory and 80K on-chip memory. In the software implementation using C-language, the data costs and the messages in the tile are allocated in the main memory, but we can still reduce the memory costs to 22.32–25.07% when the data costs are precomputed and 2.58–5.08% when the data costs are calculated on the fly. This significant reduction makes it possible to perform BP on data too big to fit in the memory.

A stereo matching result on a high-resolution data “Art” is

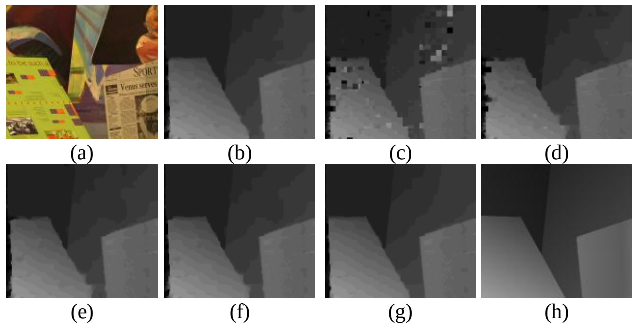


Fig. 8. Stereo matching of “Venus”. (a) The left image. (b) Result of BP-M after 500 iterations. (c) Result of block-based BP. (d) Result of tile-based BP without inverse raster scan. (e) Result of tile-based BP with one outer iteration and (f) 20 outer iterations. (g) Result of TRW-S. (h) Ground truth.

shown in Fig. 9. Because BP-M requires 6.9 GBytes memory, it cannot be performed on a 32-bit platform. However, tile-based BP can be successfully performed using less than 2 GBytes memory. If we compute the data costs on the fly, only less than 400 MBytes memory is required.

The memory reduction and regular tile-wise process also lead to a great saving of bandwidth. In hardware implementation, we can put the messages of the current tile in the fast on-chip buffer. When we process this tile, we only have to load the boundary messages from the off-chip main memory. Because the message-passing process of T_I iterations can be performed without accessing the main memory, the processing speed is no longer limited by the available bandwidth but the degree of parallelism, which can be large due to the simplicity of the operations. After the tile is processed, we only store the outgoing boundary messages. Because in one outer iteration, each tile is processed twice, the bandwidth cost of messages is

$$2\cdot\left(NB^{-2}\right)\cdot\left(4BL+4BL\right)\cdot T_O = 16NLT_OB^{-1}, \qquad (6)$$

where the first $4BL$ term is due to the loading of the incoming boundary messages and the second one is due to the saving of the outgoing boundary messages. The cost only depends on the number of outer iterations T_O regardless of the number of inner iterations T_I. As we have shown in the previous subsection, T_O is much smaller than the number of iterations T in traditional BP algorithms such as BP-M.

The bandwidth cost of data and smoothness costs is more dependent on the design choices and the applications. When the data costs are precomputed in the beginning, we have to load B^2L elements for processing each tile, and overall the bandwidth cost is $2NLT_O$. Again, because $T_O << T$, this cost is much smaller than that of BP-M. For many applications, the bandwidth can be further reduced if the data costs are computed on the fly. For example, in stereo matching and image denoising, we can just load $O(B^2)$ pixels to generate the data costs and reduce the bandwidth to $O(NT_O)$ regardless of L. This technique for the stereo matching is described in Section VI.A. On the contrary, if we compute the data costs on the fly in BP-M, the bandwidth cost is $O(NT)$ because there is no simple way to reuse the input data or the computed data costs.

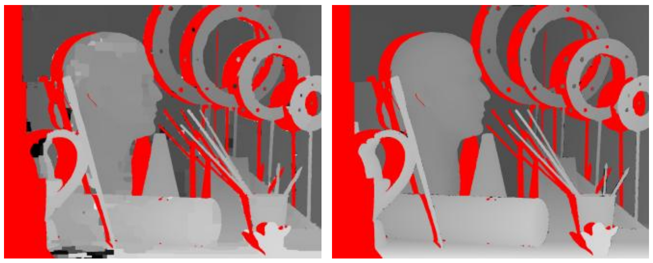


Fig. 9. Stereo matching of a high-resolution dataset “Art”. The image size (N) is 1390-by-1110 and the disparity range (L) is 240. (a) Result of tile-based BP with (B, T_I, T_O) = (16, 20, 12). (b) The ground truth. For better visualization, the occluded regions which are not handled by the adopted simple energy functions are masked as red.

TABLE III
MEMORY COSTS OF TILE-BASED BELIEF PROPAGATION

Tile Size (B)	A+B+C	A+B+D	A*
16	25.07%	5.08%	5.00%
32	22.77%	2.83%	2.50%
64	22.32%	2.58%	1.25%

The numbers are given relative to the memory cost of the BP-M (all messages + precomputed data costs). The problem size is N=640×480 and L= 64.
A: Boundary messages. B: Messages in the current tile. C: Precomputed data costs for the whole MRF. D: Data costs for the current tile.
** B and D are allocated on the on-chip memory.*

We use the stereo matching of a VGA-sized 30 fps video to compare bandwidth costs of several BP algorithms and design methods. Because the goal of this application is real-time performance, we reduce the iteration number of BP-M to 20 and the outer iteration of tile-based BP to 5. Under this setting, the estimated disparity maps of BP-M are worse than the results of tile-based BP but are still much better than those of the local or scanline-based algorithms.

The bandwidth costs are summarized in Table IV. In this example, BP-M has to access 219.73 GBytes per second, which is impossible to achieve on most high-end platforms. Therefore, the processing speed is severely bandwidth-limited. For tile-based BP, the required bandwidth is a few gigabytes per second, which is acceptable on modern desktops and graphics cards. We can also see the data costs dominate the bandwidth now. If we perform the on-line data computation for each tile, the bandwidth consumption can be further reduced. Because the total bandwidth cost of tile-based BP plus the on-line data computation is only 0.39–1.33% of BP-M, the main bottleneck of the belief propagation is resolved.

Finally, when using the same number of iterations, the arithmetic complexity of tile-based BP is similar to that of BP-M. That is, if tile-based BP is performed for T_O iterations and in each tile BP-M is performed for T_I iterations, the number of computed messages is identical to the case where the global BP-M is performed for $2T_OT_I$ iterations. In the software implementation the timing matches this claim. However, because tile-based BP requires much less bandwidth, the processing cores can work extremely fast without waiting for the data retrieval. Moreover, we can apply a tile-wise pipeline to eliminate the data latency, as described in Section VI.A.

TABLE IV
BANDWIDTH COSTS OF DIFFERENT BELIEF PROPAGATION ALGORITHMS

		Data cost (GBytes/s)	**Message (GBytes/s)**	**Total (GBytes/s)**
BP-M		43.95	175.78	219.73 (100.00%)
Tile-based BP	B = 16	5.49	2.75	8.24 (3.13%)
	B = 32	5.49	1.37	6.87 (2.81%)
	B = 64	5.49	0.69	6.18 (2.66%)
Tile-based BP+ On-line data cost computation	B = 16	0.17	2.75	2.92 (1.33%)
	B = 32	0.17	1.37	1.54 (0.70%)
	B = 64	0.17	0.69	0.86 (0.39%)

The application is stereo matching. The input data is a VGA-sized video (N=640×480, L= 64, and the frame rate is 30 frames per second).
The number of iterations of the BP-M is 20, and the number of the outer iterations of the tile-based BP is 5. All messages, input images, and data costs are stored as 8-bit integers. The smoothness costs are analytic functions of the labels which can be computed on-the-fly.

V. PARALLEL MESSAGE CONSTRUCTION

For constructing an L-D message, according to (2) and (3) we have to calculate L^2 hypotheses. Therefore, the arithmetic complexity of this operation is $O(L^2)$. For many applications such as stereo matching, the smoothness cost $V_{pq}(l_p, l_q)$ can be represented in a simple form:

$$V_{pq}\left(l_p,l_q\right)=w_{pq}\,f\left(\left|l_p-l_q\right|\right)=w_{pq}\,f\left(\Delta l\right), \qquad (7)$$

where w_{pq} is a spatially varying weighting coefficient, f is an analytic function of the absolute difference of the labels of two nodes, which is denoted by Δl for convenience. When the function is linear or quadratic, one can construct the messages using only $O(L)$ hypotheses [13]. However, this so-called min-convolution method is a strictly sequential operation. Each entity of the message must be calculated after the preceding one is done. Here, we propose a fast message construction method also with complexity $O(L)$, but it can be parallelized.

The cost functions serve for measuring the compatibility between the labels and the observations or prior knowledge. In certain situations, the observed data are *outliers* that are incompatible with the models. For examples, we commonly assume in the stereo matching that the disparity values vary smoothly between neighboring pixels. However, this assumption is valid only when the corresponding pixels belong to the same object. Otherwise, the difference between their disparity values can be arbitrarily large, resulting in a high cost value.

To suppress the effects of the outliers, we can use robust functions as cost functions instead of typical linear or quadratic estimators [5], [6]. It has been shown that the usage of the robust functions is critical in many problems, such as stereo matching [32], [24], [29] and image restoration [6], [13]. A number of popular robust functions are depicted in Fig. 10. We can clearly see that the robust functions can be split into two regions. In the *inlier* region, the function value gradually increases. It corresponds to the case where the cost function should increase as the noises in the data or estimation increase.

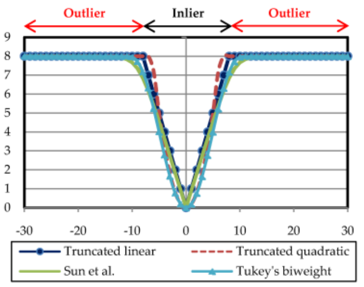


Fig. 10. Illustration of several typical robust functions. We choose the parameters to match the cut-off regions and the penalty value of the outlier of different functions.

On the other hand, in the *outlier* region, the function value is nearly constant. It is because when the input is an outlier which can hardly be modeled, the only way to penalize it is assigning a uniform cost.

A. Using Robust Function as Smoothness Cost

When we use a robust function as the smoothness cost, without loss of generality, we may assume w_{pq} = 1 and rewrite the smoothness cost in (7) as

$$V_{pq}\left(l_p,l_q\right)=\begin{cases} e(\Delta l), & \text{if } \Delta l \le M \\ K, & \text{otherwise} \end{cases}, \qquad (8)$$

where the constant $K \ge e(M)$ as shown in Fig. 10. For certain robust functions, the outlier regions are not exactly flat, but approximating them by constants does not make noticeable difference in the experimental results.

We notice that, for most applications, the range of the inlier M is much smaller than the size of the label set L. This comes with no surprise, because the pairwise smoothness cost can represent simple models that are valid only when two labels are close. For example, in image restoration, the label represents image intensity (i.e., L = 255). When the neighboring pixels should belong to the same patch, the intensity difference is mainly due to noise. In this case the robust function serves to make Δl smaller. On the contrary, when Δl is larger than the noise level, these two pixels should be uncorrelated.

When we construct an entity of a message using (2), we first calculate L hypotheses, which can be grouped into two subsets according to the value of Δl. Therefore, we can rewrite (2) as

$$m_{p\to q}\left(l_q\right)=\min_{l_p}\left(S_{q,i},S_{q,o}\right), \qquad (9)$$

$$S_{q,i}=\left\{e(\Delta l)+H\left(l_p\right) \mid \Delta l \le M\right\}, \qquad (10)$$

$$S_{q,o}=\left\{K+H\left(l_p\right) \mid \Delta l > M\right\}, \qquad (11)$$

where the time index of m and subscript of H are removed for simplicity. The first set contains the hypotheses generated using the inlier region of the robust function, and the second one contains those generated using the outlier region of the robust function.

We show that the minimum of $S_{q,o}$ is independent of q and can be calculated once and applied to all q's. Let S_o denote the set of $H(l) + K$ for all l_p, then we have

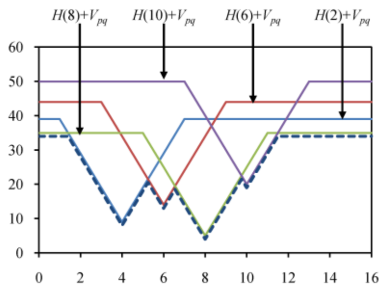


Fig. 11. Illustration of the message construction process for the case where the smoothness cost is a robust function. In this figure, $e(\Delta l)$ = 10min(|Δl|, 3), K = 10, and T = 2. For clarity, only 4 out of 16 $H + V_{pq}$ are shown. The lower envelop shown as the dotted line is the final message.

$$\min_{l_p}\left(S_o\right)=\min_{l_p}\left(H\left(l_p\right)\right)+K \le \min\left(S_{q,o}\right). \qquad (12)$$

According to the definition in (8), when argmin($H(l)$) = l_i and $|l_i - l_q| < M$, $H(l_i) + e(|l_i - l_q|) \le H(l_i) + K$. Therefore, (9) is equivalent to

$$m_{p\to q}\left(l_q\right)=\min_{l_p}\left(S_{q,i},S_o\right)=\min_{l_p}\left(S_{q,i},\min_{l_i}\left(H\left(l_i\right)\right)+K\right), \qquad (13)$$

where the second term can be calculated only once in constructing the message.

In summary, we first find the minimum of H's. Then we find the minimum of $S_{q,i}$ for each l_q, and compare it with the minimum of H's plus K to obtain the final message. The message constructed this way is identical to the one constructed by (2).

B. Cost Analysis

Because the set of hypotheses from the inlier region is much smaller than L, and the minimum of the hypotheses is independent of q, we can avoid calculating many redundant hypotheses. This is illustrated in Fig. 11, where the hypotheses from four H's are shown, and the message is the lower envelop of these hypotheses. We can see that, among the sets $S_{q,o}$ of all q's, only the one of the minimum of H's would present in the final messages. The minima of all other sets are always larger than min(S_o), so we can simply avoid calculating them. Also, min(S_o)+K is present many times in the final message, but it only has to be computed once.

Specifically, the proposed method requires $O(L)$ operations to calculate min(S_o)+K, $O(ML)$ operations to find min($S_{q,i}$) for all l_q's, and $O(L)$ operations to calculate the final message values. Therefore, the overall complexity is $O(ML)$. When M<<L, the complexity becomes $O(L)$.

C. Parallel Computation

While the complexity is similar to the min-convolution method, our parallel method is more suitable for hardware because it provides more freedom on disparity-parallel order.

For comparison, we give the pseudo codes of the min- convolution algorithm and our algorithm in Figs. 12 and 13, re-

```
Algorithm Min-Convolution Message Construction
Input: $M_{rp}, M_{sp}, M_{tp}, D_p, S, T$    Output: $M_{pq}$
1  for $l = 0,...,L-1$ in parallel
2      $H[l] = (M_{rp}[l] + M_{sp}[l]) + (M_{tp}[l] + D_p[l])$
3  end for
4  M_min = $min\_all(H, L) + S*T$
5  for $l = 1,...,L-1$
6      $H[l] = \min(H[l-1] + S, H[l])$
7  end for
8  for $l = L-2,...,0$
9      $H[l] = \min(H[l+1] + S, H[l])$
10 end for
11 for $l = 0,...,L-1$ in parallel
12     $M_{pq}[l] = \min(H[l]$, M_min)
13 end for
```

Fig. 12. The pseudo code of the min-convolution message construction. The smoothness cost is a truncated linear function: $f(\Delta l) = S\cdot\min(|\Delta l|, T)$.

```
Algorithm Parallel Message Construction
Input: $M_{rp}, M_{sp}, M_{tp}, D_p, S, T$    Output: $M_{pq}$
1  for $l = 0,...,L-1$ in parallel
2      $H[l] = (M_{rp}[l] + M_{sp}[l]) + (M_{tp}[l] + D_p[l])$
3  end for
4  M_min = min_all$(H, L) + T$                      // M_min = min($S_O$)
5  for $l = 0,...,L-1$ in parallel
6      for $m = -M,...,M$ in parallel
7          M0[l][m] = $H[l+m] + S*\text{abs}(m)$      // M0[$l$] = ($S_{l,i}$)
8      end for
9  end for
10 for $l = 0,...,L-1$ in parallel
11     M1[$l$] = min_all(M0[$l$], $2M+1$)            // M1[$l$] = min($S_{l,i}$)
12 end for
13 for $l = 0,...,L-1$ in parallel
14     $M_{pq}[l]$ = min(M1[$l$], M_min)
15 end for
```

Fig. 13. The pseudo code of the proposed parallel message construction algorithm. The smoothness cost is a truncated linear function: $f(\Delta l) = S\cdot\min(|\Delta l|, T)$.

spectively, where the for loops that can be performed in parallel are explicitly annotated. The sub-routine min_all (A, L) returns the minimum of the array A of length L. While its arithmetic complexity is $O(L)$, it can be performed in parallel by using a comparator tree or a min-reduction operation. We can see that there are two strictly sequential for loops in the min-convolution algorithm (line 5-10 in Fig. 12). On the contrary, while the proposed algorithm has the same number of first-level for loops, all loops can be performed in parallel. The loop that generates all hypotheses of $S_{q,i}$ can be totally unrolled into $(2M+1)L$ independent operations (lines 5-9 in Fig. 13).

Our parallel algorithm can also be visualized as shown in Fig. 14, where the variables names are the same as those in the pseudo code (Fig. 13). This figure represents the implementation of our algorithm at the maximum degree of parallelism. We first compute H's using (3) and generate the hypotheses in $S_{q,i}$ of all q's in a 2D array M0. We then generate the minimum of M0 using L min_all sub-routines in parallel. Meanwhile, the minimum of H's is calculated. Finally, the message is the minimum of M_min and M1. There are total $(3+2M)L$ adders and $(2ML+2L-1)$ comparators. Compared with the on-chip memory for storing the messages and data costs, these elements are infinitesimally small. Also, all the temporal variables, including H, M0, M1, and M_min, do not have to be explicitly stored in

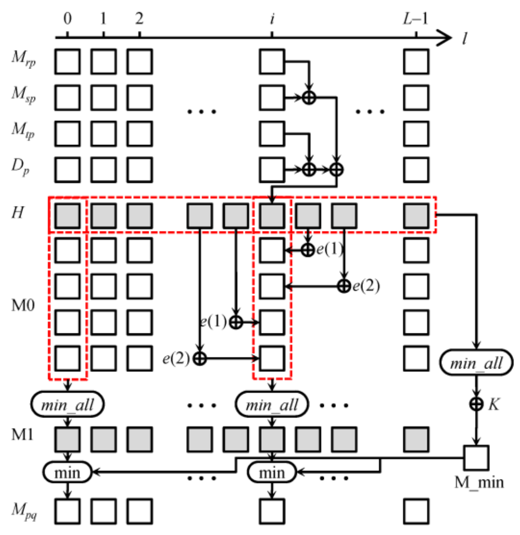


Fig. 14. The parallel implementation of the proposed fast message construction algorithm. In this case, $M = 2$ and e can be any monotonic function.

registers. This means that all calculations can be wired together. This approach is used in our VLSI implementation.

VI. Hardware Realizations

In this section, we present two implementations of the proposed techniques, a VLSI circuit and a GPU program, for the stereo matching application.

A. Very Large Scale Integration (VLSI) Circuit

The system overview of the stereo matching processor and the off-chip memory is shown in Fig. 15. The processor consists of three major components, an on-chip buffer, a data cost generator, and a message construction cluster. We use the UMC 90 nm manufacture technology and list the specification in Table V. The floorplan of the circuit is shown in Fig. 16. In this design, the tile size (B) is 16 and L is 64. The design supports coarse disparity sampling, and hence the disparity range can be either 0–63 or 0–126, which is larger than most existing GPU, VLSI, and FPGA implementations [12], [17]. The circuit itself only consumes 445.4 mW when working at 185 MHz. It consists of 2.5M gates (about 10M transistors) and the die size is about 25 mm^2. Because our design consumes very little power, it is suitable for embed system applications.

The off-chip memory stores the left image and right image, which can be either captured from an imaging sensor or streamed from the host processor. It also stores the boundary messages and the disparity map, which only takes a few megabytes (see the analysis in Section IV.D).

The inlier region of the smoothness cost is fixed in this design. Specifically, we set $M = 3$ in (8). This setting works well in our experiment (see Fig. 19). We use the Birchfield-Tomasi cost sas the data cost [4].

We exploit the pixel-overlapping nature of the tile-wise processing to minimize the bandwidth consumption. When we calculate the data costs of the first tile of a row, we only need to

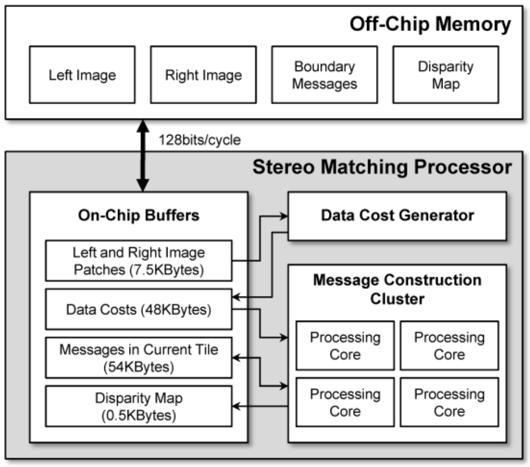


Fig. 15. Systematic overview of the developed VLSI circuit.

load 1) $B\times B$ pixels in the left image and 2) $B(B+L)$ pixels in the right image from the off-chip memory. However, when we calculate the data costs of the next tile, many pixels in the right image are repetitive and already loaded. Therefore, we only have to load the extra $B\times B$ pixels, as illustrated in Fig. 17. This rule applies to all subsequent tiles of the same row. In this way, when we perform tile-based BP for one raster scan (or inverse raster scan), every pixel of the input images is loaded only once. This data reuse scheme was previously used video codec [36]. Because we transform the pixel-wise BP into a tile-wise process, this scheme can be readily adopted.

The message passing of the tile is performed by a message construction cluster, which consists of four processing cores. Each processing core is a realization of the proposed parallel message construction algorithm at the maximum degree of parallelism, as shown in Fig. 14. Therefore, we can calculate four messages in a single cycle. Including the initialization and controlling, it takes 423 cycles to perform one-iteration BP-M for the 16^2 tile (i.e., generate 1024 messages). Generating the beliefs and the disparity values can be done using the same processing cores without additional cycles.

We use a two-stage pipeline as shown in Fig. 18 to improve hardware utilization and efficiency. Specifically, when we perform message passing for the current tile, we also write the boundary messages (and the disparity values if necessary) of the previous tile to the off-chip memory, then load the boundary messages and calculate the data costs of the next tiles. In this way, we improve the processing speed by two-fold and make all processing cores work at maximal rate. The additional cost of the pipeline is a few ping-pong buffers for the data costs, disparity values, and boundary messages.

Because we have greatly reduced the bandwidth consumption, the performance is limited by the computational power. That is, the processing speed is bounded by the message passing step (first row in Fig. 18) unless $T_I < 3$. Although the functionalities of the processing cores are fixed, the processing speed can be adjusted by changing the numbers of the inner and outer iterations, as shown in Table VI.

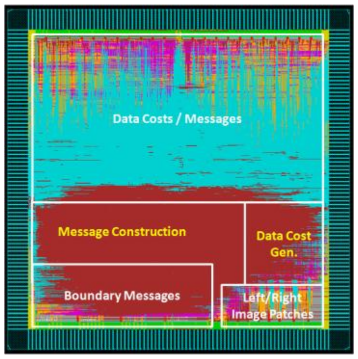


Fig. 16. The floorplan of the developed VLSI circuit.

TABLE V
SPECIFICATION OF THE VLSI CIRCUIT FOR STEREO MATCHING

Technology		UMC 90 nm
Operating voltage		1.0 V
Power consumption		445.4mW (at 185MHz)
Chip area		5×5 mm^2
Gate count	Processing cores	0.633M
	Memory	1.871M
	Total	2.505M
Disparity range (L)		64 ({0,1,2,…,63}) or 128 ({0,2,4,…,126})
Tile size (B)		16

When high-quality results are desired, as our analysis in Section IV.C shows, we should set $(T_I, T_O) = (20, 5)$. In this case, our implementation can generate QVGA-sized disparity maps at 7.28 frames per second (fps) and VGA-sized disparity map at 1.82 fps. If we only want a reasonable result, we can set $(T_I, T_O) = (20, 1)$ and increase the frame-rate five-fold. As shown in Figs. 7, 8, and 19, the results after the first outer iteration is already very close to the final results. If we want the maximal-speed performance, we can set $(T_I, T_O) = (3, 1)$. The degraded result is still better than most fast local methods and is suitable for applications such as robot navigation.

For comparison, generating a QVGA-size disparity map on a Xeon 2.0GHz desktop using tile-based BP with $(T_I, T_O) = (20, 5)$ takes 150 seconds and using BP-M with 200 iterations takes 142 seconds. Our VLSI circuit runs 1000 times faster.

Because the operations in our design are very simple, techniques such as the data reuse for data cost generation and the two-stage pipeline described here can be adopted in other multi-core processors, such as Imagine [19] and Larrabee [31].

B. Graphics Processing Unit (GPU) Program

We use the CUDA technique [26] to implement a GPU disparity estimation system. CUDA provides a C language environment to write programs for execution on GPU. In the CUDA programming model, the programmer can call the functions, or *kernels*, to be executed N times in parallel by N individual *threads*. There are two sets of inputs to the threads. The first set is broadcasted to all threads and defined as typical function arguments. The second set is the *thread index*, which is an n-D vector ($n < 4$). The index of every thread is unique. Therefore, although the programs of all threads are identical, we can use the

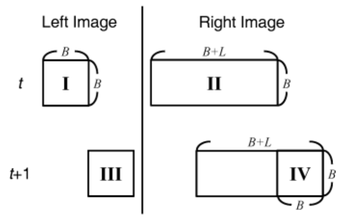


Fig. 17. The data reuse scheme for calculating the data costs. When we calculate the data costs of the first tile of a row, we need to load B^2 pixels in the left image (**I**) and $B(B+L)$ pixels in the right image (**II**). However, in calculating the data costs of the next tile, only B^2 additional pixels are loaded (**IV**).

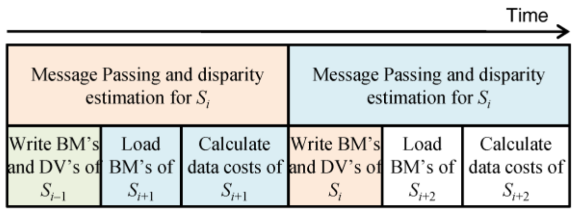
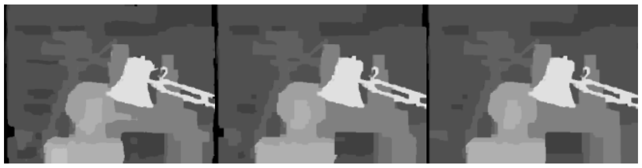

Time
Message Passing and disparity estimation for S_i
Message Passing and disparity estimation for S_i
Write BM's and DV's of S_{i-1}
Load BM's of S_{i+1}
Calculate data costs of S_{i+1}
Write BM's and DV's of S_i
Load BM's of S_{i+2}
Calculate data costs of S_{i+2}

Fig. 18. Two-stage pipeline in the VLSI circuit. Here BM denotes boundary message and DV denotes disparity value.



(a) (b) (c)

Fig. 19. The disparity maps of the dataset "Tsukuba" by the hardware implementation using different numbers of iterations. (a) (T_I, T_O) = (3, 1), (b) (T_I, T_O) = (20, 1), and (c) (T_I, T_O) = (20, 5).

thread index to distinguish different threads for accessing different data or invoking branches.

In the CUDA hardware model, the GPU is split into several *multiprocessors*, where each multiprocessor contains many processors, and an on-chip shared memory. The programmer can explicitly define the usage of the shared memory to minimize the bandwidth consumption. The threads that use the same shared memory are arranged in a single *block*. The threads in different blocks cannot be synchronized; they can only communicate to others by using the off-chip memory.

Therefore, we can activate many different kernels, one for each multiprocessor, or activate many identical kernels, if the threads in different blocks do not have to communicate with others frequently. The number of blocks and threads can be much larger than the available resources of the GPU; the GPU automatically schedules the threads to hide the latency of memory access and maximizes the performance.

We first describe the implementation of BP-M. We arrange the computations for each row/column in a separate block. Because the min-convolution method is strictly sequential, each block contains only one thread. That is, degree of parallelism is $O(W)$, where W is the image width/height, or $\approx O(N^{1/2})$.

On the contrary, our method can be efficiently performed by activating L threads in each block. A thread with index l_q first calculates $H(l_q)$ and then the hypotheses in the set $S_{q,i}$. Since M is small, we use each thread to calculate min_all($S_{q,i}$, $2M + 1$) sequentially. We than use all threads to calculate min_all(S_O, L)

TABLE VI
PROCESSING SPEED OF THE VLSI CIRCUIT

Image size	T_I	T_O	Frames/sec	DE/s[*]
320×240	20	5	7.28	35.78M
	20	1	36.45	178.91M
	3	1	233.23	1146.38M
640×480	20	5	1.82	35.78M
	20	1	9.11	178.91M
	3	1	58.31	1146.38M

[*]*DE/s denotes the number of the disparity estimation per second as in [40].*

TABLE VII
THE EXECUTION TIME OF DISPARITY ESTIMATION FOR THE BENCHMARK "TSUKUBA" (UNIT: MS)

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

TABLE VIII
THE EXECUTION TIME OF DISPARITY ESTIMATION FOR THE BENCHMARK "CONES" (UNIT: MS)

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

in parallel by min-reduction. Finally, each thread generates the message entity $M_{pq}(l)$. The latency of this approach is $O(M + \log L)$, much smaller than $O(L)$ of the min-convolution method.

Tables VII and VIII list the execution times for the benchmarks "Tsukuba" and "Cones", respectively. For fairness, the time required to transfer data from the main memory to the GPU texture memory is included. The platform uses a 3.0 GHz CPU with 2 GB main memory and an NVidia GeForce 8800GTS with 512 MB texture memory. As we can see, the proposed method is about four times faster than the min- convolution method because it can activate more threads. Note that our CUDA programs are not fully optimized. A faster version can be obtained by considering the hardware- and driver-dependent effects [7]. Nevertheless, the ratio of processing speeds between those methods shall be unaffected.

Implementing tile-based BP on current-generation GPU's is nontrivial. The main difficulty is, although the GPU on-chip memory is large, it is explicitly separated into several shared memories, one for each multiprocessor. The data cannot be directly transferred from one shared memory to another. For the platform we use, the maximal tile size B is only 12 when L = 16. The computational power of GPU is not fully utilized with this setting, and hence the reduction of the bandwidth is overwhelmed by the frequent task switches. Therefore, our current implementation is only slightly faster than BP-M. Nevertheless, tile-based BP makes it possible to process large dataset such as the one in Fig. 9. This obstacle shall be removed in the next generation GPU's with multi-level caches [31].

VII. Discussion

As we try to improve the efficiency of the BP algorithm by enforcing locality on processing flow and reducing computational complexity, we notice that the messages of most nodes become stable after a few iterations and do not have to be updated. Thus, subsequent computations can be saved [40]. The execution time of tile-based BP can be further reduced by terminating the processing of a tile after certain stability or convergence criteria are met. The termination criteria is application- and data-dependent.

Moreover, although we only compare tile-based BP with BP-M, the approach can be applied to other message schemes as well. For example, we can combine tile-based BP with hierarchical BP [13] by applying the tiled-based BP in each level (or only at a finer level where the storage of the messages is too large). The messages at the next level can be initialized by using only the boundary messages at the current level. In [25] we have shown that this approach works well for stereo matching. However, finding the optimal parameters of this combination for various applications requires further study.

Tile-based BP requires an on-chip buffer of size $O(B^2L)$ to store the messages and data costs of the tile. To further reduce the memory requirement for cost-sensitive devices, complementary methods can be incorporated [38], [43].

Finally, it should be pointed out that the simple energy function we used in the stereo matching described in Section VI is only for illustration purpose. A high-performance stereo matching system usually combines several independent algorithms [39]. Nevertheless, tile-based BP is readily applicable to reduce the memory cost and increase scalability of the system.

VIII. Conclusion

In this paper, we have developed tile-based message passing and parallel message construction algorithms. These techniques greatly reduce the memory, bandwidth, and computational costs of BP and enable the parallel processing. With these two techniques, belief propagation becomes more suitable for low-cost and power-limited consumer electronics. We have demonstrated the applicability of the proposed techniques for various applications in the Middlebury MRF benchmark. A VLSI circuit and a GPU program for stereo matching based on the proposed techniques have been developed. These techniques can also be applied to other parallel platforms.

Acknowledgement

We thank the anonymous reviewers for insightful comments, which greatly help improve the quality of this paper.

References

[1] A. Adams, N. Gelfand, K. Pulli, "Viewfinder alignment," *Computer Graphics Forum*, vol. 27, no. 2, pp. 597-606, 2008.

[2] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen, "Interactive digital photomontage," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 294-302, Aug. 2004.

[3] J. Besag, "On the statistical analysis of dirty pictures (with discussion)," *J. Royal Statistical Soc.*, Series B, vol. 48, no. 3, pp.259-302, 1986.

[4] S. Birchfield and C. Tomasi, "A pixel dissimilarity measure that is insensitive to image sampling," *IEEE Trans. PAMI*, vol. 20, no. 4, pp. 401-406, Apr. 1998.

[5] M. J. Black and A. Rangarajan, "On the unification of line processes, outlier rejection, and robust statistics with applications in early vision," *IJCV*, vol. 19, no. 1, pp. 57-91, July 1996.

[6] M. Black, G. Sapiro, D. H. Marimont, and D. Heeger, "Robust anisotropic diffusion," *IEEE Trans. IP*, vol. 7, no. 3, pp. 412-432, March 1998.

[7] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder, "Sparse matrix solvers on the GPU: conjugate gradients and multigrid," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 917-924, Jul. 2003.

[8] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. PAMI*, vol. 23, no. 11, pp. 1222-1239, Nov. 2001.

[9] A. Brunton, C. Shu, and G. Roth, "Belief propagation on the GPU for stereo vision," in *Proc. CRV'06*, pp.76-81, 2006.

[10] 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*, vol. 16, no. 6, pp. 673-688, June 2006.

[11] A. Criminisi, A. Blake, and C. Rother, "Efficient dense stereo with occlusions for new view-synthesis by four-state dynamic programming," *IJCV*, vol. 71, no. 1, pp. 89-110, Jan. 2007.

[12] J. Díaz, E. Ros, R. Carrillo, and A. Prieto, "Real-time system for high-image resolution disparity estimation," *IEEE Trans. IP*, vol. 16, no. 1, pp. 280-285, Jan. 2007.

[13] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *IJCV*, vol. 70, no. 1, pp. 41-54, Oct. 2006.

[14] W. T. Freeman, E. C. Pasztor, and O. T. Carmichael, "Learning low-level vision," *IJCV*, vol. 40, no. 1, pp. 25-47, Oct. 2000.

[15] B. J. Frey and D. MacKay, "A revolution: belief propagation in graphs with cycles," In *Proc. NIPS*, pp. 479-485, 1998.

[16] M. Gong, R. Yang, L. Wang, and M. Gong, "A performance study on different cost aggregation approaches used in real-time stereo matching," *IJCV*, vol. 75, no. 2, pp. 283-296, Nov. 2007.

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

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

[19] B. Khailany, W. J. Dally, U. J. Kapasi, P. Mattson, J. Namkoong, J. D. Owens, B Towles, A. Chang, and S. Rixner, "Imagine: media processing with streams," *IEEE Micro*, vol. 21, no. 2, pp. 35-46, March 2001.

[20] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE Trans. PAMI*, vol. 28, no. 10, pp. 1568-1583, Oct. 2006.

[21] N. Komodakis and G. Tziritas, "Image completion using efficient belief propagation via priority scheduling and dynamic pruning," *IEEE Trans. IP*, vol. 16, no. 11, pp. 2649-2661, Nov. 2007.

[22] M. P. Kumar and P.H.S. Torr, "Fast memory-efficient generalized belief propagation," In *Proc. ECCV*, pp. 451-463, 2006.

[23] S. Li, *Markov Random Field Modeling in Computer Vision*. Springer, 1995.

[24] C.K. Liang, T.S Lin, B.Y. Wong, C. Liu, and H. H. Chen, "Programmable aperture photography: multiplexed light field acquisition," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 55:1-55:10, Aug. 2008.

[25] C.K. Liang, C.C. Cheng, Y.C. Lai, L.G. Chen, and H. H. Chen, "Hardware-efficient belief propagation," in *Proc. CVPR*, pp. 80-87, 2009.

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

[27] S. C. Park, C. Chen, and H. Jeong, "VLSI architecture for MRF based stereo matching," in *LNCS*, vol. 4599, pp. 55-64, 2007.
[28] C. Rother, V. Kolmogorov, and A. Blake, "'GrabCut' − interactive foreground extraction using iterated graph cuts," *ACM Trans. Graph.*, vol. 23, no. 3, pp. 277-286, Aug. 2004.
[29] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *IJCV*, 47, no. 1-3, pp. 7-42, Apr. 2002.
[30] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," In *Proc. CVPR*, vol. 1, pp. 195-202, 2003.
[31] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugerman, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan, "Larrabee: a many-core x86 architecture for visual computing," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 1-15, Aug. 2008.
[32] J. Sun, N.N. Zhang, and H.Y. Shum, "Stereo matching using belief propagation," *IEEE Trans. PAMI*, vol. 25, no. 7, pp. 787-800, Jul. 2003.
[33] 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*, vol. 30, no. 6, pp. 1068-1080, June 2008.
[34] M. F. Tappen and W. T. Freeman, "Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters," In *Proc. ICCV*, vol. 2, pp. 900-907, 2003.
[35] Y.C. Tseng, N. Chang, and T.S. Chang, "Low memory cost block-based belief propagation for stereo correspondence," In *Proc. ICME*, pp. 1415-1418, 2007.
[36] 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*, vol. 2, no. 1, pp. 61-72, Jan. 2002.
[37] V. Vineet and P. J. Narayanan, "CUDA cuts: fast graph cuts on the GPU," In *CVPR Workshop on Visual Computer Vision on GPU's*, 2008.
[38] L. Wang, H. Jin, and R. Yang, "Search space reduction for MRF stereo," In *Proc. ECCV*, 2008.
[39] Q. Yang, L. Wang, R. Yang, H. Stewénius, D. Nistér, "Stereo matching with color-weighted correlation, hierarchical belief propagation, and occlusion handling," *IEEE Trans. PAMI*, vol. 31, no. 3, pp. 492-504, Mar. 2009.
[40] 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.
[41] J.S. Yedidia, W.T. Freeman, and Y. Weiss, "Generalized belief propagation," In *Proc. NIPS*, pp. 689-695, 1998.
[42] K.J. Yoon and I.S. Kweon, "Locally adaptive support-weight approach for visual correspondence search," In *Proc. CVPR*, pp. 924-931, 2005.
[43] T. Yu, R.S. Lin, B. Super, and B. Tang, "Efficient message representations for belief propagation," In *Proc. ICCV*, pp. 1-8, Oct. 2007.
[44] Project website, http://mpac.ee.ntu.edu.tw/~chiakai/hebp

Chia-Kai Liang received the B.S. degree in Electrical Engineering and Ph.D. degree in Communication Engineering in 2004 and 2009 respectively, both from National Taiwan University, Taipei, Taiwan. From 2006 to 2007, he was a VLSI design engineer at Avisonic Corp. In 2009, he was a research intern in Nokia research center, Palo Alto. His current research interests are image processing, computer vision, computational photography, and multimedia hardware architecture. He received IEEE CSVT Transactions Best Paper Award in 2008, IPPR Best Dissertation Award in 2009, and NTU GICE Best Dissertation Award in 2009. He is a member of Phi-Tau-Phi, IEEE, and ACM SIGGRAPH.

Chao-Chung Cheng received the B.S. and M.S. degrees in Electronics Engineering from National Chiao-Tung University, Hsinchu, Taiwan, R.O.C., in 2003 and 2005, respectively. He is currently a Ph.D. student of Graduate Institute of Electronics Engineering in National Taiwan University. His research interests include video coding system design, stereo vision, and 2D-to-3D Conversion.

Yen-Chieh Lai received the B.S. degree in Electrical Engineering in 2009 from National Taiwan University, Taipei, Taiwan. Currently he is a graduate student of the Graduate Institute of Electronics Engineering in National Taiwan University. His research interest is 3D signal processing.

Liang-Gee Chen received the B.S., M.S., and Ph.D. degrees in Electrical Engineering from National Cheng Kung University in 1979, 1981, and 1986, respectively. In 1988, he joined the Department of Electrical Engineering, National Taiwan University. Currently, he is the Distinguished Professor of Department of Electrical Engineering and the Deputy Dean of office of Research and Development in National Taiwan University. Since 2007, he also serves as a Co-Director General of National SoC Program. He is the IEEE Fellow from 2001. His research interests include DSP architecture design, video processor design, and video coding systems. He has published over 400 papers and 30 patents. Dr. Chen has served as an Associate Editor of IEEE Transactions on Circuits and Systems for Video Technology and other international technical journals. He is also involved several IEEE technical committees, including the TPC Chair of 2009 IEEE ICASSP and the TPC chair of ISCAS 2012. He has received several outstanding research awards and outstanding industrial technology contribution awards from NSC. His group has won the DAC/ISSCC Student Design Contest for four times since 2004, and had the honor of Student Paper Contest at ICASSP 2006.

Homer H. Chen (S'83-M'86-SM'01-F'03) received the Ph.D. degree in Electrical and Computer Engineering from University of Illinois at Urbana- Champaign, Urbana.

Since August 2003, he has been with the College of Electrical Engineering and Computer Science, National Taiwan University, Taiwan, R.O.C., where he is Irving T. Ho Chair Professor. Prior to that, he held various R&D management and engineering positions with US companies over a period of 17 years, including AT&T Bell Labs, Rockwell Science Center, iVast, and Digital Island. He was a US delegate for ISO and ITU standards committees and contributed to the development of many new interactive multimedia technologies that are now part of the MPEG-4 and JPEG-2000 standards. His professional interests lie in the broad area of multimedia signal processing and communications.

Dr. Chen is an Associate Editor of IEEE Transactions on Circuits and Systems for Video Technology. He served as Associate Editor of IEEE Transactions on Image Processing from 1992 to 1994, Guest Editor of IEEE Transactions on Circuits and Systems for Video Technology in 1999, and an Associate Editorial of Pattern Recognition from 1989 to 1999.