HARDWARE-EFFICIENT STEREO ESTIMATION USING A RESIDUAL-BASED APPROACH

Abhishek A. Sharma[1], *Kaustubh Neelathalli*[1], *Diana Marculescu*[1], *Eriko Nurvitadhi*[2]

Carnegie Mellon University[1],
Intel Science and Technology Center on Embedded Computing[2]

ABSTRACT

Many promising embedded computer vision applications, such as stereo estimation, rely on inference computation on Markov Random Fields (MRFs). Sequential Tree-Reweighted Message passing (TRW-S) is a superior MRF solving method, which provides better convergence and energy than others (e.g., belief propagation). Since software TRW-S solvers are slow, custom TRW-S hardware has been proposed to improve execution efficiency. This paper proposes hardware mechanisms to further optimize TRW-S hardware efficiency, by tracking differences in input message values (residues) and skipping computation when values no longer change (residue is zero). Evaluations of our hardware mechanisms using Middlebury benchmark show 1.6x to 6x potential reduction in computation (depending on design parameters) while increasing energy by only 0.4% to 4.8%.

Index Terms— Hardware optimization, stereo estimation, Markov Random Fields

1. INTRODUCTION

Many computer vision applications, such as 3D stereo estimation, rely on inference computation on Markov Random Fields (MRFs) formulated as graphs, where computation is done through message passing over the nodes of the graph. These applications promise disruptive new capabilities for embedded systems. For example, smart phones, security cameras, and even glasses [1] that can view the world in 3D would open up new usage scenarios and market opportunities.

Among the various MRF solving methods, Sequential Tree-Reweighted Message passing (TRW-S) has been shown [2] to provide reliable convergence and yield better final energy than others (e.g., belief propagation). However, conventional TRW-S software solvers are slow due to the high computational demand of the algorithm. To this end, prior work has proposed a custom TRW-S hardware solution [3] to improve execution efficiency, thereby making it more amenable for embedded implementation.

This paper proposes a technique to further optimize TRW-S hardware efficiency. The technique works by tracking differences in input messages (residue) to identify converging nodes whose inputs no longer change (residue is zero). Once such nodes are identified, the computation on these nodes can be skipped safely without sacrificing output quality. However, identifying node convergence perfectly is impossible without observing full execution traces, thereby necessitating novel hardware mechanisms to predict convergence and to utilize the prediction to avoid unnecessary computation on already converging nodes.

Our contributions are as follows. First, using the standard Middlebury benchmark [2], we studied the opportunity of the proposed optimization technique on an ideal system and demonstrate potential reduction in computation by 7.2x on average. Second, we propose hardware mechanisms to predict convergence and skip computation on nodes that have been predicted to converge. Third, we evaluate the effectiveness of the proposed hardware mechanisms, and show that they can reduce computation by 1.6x to 6x (depending on design parameters), while sacrificing only 0.4% to 4.8% in energy.

The rest of the paper is organized as follows. Section 2 reviews the TRW-S algorithm. Section 3 presents the opportunity study, which quantifies the potential benefit of the proposed technique under a perfect system. Section 4 elaborates on the proposed method for predicting convergence. Section 5 discusses the hardware mechanisms needed to support the proposed technique. Evaluations of the effectiveness of these mechanisms are presented in Section 6. Finally, Section 7 concludes.

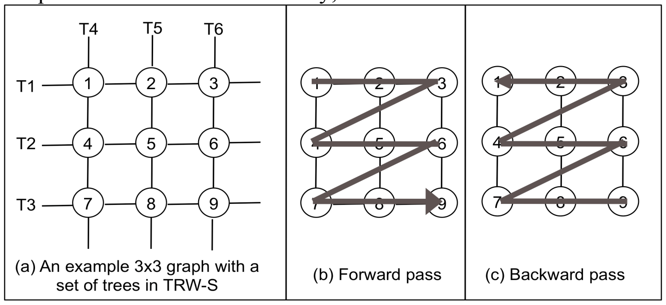


Figure 1: TRW-S go over a set of trees (T1 to T6) in a monotonically increasing order, through forward/backward passes.

2. BACKGROUND ON TRW-S

Many computer vision applications involve assigning labels optimally to nodes in a graph that represent an image. For example, in stereo estimation, the nodes represent pixels from a pair of stereo images, and the labels denote the 3D depth inferred from the image pair. The optimal labeling problem is typically formulated as an energy minimization on Markov Random Field [2]. Among various MRF solving methods, TRW-S [6][7] is known to provide better convergence and energy than others [2].

In TRW-S, the energy minimization problem is cast as a set of minimization problems on trees that cover the graph. Figure 1(a) shows an example MRF graph with the associated set of trees, T1 to T6, representing a 3x3 pixel image. The TRW-S computation follows a sequence of update functions applied to the nodes in the trees in a monotonically increasing order. This "sequential" update order impacts convergence as it avoids oscillating energy value. As a specific example, T1 tree in Figure 1 would update node 1 and sends the output message to node 2, then from node 2 to 3. For T4 tree, the order is node 1, 4, and 7. To achieve monotonically increasing order for all the trees in the graph, the typical approach is to iterate over the graph by passing messages from top-left to bottom-right (forward pass), and then in the opposite direction (backward pass), as illustrated in Figure 1(b) and 1(c).

3. RESIDUAL-BASED APPROACH

The proposed residual-based approach relies on the observation that the function to update the nodes in TRW-S trees is stateless. That is, if the inputs to the function do not change, the outputs will remain the same. Therefore, if the input message values to a node no longer change, there is no need to compute the update function for that node anymore.

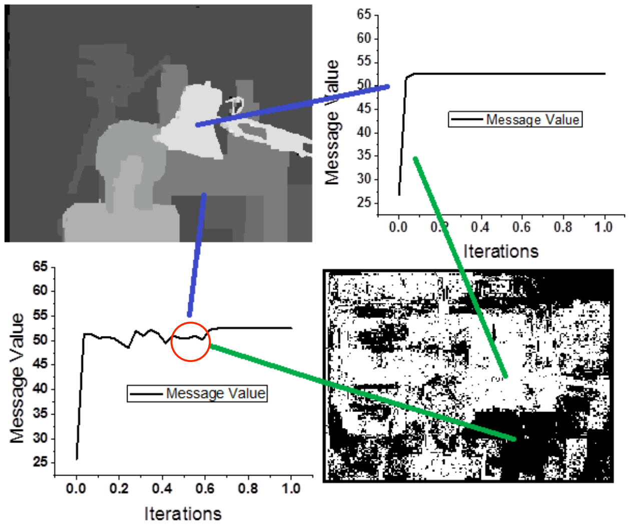


Figure 2: Plots of message values at two nodes in the true disparity map for Tsukuba image from Middleburry benchmark [4][5]. The X-axis is normalized to 1 with respect to the total number of iterations needed for all nodes in the graph to converge. The bottom right shows a convergence-map for the nodes in the image. The white nodes are those that have already converged after the first ~30% of the total number of iterations.

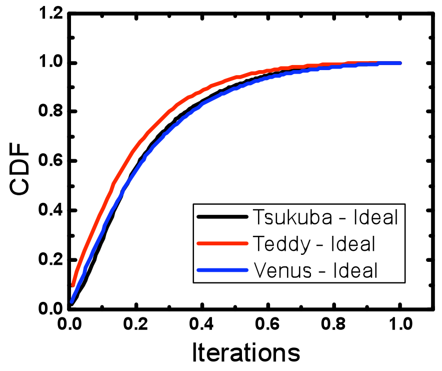


Figure 3: Cumulative distribution function (CDF) of node convergence for the three Middlebury images. For each image, we run TRW-S iterations until the entire graph converges, and normalize the iteration number to 1.

To this end, we define the *residue* as the difference between the message values of a particular node across two consecutive iterations. We further define a node to have reached *convergence* if the residue has reached zero, and no longer changes. We propose tracking node residues and identifying node convergence, in order to save computation by not updating these already converging nodes. Similar residual-based optimization has been proposed before [8], however it was in the context of belief propagation.

Figure 2 illustrates the opportunity of applying residual-based optimization in the context of TRW-S for stereo estimation. The figure includes plots of message values from two nodes in the Tsukuba image from the Middlebury benchmark [4][5] over a number of iterations necessary for all the nodes in the entire graph to converge. The number of iterations is normalized to 1 (i.e., 0 and 1 is the first and last iteration, respectively). One node corresponds to a pixel in a flat (low-frequency) image region (i.e., the lamp), while another node belongs to a pixel at an edge (high-frequency) region in the image. As the plots show, both nodes converge within the first ~65% of the total number of iterations, with the node from the non-varying (lamp) area of the image converges much quicker (at ~10% of the total iterations) than the edge node (at ~65%). The figure also shows a black-white snapshot at the first 30% of execution, where the white area represents nodes that have already converged. As shown, a significant amount of nodes are white.

Figure 3 depicts cumulative distribution function (CDF) of node convergence for the three Middlebury benchmark images. On average, 86% of nodes have already converged in the first 40% of the total number of iterations, and 99% of nodes converge within the first 80% of the total number of iterations. This presents a tremendous opportunity to reduce computation by skipping the update function for these already converging nodes. For the three Middlebury benchmark images, if we skip the update on the converging nodes, the total number of updates is on average 7.2x smaller than the total updates if we did not do any skipping.

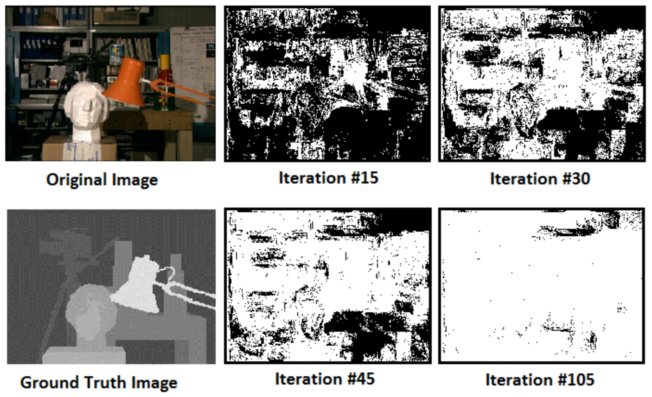


Figure 4: Convergence-maps from different iterations of Tsukuba. White regions represent converged nodes.

4. PREDICTING NODE CONVERGENCE

In practice, it is non-trivial to know when a node has converged. Without knowing the input message values for all the iterations needed for the graph to converge, it is impossible to identify the point of convergence for a given node. This is because at some iterations, message values may not change. However, at later iterations, they may change again. The bottom-left graph in Figure 2 illustrates this situation. At the first 40% to 50% of execution, the message value remains constant (indicated by a red circle in the figure). Following that, the message value changes again, before it reaches convergence at ~65% of execution.

Since it is impossible to know for sure when a node has converged without looking ahead, we propose making a prediction instead. If our prediction is correct, then we can safely skip the node that has been predicted to converge. The downfall occurs if we predict incorrectly. At this point, we may sacrifice the quality of the TRW-S output by skipping nodes that have not actually converged. However, if the prediction is accurate enough, the degradation in quality can still be acceptable.

4.1. Predictor Design

The most straightforward way to make a prediction is based on history, or *temporal* behavior. If over X number of iterations, message value remains the same, we may predict that the node has converged. However, we found that the ideal value of X varies widely among different nodes. A node with long intermittent periods of constant message values (as mentioned above) would require a large X to provide good prediction accuracy. However, the larger the X, the more we miss the opportunity to skip computations since we have to wait for X iterations before any prediction can be made.

Therefore, instead of relying on temporal behavior, we propose a predictor based on *spatial* information. That is, we observe that nodes in an image area with edges (high frequency) converge at a much later iteration than the nodes at the flat image regions (low frequency). Further, flat image regions tend to converge *together*. This behavior is shown in Figure 4, which shows the original Tsukuba image along with snapshots from different iterations. Therefore, instead of predicting a single node, we propose predicting convergence at a group (or tile) level.

```
PredictConvergence($G^k$, $m_{sum}^{k-1}$)
    $m_{sum}^{k}$ = sum_msgs($G^k$)
    $residue = m_{sum}^{k} - m_{sum}^{k-1}$
    return($residue$ == 0)
```

Figure 5: Pseudo-code for the convergence prediction algorithm

Figure 5 shows the pseudo-code of our prediction. At iteration k, given a group of nodes (G) and the sum of the messages of G from the previous iteration (m_{sum}^{k-1}), the algorithm returns true if it predicts that all the nodes in G has converged. Internally, the algorithm calculates the sum of messages in G for the current iteration (m_{sum}^{k}), and compares it with the sum from the previous iteration (m_{sum}^{k-1}) to calculate the residue. If the *residue* is zero, then the prediction returns true to indicate convergence. Once a group has been predicted to converge, then the updates for all the nodes in the group can be skipped entirely.

4.2. Predictor Performance

The prediction accuracy of our predictor for varying group sizes, where a group is a square tile of *NxN* size is shown in Figure 6(a). The accuracy is in terms of pixel misprediction, indicating the percentage of incorrect predictions. For example, a 5% misprediction means that 5% out of the total predictions were made incorrectly (i.e., prediction says that a node has converged, while in actuality the node has not yet converged). As expected, larger group size results in better prediction accuracy (i.e., lower misprediction). And, a node-level prediction (1x1 group size) yields a much worse accuracy than a group-level prediction.

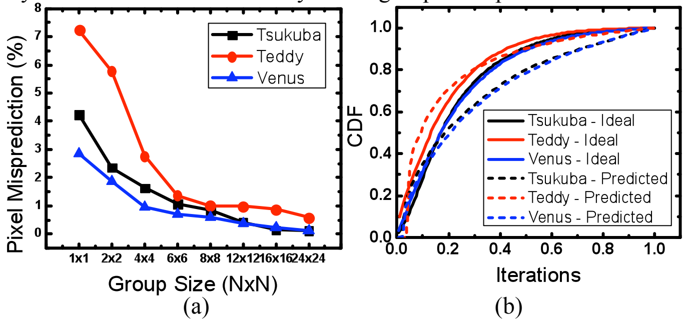


Figure 6: (a) Predictor accuracy for varying group sizes. (b) Cumulative distribution function (CDF) of node convergence for the three Middlebury images. Solid lines indicate CDF calculated based on predicted convergence, and dashed lines indicate CDF from actual convergence (taken from Figure 3).

Figure 6(b) shows the CDF of node convergence. The solid lines show CDF calculated based on predicted convergence, while dashed lines are CDF based on actual convergence (taken from Figure 3). For Tsukuba and Venus, our predictor requires only a short time to make prediction (i.e., their CDF lines lagging only slightly behind the actual CDF lines). For Teddy, our predictor (inaccurately) makes prediction slightly too early. In overall, our predictor performs quite well.

5. PROPOSED HARDWARE

5.1. Baseline Hardware

To the best of our knowledge, the hardware-based TRW-S system proposed in [3] provides the best results (i.e., from using TRW-S algorithm) as well as performance compared to other prior work that accelerate energy minimization algorithms [9]-[12]. Therefore, we use this work as our baseline design.

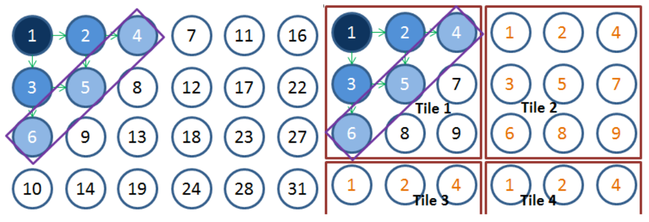
The system proposed in [3] utilizes a FPGA-based platform, where the graph data is streamed to an FPGA that implements a custom hardware processing-element to perform the TRW-S update function in a pipelined fashion. Since TRW-S imposes sequential ordering (see Figure 1), data dependencies make parallelized implementation non-trivial. To address this, the TRW-S hardware adopts diagonal-style processing shown in Figure 7(a).

In this example, the diagonal processing style goes through node {1}, then nodes {2,3}, followed by nodes {4, 5, 6}, and so on. Notice that nodes in a given diagonal stripe (e.g., {2,3}, {4,5,6}) are independent of each other and can be processed in parallel. Such diagonal processing maintains monotonically increasing order of updates, while exposing parallelization opportunity for the nodes within each diagonal stripe.

5.2. Proposed Mechanism for Residual Approach

5.2.1. Prediction

We use a simple method to implement the prediction algorithm described previously in Section 4. For each group of nodes, we carry the sum of message values from the previous iteration (i.e., m_{sum}^{k-1} in the pseudo-code in Figure 5). Thus, we only add very little storage (i.e., one value per group). To calculate the sum of messages, TRW-S already calculates the sum of its input messages in the first part of its update function [2]. Therefore, we can piggyback on this computation. The only additional operations needed are to sum the labels together and to calculate the residue (a single subtraction), which are relatively trivial compared to the rest of the update function.



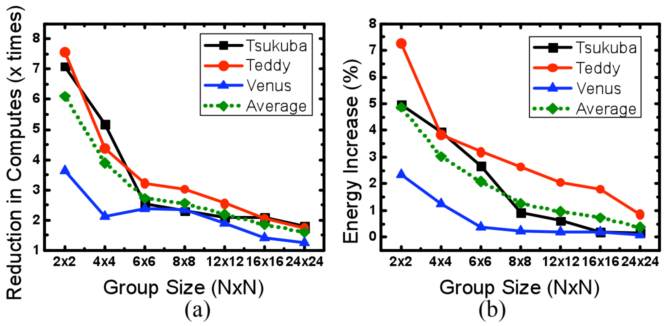
(a) (b)

Figure 7: (a) Baseline node-level diagonal processing from [3]. (b) The proposed tile-level approach, which performs diagonal processing across nodes (within a tile) as well as tiles.

5.2.2. Scheduling

Unlike the baseline TRW-S hardware, our predictor favors a group/tile level processing. Therefore, we propose breaking the graph into tiles (Figure 7(b)), and perform tile-level processing. Such a tile-level approach can still take advantage of diagonal style processing. First, for a given tile, we can still process the nodes inside the tile diagonally, therefore benefiting from the same type of parallelism as in the baseline TRW-S hardware. Second, we can process the tiles themselves diagonally as well. In Figure 7(b) example, we first process Tile 1, then Tile 2 and 3 in parallel, and so on. Notice that with tile-level processing, we still maintain the monotonically increasing order of updates required by TRW-S.

The tile processing also helps ease scheduling data transfers between system memory (e.g., DDR) and FPGA, since we can utilize bulk transfers, which is not the case if we had to skip data transfers for individual nodes.



(a) (b)

Figure 8: (a) Reduction in computations and (b) increase in energy from the proposed residual-based TRW-S optimization approach, with respect to the original TRW-S algorithm.

6. EVALUATION

Here, we evaluate the cost-benefit trade-off of the proposed residual-based approach, for various prediction group sizes. We modified the Middlebury benchmark code to include our prediction mechanism and to skip update functions when a group has been predicted to converge. We use default Middlebury TRW-S parameters for the characterization [2].

Figure 8 shows (a) the reduction in computation and (b) the increase in energy for varying prediction group sizes, relative to baseline TRW-S without residual tracking. The figure shows the results for the three Middlebury benchmark input images, as well as the average among them. Tsukuba and Teddy have higher potential reduction in computation than Venus, while also suffering from higher increase in energy. This is because these two images have more features than Venus, which is relatively flat, which leads to higher variation in terms of node convergence. i.e., some nodes at the feature edges require more iterations to converge than other nodes at flat image regions. Therefore, there is more opportunity for skipping early-converging nodes. In overall, the residual-based approach only incurs 0.4% to 4.8% average increase in energy for the group sizes under study, while providing 1.6x to 6x average reduction in computation.

7. CONCLUSION

This paper proposes a technique to improve the efficiency of Sequential Tree-Reweighted Message passing (TRW-S) algorithm implementation in hardware. TRW-S is a superior MRF solving method that is widely used in computer vision applications, such as stereo estimation. Our evaluations using Middlebury benchmark show that the technique has the potential to reduce computation by 7.2x if we assume a perfect system. Furthermore, we show that the proposed hardware mechanisms can reduce computation by 1.6x to 6x, while exacerbating energy by only 0.4% to 4.8%.

8. ACKNOWLEDGMENTS

This research has been funded (in part) by the Intel Science and Technology Center on Embedded Computing.

9. REFERENCES

[1] Google Glass, URL: https://plus.google.com/+projectglass

[2] R. Szeliski, et al., "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," IEEE Trans. on Pattern Analysis and Machine Intelligence, 2008.

[3] J. Choi and R. Rutenbar, "Hardware implementation of MRF map inference on an FPGA platform," Field Programmable Logic and Applications, 2012.

[4] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," International Journal of Computer Vision, 2002.

[5] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003.

[6] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," IEEE Transaction on Pattern Analysis and Machine Intelligence, 2006.

[7] M. J. Wainwright, et al., "MAP estimation via agreement on trees: message-passing and linear-programming approaches," IEEE Transactions on Information Theory, 2005.

[8] G. Elidan, I. McGraw, and D. Koller, "Residual belief propagation: informed scheduling for asynchronous message passing," Uncertainty in Artificial Intelligence, 2006.

[9] L. Zhou, et al., "Accelerated Belief Propagation for hardware implementation," Int. Conference on Multimedia Technology, 2011.

[10] A. Brunton, C. Shu, and G. Roth, "Belief propagation on the GPU for stereo vision," Computer and Robot Vision, 2006.

[11] M. P. Kumar and P.H.S. Torr, "Fast memory-efficient generalized belief propagation," European Conf. on Computer Vision, 2006.

[12] C. Liang, et al., "Hardware-Efficient Belief Propagation," IEEE Transactions on Circuits and Systems for Video Technology, 2011.