Appears in The 25th Intl. Conf. on Field Programmable Logic and Applications (FPL), 2015

Fast Hierarchical Implementation of Sequential Tree-reweighted Belief Propagation for Probabilistic Inference

Skand Hurkat* Jungwook Choi† Eriko Nurvitadhi‡ José F. Martínez* Rob A. Rutenbar§

*Computer Systems Laboratory
Cornell University
Ithaca, NY 14853 USA

†Dept. of Electrical and Computer Engineering
§Dept. of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801 USA

‡Intel Labs
Hillsboro, OR 97124 USA

***Abstract*—Maximum a posteriori probability (MAP) inference on Markov random fields (MRF) is the basis of many computer vision applications. Sequential tree-reweighted belief propagation (TRW-S) has been shown to provide very good inference quality and strong convergence properties. However, software TRW-S solvers are slow due to the algorithm's high computational requirements. A state-of-the-art FPGA implementation has been developed recently, which delivers substantial speedup over software.**

In this paper, we improve upon the TRW-S algorithm by using a multi-level hierarchical MRF formulation. We demonstrate the benefits of Hierarchical-TRW-S over TRW-S, and incorporate the proposed improvements on a Convey HC-1 CPU-FPGA hybrid platform. Results using four Middlebury stereo vision benchmarks show a 21% to 53% reduction in inference time compared with the state-of-the-art TRW-S FPGA implementation. To the best of our knowledge, this is the fastest hardware implementation of TRW-S reported so far.

I. Introduction

Many applications in computer vision rely on maximum a posteriori probability (MAP) inference computation on Markov Random Fields (MRF). Among the available methods for MAP inference, prior work [6, 13] has shown that the sequential tree-reweighted belief propagation (TRW-S) algorithm can provide good inference with reliable convergence properties.

However, software TRW-S implementations are typically slow because the algorithm is computationally intensive and challenging to parallelize due to its inherent sequential nature. Choi and Rutenbar [2, 3] recently implement a hardware TRW-S solver on a Convey HC-1 system. Their implementation achieves significantly faster inference compared to a software implementation.

In this paper, we apply an algorithmic optimization to TRW-S that speeds up its rate of convergence, resulting in improved performance. Our technique is based on formulation of the MRF graph as a multi-level hierarchy, along the lines of hierarchical belief propagation proposed by Felzenszwalb and Huttenlocher, [5] which helps reduce the amount of computation involved while still providing good results. Our work explores the applicability and effectiveness of the hierarchical approach for TRW-S in the context of a hardware implementation.

The contributions of this work are twofold. First, we describe the application of a hierarchical approach to TRW-S and evaluate its benefits. To this end, we extend the hierarchical approach described by Felzenszwalb and Huttenlocher [5] to accommodate contrast-based spatially varying discontinuity costs, which results in better accuracy. We study the effect of tuning various parameters (block size, number of levels, and iterations per level) for an implementation of our proposed Hierarchical-TRW-S algorithm. Second, we implement Hierarchical-TRW-S on a Convey HC-1 CPU-FPGA hybrid platform. We experiment with and create efficient hardware to manage the operations and amortize the overhead involved.

Experiments using four Middlebury stereo vision benchmarks [12, 13] show that Hierarchical-TRW-S reduces inference time by 21% to 53% compared with Choi and Rutenbar's hardware TRW-S implementation. [2] To the best of our knowledge, this is the fastest hardware implementation of TRW-S reported so far.

This paper is organized as follows. In Section II, we define the energy minimization problem used for inference, and introduce notation that will be used throughout the paper. We also discuss the hierarchical belief propagation algorithm proposed by Felzenszwalb and Huttenlocher, [5] and the sequential tree-reweighted belief propagation algorithm by Kolmogorov. [6] In Section III, we describe algorithmically a Hierarchical-TRW-S implementation. In Section IV, we explain our hardware Hierarchical-TRW-S implementation. We discuss results from running Hierarchical-TRW-S on a Convey HC-1 CPU-FPGA hybrid platform in Section V. Finally, we discuss pertinent related work in Section VI.

II. Background

In this section, we describe the MRF energy minimization problem, the tree-reweighted belief propagation algorithm, and we introduce notation that will be used through the paper.

A. The MRF Energy Minimization Problem

Many problems in early vision and other applications can be formulated as an energy minimization task over a graph. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph defined by vertices $\mathcal{V}$ and edges $\mathcal{E}$. Further, let energy potentials be defined for vertices and edges, $\theta_p \ \forall \ p \in \mathcal{V}$ (also referred to as data costs) and $\theta_{(p,q)} \ \forall \ (p,q) \in \mathcal{E}$ (also referred to as discontinuity costs or smoothness costs). The MRF energy minimization problem is the task of assigning

labels from a label set $\mathcal{L}$ to the vertices $\mathcal{V}$ in order to minimize an energy function defined in the following manner:

$$E(\boldsymbol{l}|\boldsymbol{\theta}) = \sum_{p\in\mathcal{V}} \theta_p(l_p) + \sum_{(p,q)\in\mathcal{E}} \theta_{(p,q)}(l_p, l_q) + \theta_{\text{const}} \quad (1)$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is the parameter/cost vector as a concatenation of data costs, discontinuity costs, and θ_{const}; $l_s \in \mathcal{L}$ is the label assigned to $s \in \mathcal{V}$; and $\boldsymbol{l} \in \mathcal{L}^{|\mathcal{V}|}$ is a vector of labels assigned to all vertices in $\mathcal{V}$.

Loopy belief propagation (BP), along with methods like graph-cuts and tree-reweighted belief propagation, have been shown to provide *good*, approximate solutions to the energy minimization problem on graphs with loops. [6, 7, 15]

While BP based methods are popular because they adapt easily to parallel execution, a major drawback is the large number of iterations required to reach convergence. Hierarchical BP, proposed by Felzenszwalb and Huttenlocher, [5] enables convergence to be reached in a small number of iterations for grid graphs typically used in image processing tasks. Hierarchical BP borrows from the concept of image pyramids, grouping $\epsilon \times \epsilon$ nodes from a level into one node on the graph at a higher level. This process is repeated to form additional levels in the hierarchy, followed by belief propagation at the highest level. Messages from higher levels are then copied to lower levels to serve as a starting *guess* reasonably close to convergence (as nearby pixels in an image are expected to be strongly correlated). As a result, fewer iterations are needed at any level. Additionally, the overhead due to the multi-level nature of the graph is bounded. [5]

B. Tree-reweighted Belief Propagation

Tree-reweighted belief propagation (TRW) was introduced by Wainwright et al. [14] for graphs with cycles. A sequential version of this algorithm (TRW-S) was proposed by Kolmogorov. [6] Both methods work by decomposing a graph with loops into a convex combination of trees, and performing modified BP on the tree decomposition.

TRW-S has some advantages over BP: it provides a lower bound for the energy function, which is guaranteed to be non-decreasing; and a *weak tree agreement (WTA)* over the tree decomposition guarantees a local optimum for the energy function. A comparative study of energy minimization techniques by Szeliski et al. [13] shows that the TRW-S algorithm produces one of the best results for a stereo matching task.

Our work applies a hierarchical approach to TRW-S to achieve faster convergence. Further, we generalize the hierarchical approach to the case when discontinuity costs vary by a contrast-based scaling factor over the grid graph, as is common in early-vision tasks such as stereo matching.

III. Hierarchical-TRW-S

In this section, we describe our proposed Hierarchical-TRW-S algorithm, and select a set of parameters that provide good convergence and are amenable to a hardware implementation.

A. Description of Our Algorithm

In order to adapt the hierarchical approach to the TRW-S algorithm, we group $\epsilon \times \epsilon$ nodes at the lower level into a node at the higher level. We add up data costs at the nodes on the lower level to get the data cost for the corresponding node at the higher level.

The Middlebury reference software [13] uses a contrast-based scaling factor for discontinuity costs (referred to as *gradient cues*) in order to improve the performance of the stereo matching algorithm. These are multiplicative penalties $w_{(p,q)}$ applied to the global finite-discontinuity cost function θ_V, as shown in Equation 2a, to obtain the local discontinuity cost functions $\theta_{(p,q)}$. Felzenszwalb and Huttenlocher [5] describe a method to scale the finite-discontinuity cost function (between levels L and $L+1$) as shown in Equation 2b. We modify this to incorporate gradient cues as described in Equation 2c, where n_p, n_q refer to the nodes at the higher level in the hierarchy. We can see that in the case when we don't use gradient cues ($w_{(p,q)} = 1 \ \forall \ (p,q) \in \mathcal{E}$), our method degenerates to the one proposed by Felzenszwalb and Huttenlocher.

$$\theta^L_{(p,q)}(l_p, l_q) = w^L_{(p,q)} \times \theta^L_V(l_p - l_q) \quad (2a)$$

$$\theta^{L+1}_V(l_{n_p} - l_{n_q}) = \epsilon \times \theta^L_V\left(\frac{l_p - l_q}{\epsilon}\right) \quad (2b)$$

$$w^{L+1}_{(n_p,n_q)} = \frac{1}{\epsilon} \times \sum_{\{(p,q)\in\mathcal{E}: p\in n_p, q\in n_q\}} w^L_{(p,q)} \quad (2c)$$

We perform TRW-S on the higher level, and then copy the messages from the higher level down to the lower level. This process could be repeated multiple times to create more levels in the hierarchy.

We note that Kolmogorov's work [6] does not depend on the initial value of messages. As the hierarchical approach merely sets the initial value of messages (to a *guess* reasonably close to the converged values—see Section II-A), it does not interfere with the convergence properties of the TRW-S algorithm on the lowest level of the hierarchy, but it expedites the TRW-S algorithm on the lowest level.

B. Parameter Exploration

We apply the hierarchical approach as described in Section III-A to a series of Middlebury benchmarks. We run a software implementation of the described Hierarchical-TRW-S algorithm in order to explore the effects of various parameters that can be *tuned* in the hierarchical approach. We observe that choosing a *good* set of parameters is critical to the success of a hardware implementation of Hierarchical-TRW-S, as we must carefully trade hierarchical overheads for convergence speed and accuracy.

Two parameters that can be manipulated are the number of levels in the hierarchy and the amount of computation done at each level. It may be profitable to increase one or both, if this means trading off the additional computation for faster convergence. We varied the number of levels between two and seven, and for each case, allowed the number of iterations

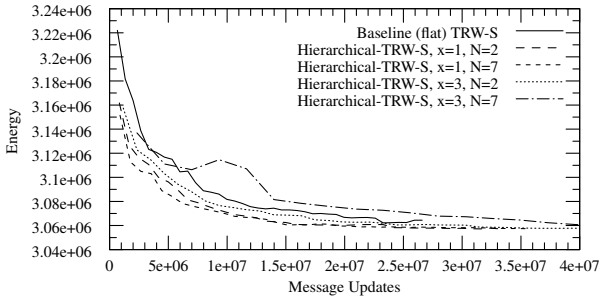
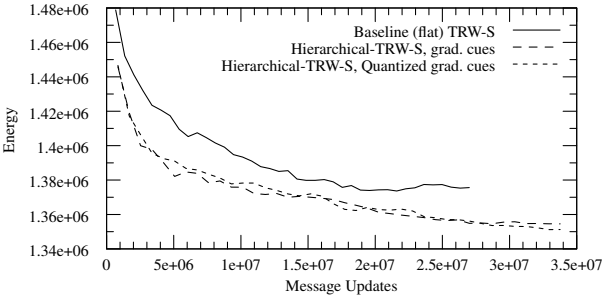
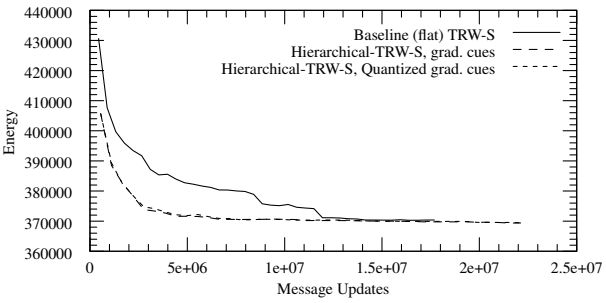


Figure 1. Convergence rates of a software Hierarchical-TRW-S implementation, compared with Baseline (flat) TRW-S, shown as energy-vs-message update plots, for the Venus benchmark. We observe that extra computation from adding levels to the hierarchy or performing additional iterations at higher levels can offset benefits of the Hierarchical-TRW-S algorithm.



(a) Teddy



(b) Tsukuba

Figure 2. Convergence rates for two Middlebury benchmark images using gradient cues in Hierarchical-TRW-S compared with Baseline (flat) TRW-S. Also shown the rate of convergence for the case with gradient cues quantized to one bit (at most two values). The hierarchical approach uses two levels ($N = 2$) and as many iterations on the higher level as on the lower level ($x = 1$). Quantization of gradient cues does not significantly affect the rate of convergence.

at each level to be one, two, or three times the number of iterations on the immediately lower level. As a result, we end up with 18 different configurations.

Figure 1 shows the convergence curves of the four corners of such set of configurations for the *Venus* benchmark. [13] Based on this plot and similar plots from other Middlebury benchmarks, we see that energy is minimized with the least amount of message updates when the same number of iterations are performed at each level. On the other hand, adding levels to the hierarchy does not add any significant performance gains, it may result in loss of performance due to overheads not captured through the metric of number of

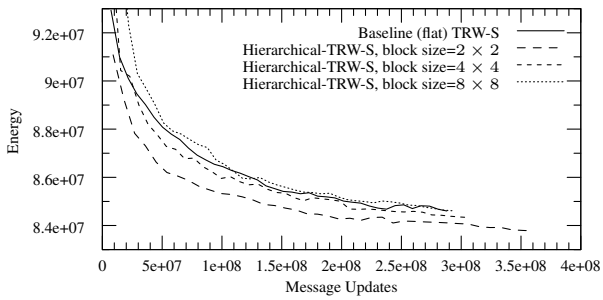


Figure 3. Effect of block size used in forming the hierarchy on the convergence of the Hierarchical-TRW-S algorithm for the *ConesF* (cropped) benchmark. Increasing the block size results in slower convergence, sometimes slower than the Baseline (flat).

message update operations performed, like constructing the hierarchy and copying messages between levels. Consequently, we set the number of levels at two and perform the same number of iterations at each level, as this configuration results in the least amount of computational overhead. Figure 2 shows convergence results for two additional benchmarks from the Middlebury suite that use gradient cues for better stereo estimation.

The Middlebury reference software [13] uses at most two values for gradient cues, however, our method of scaling gradient cues in Equation 2c will create additional values. In order to simplify our implementation, we quantize the averaged gradient cue to be one of the two allowed values for the original problem. Figure 2 shows that this does not adversely affect convergence.

Felzenszwalb and Huttenlocher [5] make a general case for $\epsilon \times \epsilon$ blocks, but restrict their experiments to use 2×2 blocks, which works well in practice. Larger block sizes may be more profitable for images where the feature size is fairly large. We experimented with varying block sizes for various benchmarks. Convergence results for *ConesF* (cropped) (an image from the Middlebury suite [12] cropped to 1800×1000 pixels in size, with 64 disparity levels), for blocks of sizes 2×2, 4×4, and 8×8, are plotted in Figure 3, which shows that larger block sizes lead to a slower rate of convergence.

This surprising result may be attributed to the fact that the probability of object boundaries in the image passing through (as opposed to being aligned at the edges of) larger blocks is more than smaller blocks. As a result, the hierarchical algorithm will produce messages that do not capture such object boundaries well enough, and produce a poor starting guess for the lower layer.

In order to study this effect of edge alignment, we created two pairs of synthetic stereo benchmarks with a square foreground, aligned with an 8×8 and a 2×2 block boundary for one pair, and not aligned with these boundaries for the other pair. Figure 4 shows the difference (L1 norm) between the messages from the last iterations at the higher and lower levels as a heat map. Black indicates no or very little difference; white indicates a significant difference in message values. These results confirm our suspicions: if all object boundaries are aligned with the blocks used to create levels in the hierarchy, both larger and smaller block sizes lead to a good initial value

for the lower level, and Hierarchical-TRW-S performs well. On the other hand, if object boundaries are not aligned with blocks used, both small and large blocks perform equally poorly. In a real image, smaller block sizes will result in much faster convergence than larger block sizes, because they are more likely to be aligned with object boundaries.

As a result, we fix the block size to be 2×2 for the rest of this work. Exploring larger block sizes for Hierarchical-TRW-S with more sophisticated handling of gradient cues is the subject of future work.

IV. Hardware Implementation of Hierarchical-TRW-S

In this section, we describe our hardware implementation of Hierarchical-TRW-S using a hybrid-core computing system, using an optimal set of parameters obtained in Section III. To implement the proposed Hierarchical-TRW-S, we need to consider not only TRW-S inference for different hierarchical levels, but also data management across the levels to construct additional MRFs for higher levels and copy messages back to lower levels. Our implementation platform, Convey HC-1, [4] is equipped with an Intel Xeon dual-core processor and four Xilinx Virtex-5 (V5LX330) FPGAs which are tightly coupled via a cache-coherent virtual memory. We build Hierarchical-TRW-S on top of Choi and Rutenbar's single-level design—Streaming TRW-S (STRM-TRW-S), [2] and explore options to orchestrate data management and inference for best performance.

A. Hierarchical-TRW-S Implementation

To implement Hierarchical-TRW-S, we leverage the STRM-TRW-S architecture. Implementing Hierarchical-TRW-S essentially consists of adding two new operations:

1) **Construct:** Constructs the higher level of the hierarchy by adding data costs and averaging gradient cues.
2) **Copy:** Consists of copying messages from the upper levels to the lower levels in the hierarchy.

We discuss two possible approaches to implementing the Hierarchical-TRW-S modifications:

1) **Naïve approach:** We can employ a Naïve approach and use STRM-TRW-S to run the inference algorithm on FPGA, and let the CPU handle the Hierarchical-TRW-S modifications, namely the Construct and Copy tasks. In addition to reading stereo images and writing a disparity map, the CPU will construct a new MRF for the higher-level inference, then call STRM-TRW-S to run inference in FPGA, and finally copy computed message values to be used in the next hierarchy. However, as we can see from Table II, the CPU operations for Construct and Copy incur significant overhead in total execution time.
2) **Optimized approach:** We move the Construct and Copy operations required for Hierarchical-TRW-S to dedicated FPGA modules. In this approach, the CPU is responsible only for handling inputs and outputs and for coordinating the different phases of the Hierarchical-TRW-S algorithm between the FPGA units. This minimizes Hierarchical-TRW-S' overheads, as shown in Table II.

Table I. Device utilization summary for Optimized Hierarchical-TRW-S on Xilinx Virtex-5 (V5LX330) FPGA

Resources	STRM-TRW-S	Construct	Copy	Convey infrastructure	Total
Slice Registers (FF)	25,745	920	1,027	53,531	**81,223**
Slice LUTs (6 input)	24,967	1,362	698	58,916	**86,125**
Block RAM (36 kbit)	132	8	22	75	**237**

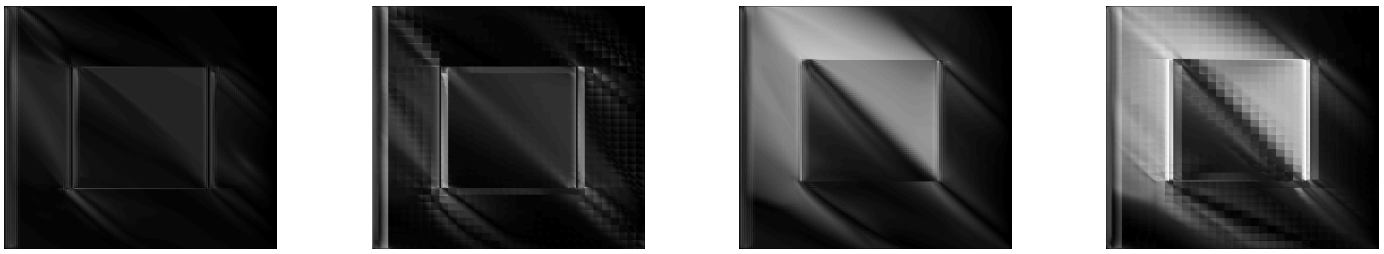
B. Architecture Details for Optimized Approach

The overall architecture for Optimized Hierarchical-TRW-S is shown in Figure 5. A modification on the STRM-TRW-S architecture of Choi and Rutenbar, [2, 3] Optimized Hierarchical-TRW-S shares the memory interface between STRM-TRW-S, Construct, and Copy. Data costs for each node (16 bits per label) in the MRF and its horizontal and vertical messages (24 bits per message per label, i.e., 48 bits per label) are packed together. The 1 kbit-per-cycle external memory bandwidth can transfer data cost (via memory ports 0–3) and messages (via memory ports 4–15) for up to 16 labels in each clock cycle. If the number of labels is larger than 16, it will take multiple cycles to fetch data for the entire data set. STRM-TRW-S, Construct, and Copy employ a folded pipeline architecture [9] with a variable folding factor of up to four to efficiently fetch and process data. Table I shows the device utilization summary of the FPGA modules.

Streaming TRW-S Module: As explained in Section II-B, to guarantee the convergence of TRW-S, messages need to be updated in a sequential manner. Choi and Rutenbar showed that a diagonal order of message updates preserves the sequential dependencies while providing opportunities for parallelism. [2] Their STRM-TRW-S architecture exploits this parallelism within the diagonal nodes to maximize throughput. STRM-TRW-S consists of the message-passing unit and a FIFO interface. A stream of node data (stored in the diagonal order) is fetched from the external memory to STRM-TRW-S via the FIFO interface, and then processed in parallel through multiple pipeline stages of the message-passing unit. Memory access by the FIFO interface and computation by the message-passing unit are done in parallel to exploit as much memory bandwidth as possible.

Construct and Copy for Streaming TRW-S: The Construct and Copy operations necessary for Hierarchical-TRW-S deal with data transfer between hierarchical levels—Construct transfers data from the lower level to the higher level, and Copy performs the inverse. Therefore, to design dedicated modules for the Optimized approach, we need to consider the data layout between levels.

As discussed previously, STRM-TRW-S requires data to be stored in a diagonal fashion. However, Construct and Copy need to access the data as square 2×2 blocks, i.e. from multiple diagonals. This means that given a diagonal order for data storage on any level, Construct and Copy must necessarily access data in a irregular and non-sequential order



(a) Aligned, Block size = 2×2 (b) Aligned, Block size = 8×8 (c) Not aligned, Block size = 2×2 (d) Not aligned, Block size = 8×8

Figure 4. Heat maps show the L1 norm between messages (black indicates very small or no difference while white indicates a large difference) at the 9th iteration on the lower level and the 10th iteration on the higher level when the foreground object is aligned (Figure 4a and Figure 4b) and not aligned (Figure 4c and Figure 4d) with the blocks used to create the hierarchy. When the foreground object is aligned, both small and large block sizes result in messages fairly close to their respective final values on the lower level. On the other hand, when the foreground is not aligned, both larger and smaller block sizes result in poor starting guesses for the TRW-S algorithm on the lower level.

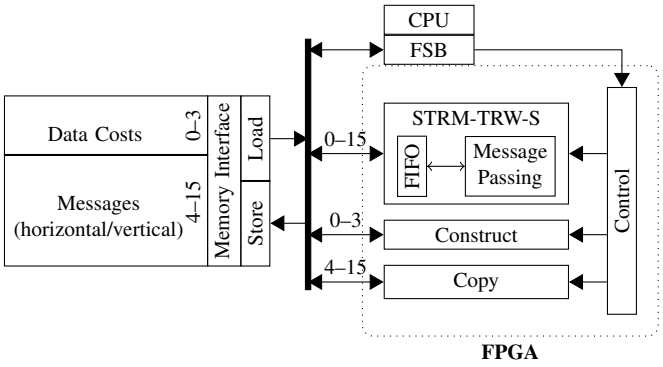


Figure 5. Overall architecture of Optimized Hierarchical-TRW-S

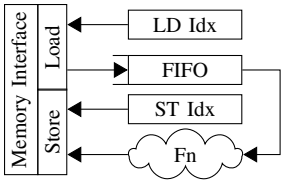


Figure 6. Architecture for Construct and Copy units

to transfer data between levels. This, however, causes little performance degradation thanks to Convey's scatter-gather DIMMs (SG-DIMMs), [1] which provide 8-byte accesses to physical memory, thereby reducing the inefficiencies involved in non-sequential memory accesses on a conventional system. As our message data is packed in multiples of 8 bytes, the SG-DIMMs can be exploited to achieve near-peak bandwidth for irregular memory accesses. As an example, experiments on the Copy operation for the *Tsukuba* benchmark show memory bandwidth utilization at 92.1% of the allotted capacity (see Table V).

Figure 6 shows a streaming architecture used for Construct and Copy. Both consist of index control logic, a FIFO, and functional logic. The index control logic computes proper indices for transferring data between a diagonal in the higher level and the corresponding multi-diagonals in the lower level. To this end, the load index control of Construct and the store index control of Copy include a logic block for index increment with varying strides. On the other hand, the store index control of Construct and the load index control of Copy need only one simple counter for memory indexing. The functional logic for Construct consists of arithmetic units for accumulation of data cost and averaging of gradient cues, whereas the function logic for Copy is simply a buffer for copying the same messages four times. From Table I, we see that the hardware overheads for Construct and Copy are small.

Table II. COMPARISON OF TIMES SPENT ON VARIOUS TASKS FOR BASELINE (FLAT) TRW-S AND HIERARCHICAL-TRW-S ($N = 2, x = 1$) ALGORITHMS FOR THE TSUKUBA IMAGE PAIR BENCHMARK. TIMES REPORTED ARE FOR 29 ITERATIONS FOR BASELINE TRW-S AND 12 FOR HIERARCHICAL-TRW-S, AS THEY RESULT IN NEARLY IDENTICAL ENERGY VALUES (AROUND 100.5% OF THE LOWER BOUND). STRM-TRW-S TIMES FOR HIERARCHICAL-TRW-S INCLUDE TIME SPENT IN BOTH LEVELS OF THE HIERARCHY.

Task	Time Required (s)		
	Baseline	Hierarchical-TRW-S	
		Naïve	Optimized
Construct	N/A	0.015	0.001
Copy	N/A	0.027	0.001
STRM-TRW-S	0.093	0.048	0.048
Total	**0.093**	**0.090**	**0.050**

Table III. A COMPARISON BETWEEN BASELINE TRW-S AND HIERARCHICAL-TRW-S ($N = 2$, $x = 1$) FOR THE MIDDLEBURY STEREO BENCHMARK IMAGES. THE TIMES REPORTED DISCOUNT THE INITIALIZATION TIME, AS IT IS NOT PART OF THE MRF INFERENCE PROBLEM. THE TIMES REPORTED FOR BASELINE INCLUDE THE TIME SPENT IN STRM-TRW-S, WHILE TIMES REPORTED FOR HIERARCHICAL-TRW-S INCLUDE THE TIMES SPENT IN STRM-TRW-S FOR BOTH LEVELS, ALONG WITH TIMES REQUIRED FOR CONSTRUCT AND COPY OPERATIONS.

Benchmark	Convergence	Runtime (s)	
		Baseline	Hierarchical
Teddy	Quick	0.233	0.159 (-32%)
	Better	1.436	1.135 (-21%)
Tsukuba	Quick	0.022	0.014 (-38%)
	Better	0.093	0.050 (-46%)
Venus	Quick	0.115	0.069 (-40%)
	Better	0.336	0.199 (-41%)
ConesF (Cropped)	Quick	2.529	1.313 (-48%)
	Better	11.080	5.185 (-53%)

V. EXPERIMENTAL RESULTS

This section discusses the experimental results obtained on our Convey CPU-FPGA hybrid platform.

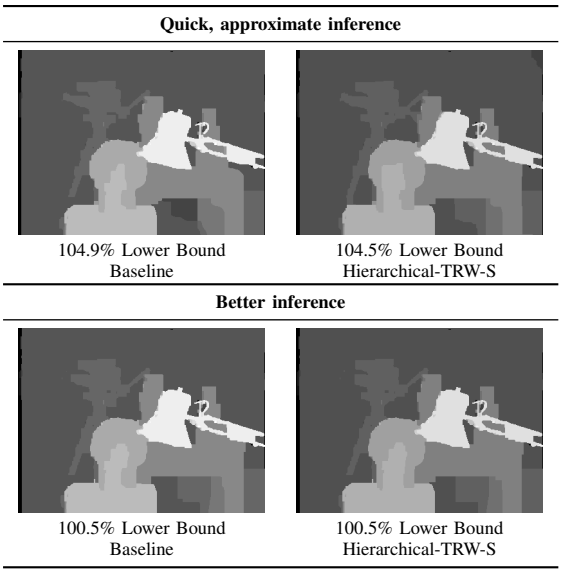


Figure 7. A comparison of resulting disparity maps for Baseline TRW-S and Hierarchical-TRW-S for the case when quick, approximate inference is needed, and for the case when energy is allowed to reach sufficiently close to the bound.

Table II lists the time required for various tasks in both Baseline (flat) TRW-S and Hierarchical-TRW-S for the *Tsukuba* benchmark. Because Hierarchical-TRW-S requires fewer iterations to achieve the same energy value as Baseline TRW-S, the time spent in the STRM-TRW-S hardware on the FPGA is significantly reduced, from 93 ms to 48 ms. Naïve, which leaves the preparation and copy tasks to the CPU, achieves only a modest improvement over Baseline, due to overheads associated with performing the Construct and Copy tasks on the CPU. Optimized, in which we moved Construct and Copy tasks to FPGA, performs better. For the purpose of reporting relative improvements in performance, we discount the initialization time because it is the time required to compute the inputs to the MRF inference problem and is not a part of the MRF inference task.

Table III lists times required by Baseline TRW-S and Hierarchical-TRW-S for the Middlebury benchmarks to reach a certain level of convergence. There are two sets of results mentioned for each benchmark. The first one is for the case when a quick, rough estimate is sufficient. In this case, we perform approximately 10 (Baseline) iterations on the MRF. The second set is for when we want better inference and allow the energy to reach sufficiently close to the bound. (Figure 7 shows the resulting disparity map using the *Tsukuba* benchmark for the two cases, for both Baseline TRW-S and Hierarchical-TRW-S.) The hierarchical approach speeds up the rate of convergence, but has its own overheads (Construct and Copy). In the first case (quick, rough estimate), the overheads may not be as easily amortized due to the relatively short inference time. As we get closer to convergence, the overheads are amortized, but the hierarchical algorithm provides diminishing returns. Thanks to Optimized, the overheads (Construct and Copy) are rendered extremely small to cause any deterioration in the performance of Hierarchical-TRW-S (e.g.: 0.002 s overhead vs 0.048 s inference for the *Tsukuba* benchmark, better inference, see Table II). From Table III, we note that Optimized Hierarchical-TRW-S offers a 21% to 53% reduction in inference time compared with Baseline.

VI. Related Work

A significant amount of related work deals with stereo matching either on FPGAs, GPUs, or VLSI circuits. In this section, we restrict our study to related work involving MAP inference on MRFs. It is difficult to make an exact comparison with related work due to the plethora of algorithms and benchmarks used. In this section, we report relevant performance numbers, and make a best estimate of performance normalized to the benchmarks used in this paper when the two sets are not the same. Because this section can necessarily discuss only a limited number of prior work, we discuss only the most relevant related work.

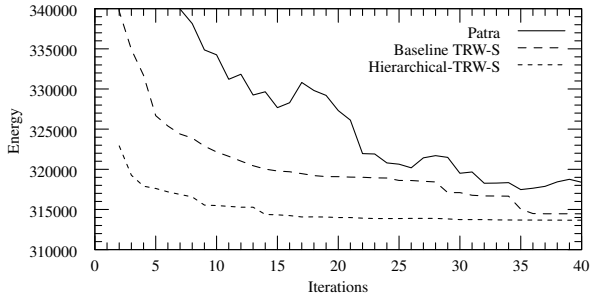
A. Comparison against Patra

Recently Zhao et al have proposed Patra, [18] which is an FPGA implementation of the TRW algorithm by Wainwright et al. [14] Our Hierarchical-TRW-S implementation is superior than Patra in two ways: convergence speed in time and platform resource utilization.

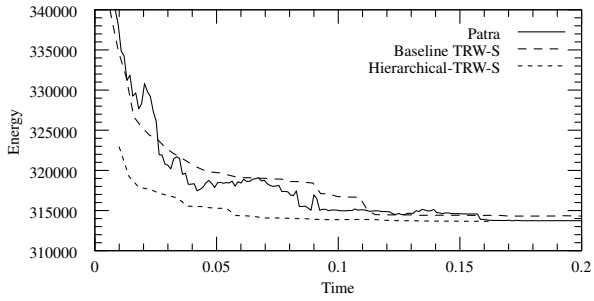
Convergence speed in time: In order to compare our implementation against Patra, we modify the Middlebury reference software [13] to implement Patra, and use it to obtain energy-iteration plots. We ensure that our software implementation matches the final (converged) energy values reported in the Patra paper exactly. Next, we use the per-iteration times reported in the Patra paper to compute the energy-time plots. Note that Patra does not support gradient cues or L2 smoothness costs. As a result, we disable these features in Optimized Hierarchical-TRW-S as well, and obtain energy-iteration and energy-time curves in order to make a fair comparison against Patra. As a result, the convergence curves for our figures in this section may appear different from those in the rest of this paper.

As Patra is based on TRW and not on TRW-S, it cannot provide any of the guarantees on convergence that come with TRW-S. While Patra may run each iteration faster than our streaming algorithm, Patra converges at a much slower rate. As we can see in Figure 8a, Patra takes many more iterations to reach the same energy levels as Baseline TRW-S or Hierarchical-TRW-S. Even when we factor in the lower time per iteration for Patra, we see that our Hierarchical-TRW-S algorithm outperforms Patra, as can be seen in Figure 8b. This is true for the other benchmarks as well.

Device and memory bandwidth utilization: Patra is implemented on a CPU-FPGA hybrid platform, equipped with one Xilinx Virtex-6 FPGA and 38 Gigabyte/sec of memory bandwidth. Similarly, we use one Xilinx Virtex-5 FPGA available in Convey HC-1 with 19.2 Gigabyte/sec of memory bandwidth. Efficient hardware implementations should provide good performance while utilizing as little area and memory bandwidth resources; our Hierarchical-TRW-S implementation outperforms Patra on both fronts. The numbers are reported in Table IV and Table V. As our architecture is optimized for multiples of 16 labels, *Venus* and *Teddy* aren't able to use all available bandwidth, as they use 20 and 60 labels respectively. Nevertheless, our system provides better convergence while using less FPGA resources and less bandwidth.



(a) Energy vs Iterations



(b) Energy vs Time

Figure 8. Comparison of convergence rates for Hierarchical-TRW-S, Baseline TRW-S, and Patra for the *Tsukuba* benchmark.

Table IV. COMPARISON OF FPGA RESOURCE UTILIZATION BETWEEN PATRA [18] AND OPTIMIZED HIERARCHICAL-TRW-S (OUR SYSTEM). NUMBERS LIST TOTAL DEVICES USED ON THE FPGA.

System	FPGA	LUT	FF	BRAM
Patra	Virtex-6 (SX475T)	73,217	110,700	380
Hierarchical-TRW-S	Virtex-5 (V5LX330)	86,125	81,223	237

B. Comparison against other related work

Our Baseline (flat) TRW-S implementation is exactly the same as Choi and Rutenbar's work. [2, 3] This has been discussed in Section IV-B. Most other related work uses belief propagation (BP) and its hierarchical variant, e.g.: tile-based BP on a VLSI chip and on a GPU by Liang et al., [8] on a GPU and on FPGAs by Park et al., [10, 11] real-time BP on the GPU by Yang et al., [17] fast-BP by Xiang et al. [16] Choi and Rutenbar [2] extensively compare some of these works with Baseline (flat) TRW-S implementation.

In particular, Liang et al. [8] use a tile-based BP algorithm on a UMC 90 nm technology, which processes QVGA (320×240) images with 64 labels at 14 frames per second. We estimate that for the same set of input images, our hardware will operate over 17 frames per second. We can, however, make a direct comparison with their GPU implementation on a 8800 GTS GPU, which requires a reported 124 ms on the Tsukuba benchmark, compared to 14 ms required by our hardware Hierarchical-TRW-S solver.

Table V. COMPARISON OF MEMORY BANDWIDTH UTILIZATION BETWEEN PATRA [18] AND OPTIMIZED HIERARCHICAL-TRW-S (OUR SYSTEM). NUMBERS ARE REPORTED AS ABSOLUTE BANDWIDTH IN GIGABYTES PER SECOND.

System	Tsukuba $384\times288\times16L$	Venus $434\times383\times20L$	Teddy $450\times375\times60L$
Patra	26.63 GB/s	12.50 GB/s	19.41 GB/s
Hierarchical-TRW-S	17.69 GB/s	11.08 GB/s	16.70 GB/s

Park et al. [10, 11] have developed a QVGA stereo matching algorithm using Fast BP on two Xilinx Virtex II FPGAs [11] and on a 0.18 μm CMOS process. [10] Their implementations achieve 30 frames per second at QVGA size (320×240) images with 32 disparity levels. We estimate that our system can operate at faster than 35 frames per second.

Similarly, Yang et al. [17] with their real-time BP algorithm can achieve up to 16 frames per second for Tsukuba on a 7900 GTX GPU. Xiang et al. [16] can process Tsukuba in 61 ms on a Geforce GTX 260 GPU, compared with 14 ms for our implementation.

VII. SUMMARY

In this paper, we proposed a hierarchical modification to the TRW-S algorithm. We have tested this modification in software and on a Convey HC-1 CPU-FPGA hybrid platform. Our experiments show a definite improvement in both software and hardware implementations. Specifically, we get a 21% to 53% reduction in computation time on our CPU-FPGA hardware implementation for four Middlebury stereo-image benchmarks. At the same time, we've observed that using larger blocks to create the hierarchy worsens convergence. Further investigation into a method to speed up convergence using larger blocks will be the subject of future work.

ACKNOWLEDGMENTS

This work was supported in part by Intel's Science and Technology Center for Embedded Computing, and by SRC STARnet's Center for Systems on Nanoscale Information fabriCs (SONIC), through MARCO/DARPA Grant 2013-MA-2385.

REFERENCES

[1] T. M. Brewer, "Instruction set innovations for the Convey HC-1 computer," *IEEE Micro*, no. 2, pp. 70–79, 2010. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/MM.2010.36

[2] J. Choi and R. A. Rutenbar, "Hardware implementation of MRF MAP inference on an FPGA platform," in *22nd International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2012, pp. 209–216. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6339183

[3] ——, "Video-rate stereo matching using Markov random field TRW-S inference on a hybrid CPU+FPGA computing platform," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays (FPGA)*. ACM, 2013, pp. 63–72. [Online]. Available: http://dl.acm.org/citation.cfm?id=2435264.2435278

[4] *Convey Reference Manual*, Convey Computer, May 2012. [Online]. Available: http://www.conveysupport.com/alldocs/ConveyReferenceManual.pdf

[5] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient belief propagation for early vision," *International Journal of Computer Vision (IJCV)*, vol. 70, no. 1, pp. 41–54, 2006. [Online]. Available: http://link.springer.com/10.1007/s11263-006-7899-4

[6] V. Kolmogorov, "Convergent tree-reweighted message passing for energy minimization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 10, pp. 1568–1583, 2006. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1677515

[7] V. Kolmogorov and R. Zabin, "What energy functions can be minimized via graph cuts?" *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 26, no. 2, pp. 147–159, 2004. [Online]. Available: http://ieeexplore.ieee.org/articleDetails.jsp?arnumber=1262177

[8] C.-K. Liang, C.-C. Cheng, Y.-C. Lai, L.-G. Chen, and H. H. Chen, "Hardware-efficient belief propagation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 21, no. 5, pp. 525–537, 2011. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5733391

[9] K. K. Parhi, C.-Y. Wang, and A. P. Brown, "Synthesis of control circuits in folded pipelined DSP architectures," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 29–43, 1992. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=109555

[10] J. Park, S. Lee, and H.-J. Yoo, "A 30fps stereo matching processor based on belief propagation with disparity-parallel PE array architecture," in *Proceedings of 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2010, pp. 453–456. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5537657

[11] S. Park, C. Chen, and H. Jeong, "VLSI architecture for MRF based stereo matching," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*. Springer, 2007, pp. 55–64. [Online]. Available: http://www.springerlink.com/index/0M3VW65K75387513.pdf

[12] D. Scharstein and R. Szeliski, "High-accuracy stereo depth maps using structured light," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 1. IEEE, 2003, pp. I–195. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1211354

[13] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for Markov random fields with smoothness-based priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 6, pp. 1068–1080, 2008. [Online]. Available: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4420084

[14] M. J. Wainwright, T. S. Jaakkola, and A. S. Willsky, "MAP estimation via agreement on trees: message-passing and linear programming," *IEEE Transactions on Information Theory*, vol. 51, no. 11, pp. 3697–3717, 2005. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1522634

[15] Y. Weiss and W. T. Freeman, "On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs," *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 736–744, 2001. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=910585

[16] X. Xiang, M. Zhang, G. Li, Y. He, and Z. Pan, "Real-time stereo matching based on fast belief propagation," *Machine Vision and Applications*, vol. 23, no. 6, pp. 1219–1227, 2012. [Online]. Available: http://link.springer.com/10.1007/s00138-011-0405-1

[17] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nistér, "Real-time global stereo matching using hierarchical belief propagation." in *Proceedings of the British Machine Vision Conference (BMVC)*, vol. 6. BMVA Press, 2006, pp. 989–998. [Online]. Available: http://www.bmva.org/bmvc/2006/papers/324.html

[18] W. Zhao, H. Fu, G. Yang, and W. Luk, "Patra: Parallel tree-reweighted message passing architecture," in *24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6927506