

© 2015 Jungwook Choi

HIGH PERFORMANCE AND ERROR RESILIENT PROBABILISTIC INFERENCE SYSTEM
FOR MACHINE LEARNING

BY

JUNGWOOK CHOI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Doctoral Committee:

Professor Rob A. Rutenbar, Chair
Associate Professor Deming Chen
Professor David A. Forsyth
Professor Naresh R. Shanbhag

ABSTRACT

Many real-world machine learning applications can be considered as inferring the best label assignment of maximum a posteriori probability (MAP) problems. Since these MAP problems are NP-hard in general, they are often dealt with using approximate inference algorithms on Markov random field (MRF) such as belief propagation (BP). However, this approximate inference is still computationally demanding, and thus custom hardware accelerators have been attractive for high performance and energy efficiency.

There are various custom hardware implementations that employ BP to achieve reasonable performance for the real-world applications such as stereo matching. Due to lack of convergence guarantees, however, BP often fails to provide the right answer, thus degrading performance of the hardware. Therefore, we consider sequential tree-reweighted message passing (TRW-S), which avoids many of these convergence problems with BP via sequential execution of its computations but challenges parallel implementation for high throughput. In this work, therefore, we propose a novel streaming hardware architecture that parallelizes the sequential computations of TRW-S. Experimental results on stereo matching benchmarks show promising performance of our hardware implementation compared to the software implementation as well as other BP-based custom hardware or GPU implementations.

From this result, we further demonstrate video-rate speed and high quality stereo matching using a hybrid CPU+FPGA platform. We propose three frame-level optimization techniques to fully exploit computational resources of a hybrid CPU+FPGA platform and achieve significant speed-up. We first propose a message reuse scheme which is guided by simple scene change detection. This scheme allows a current inference to be made based on a determination of whether the current result is expected to be similar to the inference result of the previous frame. We also consider frame level parallelization to process multiple frames

in parallel using multiple FPGAs available in the platform. This parallelized hardware procedure is further pipelined with data management in CPU to overlap the execution time of the two and thereby reduce the entire processing time of the stereo video sequence. From experimental results with the real-world stereo video sequences, we see video-rate speed of our stereo matching system for QVGA stereo videos.

Next, we consider error resilience of the message passing hardware for energy efficient hardware implementation. Modern nanoscale CMOS process technologies suffer in reliability caused by process, temperature and voltage variations. Conventional approaches to deal with such unreliability (e.g., design for the worst-case scenario) are complex and inefficient in terms of hardware resources and energy consumption. As machine learning applications are inherently probabilistic and robust to errors, statistical error compensation (SEC) techniques can play a significant role in achieving robust and energy-efficient implementation. SEC embraces the statistical nature of errors and utilizes statistical and probabilistic techniques to build robust systems. Energy-efficiency is obtained by trading off the enhanced robustness with energy.

In this work, we analyze the error resilience of our message passing inference hardware subject to the hardware errors (e.g. errors caused by timing violation in circuits) and explore application of a popular SEC technique, algorithmic noise tolerance (ANT), to this hardware. Analysis and simulations show that the TRW-S message passing hardware is tolerant to small magnitude arithmetic errors, but large magnitude errors cause significantly inaccurate inference results which need to be corrected using SEC. Experimental results show that the proposed ANT-based hardware can tolerate an error rate of 21.3%, with performance degradation of only 3.5% with an energy savings of 39.7%, compared to an error-free hardware.

Lastly, we extend our TRW-S hardware toward a general purpose machine learning framework. We propose advanced streaming architecture with flexible choice of MRF setting to achieve 10-40x speedup across a variety of computer vision applications. Furthermore, we provide better theoretical understanding of error resiliency of TRW-S, and of the implication of ANT for TRW-S, under more general MRF setting, along with strong empirical support.

Acknowledgments

First of all, I would like to thank my advisor, Professor Rob Rutenbar, for his continuous support and guidance throughout my PhD. He has provided me with invaluable advice, and I cannot imagine completing this work without his patience and motivation. I would also like to express my appreciation to the rest of my committee: Professor Naresh Shanbhag, Professor Deming Chen, and Professor David Forsyth for their insightful comments and thoughtful help on various issues.

During my PhD, I have been privileged to work with many brilliant people. I sincerely thank Eric Kim, Ameya Patil and Naresh Shanbhag for the stimulating discussions and invaluable advice on error resilience study. I am also grateful to Skand Hurkat, Eriko Nurvitadhi, and José Martínez for their tremendous help on implementing advanced probabilistic inference machines.

My sincere thanks go to my fellow lab mates and colleagues, Glenn Ko, Tianqi Gua, Abner Guzman-Rivera, Shang-nien Tsai, Minje Kim and Sai Zhang as they always gladly provide their help and advice on my research. I would also like to thank the people in KCSA and my beloved friends in South Korea and the U.S. for their friendship. I thank the administrative staff in ECE and CS, especially Laurie Fisher, Karen Stahl, Julie Gustafson, and Erin Henkelman for assisting me many times with the miscellaneous issues.

Finally, I would like to express my sincere gratitude to my family for their consistent support. I must acknowledge my wife and best friend, Na Young; without her love, encouragement, and support, I would not have completed this work. I would also like to recognize the invaluable contributions of my parents and parents-in-law who have provided unconditional love and support. Last but not least, my deepest love goes to my little angel Antonia.

TABLE OF CONTENTS

List of Tables	vii
List of Figures	viii
Chapter 1 Introduction	1
1.1 Related Work	4
1.2 Thesis Contribution	6
1.3 Dissertation Organization	7
Chapter 2 Background	9
2.1 Markov Random Field Inference for Stereo Matching	9
2.2 Statistical Error Compensation	15
2.3 Hybrid Multicore + FPGA Platform	17
2.4 Summary	18
Chapter 3 Hardware Implementation of Sequential Tree-Reweighted Message Passing	20
3.1 Hardware Architecture of MRF Parameter Computation	20
3.2 Design Issues in Hardware Implementation of TRW-S	23
3.3 Hardware Architecture of TRW-S	25
3.4 Experimental Results	28
3.5 Summary	32
Chapter 4 Frame-Level Optimizations for High Speed Stereo Matching	33
4.1 Message Reuse based on Scene Change Detection	33
4.2 Frame-Level Parallelization	36
4.3 Function-Level Pipelining	37
4.4 Experimental Results	40
4.5 Summary	47
Chapter 5 Error Resilient MRF Message Passing Architecture for Stereo Matching .	48
5.1 Error Analysis of TRW-S	48
5.2 Simulation Methodology and System Evaluation Setup	53
5.3 Results	55
5.4 Summary	58

Chapter 6	Toward a General Purpose Probabilistic Inference System	59
6.1	High Performance TRW-S Accelerator for Computer Vision	59
6.2	Error Resilient TRW-S for Computer Vision via ANT	76
6.3	Summary	83
Chapter 7	Conclusion	87
References	89

List of Tables

1.1	Taxonomy of probabilistic inference platforms.	6
3.1	List of stereo matching sub-functions.	20
3.2	Device utilization summary.	28
3.3	Execution cycle and time of streaming TRW-S for stereo matching.	29
3.4	Execution time (in ms) of stereo matching functions.	30
3.5	Comparison of speed between GPU implementations [1, 2, 3] and streaming TRW-S for Tsukuba task.	31
4.1	Analysis of total execution time (in second) for Flower video stereo matching (276 frames).	43
4.2	Impact of message reuse on energy minimization performance and execution time (in second). Flower video stereo matching task (360x262x276 frames) is used for example.	44
5.1	disparity map and BPR comparison for error-free, conventional and ANT at various V_{dd}	57
5.2	Estimated compensation overhead and power consumption of ANT obtained via synthesis in a commercial 45 nm CMOS process.	58
6.1	MRF parameter configurations for target computer vision applications	60
6.2	Relative resource overhead of memory interface for different parallel factors.	65
6.3	Summary for FPGA utilization and specification of PE.	72
6.4	Comparison between execution time (in second) of 1 iteration of TRWS SW, STRM-TRWS and STRM-TRWS-CV for various computer vision tasks.	73
6.5	Performance evaluation of streaming TRW-S hardwares within the taxonomy of probabilistic inference platforms.	75
6.6	Performance evaluation of streaming TRW-S hardwares with other non-BP stereo matching implementations. *MDE/s is calculated for one iteration of TRW-S inference in order to compensate the fact that the other methods (e.g., SGM, DP and ADSW) perform one-time search for each pixel.	77

List of Figures

1.1	Simulation results of voltage overscaling (VOS) for a 4-tap correlation filter (a sensor) at 50 MHz in a 45 nm CMOS process [4].	3
2.1	Representation of a posterior in a pairwise 4-connected (grid) MRF.	11
2.2	Comparison of energy minimization performance between BP and TRW-S for Tsukuba stereo matching benchmark [5].	14
2.3	Statistical error compensation: (a) general form, (b) algorithmic noise tolerance, and (c) error distributions.	16
2.4	Overall architecture of Convey HC-1 [6].	18
3.1	(a) Overall architecture of matching cost computation unit (MCC) and gradient cue computation unit (GCC). (b) Example of image data. Nodes (circles with numbers) inside the red dotted line and blue dotted line are used as input for MCC and GCC, respectively. (c) Detailed architecture of MCC.	21
3.2	Detailed architecture of GCC.	22
3.3	Energy minimization results of TRW-S for different fractional bits.	24
3.4	Diagonal ordering for parallelization of TRW-S.	24
3.5	Streaming TRW-S hardware architecture: (a) overall architecture, (b) reparameterize unit, (c) message update unit.	26
3.6	Comparison of disparity maps.	30
4.1	Impact of message reuse in energy minimization performance and histogram based scene change detection. (Flower video stereo matching task ([5]) is used for example.)	34
4.2	Pipelining of stereo matching functions.	38
4.3	Video stereo matching task (Flower, [7]) and disparity map results. (Lighter grey means closer to camera.)	41
4.4	Possible reduction versus actual reduction in overall execution time due to function-level pipelining.	41
4.5	Per-frame execution time of CPU and FPGA functions.	42
4.6	Video stereo matching with scene change detection (SCD) based message reuse.	45
5.1	Effect of AS error on updated message when (a) $\Delta H > 0$, and (b) $\Delta H < 0$	51

5.2	Verification of error analysis: (a) effect of error on message updated via a box plot of e_{AS} vs. \tilde{e}_{AS} , and (b) effect of error on energy minimization performance of message passing via a plot of minimum energy (after 10 iterations) vs. e_{max}	52
5.3	Energy minimization performance against various precision in computation of STRM-TRWS.	53
5.4	TRWS-HW simulation: (a) methodology, and (b) error statistics for AS at $V_{dd} = 0.78$ V.	54
5.5	Simulation results: (a) performance of ANT with injection of uniform errors with different magnitude using $M[20, 0, 0]$ and $E[4, 12, 4]$, and (b) error injection rate vs. energy minimization for different ANT estimators with $M[8, 8, 4]$	56
6.1	Overall architecture of block-parallel memory interface.	62
6.2	Example of processing 3x3 MRF using two PEs. (a) Node and edge data for a 3x3 MRF mapped to two sets of streams for nodes ($\{0,2,4,6,8\}$, $\{1,3,5,7\}$), horizontal edges ($\{0,2,4\}$, $\{1,3,5\}$), and vertical edges ($\{0,2,4\}$, $\{1,3,5\}$) are loaded to the corresponding Read buffers. (b) The loaded streams are used by two PEs to compute messages for each node. The diagram shows indices of nodes that go in and out of each PE. Assume that Read buffers are preloaded. (c) Processing nodes using two PEs with three cycles of unexpected delay in Read buffer of PE0 at cycle 5.	66
6.3	(a) Illustration of Jump Flooding algorithm. (b) Comparison of minimum energy for an image denoising task (House): Full search vs. Jump Flooding.	68
6.4	Streaming architecture for Jump Flooding-based message computation (JF-UNIT). (a) Message passing unit as a PE. (b) An example for operation of Jump Flooding. Solid and dashed arrows represent labels chosen and not chosen at each level, respectively. (c) Architecture for Jump Flooding-based message update.	71
6.5	Comparison of inference quality between TRWS-SW and STRM-TRWS-CV for various computer vision tasks.	73
6.6	Impact of error injection on inference quality for different coupling strengths.	82
6.7	(a) Inference quality vs. error injection for different coupling strength and ANT, (b) Inference quality vs. coupling strength for different error magnitude and ANT.	84
6.8	Tsukuba: (a) Inference quality vs. error injection for different coupling strength and ANT, (b) Inference quality vs. coupling strength for different error magnitude and ANT.	85
6.9	Venus: (a) Inference quality vs. error injection for different coupling strength and ANT, (b) Inference quality vs. coupling strength for different error magnitude and ANT.	85
6.10	Teddy: (a) Inference quality vs. error injection for different coupling strength and ANT, (b) Inference quality vs. coupling strength for different error magnitude and ANT.	86

6.11 Comparison of bad pixel ratio (BPR) for stereo matching of Teddy task using TRW-S with different λ s.	86
---	----

Chapter 1

Introduction

We are living in the “Big Data” era where “enormous amounts of heterogeneous, semistructured and unstructured data are continually generated at unprecedented scale” [8]. To analyze and extract information from the data at scale, many tools from machine learning have been successfully employed. In particular, probabilistic graphical models (PGMs) are “an elegant framework which combines uncertainty (probabilities) and logical structure (independence constraints) to compactly represent complex, real-world phenomena” [9], and probabilistic inference is a way to derive useful insights from the PGM. Thanks to their flexibility and representation power, PGMs have been widely used in machine learning to model a large variety of real-world applications. We are especially interested in undirected graphical models called Markov random fields (MRFs) for modeling maximum a posteriori (MAP) problems, which include stereo matching and image denoising in computer vision, medical image segmentation in bio-engineering, and sound source separation in machine listening [10, 11, 12]. In these applications, probabilistic inference seeks the most probable label assignment which describes a set of related, random observations. Thus, probabilistic inference is practically attractive as one unified framework for machine learning, since one good implementation can be used for a variety of machine learning applications.

The main focus of this work is to build a probabilistic inference system that can be used as a unified framework for solving MAP problems in machine learning. To this end, custom hardware acceleration is promising for two reasons. First, custom hardware acceleration achieves both high performance and energy efficiency. For example, application specific integrated circuits (ASICs) are 500x more energy efficient than CPU implementation for video decoding [13], and field programmable gate array (FPGA) acceleration minimizes latency and maximizes energy efficiency for intelligent personal assistant applications [14].

In our case, the MAP problems are mapped to MRFs and solved using MRF inference algorithms. Since finding the exact solution for this MRF inference is known to be NP-hard [15], we exploit message passing based approximate inference methods such as belief propagation (BP) [16] and sequential tree-reweighted message passing (TRW-S) [17]. But such approximate inference algorithms are still computationally demanding in general, which is one significant motivation for custom hardware implementation.

Next, we note that custom hardware acceleration can effectively address the reliability issue caused by nanometer imperfections. In the deep sub-nanometer regime, circuits become increasingly vulnerable to supply noise, leakage, and interconnect noise. As an example, it is known that variation of threshold voltage of the metal-oxide-semiconductor field-effect transistor (MOSFET) fabrication is increasing as the feature size shrinks down [18]. Emerging devices are no better, as a line of research shows that the carbon nanotube field-effect transistors (CNFETs) suffer huge variation in I-V curves [19]. These statistical variations in devices pose huge challenges in hardware implementation, since the overhead of worst case design has become prohibitively expensive [20]. However, we can employ statistical error compensation (SEC) in our custom hardware design to enable nominal design with minimal performance degradation and resource overhead, since SEC analyzes errors caused by statistical variation of the circuit and uses the analysis for effective error detection and compensation [21].

Furthermore, energy savings can be achieved by carefully trading the enhanced robustness for energy. Figure 1.1 shows HSPICE simulation results of a 4-tap filter in 45 nm CMOS subject to voltage overscaling (VOS) [4]. If the errors are fully compensated for without additional overhead, energy reduction up to $9\times$ can be achieved over a system operating at the point of first failure (PoFF). As we show throughout this dissertation, SEC techniques provide the much needed low overhead error compensation. In this work, therefore, we first propose a new hardware architecture for a parallel implementation of TRW-S on an FPGA platform. We discuss issues in hardware implementation of TRW-S, which includes the aforementioned sequential manner of computation as well as floating to fixed point conversion. Then, we propose a pipelined hardware architecture in which the order of computation is devised to execute computation in parallel while preserving the original convergence prop-

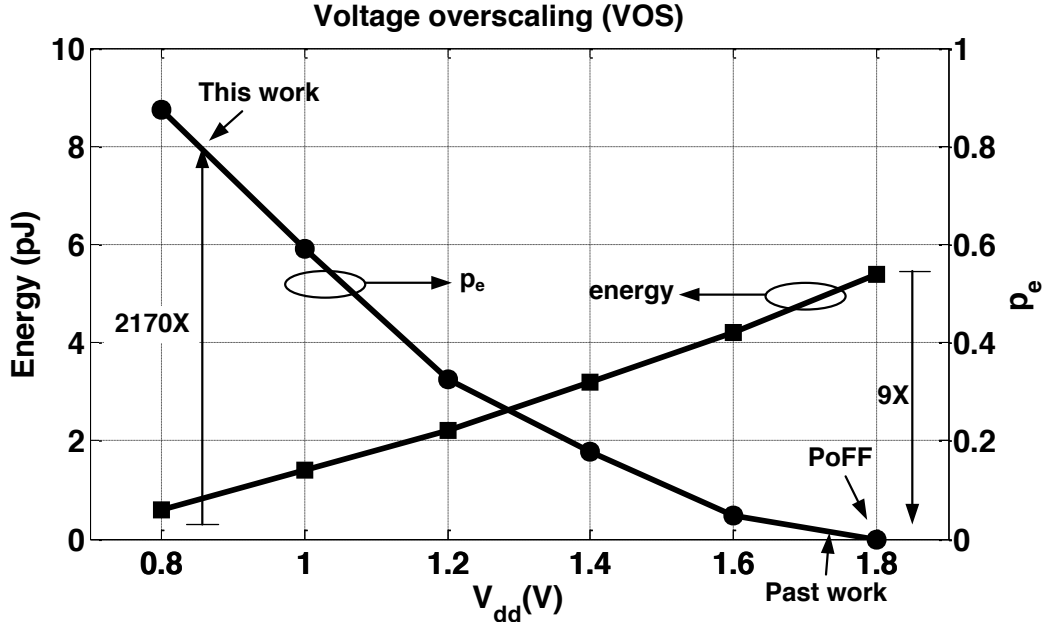


Figure 1.1: Simulation results of voltage overscaling (VOS) for a 4-tap correlation filter (a sensor) at 50 MHz in a 45 nm CMOS process [4].

erty of TRW-S. The proposed hardware architecture exploits a novel FIFO based streaming memory interface to manage streaming data input and output, and the computation inside is fully pipelined to parallelize data processing (i.e., message passing) over the multiple pipeline stages.

We take stereo matching as a showcase for performance of our hardware. Stereo matching extracts depth information from a stereoscopic image pair with different viewpoints. This fundamental problem has been studied for many years in computer vision and appears in a wide variety of applications with challenges for accuracy and speed. For example, desire for accurate video-rate stereo matching comes from applications as diverse as gesture recognition [22] and automotive navigation [23], since both depend for their quality of service on accuracy and speed of the stereo matching. We first show that our TRW-S based MRF inference hardware achieves significant speed-up over a conventional software implementation and outperforms BP based custom hardware or GPU implementations in terms of both quality of inference and speed.

Second, we further improve speed-up of our TRW-S based stereo matching hardware to demonstrate video-rate stereo matching using a hybrid CPU+FPGA platform. The goal is

to achieve high frame rate for the stereo video sequence while maintaining the accuracy of TRW-S. We propose several frame-level optimization techniques that fully utilize capability of our hybrid CPU+FPGA platform; the techniques are to properly reuse results from the previous frame and overlap CPU and FPGA execution times for processing multiple frames in parallel. From experimental results on a famous stereo matching benchmark suite as well as real-world stereo video samples, we demonstrate that our TRW-S based MRF inference system is able to satisfy the demanding requirement of high-quality video-rate stereo matching for real-world applications.

Third, we study the performance of the TRW-S architecture when subject to nanometer imperfections. We first analyze the error propagation characteristics of a message passing algorithm. Then, we explore the impact of error resiliency techniques when applied to MRF inference applications. We utilize our TRW-S based stereo matching hardware architecture as a test case, and explore its error resilience as well as the impact of a popular SEC technique, algorithmic noise tolerance (ANT) for error resilient and energy efficient probabilistic inference.

Last, we extend our TRW-S hardware toward a general purpose machine learning framework. To support a wider variety of computer vision applications beyond stereo matching, we propose an advanced streaming architecture with flexible choice of MRF setting. We perform a thorough evaluation of the performance of our hardware in order to demonstrate its superiority to our prior implementation. Furthermore, we provide better theoretical understanding of error resiliency of TRW-S, and of the implication of ANT for TRW-S, under more general MRF setting, along with strong empirical support.

1.1 Related Work

1.1.1 Taxonomy of probabilistic inference platforms

There have been numerous implementations of probabilistic inference using various types of platforms. Table 1.1 categorizes previous work into these types. General purpose CPUs are the most common option for implementation of probabilistic inference because of flexi-

ble software (SW) implementation. However, such flexibility incurs significant overhead in computation, leading to inefficiency in energy usage. Thus, the general purpose CPUs are often used for evaluating various probabilistic inference algorithms [10, 24] or realizing a large scale application using datacenter level computing power [25].

On the other hand, there have been application specific implementations for probabilistic inference algorithms. In this case, the implementation often exploits application-specific characteristics to optimize speed or energy efficiency of computation. In the case of stereo matching, researchers have simplified message computation and customized scheduling of the message passing to reduce memory access and achieve high throughput. [1, 2, 3] and [26] utilized graphic processing unit (GPU) and FPGA for parallel implementation of BP and TRW, respectively. Also, [27, 28, 1] designed ASICs for high throughput and energy efficient implementations of BP. These BP and TRW based inference hardwares achieved high throughput in message computation thanks to their parallel processing elements. However, they suffer inferior convergence speed compared to TRW-S implementation, resulting in slower performance in practice. Also, aggressive customization of message computation in these implementations has restricted their application to stereo matching.

To achieve better usability, speed and energy efficiency, researchers have proposed configurable hardwares that are implemented in FPGA. [29] demonstrated remarkable performance on probabilistic inference with directed acyclic graphs (DAGs). Also, [30] proposed Gaussian BP based MRF inference hardware for signal processing application. These configurable platforms allow flexible setup for various problems, but do not cover MAP problems which are not well mapped to both DAGs and Gaussian BP. GraphGen in [31] and GP5 in [32] reported wide coverage for MAP problems, but their performance is not optimal.

In this work, we propose a custom hardware strategy that can achieve the best performance out of available hardware resources (FPGA slices and memory bandwidth) for wide range of computer vision applications.

Table 1.1: Taxonomy of probabilistic inference platforms.

Application Specific HWs: Stereo Matching	Configurable Hardwares	General Purpose CPUs
FPGA [26] GPU [1, 2, 3] ASIC [27, 28, 1]	Bayesian computing machine [29] Factor graph processor [30] GP5 [32] GraphGen [31]	Middlebury API [10] OpenGM [24] GraphLab [25]

1.1.2 Error resiliency of probabilistic inference

Previously, SEC was studied for signal processing, such as PN code acquisition filter [4] and Viterbi decoders[33]. Previous work [34] proposed to apply algorithmic noise tolerance (ANT), which is a popular technique of SEC, to low density parity check (LDPC) decoder with BP as a core inference for error resiliency and energy efficiency. In this work, we extend the results of [34] to a more general MRF message passing inference, with a more complex graph and a larger hypothesis space in order to cover a wide range of machine learning applications.

1.2 Thesis Contribution

The list of our core contributions is as follows:

1. Our TRW-S based MRF inference hardware is 34.5~49 times faster than a standard SW implementation. It also outperforms other BP based ASIC/GPU implementations in terms of quality of result and speed.
2. Our TRW-S based MRF inference system implemented on a Convey HC-1[6] performs stereo matching of a QVGA-size mixed-scene 168-frame video stereo sequence in 7.34 seconds, or equivalently, with a frame rate of 22.9 frame per second.
3. SEC has been applied to our TRW-S stereo matching hardware architecture. Analysis and simulations show that for a 20-bit architecture, small errors ($e \leq 1024$) are tolerable, while large errors ($e \geq 4096$) degrade the performance significantly. By applying ANT, experimental results show that the proposed ANT based hardware can tolerate

an error rate of 21.3%, with performance degradation of only 3.5% compared to an error-free full precision hardware with an energy savings of 39.7%.

4. We exploit MRF inference as a general framework for machine learning to extend our scope to variety of computer vision problems. To this end, we propose a novel hardware architecture that can support various MRF configurations for solving computer vision problems, which achieves $10 - 40\times$ speedup over a conventional SW implementation for stereo matching, image denoising, and object segmentation benchmarks ([10, 24]) without sacrifice in inference quality. Also, we extend our understanding of error resiliency of TRW-S message passing and the impact of ANT under more general MRF settings, providing asymptotic bounds on error propagation and analyzing implication of ANT for the error resiliency.

1.3 Dissertation Organization

This dissertation is organized as follows:

- **Chapter 1** has introduced promises and challenges of MRF based inference for machine learning problems and motivation to build a custom hardware accelerator for them, as the stereo matching problem for a test application.
- **Chapter 2** reviews how stereo matching is formulated in an MRF framework and solved by message passing based inference algorithms, and highlights inference advantages of TRW-S over BP.
- **Chapter 3** discusses issues of hardware implementation of TRW-S and proposes an efficient hardware architecture.
- **Chapter 4** offers frame-level optimization techniques considering both the inference algorithm and the implementation platform to further improve performance of the TRW-S hardware for our hybrid CPU+FPGA platform.

- **Chapter 5** applies SEC to the TRW-S hardware to enhance error robustness and save power consumption.
- **Chapter 6** extends our TRW-S hardware to more general purpose computing for machine learning applications. Furthermore, theoretical analysis on the error resiliency of message passing inference for a more general MRF inference setting is provided.
- **Chapter 7** provides brief conclusions.

Chapter 2

Background

In this chapter, we introduce fundamental concepts of MRF inference and SEC. We first explain how a stereo matching problem can be reformulated as an MRF energy minimization and solved by MRF inference. To this end, we discuss construction of MRFs as well as motivation of using TRW-S over BP for MRF inference. Next, we discuss SEC and one of its important variants called algorithmic noise tolerance (ANT). We also introduce our target platform, the Convey HC-1 hybrid-core computing device.

2.1 Markov Random Field Inference for Stereo Matching

2.1.1 MRF formulation of stereo matching problem

The goal of stereo matching is to find matching pixels from a pair of stereo images and statistically infer depth based on their pixel-wise horizontal displacement, which is inversely proportional to depth. Thus, stereo matching can naturally be formulated as a maximum a posteriori (MAP) discrete-label inference problem [35], where we seek the most probable displacement (i.e., a set of disparity labels), $x = \{x_s, s \in \mathcal{V}\}$, given stereo images as observation, y , written as:

$$\arg \max_x P(x|y) = \arg \max_x P(y|x)P(x). \quad (2.1)$$

x_s is a label in a discrete domain χ associated with pixel s in the reference image, and \mathcal{V} corresponds to all the pixels. $P(x|y)$ is called the posterior, and it is rephrased as a product of the likelihood, $P(y|x)$, and the prior, $P(x)$, using Bayes rule.

Of great practical interest are the cases wherein $P(y|x)$ and $P(x)$ can be defined by a

product of small functions. In particular, if the likelihood is determined by a pixel-wise relationship between the label and the observation, and the prior by a pairwise relationship between labels of two adjacent pixels, then we can formulate them as a product of factors:

$$P(y|x)P(x) \propto \prod_{s \in \mathcal{V}} \phi_s(x_s, y_s) \prod_{(s,t) \in \mathcal{E}} \phi_{st}(x_s, x_t) \quad (2.2)$$

$$\propto \exp \left(- \left(\sum_{s \in \mathcal{V}} \theta_s(x_s, y_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \right) \right), \quad (2.3)$$

where factors ϕ_s and ϕ_{st} represent the likelihood of node s and the prior of edge (s, t) , respectively. These factors are defined in an MRF, which is an undirected graph, $G = (\mathcal{V}, \mathcal{E})$, with a set of nodes (i.e., all the pixels), \mathcal{V} , and a set of edges, \mathcal{E} , corresponding to the random variables and the statistical relationship between nodes, respectively. Fig. 2.1 shows an example of representing a posterior in a pairwise 4-connected (grid) MRF. Note that the factors are represented in the log space as $\theta = \{\theta_s, \theta_{st}\}$ in (2.3) to replace the product with summation for less expensive arithmetic. We call $E(x) = \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t)$ the energy function over x defined on an MRF with parameters θ . Then the original MAP problem is formulated as an energy minimization problem on the MRF as follows:

$$\arg \min_x E(x) = \arg \min_x \left\{ \sum_{s \in \mathcal{V}} \theta_s(x_s) + \sum_{(s,t) \in \mathcal{E}} \theta_{st}(x_s, x_t) \right\}, \quad (2.4)$$

where $\theta_s(x_s)$ and $\theta_{st}(x_s, x_t)$ are parameters which penalize certain choices of labels x , and are called the data cost and the smoothness cost, respectively. The goal of (2.4) is to find the set of labels x that minimizes the overall summation over $\theta_s(x_s)$ and $\theta_{st}(x_s, x_t)$ terms (i.e., energy) among all possible per-pixel label choices.

2.1.2 MRF parameter computation

As prologue to MRF inference, we must first compute the MRF parameters (i.e., data cost and smoothness cost) defined in (2.4). For stereo matching, the data cost models

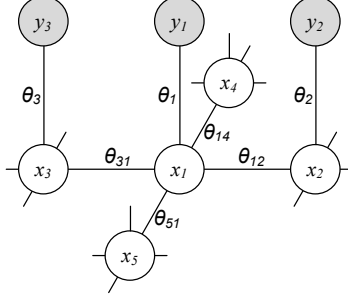


Figure 2.1: Representation of a posterior in a pairwise 4-connected (grid) MRF.

the (dis)similarity of color between a pixel in one image and each of several horizontally displaced pixels in the other image. The disparity information we seek (called a disparity map) is recovered from this horizontal separation (i.e., where a left-image pixel is to be found in the right image). We employ the sampling insensitive matching model of [36] for the data cost. When comparing dissimilarity of one pixel s in the left image I_L and a pixel $s' = (s - x_s)$ displaced by distance x_s in the right image I_R , we first compute interpolated pixel intensity values for neighboring pixels as follows:

$$I_L^-(s) = \frac{1}{2} (I_L(s), I_L(s - 1)), \quad (2.5)$$

$$I_L^+(s) = \frac{1}{2} (I_L(s), I_L(s + 1)), \quad (2.6)$$

$$I_R^-(s') = \frac{1}{2} (I_R(s'), I_R(s' - 1)), \quad (2.7)$$

$$I_L^-(s') = \frac{1}{2} (I_L(s'), I_L(s' - 1)). \quad (2.8)$$

These interpolated intensity values are used to compute the intensity difference of two pixels s and s' as follows:

$$di_L(s, s') = \max \{0, I_L(s) - I_R^{max}(s'), I_R^{min}(s') - I_L(s)\}, \quad (2.9)$$

$$di_R(s, s') = \max \{0, I_R(s') - I_L^{max}(s), I_L^{min}(s) - I_R(s')\}, \quad (2.10)$$

where I^{min} and I^{max} are $\min \{I^-, I^+, I\}$ and $\max \{I^-, I^+, I\}$, respectively. Then, the dissimilarity vector for the node s (which contains one element for each possible disparity label) is computed as ($k=1$ or 2):

$$\theta_s(x_s) = \min \left\{ di_L(s, s - x_s)^k, di_R(s, s - x_s)^k \right\}. \quad (2.11)$$

Note that $\theta_s(x_s)$ is a vector of costs measured from pixel s in the left image to the horizontally displaced pixel $s' = s - x_s$ in the right image. It is shown in [37] that this matching cost is well suited to MRF inference since it corrects blurring effects.

The smoothness cost models the fact that neighbor pixels are more likely to have the same (or similar) depth, because most pixels are not on the boundary in the disparity map. We use truncated smoothness functions with image boundary information considered as follows:

$$\theta_{st}(x_s, x_t) = w_{st} \cdot \min \left\{ |x_s - x_t|^k, V_{max} \right\}, \quad (2.12)$$

where k is 1 or 2, V_{max} is the maximum smoothness cost, and w_{st} is a per-edge weight for gradient cue based boundary information. The smoothness cost is limited by V_{max} in order to allow disagreement in labels between two nodes across the object boundary in the image. The per-edge weight has a value of 1 only for the case where the gradient of image intensity is larger than a threshold, which implies the existence of the boundaries. Note that w_{st} exists for every edge. Thus there are weights in four directions from one node. It is shown in [35] that considering lower-level visual cues such as object boundary information in the smoothness cost improves stereo matching. Thus, we use truncated functions with the gradient cues as our smoothness cost.

2.1.3 MRF inference methods: BP vs. TRW-S

Belief propagation (BP) is one of the most popular algorithms to solve MRF energy minimization problems. In BP, a node propagates its belief of which label is the most probable to its neighbors by passing messages. Thus, a message is a vector of size $|x_s|$, where each value corresponds to the probability of a label being chosen. (Note that in energy minimization

formulation, we take $-\log$ to represent this vector as the “cost,” as described in (2.3). To infer the best label assignment, BP repeats messages passing along the edges of an MRF until the messages converge to a fixed point. (It is called “one iteration” when message passing covers the entire graph.) One can apply the message passing in any preferred order (e.g. all messages at the same time), and thus computation of BP can be well parallelized.

In case of a tree-structured MRF graph, BP is guaranteed to find the global optimal solution [38]. In case of a loopy graph, however, BP is often trapped in a local optimum due to double counting, a situation in which the belief of a node is propagated along a loop and passed back to itself, over-emphasizing its own belief [38]. In practice, BP often suffers from this effect of double counting, and thus the messages may oscillate or converge to obtain a final energy value much higher than the global minimum.

On the other hand, sequential tree-reweighed message passing (TRW-S) is designed to avoid double counting. In TRW-S, the original loopy graph is decomposed into a set of trees that cover the graph, and message passing is now based on these trees in order to reduce the effect of double counting, and there is some new tree-to-tree communication of evolving results. This tree decomposition leads to the lower bound of the energy minimization problem (2.4) as follows [17]:

$$\min_x E(x : \theta) \geq \sum_T \omega_T \min_x E(x_T : \theta_T), \quad (2.13)$$

where θ_T is a set of parameters defined on a tree T , and ω_T is a distribution on trees satisfying $\theta = \sum_T \omega_T \cdot \theta_T$. The lower bound (RHS of (2.13)) consists of energy minimization problems over the decomposed trees $\min_{x_T} E(x_T : \theta_T)$, which BP can find the optimal label assignment for each tree without double counting. If the best label assignments for the trees agree on all the nodes, the equality is satisfied in (2.13), and the corresponding assignment is the optimal solution of the original energy minimization problem. Furthermore, if message passing is performed sequentially, the lower bound in (2.13) is guaranteed not to decrease [17]. In other words, by passing messages sequentially, we can restrain the energy value from oscillating. In practice, therefore, TRW-S often finds lower minimum energy than BP [10]. As an example, Fig. 2.2 shows comparison of energy minimization performance between BP

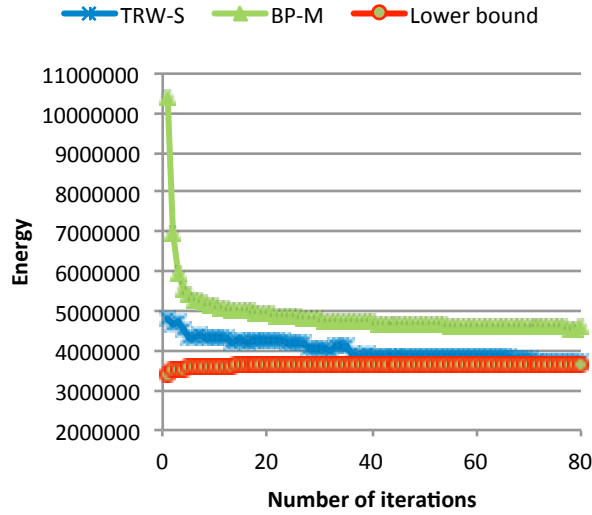


Figure 2.2: Comparison of energy minimization performance between BP and TRW-S for Tsukuba stereo matching benchmark [5].

and TRW-S for the Tsukuba stereo matching benchmark [5]. TRW-S obtains lower minimum energy than BP. The minimum energy of TRW-S is close to the lower bound, implying that this inference result is almost the global optimum.

To exploit this favorable convergence property, we use TRW-S for our inference algorithm. The two-step message passing rule for a pairwise grid MRF can be written as follows:

$$\bar{\theta}_p(x_p) = \theta_p(x_p) + \sum_{s \in Nb(p)} M_{sp}(x_p), \quad (2.14)$$

$$M_{pq}(x_q) = \min_{x_p} \left\{ \frac{1}{2} \theta_p(x_p) - M_{qp}(x_p) + \theta_{pq}(x_p, x_q) \right\}, \quad (2.15)$$

where $\bar{\theta}_p(x_p)$ and $M_{pq}(x_q)$ are the reparameterize step and the message update step, respectively. A ratio of 1/2 is multiplied to the reparameterization $\bar{\theta}_p(x_p)$ to compensate double counting of nodes in the row-column decomposed graph, where a row and a column trees duplicate each node. The convergence property of TRW-S is guaranteed by executing (2.14)

and (2.15) for all the nodes sequentially, but this sequential nature of computation obstructs parallelization of the algorithm.

2.2 Statistical Error Compensation

Statistical error compensation (SEC) techniques, such as algorithmic noise tolerance (ANT) [39], employ statistical estimation and detection techniques to compensate for errors approximately. Thus, they are best suited for applications where the performance metrics themselves are statistical. When SEC techniques are applied to these applications, significant power savings, e.g., up to 67% for an FIR filter [39], and robustness enhancement can be achieved, compared to logic/architectural level techniques such as Razor.

A high level depiction of SEC is given in Fig. 2.3(a). SEC utilizes the statistics of errors to perform detection and estimation to compensate for errors. It also incorporates system level statistical metrics, such as signal-to-noise ratio (SNR), or bit error rate (BER). SEC operates on multiple observations, where each observation is generated by erroneous hardware, an error free estimator, or an erroneous estimator. Each observation y_i is a corrupted version of the correct output y_o , i.e., $y_i = y_o + \eta_i + e_i$, where η_i denotes hardware errors and e_i denotes estimation errors. Based on these observations, detection and estimation techniques are employed in conjunction with the statistical information of η_i and e_i to obtain the output most likely to be correct. Errors that have a large effect on the system level performance are detected and compensated, while errors with minimal effect on performance are considered benign and permitted.

2.2.1 Algorithmic noise tolerance

Statistical error compensation (SEC) in the form of algorithmic noise-tolerance (ANT) [39, 33] in Fig. 2.3(b) incorporates a *main* block and an *estimator*. The main block, unlike the estimator, is permitted to make hardware/timing errors. The estimator is a low-complexity block (typically 5%-to-20% of the main block complexity) generating a statistical estimate

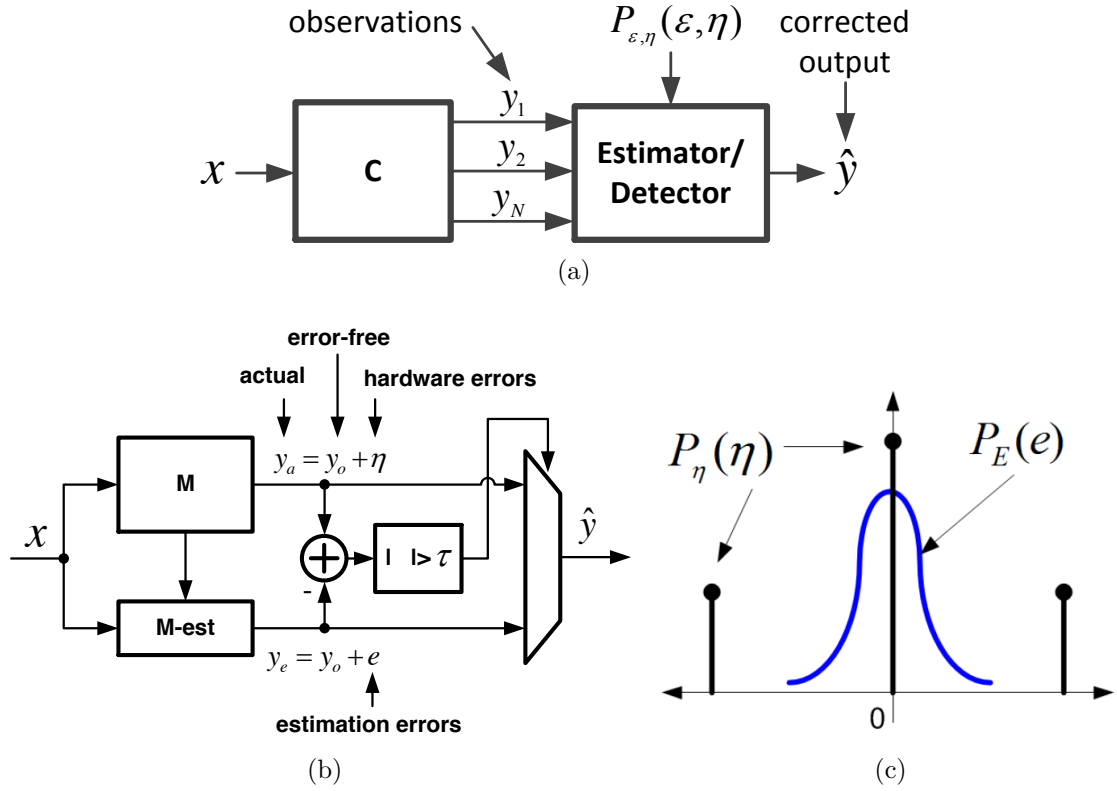


Figure 2.3: Statistical error compensation: (a) general form, (b) algorithmic noise tolerance, and (c) error distributions.

of the correct main block output, i.e.,

$$y_a = y_o + \eta \quad (2.16)$$

$$y_e = y_o + e, \quad (2.17)$$

where y_a is the actual main block output, y_o is the error-free main block output, η is the hardware error, y_e is the estimator output, and e is the estimation error. Note that the estimator exhibits estimation error e because it is simpler than the main block. ANT exploits the difference in the statistics of η and e (see Fig. 2.3(c)). To enhance robustness, it is necessary that when $\eta \neq 0$, η be large compared to e . In addition, the probability of the event $\eta \neq 0$ must be small. The final/corrected output of an ANT system \hat{y} is obtained via

the following decision rule:

$$\hat{y} = \begin{cases} y_a, & \text{if } |y_a - y_e| < \tau \\ y_e, & \text{otherwise,} \end{cases} \quad (2.18)$$

where τ is an application-dependent parameter chosen to maximize the performance of ANT. Under the conditions outlined above, it is possible to show that

$$SNR_{uc} \ll SNR_e \ll SNR_{ANT} \approx SNR_o, \quad (2.19)$$

where SNR_{uc} , SNR_e , SNR_{ANT} and SNR_o are the signal-to-noise ratios of the uncorrected main block (η dominates), the estimator (e dominates), the ANT system, and the error-free main block (ideal), respectively. Thus, ANT detects and corrects errors approximately, but does so in a manner that satisfies an application-level performance specification (SNR). Several low-overhead estimation techniques have been proposed by exploiting data correlation, system architecture, and statistical signal processing [21].

Conventional fault-tolerant techniques focus on providing complete correctness while in operation. However, communication-inspired algorithmic noise-tolerance (ANT) techniques [20] utilize the fact that some applications are tolerant to small errors and show significant improvement in robustness while providing energy efficiency. This is done not by concentrating on error-free output, but rather by trying to meet the signal-to-noise ratio (SNR) or bit error-rate (BER) specification of the application. The communication-inspired approach treats nanometer circuit fabrics as a noisy channel, and faults and errors resulting from this channel are addressed through statistical signal processing techniques that were primarily used in communication systems for several decades.

2.3 Hybrid Multicore + FPGA Platform

Our implementation targets a Convey HC-1 hybrid-core computing system containing an Intel Xeon dual core processor and four Xilinx Virtex-5 (LX330) FPGAs [6]. Fig. 2.4 shows the overall architecture of Convey HC-1. The FPGA fabric consists of three major parts:

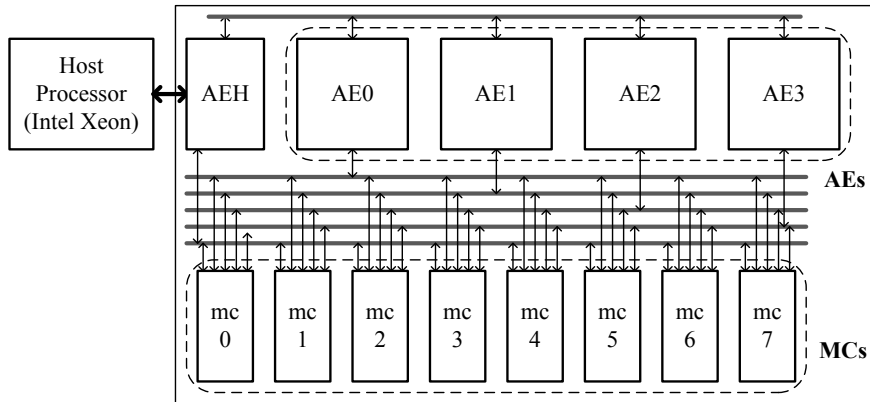


Figure 2.4: Overall architecture of Convey HC-1 [6].

the application engine hub (AEH) that interfaces the Xeon to the FPGAs; the memory controllers (MCs); and the FPGAs themselves, in which Convey refers to as application engines (AEs). Each FPGA is connected to eight MCs, which control 16- DRAM DIMMs to provide 20-GB/s (or 1-kb/cycle) of memory bandwidth per FPGA. This memory bandwidth will be fully utilized in Chapter 3 to achieve the maximum speedup for TRW-S inference.

The platform provides a single, cache-coherent virtual memory system across the host processor, and the FPGA fabric [40]. The data transfer between the host and the FPGAs is expedited using a dedicated data mover, and thus any data written by the host processor core can be used in any FPGA through the front-side bus and AEH, and vice versa. The platform also supports nonblocking calls for the FPGA functions [6], which allows overlap of the execution time for the CPU and FPGA functions. These capabilities offer a broad space of frame level optimization for video stereo matching; we shall explore this in Chapter 4.

2.4 Summary

This chapter has shown that a stereo matching problem can be formulated as an energy minimization over an MRF. Further, we reviewed two message passing based inference algorithm which can solve this energy minimization problem: belief propagation (BP) and sequential

tree-reweighted message passing (TRW-S). We showed that TRW-S has a favorable convergence property which BP lacks, and that TRW-S outperforms BP in practice. The downside of TRW-S is its inherent sequential manner of computation which makes parallelization challenging. In Chapter 3, we will show a novel way to expose necessary parallelism in the computation of TRW-S.

Chapter 3

Hardware Implementation of Sequential Tree-Reweighted Message Passing

Our MRF-based stereo matching consists of the sub-functions shown in Table 3.1. We consider hardware implementation of both the message passing algorithm of TRW-S and the MRF parameter computation, since these two steps consume most of the execution time. For example, the TRW-S inference and the MRF parameter computation take 73.3% and 21.3%, respectively, of the total CPU execution time of stereo matching for the Tsukuba task. We will show in this chapter that both the MRF parameter computation and TRW-S inference can be massively parallelized to achieve order of magnitude speed-up by hardware accelerators implemented in FPGAs.

3.1 Hardware Architecture of MRF Parameter Computation

The MRF parameter computation consists of calculating matching cost for the data cost term and finding gradient cues for the smoothness cost (so, this hardware is called COM_COST). The matching cost and the gradient cues are computed independently using a Matching Cost

Table 3.1: List of stereo matching sub-functions.

Function name	Partition	Description
RD_IMGS	CPU	Get stereo image pair
COM_COST	FPGA	Compute MRF parameters
STRM-TRWS	FPGA	Run TRW-S inference
GET_LABEL	CPU	Obtain the best labeling result
WR_DISP	CPU	Produce disparity map

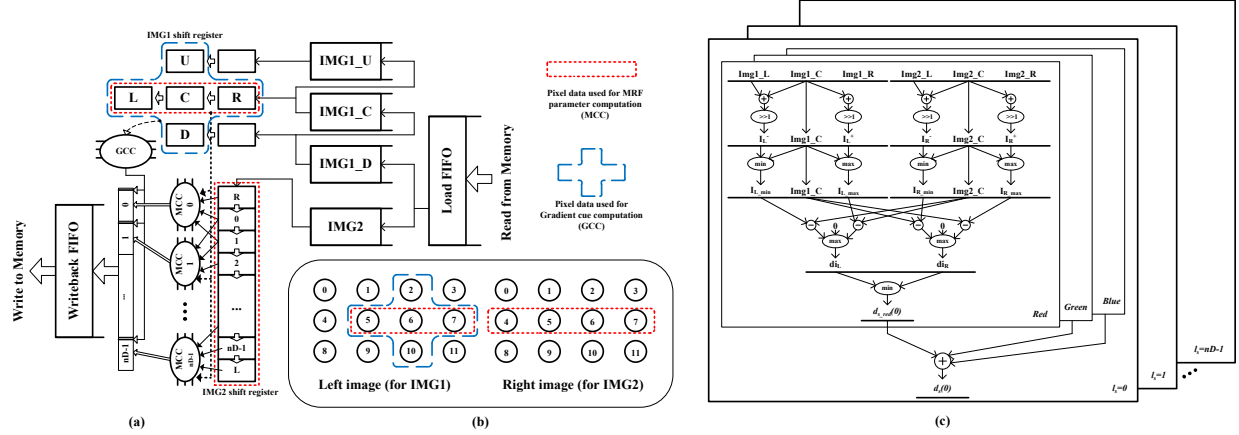


Figure 3.1: (a) Overall architecture of matching cost computation unit (MCC) and gradient cue computation unit (GCC). (b) Example of image data. Nodes (circles with numbers) inside the red dotted line and blue dotted line are used as input for MCC and GCC, respectively. (c) Detailed architecture of MCC.

Computation unit (MCC) and a Gradient Cue Computation unit (GCC), as shown in Fig. 3.1(a). To exploit the maximum parallelism, the MCC is designed to compute difference of color intensity (red, green, blue) for all the labels in parallel, and the GCC is designed to compute gradients of each color intensity in for directions (right, down, left, up) in parallel. Thus, the MCC reads three (left, center, right) pixel data (i.e., color intensity) from the left image and $nD + 2$ pixel data from the right image for the nD label data cost. The GCC reads five pixels of color (RGB) data to compute gradient of color in four directions from the center node.

FIFOs and shift registers are utilized to feed the required pixel data from the left (IMG1) and the right images (IMG2). Fig. 3.1(a) shows the overall architecture of the MCC and the GCC. The (red, green, blue) pixel data for both left and right images are loaded the load FIFO and fetched into the IMG1 and IMG2 FIFOs. There are three FIFOs for IMG1: IMG1_U, IMG1_C, and IMG_D. These IMG1 FIFOs are managed to temporarily store the pixel data for the up, center, and down rows of the left image, which are fetched from the memory in row-major order. For example, if all the pixels in the left image of Fig. 3.1(b) are loaded, IMG1_U, IMG1_C, and IMG_D will contain data for pixels $\{0,1,2,3\}$, $\{4,5,6,7\}$,

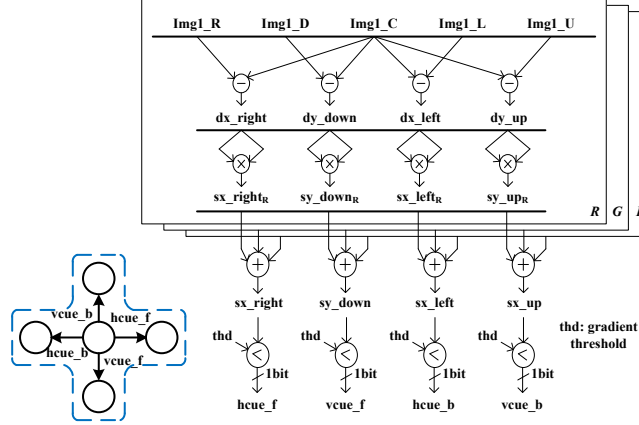


Figure 3.2: Detailed architecture of GCC.

and $\{8,9,10,11\}$, respectively. Also, the IMG2 FIFO temporarily stores the pixel data from the right image. Depth of the IMG1 and IMG2 FIFO is set as the width of the image to align the pixel data by the rows of the image.

The pixel data stored in the IMG1 and IMG2 FIFOs are used to fill the shift registers. The depth of the IMG1 shift register is 3, since the MCC and the GCC require only the center pixel from IMG1 along with its right and left neighbors. On the other hand, the depth of IMG2 shift register is $nD + 2$, since the MCC requires three (left, center, right) pixel data for each nD label. For example, if $nD = 2$ and the pixel 6 in Fig. 3.1(b) is the center pixel, the MCC_0 and the MCC_1 read $\{5,6,7\}$ and $\{4,5,6\}$ for IMG2 to compute $\theta_6(0)$ and $\theta_6(1)$, respectively.

Figure 3.1(c) shows the detailed hardware architecture of the MCC. As discussed before, the matching costs of each color for all the labels are computed in parallel, and then merged at the last stage to generate a vector of data cost with the size of nD . The computations are equivalent to equations from (2.5) to (2.11) in Section 2.1.2, which are pipelined to maximize throughput as well as to reduce the critical path.

Figure 3.2 shows the detailed architecture of the GCC. The color values are subtracted (dx_right , dy_down , dx_left , and dy_up) and squared (sx_right , sy_down , sx_left , and sy_up) to find the gradient. The computations for the four directions for the three colors

are done in parallel, and then merged at the last stages. Note that the actual value of the smoothness cost as well as the per-edge weight are not computed. This is because the smoothness cost is a function of the labels and the per-edge weight, and the weight is an on-off function so that it has non-unity value only if the gradient value is less than the threshold. Thus, we can simply store only the one bit signals to indicate whether the per-edge weight needs to be applied or not. In this way, we can again reduce the size of the input data.

The computed matching costs for a node are packed with the four bit gradient cue flags, and then they are put in the write back FIFOs to dump into the memory. These MRF parameters are used in the inference engine to compute the most probable labels.

3.2 Design Issues in Hardware Implementation of TRW-S

In this section, we discuss design issues that are critical in our hardware implementation of TRW-S: floating point to fixed point conversion and parallelization of message passing.

3.2.1 Floating point to fixed point conversion

TRW-S has floating point arithmetic inherently in the algorithm, as accounted by the fractional constant in [17]. This fractional constant is set to $1/2$ in (2.15) for the case of a 2-D grid MRF, which can be implemented with a simple bit shift instead of a floating point arithmetic unit.

We determine the proper number of fractional bits (FB) to convert floating point arithmetic in the software implementation of TRW-S [10] to fixed point arithmetic. Fig. 3.3 shows the energy minimization results of TRW-S for different FB values. As the diamond-mark curve indicates, the energy minimization results are not too sensitive to FB.

However, the lower bound shows more sensitivity to FB. Note that the lower bound is related to the convergence of TRW-S, since the higher the lower bound, the closer to the optimal label assignment. For the Tsukuba stereo benchmark ([10]), TRW-S is converged with FB of 7 and 8, but not converged with 6 or smaller FB. Thus, we choose 8-bit as FB,

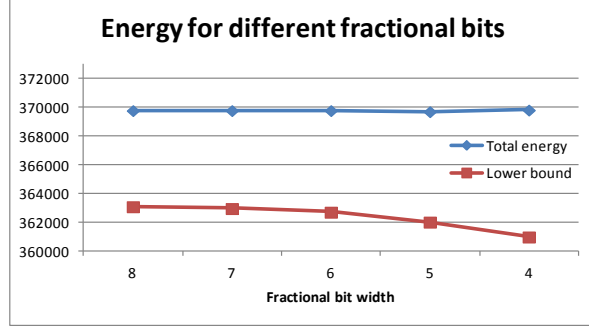


Figure 3.3: Energy minimization results of TRW-S for different fractional bits.

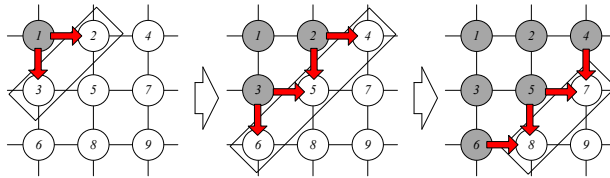


Figure 3.4: Diagonal ordering for parallelization of TRW-S.

which includes the minimum bit width (= 7-bits) plus 1-bit as a guard bit. Experiments with other benchmarks (Teddy and Venus, from the standard Middlebury suite [10]) show that this choice of FB increases at most 0.23% of the minimum energy.

3.2.2 Parallelization of TRW-S

As discussed in Section 2.1.3, parallelization of TRW-S is not straightforward due to its sequential message passing. Since the newly updated messages are used in the next message passing, there is data dependency between neighboring nodes in the tree decomposed MRF. Therefore, in the original implementation of TRW-S, the message passing is done in row-major order.

However, if we smartly choose an update order, we can more aggressively parallelize message passing on the monotonic chains. In Fig. 3.4, we show a novel *diagonal ordering* that is suitable for parallelization. The number inside the node shows the diagonal ordering of TRW-S for the forward pass, and the arrows indicate the messages passed after each belief update of a node. As we can see in the figure, message passing in a diagonal row (grouped in

rectangles) only depends on the messages passed from the previous row (arrows). Therefore, we can perform message passing for the nodes in the same diagonal row in *parallel*. This exposes significant new opportunities for acceleration.

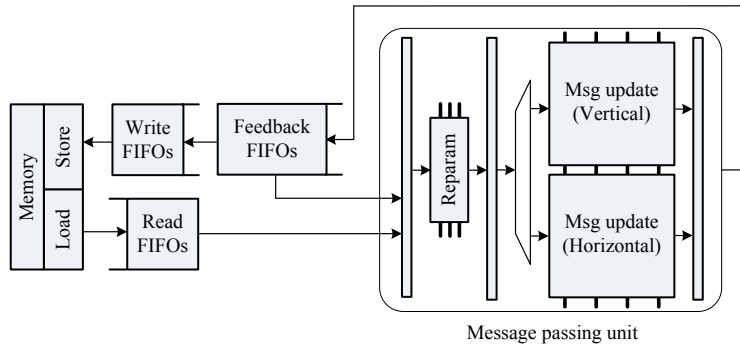
3.3 Hardware Architecture of TRW-S

There are two broad options, given the diagonal order scheme from Fig. 3.4. We might try to build many parallel computational units, each dedicated to handling one pixel node. However, as we quickly discovered, the application is extremely demanding of DRAM memory bandwidth: message passing for each node involves computations for the complete set of labels for each pixel, which means that each node is producing and consuming kilobits of data for its message passing. To render this tractable, we have devised a streaming architecture that consists of a single, deeply pipelined message passing unit and a set of FIFO memory interfaces. The deep pipeline allows us to have many (14, in our final design) pixels “in flight” in each clock period, and to start and retire a complete pixel and all its message computations in each clock tick. The FIFO memory interface simplifies control over the data dependency of message passing between contiguous diagonal rows.

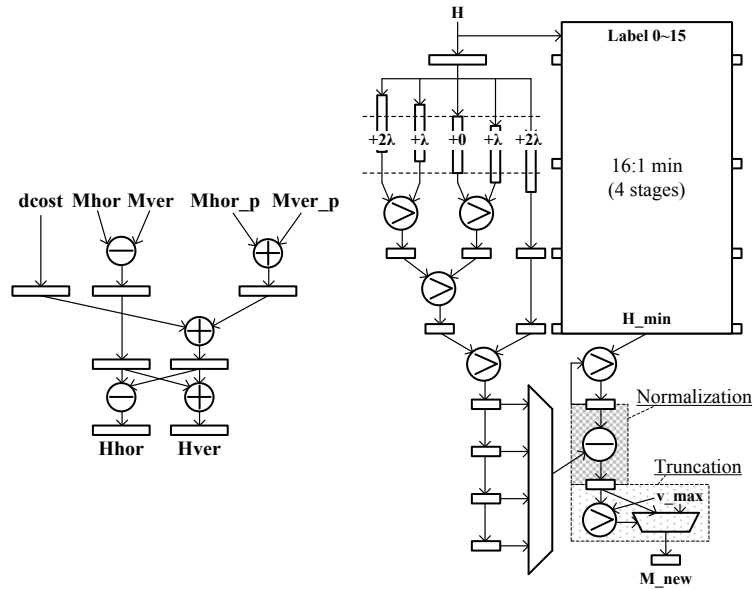
3.3.1 Overall architecture

The overall architecture of our TRW-S hardware is shown in Fig. 3.5(a), which consists of the pipelined message passing unit along with three types of FIFOs: Read FIFOs, Feedback FIFOs, and Write FIFOs. The Read FIFOs consist of FIFOs for data cost, and horizontal and vertical messages. The Feedback FIFOs consist of previously computed horizontal and vertical messages. The Write FIFOs consist of two write back buffers. The role of these FIFOs is to maintain a stream of data to be processed by the message passing unit.

The Read FIFOs keep reading data from the memory. These data are processed in the message passing unit to generate new messages. The Write FIFOs are used to store the new messages temporarily. If one of them becomes full, it starts dumping messages to the



(a)



(b)

(c)

Figure 3.5: Streaming TRW-S hardware architecture: (a) overall architecture, (b) reparameterize unit, (c) message update unit.

memory. (In the meantime, data fetching from the memory to the Read FIFOs is stalled.) The Write FIFOs employ double buffering to buffer the new messages and dump the stored messages simultaneously, which allows the message passing unit to keep working. The newly computed messages are also temporarily cached to the Feedback FIFOs. These messages are necessary in the message passing of the subsequent nodes. The Feedback FIFOs make the recently computed messages available right away, so that the message passing unit can avoid stalls.

Since this FIFO memory interface and the pipelined message passing unit perform in a decoupled, streaming manner, we refer to this hardware architecture as streaming TRW-S (STRM-TRWS). Memory access by FIFOs and message passing by the processing element are done in parallel to exploit as much memory bandwidth as possible.

3.3.2 Pipelined message passing unit

The message passing unit consists of one reparameterize unit (Reparam) and two message update units (MsgUpdate). As shown in Fig. 3.5(b,c), both processing units are fully pipelined by stages. The Reparam computes (2.14) to generate new reparameterization, which is used in the two MsgUpdate for computation of new messages (2.15) in horizontal and vertical directions. Our message update unit supports truncated linear and quadratic cost, and Potts models for the pairwise parameters [10].

We employ two hardware optimizations for efficient computation. First, we exploit a parallel message construction technique ([1]) that exploits the truncated form of smoothness cost functions and limits the search space for optimization in message computation (2.15) to reduce the complexity from $O(|\chi|^2)$ to $O(|\chi|)$ ($|\chi|$: number of assignment labels). Second, since the messages for all labels are computed in parallel, data are processed in a block manner. To deal with the case that the size of a message vector is larger than the memory bandwidth (i.e., it takes more than 1 cycle to access data for a message vector from the memory), a *folded* pipeline architecture ([41]) is employed for our message passing unit, where a long message vector is “folded” into multiple blocks to take multiple cycles for processing it. For example, if $|\chi| = 20$ and the block size $B = 16$, it takes two cycles to

Table 3.2: Device utilization summary.

Unit	COM_COST	STRM-TRWS
Slice Register	20,667	25,325
Slice LUT	21,320	25,591
Slice LUT FF	60	0
36Kb Block RAM	32	90

process a message, and thus the folding factor is two.

3.4 Experimental Results

In this section, we present performance of our TRW-S inference system. This system consists of software run by CPU for data management (e.g., image read/write) and overall control, as well as MRF computation and inference hardware implemented on FPGAs, and performs stereo matching benchmarks. Our implementation runs entirely on the Convey HC-1 platform; the host processors and all FPGA functions are run at 2.13GHz and 150MHz, respectively.

3.4.1 Hardware specification

Table 3.2 shows the device utilization summary of our FPGA implementation. The following design choices were made for this implementation.

In STRM-TRWS, 15-bit data cost and 24-bit messages in horizontal and vertical directions are packed as 63-bit data for one label of one node. The experiments show that this choice of bit-width is appropriate to avoid overflow in computation. Given the memory bandwidth of 1-Kbit/cycle, data of up to 16 labels can be fetched from or written back to the memory every cycle. By employing the folded pipeline architecture (folding factor of 4), the hardware can support up to 64-label stereo matching. The remaining 16 out of 1024 bits are used to control the data path. The depth of FIFOs is determined as follows. The depth of the Read and Write FIFOs are set to 512; a larger depth allows more pending data, in case

Table 3.3: Execution cycle and time of streaming TRW-S for stereo matching.

Task	Image size	#labels	Smoothness Cost	STRM-TRWS		SW([10])
				Cycle/#iter	second	second
Tsukuba	384x288	16	Truncated linear	478,134	0.0032	0.12
Venus	434x383	20	Truncated quadratic	1,436,257	0.0096	0.47
Teddy	450x375	60	Potts model	2,914,599	0.0194	0.67

the memory suffers long latency. The depth of the Feedback FIFOs is determined by the maximum height of the input image. Since the maximum height is set to 512, the depth should be $4 \times 512 = 2048$ (4 is the folding factor).

A similar memory interface is used in COM_COST. Furthermore, four 512-depth 32-bit width FIFOs (IMG1_U, IMG1_C, IMG1_D, IMG2) are used for buffering stereo image data to be used in MCC and GCC. In addition, 60 DSP48Es– Xilinx Virtex 5 blocks customized for efficient digital signal processing operations [42] – are used to compute square difference in color in MCC and GCC.

3.4.2 Experimental results

We first run three stereo matching tasks (Tsukuba, Teddy, and Venus, from the standard Middlebury suite [10]) in FPGA to evaluate the speed of our streaming TRW-S. Table 3.3 shows the execution cycle and time for one iteration. According to experiments, memory bandwidth utilization of our hardware architecture is about 93.1%. This shows that the FIFO interface for the message passing unit effectively hides the long latency of DRAM. As we can see in the table, the streaming TRW-S is 34.5~49 times faster than the speed of a reference software implementation of TRW-S [10] running on an Intel Core i7 machine. The disparity maps of stereo matching tasks obtained after 500 iterations and the ground truths are compared in Fig. 3.6. As we can see, the results of our hardware are almost identical to the software results, and look very similar to the ground truth.

Next, we evaluate function-by-function performance of our stereo matching system for

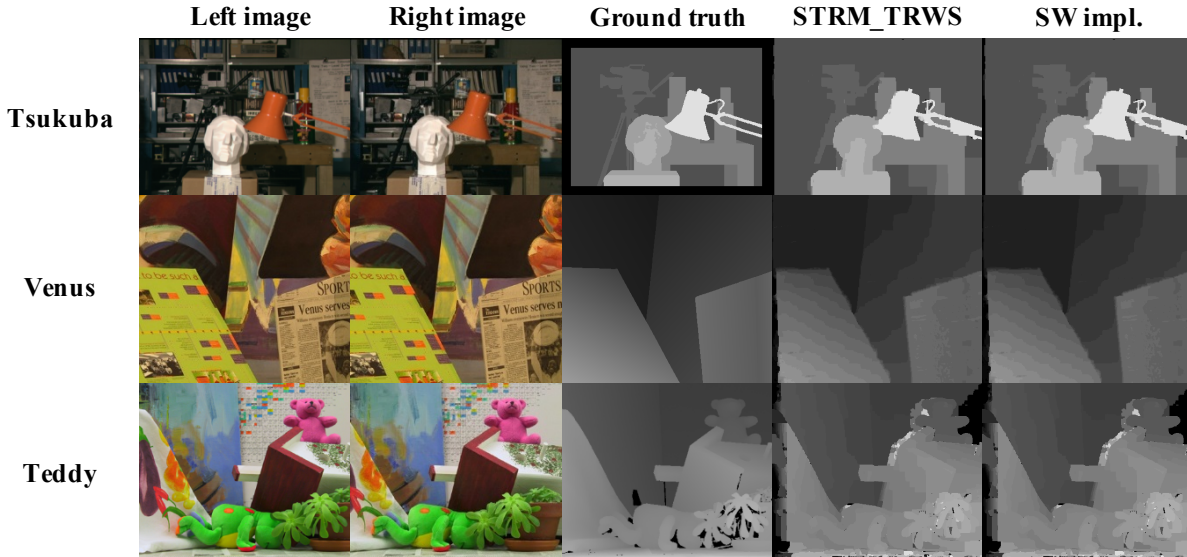


Figure 3.6: Comparison of disparity maps.

Table 3.4: Execution time (in ms) of stereo matching functions.

Tsukuba Stereo Matching Benchmark (384x288,16 labels) ([5])		
# Inference Iterations	5	40
RD_IMGS	22.3	22.4
COM_COST (CPU / FPGA)	146.9 / 4.5	147.0 / 4.5
STRM-TRWS (CPU / FPGA)	500.0 / 18.9	3720.0 / 127.1
GET_LABEL	1.9	1.9
WR_DISP	10.9	10.6
Minimum energy	394,434	370,359
Min. energy of VLSI impl. [1]	396,953 in 137.4 ms	

the well-known Tsukuba task. Table 3.4 shows the execution time of the stereo matching functions, with different number of iterations for the inference. As the MRF inference minimizes the energy over the iterations, the number of iterations can be either adaptively increased to achieve a desired minimum energy, or pre-determined for some fixed latency of the entire inference engine. In the latter case, the minimum energy is a metric to evaluate the quality of solutions, as is used in [10]. In this work, therefore, we demonstrate performance for our hardware in terms of the execution time (determined by a fixed number of iterations) and minimum energy.

Table 3.5: Comparison of speed between GPU implementations [1, 2, 3] and streaming TRW-S for Tsukuba task.

	Real-time BP [2]	Tile-based BP [1]	Fast BP [3]	Our HW
GPU platform	NVIDIA GeForce 7900 GTX	NVIDIA GeForce 8800 GTS	NVIDIA GeForce GTX 260	N/A
Number of iterations	(coarse to fine scales) = (5, 5, 10, 2)	(B, TI, TO) = (16, 20, 5)	(3 coarse to fine scales) = (9, 6, 2)	$T_o = 5$
Time (ms)	79.71	124.38	61.41	26.10

Since RD_IMGS, GET_LABEL, and WR_DISP functions are partitioned as CPU functions, they are processed in the host side. For COM_COST and STRM-TRWS functions, both CPU and FPGA execution time are measured for comparison. As shown in Table 3.4, the hardware implementation of COM_COST and STRM-TRWS run 32.6 and 26.5 times faster than the same functions run in the host processor, respectively. Thus, we can conclude that it is beneficial to accelerate both COM_COST and STRM-TRWS functions in the FPGA side.

Graphics processing units (GPUs) have also been widely explored in this application, given their obvious historical origins in the video universe. Hence, it is also worthwhile to examine comparisons between our FPGA implementation and those implementations. There are several notable efforts to use GPUs to accelerate variants of belief propagation algorithms for stereo matching [1, 2, 3]. (These include both older and newer GPU architectures.) Real-time BP ([2]) and Fast BP ([3]) exploit the idea of applying BP in a hierarchical manner (from coarse to fine scales, [43]) to speed up convergence of BP. In Tile-based BP ([1]), the entire graph is divided into small tiles; BP is performed on each tile but only messages on the boundary are kept to reduce memory access. Since all of these implementations are based on BP, there is no convergence property which leads closer to the optimum solution or the lower minimum energy, unlike TRW-S.

Table 3.5 shows execution time for each implementation for stereo matching of the standard Tsukuba single-frame benchmark with its own GPU platform and iteration settings.

Here, GPU execution time includes only the time for inference, whereas execution time of our system includes both data cost computation and inference processed in FPGA. As we can see, the execution time of our system is faster than the execution time of these other GPU implementations, although our hardware is running at a much slower clock frequency (150MHz). We believe that aggressively custom design with a tightly coupled streaming memory interface is what leads to such significant speed-ups.

Next, let us consider some recent efforts in fully custom hardware for this task. A notable VLSI implementation of tile-based BP ([1]) can run up to 64-label stereo matching of a 320x240 image in 137.4ms for a high quality result. For the Tsukuba benchmark, it obtains the lowest minimum energy of 396,953 with the same setting. Note that the lowest minimum energy corresponds to a high quality stereo result for the MRF. In comparison, for single frame performance, our system can achieve about 6 times faster speed in inference with a slightly lower minimum energy (when the number of iterations is 5), or about the same speed but with much lower minimum energy (when the number of iterations is 40), as shown in Table 3.4.

3.5 Summary

In this chapter we have implemented a stereo matching system with a novel FPGA hardware that performs TRW-S inference in a fully pipelined, streaming architecture. Experimental results show promising performance – speed, solution quality – of our streaming TRW-S compared to recent VLSI and GPU implementations of BP.

Chapter 4

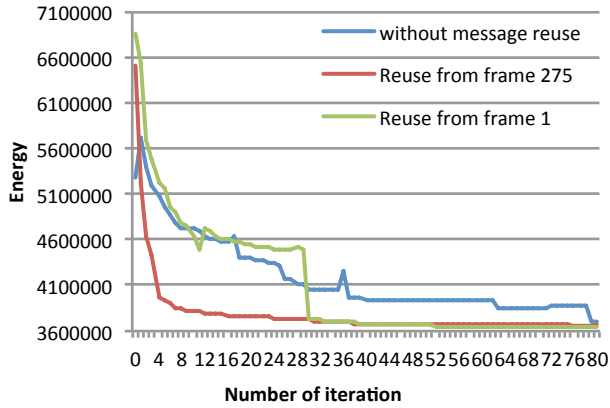
Frame-Level Optimizations for High Speed Stereo Matching

Starting from the parallel streaming architecture proposed in Chapter 3, further speed-up can be achieved by frame-level optimization in order to handle stereo video frames rapidly. In this chapter, we discuss three key optimization techniques to speed up stereo matching for multiple frames: message reuse based on scene change detection, frame-level parallelization, and function-level pipelining.




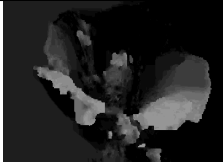
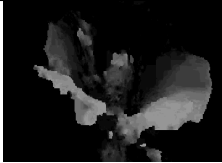
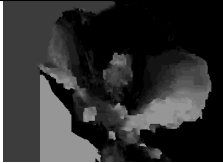
4.1 Message Reuse based on Scene Change Detection

As discussed in Section 2.1.3 the message passing operation is repeated until all the messages are converged to a fixed point. Once converged, these messages along with the initial data costs comprise the beliefs, which are used to determine the best label assignment for each pixel. Therefore, the entire computation of the message passing is to reach the correct messages based on given MRF parameters. In general, messages are initialized to zero (i.e. uniform distribution), and it takes tens to hundreds of iterations of message passing to reach the fixed point. If we know a better initial point, we can reduce such computational efforts but still reach to the proper fixed point.

It is common in video stereo matching that the contiguous frames in the video input contain similar pixel contents. Since the MRF parameters are computed based on these pixel contents, the initial parameters of the contiguous frames are also similar, leading to the similar inference results. Such similarity implies that the converged messages for the stereo matching of the previous frame can be re-used as a better starting point for the stereo matching of the current frame. The advantage of this so-called “warm start” can be



(a) Energy minimization performance for Flower task (frame 276) with/without message reuse from different frames (frame 1 or frame 275).

current input frame	Previous frame for message reuse	
 Frame 276	 Frame 275 HIST_DIST=834	 Frame 1 HIST_DIST=20,966
Without message reuse	Message reuse	
 Iteration 80	 Iteration 20	 Iteration 20

(b) Input frames (first row) and resulting disparity maps (the second row) for different settings: without message reuse, message reuse from the similar frame, and message reuse from the different frame.

Figure 4.1: Impact of message reuse in energy minimization performance and histogram based scene change detection. (Flower video stereo matching task ([5]) is used for example.)

seen in Fig. 4.1, which compares energy minimization performance for Flower video stereo matching task ([7]) with or without reusing previous messages. If we initialize messages to zero, it takes 80 iterations to achieve minimum energy lower than 3700000, at which most of the visual artifacts in the disparity map are removed. (See the left-most disparity map in the second row of Fig. 4.1(b).) In contrast, if the messages from the previous frame are reused as the initial values of the current message passing, it takes less than 20 iterations to achieve the similar minimum energy. (The center disparity map in the second row of Fig. 4.1(b).) Therefore, it is beneficial to reuse messages from the previous frame if the frames have potential to reach to a similar fixed point.

Then, the challenge is to detect when the contiguous frames have such a chance. There is a line of research on scene change detection (SCD) for efficient video compression (e.g. [44]), where the simplest approach is to construct a histogram over the color intensity to compute visual similarity between two frames. In our work, we construct a histogram not over the color intensity but over the initial maximum likelihood labels. There are two advantages in working with the maximum likelihood labels. First, we are not directly interested in the visual coherency between the frames, but rather we are interested in predicting similarity of the MRF inference results. The maximum likelihood label assignments are the result of MAP in (2.4) without considering priors, and thus the maximum likelihood labels implicitly characterize the data cost, guiding us to determine which frames will have the similar inference results. Second, the best maximum likelihood labels can be found easily with simple additional steps that can be incorporated in our MCC in Fig. 3.1. Since the histogram is built over the labels, we need a register vector of size $|x_s|$ for the histogram, which is usually smaller than a vector for the color intensity (e.g. 256 bins for each color intensity).

In our work, therefore, we attached a hardware block at the end of MCC that finds the best maximum likelihood label for each pixel and accumulates it into one of the bins of the histogram. All the computation in this block is fully pipelined so that there is little overhead in throughput. Once all the pixels are processed, the histogram is compared to the histogram of the previous frame, calculating the L-1 distance, called HIST_DIST. Based on this distance (i.e. HIST_DIST), the inference engine (i.e. STRM-TRWS) determines whether it should reuse the previous messages or not. As shown in Fig. 4.1, if the inference

reuses messages from the totally different frame (e.g. frame 1 of the Flower task), the distance of the histogram is 20,966, and thus the energy minimization is slower, compared to the distance of the histogram for the case where the messages from frame 275 are reused ($\text{HIST_DIST} = 834$). We set a threshold for this distance to avoid reusing messages with high HIST_DIST .

The idea of reusing inference results of the previous frames is also applied in other MRF inference techniques such as graph cuts ([45]), but to the best of our knowledge, this is the first attempt to incorporate the message reuse technique in a hardware implementation.

4.2 Frame-Level Parallelization

Since MRF inference for each frame is independent of the others, we can further speed up the stereo matching by utilizing the multiple FPGAs available on the platform. For example, we can process two frames of stereo input in parallel by implementing two inference engines on two different FPGAs. Note that each FPGA has its own memory bandwidth of 20-Gbyte/s. Thus, we can simply put the MRF parameters for two image frames in a different location of the shared memory and then run the two inference engines independently. Thus, a multiple-FPGA stereo matching system is attractive since 1) the frame-level parallelism is obvious and easily accessible in this video application, and 2) larger memory bandwidth can be utilized by employing multiple FPGAs, which is the prime bottleneck of performance.

In this work, COM_COST and STRM-TRWS are implemented in FPGAs, and they can be invoked by the host processor like other CPU functions to perform stereo matching. Since COM_COST and STRM-TRWS occupy one FPGA each, we can launch multiple COM_COST or STRM-TRWS functions in parallel using four FPGAs available on the Convey HC-1 to process more than one frame at the same time. The following pseudo code is an example of processing two frames in parallel using two FPGAs for COM_COST and STRM-TRWS each.

In this example, two image frames (even and odd) are read by the host processor and passed to two FPGAs (called AE1 and AE3) for COM_COST of two frames in parallel.

```

Initialize
For (idx=0; idx<numFrame; idx+=2)
    Even_frame <- RD_IMGS [idx];
    Odd_frame  <- RD_IMGS [idx+1];
    COM_COST   (Even_frame@AE0, Odd_frame@AE1); //parallel proc.
    STRM-TRWS (Even_frame@AE2, Odd_frame@AE3); //parallel proc.
    GET_DISP   (Even_frame from AE2);
    GET_DISP   (Odd _frame from AE3);
    WR_DISP    (Even_frame);
    WR_DISP    (Odd_frame);
End

```

MRF parameters computed by COM_COST are used by another two other FPGAs (AE2 and AE4) to perform inference (STRM-TRWS) for two frames in parallel. Then, disparity map results are retrieved (GET_DISP) and output (WR_DISP) by the host processor sequentially. Note that data transfer between the host processor and the FPGAs is managed by the cache coherent shared-memory, and can be expedited by using a dedicated data mover [6]. Also, we can observe that the number of calls for RD_IMGS, GET_DISP, and WR_DISP increases in proportion to the number of frames processed in parallel. The execution time for these CPU functions limits the benefit of frame-level parallelization.

4.3 Function-Level Pipelining

As the sub-functions of the stereo matching are partitioned to CPU and FPGAs, the next obvious question is how to overlap the execution time of these groups. Since the inference is done in hardware, the host processor is available while the FPGA is busy doing inference. Thus, we can apply basic software pipelining ([46]) so that the FPGA performs inference for the current frame while the host processor is preparing the input for the next frame, as well as processing the labeling results of the previous frame.

On the Convey HC-1, FPGA functions appear as callable procedures, which can be invoked by the host, and pass parameters back and forth. Because the platform supports non-

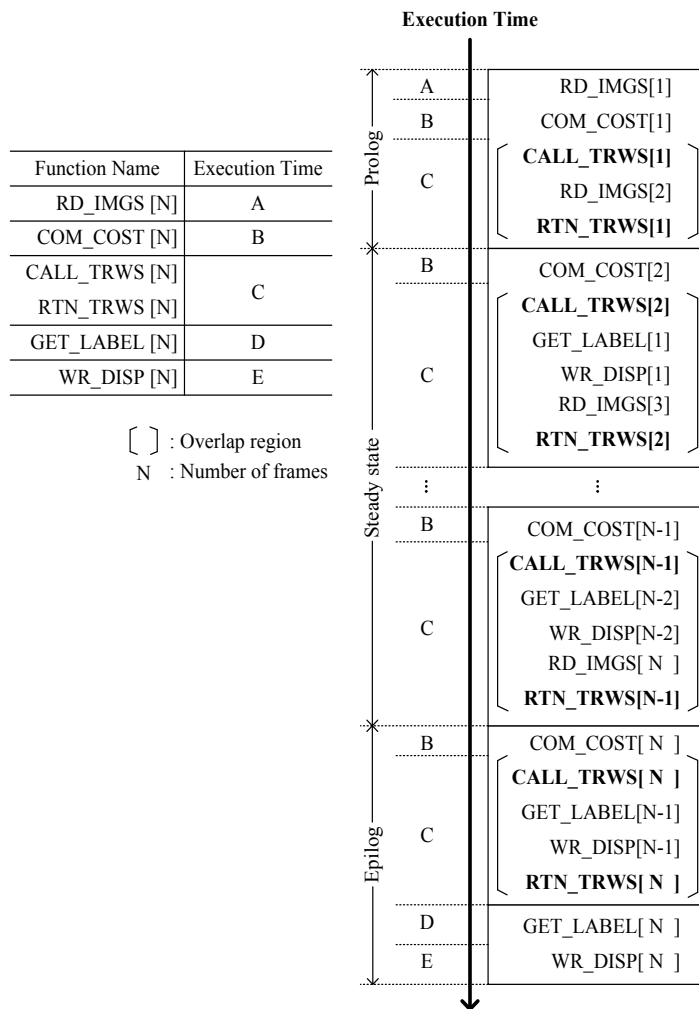


Figure 4.2: Pipelining of stereo matching functions.

blocking calls for these custom FPGA functions ([6]), we can further increase the speed of stereo matching by overlapping the execution time for the CPU functions and these hardware functions. Fig. 4.2 shows an example of applying the non-blocking call for STRM-TRWS. Note that STRM-TRWS is now divided into two sub-functions: CALL_TRWS (the “call”) and RTN_TRWS (the “return”). Thus, the computation time of the host function calls between CALL_TRWS and RTN_TRWS (overlap region in Fig. 4.2) is hidden by the execution time of the FPGAs.

It is not difficult to see that we can combine this functional pipelining and the frame-level parallelization to achieve the most speed-up. However, there is a balancing issue: the more frames that are processed by the inference engines in parallel, the longer time it takes to manage the input and output data of the inference, as observed in the previous section. If the time for input and output data management is shorter than the time for parallel inference, the total stereo matching time can be reduced because the time for data preparation can be overlapped with the inference. But if the number of frames processed in parallel is too large, then the time for data management becomes dominant in the entire execution, and the benefit of exploiting multiple FPGAs vanishes. Therefore, it is important to make the two balanced.

We can estimate the ideal execution time T for N frame stereo matching by the execution time of each function ($A \sim E$) defined in Fig. 4.2 as follows:

$$T = A + N \times (B + C) + (D + E). \quad (4.1)$$

Thus, the computation time of the host processor ($= A + D + E$) can be neglected if $(B + C)$ is greater than $(A + D + E)$ and N is large enough. This is the case when the time for data management on the host processor and the time for inference on FPGAs are well balanced.

4.4 Experimental Results

In this section, we show results from full video experiments. Benchmarks like the single-frame Middlebury set ([5]) are less common here. We use a set of real-world stereo examples from [7] for our experiments.

4.4.1 Effect of function-level pipelining

We perform stereo matching of a stereo video task (Flower) from [7] on our system with the non-blocking function calls to analyze the effect of function-level pipelining. This stereo video task consists of 276 frames of 360x262 stereo flower images as input. The 16-label (disparity levels) stereo matching is performed for 80 iterations to obtain the disparity map results in Fig. 4.3, since it requires considerable effort to propagate the depth information on the body of flower to its empty background. This task is more challenging than Tsukuba, for which TRW-S produces a reasonable outcome within only 5~40 iterations.

Next, we analyze the effect of the function-level pipelining used in our implementation. Figure 4.4 shows possible reduction versus actual reduction in overall execution time due to the function-level pipelining. The possible reduction comes from the portion of the CPU execution time relative to the total execution time. The actual reduction is calculated as the difference of execution time with and without applying the function-level pipelining. In case of single-frame stereo matching, the actual reduction is zero since there is no possible overlap between CPU and FPGA execution time. However, the actual reduction becomes closer to the portion of the CPU execution time as the number of frames increases. In the case of 64-frame stereo matching, the difference between the possible reduction and the actual reduction is only 0.5%. Thus, we can conclude that the pipelining of the functions improves the speed of the stereo matching by hiding most of the CPU execution time. (Since the portion of the CPU execution time is around 8% of the entire execution time, there is ample time for the host processor.)

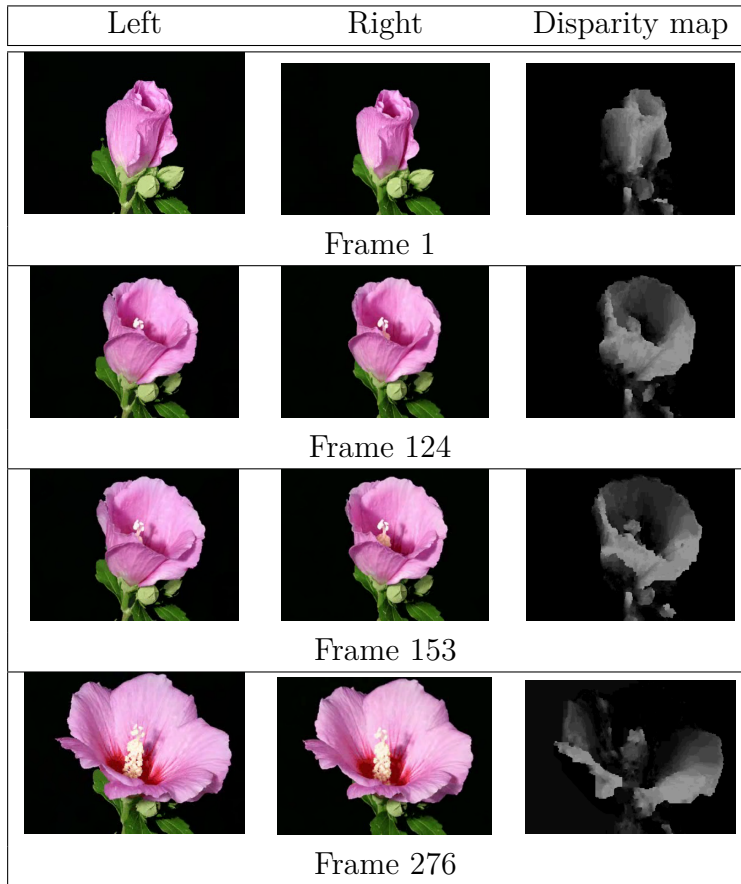


Figure 4.3: Video stereo matching task (Flower, [7]) and disparity map results. (Lighter grey means closer to camera.)

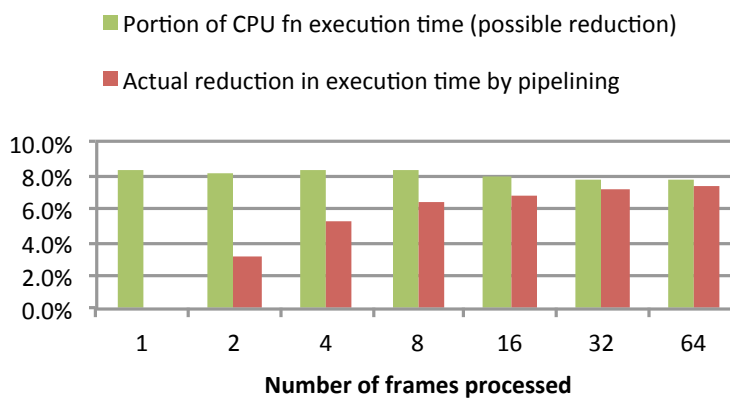


Figure 4.4: Possible reduction versus actual reduction in overall execution time due to function-level pipelining.

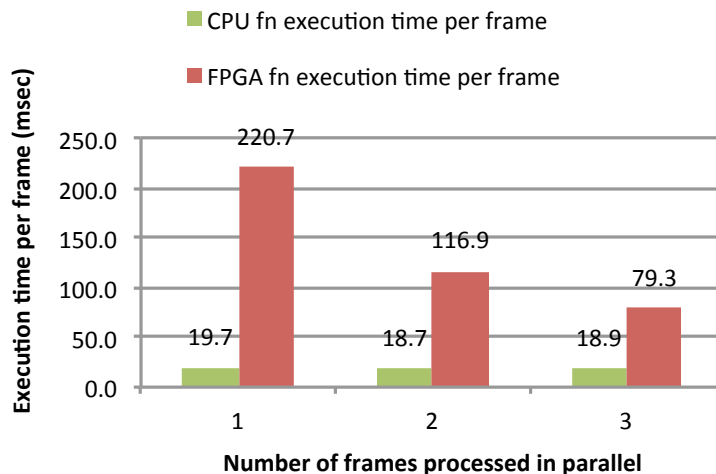


Figure 4.5: Per-frame execution time of CPU and FPGA functions.

4.4.2 Effect of frame-level parallelization

We further examine our stereo matching system for parallel processing of multiple frames. Among the four FPGAs available on the Convey HC-1, we assign one for the MCC and GCC, and the other FPGAs for STRM-TRWS. The number of FPGAs used for STRM-TRWS is equal to the number of frames processed in parallel. We compare the execution time of the CPU and FPGA functions, normalized by the number of frames processed concurrently. As shown in Fig. 4.5, the FPGA execution time per frame decreases inversely proportional to the number of FPGAs. Thus, we can conclude that the computations for multiple frames are well parallelized. In contrast, the CPU execution time per frame does not change as more hardware is utilized, as expected from Section 4.2. Since the CPU per frame execution time is smaller than the FPGA execution time, we can exploit functional pipelining to fully overlap CPU execution time with FPGA execution time.

We now analyze the total execution time for Flower video stereo matching. The execution time is measured for different numbers of frames processed in parallel, and function-level pipelining is applied for all the cases. To provide an idea of how well our optimization

Table 4.1: Analysis of total execution time (in second) for Flower video stereo matching (276 frames).

Num. FPGAs	1	2	3
Total Exec Time (T_t)	61.3	33.3	23.4
Estm. Time with Pipelining (T_p)	60.9	32.3	21.9
Estm. Time w/o Pipelining (T_{np})	66.4	37.4	27.1

techniques work for entire frames, we also present estimated execution time with and without applying function-level pipelining for each parallel processing of frames. Table 4.1 shows the total execution time of stereo matching of 276 stereo flower images (T_t) along with the estimated execution time of either applying the pipelining (T_p) or not (T_{np}) using different number of FPGAs. T_p is computed by (4.1), and T_{np} is the execution time of one parallel processing of frames multiplied by the number of parallel processing calls (e.g. multiply 92 if the number of FPGAs is 3). Thus, T_p and T_{np} are the lower and the upper bound of the total execution time. Since T_t is closer to T_p than T_{np} , we can see that our stereo matching system takes advantage of pipelining. In case of using three FPGAs for STRM-TRWS, the speed of the stereo matching is 11.8 frames per second.

4.4.3 Impact of message reuse

Next, we analyze the impact of message reuse in the total execution time and the quality of the inference results. We perform video stereo matching for the same Flower task, applying both function-level pipelining and 3-frame parallel processing. Table 4.2 shows the execution time of the stereo matching with/without enabling message reuse. As shown in the table, the total execution time depends on either CPU execution time or FPGA execution time. If the number of iteration is large (i.e. the number of iteration = 80), the FPGA execution time takes the majority of the total execution time, and the CPU time is mostly hidden by the FPGA execution time. On the other hand, if the number of iteration is small (i.e. = 20), the FPGA execution time is shorter than the CPU execution time, offsetting the benefit of FPGA acceleration. Thus, as discussed before, this emphasizes the importance of balancing.

Table 4.2: Impact of message reuse on energy minimization performance and execution time (in second). Flower video stereo matching task (360x262x276 frames) is used for example.

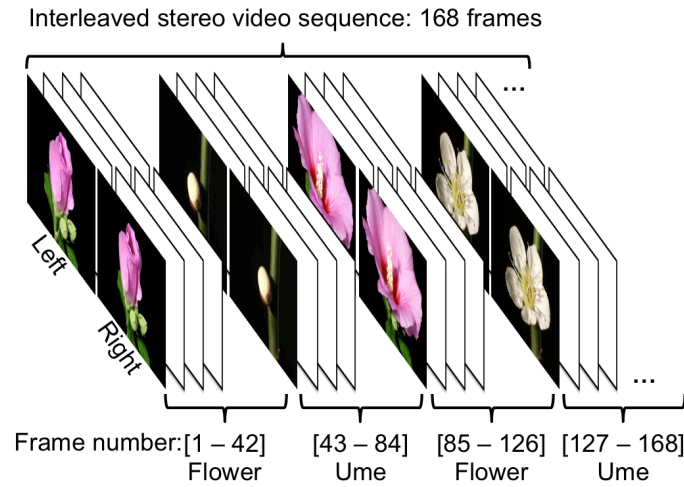
	Without Message Reuse			With Message Reuse	
Number of iterations	80	40	20	40	20
Total Exec. Time	22.0	11.9	10.1	11.8	10.1
CPU Exec. Time	9.4	9.8	9.2	9.3	9.1
FPGA Exec. Time	21.2	10.7	5.4	10.8	5.6
Avg. Incr. in Min. Energy	0.0%	4.0%	9.7%	0.4%	2.0%

One interesting question is if we can further decrease the number of iterations to reach the balancing point without sacrificing the quality of inference results. As we discussed in Section 4.1, the Flower task requires 80 iterations to get rid of the artifacts on the background of the disparity map (Fig. 4.1(b)). Therefore, if the number of iterations is decreased to 40, where the balancing point between CPU and FPGA execution time resides, the minimum energy is increased 4.0% on average (compared to the 80-iteration case), causing defects in the disparity map results. However, if message reuse is applied, we can achieve the same balancing point (i.e. the number of iterations = 40) without much increase in the minimum energy. Therefore, the message reuse based on the scene change detection can help improving both the speed of stereo matching and the quality of inference. In our platform, the maximum speed of high quality stereo matching for Flower task is 22.8 frame per second.

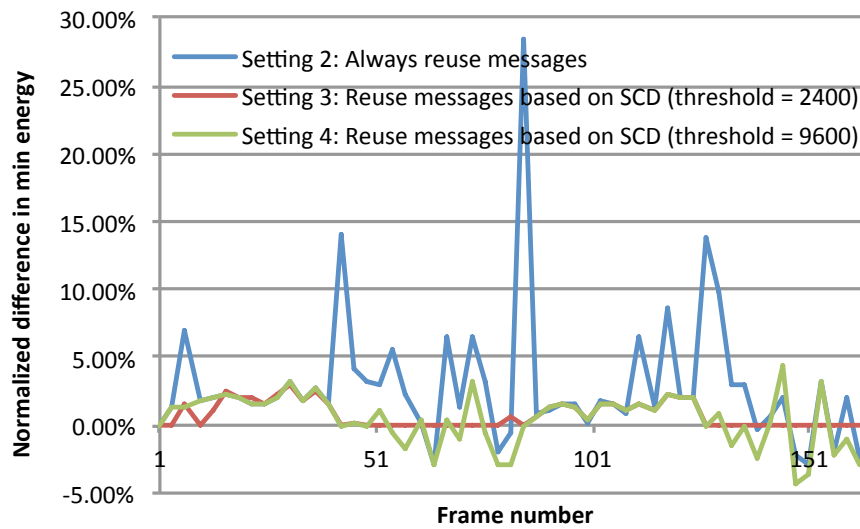
4.4.4 Impact of scene change detection based message reuse

Lastly, we demonstrate the effectiveness of our scene change detection (SCD) based message reuse for a video stereo matching scenario. The scenario consists of two video stereo sequences, Flower (frames resized to 320x240, QVGA) and Ume from [7]. To emulate scene changes in the video, these sequences are interleaved as follows: frame 1 to 42 and 85 to 126 from the Flower sequence, and frame 43 to 84 and 127 to 168 from the Ume sequence, as described in Fig. 4.6(a).

We run our stereo matching hardware with four different settings: 1) run 80 iterations



(a) Interleaved stereo video sequence (320x240x168 frames).



(b) Normalized difference in energy for different settings (setting 1 as a reference).

Figure 4.6: Video stereo matching with scene change detection (SCD) based message reuse.

without reusing messages, 2) run 20 iterations with reusing messages all the time, 3) run 20 iterations with reusing messages if HIST_DIST is less than the threshold of 9600, otherwise run 80 iterations, 4) same as 3) but with the threshold of 2400. We measure the minimum energy for each case and use the minimum energy of the first setting to compute normalized difference with the other settings. Per frame normalized difference in energy is shown in Fig. 4.6(b). Since there are three scene changes in the interleaved video sequence, if we reuse messages all the time (setting 2), the incorrect messages are reused at those scene changes (frame 43, 85 and 127), and thus the min energy curve rises rapidly (12~28% increase in energy). This undesirable outcome of the uninformed message reuse can be avoided by our simple scene change detection scheme described in Section 4.1. As shown in Fig. 4.6(b), the minimum energy curves of both setting 3 and 4 do not show spikes at the scene changes. The minimum energy curves of setting 3 and 4 are no higher than 5% of the minimum energy curve of setting 1. The energy curve of setting 3 is closer to the energy curve of setting 1 (i.e., the normalized difference is zero in many frames), since its threshold is more conservative. On the other hand, the energy curve of setting 4 sometimes shows even lower energy, despite the fact that it takes one-quarter of the number of iterations of setting 1.

The execution time for the setting 1 to run entire frames (= 168) is 11.05s, using all the other frame-level optimization techniques in this chapter as well. In comparison, the execution time of setting 2 is only 6.56s, but it suffers spikes in the energy curve due to incorrect message reuse at the scene changes. Our scene change detection based message reuse can provide good middle points. If we want conservative inference results, we can set the threshold low to achieve low minimum energy more reliably and also reduce a little execution time. Otherwise, we can set a higher threshold to save significant time but still avoid spikes in the energy curve. For example, setting 3 and 4 with the threshold of 2400 and 9600, respectively, result in the execution time of 9.19 and 7.34s, respectively. Therefore, we can conclude that our scene change detection based message reuse can be properly applied to the TRW-S inference for more efficient video stereo matching.

4.5 Summary

We have designed and benchmarked a video-rate stereo matching system implemented on a hybrid CPU+FPGA platform (Convey HC-1). We model the stereo task as statistical inference on a Markov Random Field model, and show how to implement TRW-S style inference at video rates on the hybrid platform. We employ both parallelization of TRW-S and various frame-level optimization techniques for video-rate stereo matching. Experimental results show that this system performs stereo matching of QVGA size 168 frame mixed stereo video sequence in 7.34 seconds (i.e., 22.8 frame per second) with minimum energy difference less than 5%.

Chapter 5

Error Resilient MRF Message Passing Architecture for Stereo Matching

In this chapter, we consider error resilient hardware implementation of TRW-S inference. As machine learning applications are inherently probabilistic and robust to errors, statistical error compensation (SEC) techniques can play a significant role in achieving robust and energy-efficient implementation [34, 47]. By trading the increased robustness for energy, significant energy savings can be achieved as well. In [34], algorithmic noise tolerance (ANT) has been applied to a message passing based low density parity check (LDPC) decoder and shown to achieve 45.7% energy savings while maintaining less than 4.7 dB degradation in bit error rate (BER) at a HW error rate (percentage of clock cycles in which an erroneous output exists) of 30%.

Motivated by this prior work, therefore, we utilize our TRW-S based stereo matching hardware architecture (STRM-TRWS) we discussed in Chapter 3 as a test case to explore error resiliency and energy efficiency of message passing based MRF inference hardware. We first analyze the error propagation characteristics of our message passing algorithm. Then, we explore the impact of SEC when applied to MRF inference applications.

The remainder of the chapter is organized as follows. Section 5.1 describes the error resilience characteristic of the message passing hardware in detail. The implementation of TRW-S and the methodology to generate and compensate HW errors are discussed in Section 5.2. Section 5.3 shows the results, while Section 5.4 summarizes the chapter.

5.1 Error Analysis of TRW-S

It is well known in the literature (e.g. [34]) that iterative message passing algorithms such as TRW-S are intrinsically robust to errors. In this section, we analyze the error propagation

of TRW-S as a test case to understand the basis of the inherent error robustness and how to exploit it to find the optimal precision for TRW-S hardware implementation.

In this section, we provide a simple analysis on error propagation of TRW-S to understand the basis of the inherent error resiliency of iterative message passing algorithms. We further explore the optimal precision for the TRW-S hardware, which is motivated by the inherent error resiliency of TRW-S inference.

5.1.1 Error analysis of TRW-S

We start our analysis of the error propagation of TRW-S from the message update equations (2.14) and (2.15). For simplicity, we assume binary labels for all the nodes and the Potts model ([10], with penalty of C) for the pairwise cost. The message update (2.15) can then be represented as follows:

$$M_{st}(0) = \min\{H_{st}(0), H_{st}(1) + C\} \quad (5.1)$$

$$M_{st}(1) = \min\{H_{st}(0) + C, H_{st}(1)\}. \quad (5.2)$$

Note that other message passing algorithms with a truncated smoothness cost (e.g. BP in [35, 48]) can also be represented this way.

To see the effect of arithmetic error on the result of the final message, we assume that arithmetic errors only occur in add/subtract (AS) and compare and select (CS) operations. An erroneous message for label 0, $\tilde{M}_{st}(0)$, can be represented as follows:

$$\begin{aligned} \tilde{M}_{st}(0) = \min\{ & H_{st}(0) + e_{AS}(0), H_{st}(1) + e_{AS}(1) + C\} \\ & + e_{CS}, \end{aligned} \quad (5.3)$$

where $e_{AS}(0)$ and $e_{AS}(1)$ represent the error propagated from neighbors as well as any arithmetic errors that occurred during computation of H_{st} , and e_{CS} indicates errors that occur in CS. By setting $e_{AS} = e_{AS}(0) - e_{AS}(1)$, and $\tilde{e}_{CS} = e_{CS} + e_{AS}(1)$, $\tilde{M}_{st}(0)$ can be rewritten

as:

$$\begin{aligned}
\tilde{M}_{st}(0) &= \min\{\overbrace{H_{st}(0) + e_{AS}}^{arg1}, \overbrace{H_{st}(1) + C}^{arg2}\} + \tilde{e}_{CS} \\
&= M_{st}(0) + \tilde{e}_{AS} + \tilde{e}_{CS},
\end{aligned} \tag{5.4}$$

where \tilde{e}_{AS} and \tilde{e}_{CS} are the effective error in the message M_{st} generated by AS and CS, respectively. We will refer to $H_{st}(0) + e_{AS}$ and $H_{st}(1) + C$ as *arg1* and *arg2*, respectively, and define $\Delta H \triangleq H_{st}(1) + C - H_{st}(0)$.

We analyze the relationship between e_{AS} and \tilde{e}_{AS} to understand error propagation characteristic of the message passing inference. First, assume that $H_{st}(0) < H_{st}(1) + C$, i.e., $\Delta H > 0$. Then, when e_{AS} is small, such that $e_{AS} < \Delta H$, *arg1* will be chosen as the minimum and $\tilde{e}_{AS} = e_{AS}$. Once $e_{AS} \geq \Delta H$, min will now choose *arg2* and \tilde{e}_{AS} will be fixed to ΔH (see Fig. 5.1(a)). Next assume that $H_{st}(0) > H_{st}(1) + C$, i.e., $\Delta H < 0$. In this case, if $e_{AS} \geq \Delta H$, *arg2* will be selected as the minimum, which is the correct minimum, and $\tilde{e}_{AS} = 0$. When $e_{AS} < \Delta H$, min will select *arg1*, and $\tilde{e}_{AS} = e_{AS} - \Delta H$ (see Fig. 5.1(b)). Note that the effect of e_{AS} depends only on ΔH which, in turn, only depends on the error free computation of H_{st} . Based on Fig. 5.1, we can deduce two error characteristics of TRW-S:

1. TRW-S is affected more by negative errors than positive errors, since the effect of positive errors is either bounded (Fig. 5.1(a)) or removed (Fig. 5.1(b)).
2. If the magnitude of an error is small ($|e_{AS}| < |\Delta H|$), it can either be preserved (Fig. 5.1(a)) or removed (Fig. 5.1(b)).

Similar conclusions can be derived for $\tilde{M}_{st}(1)$.

The error generated by CS, \tilde{e}_{CS} , can be viewed as part of the overall error in a message, $\tilde{e} = \tilde{e}_{AS} + \tilde{e}_{CS}$. The error \tilde{e} will affect the message computation of the adjacent node and thus can be regarded as a propagated error for the next message computation. Normalization

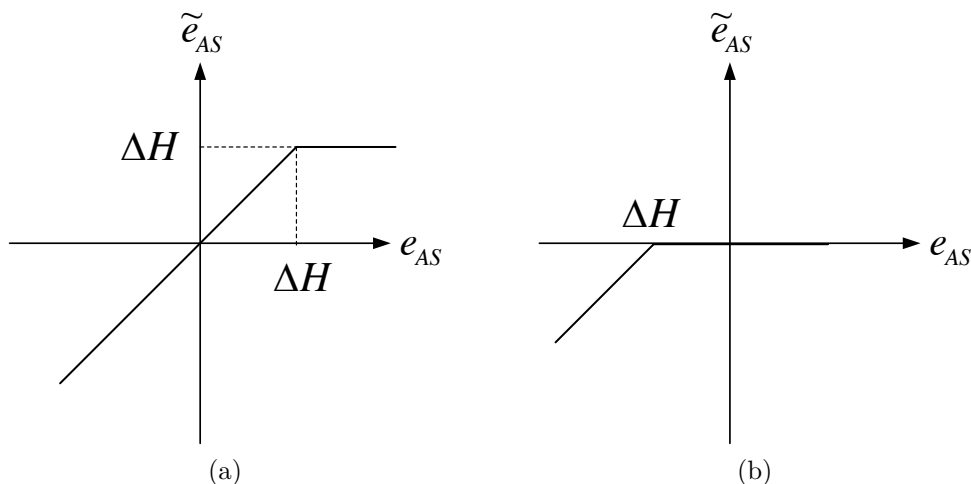


Figure 5.1: Effect of AS error on updated message when (a) $\Delta H > 0$, and (b) $\Delta H < 0$.

that occurs at the final step of MSG_UPD can also be shown to significantly reduce the magnitude of the propagated errors.

5.1.2 Verification of error analysis

Two experiments are performed on TRW-S stereo matching hardware to verify our analysis. We first run Tsukuba stereo matching [10] on our software simulator of STRM-TRWS with error injection on AS in (5.4) to see the effect of error on the message update. The Potts model with $C = 20$ is used as the smoothness cost, and the computation in hardware has 20-bit fixed point precision. The injected errors, e_{AS} , are drawn from $u(-128, 128)$, where $u(a, b)$ is a uniform distribution between a and b . Figure 5.2(a) shows the resulting relationship between e_{AS} and \tilde{e}_{AS} summarized as a box plot. Note that $|\tilde{e}_{AS}|$ cannot be larger than C due to truncation and normalization in message update. It can be seen that \tilde{e}_{AS} is more likely to be close to zero for $e_{AS} > 0$ than for $e_{AS} < 0$, which agrees with our analysis.

For a macroscopic view of the effect of error on the message passing performance, we further apply errors with different magnitude on both e_{AS} and e_{CS} and run the same Tsukuba stereo matching. We apply uniform errors as follows: $e_{AS}, e_{CS} \sim u(\min[0, e_{max}], \max[0, e_{max}])$, and $-2^{15} \leq e_{max} \leq 2^{15}$. Note that all injected errors are of the same sign. Fig. 5.2(b) shows the effect of errors on the energy minimization performance for different error magnitudes

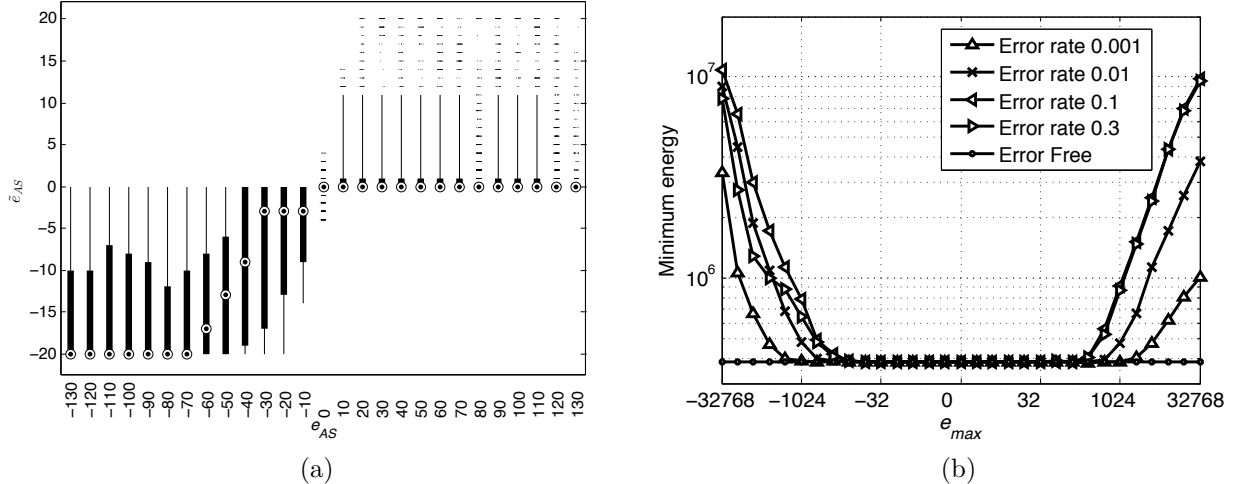


Figure 5.2: Verification of error analysis: (a) effect of error on message updated via a box plot of e_{AS} vs. \tilde{e}_{AS} , and (b) effect of error on energy minimization performance of message passing via a plot of minimum energy (after 10 iterations) vs. e_{max} .

and error injection rates. If the error magnitude is small (e.g., $|e_{max}| \leq 512$), the minimum energy given by (2.4) approaches the error free performance after 10 iterations. However, if the error magnitude is large (e.g., $|e_{max}| \geq 1024$), the minimum energy does not decrease with iterations, which indicates that message passing fails to find the best label assignment. Furthermore, the slope of the minimum energy is steeper in case of the negative errors, as expected by our analysis. Therefore, we can conclude that TRW-S is tolerant to small magnitude errors but suffers from large magnitude errors.

We can exploit this intrinsic error robustness of TRW-S to optimize the fixed point precision of arithmetic computations in (2.14) and (2.15). Figure 5.3 shows energy vs. precision used in the computation of STRM-TRWS when running the Tsukuba stereo matching. $M[a, b, c]$ represents different precision options applied to the main computation of STRM-TRWS: a precision of a -bits is used to perform the computation of (2.14) and (2.15) after b -bit LSB truncation and c -bit MSB saturation. The baseline precision is $M[20, 0, 0]$ in [49], which outputs the same energy minimization results as the floating point implementation [10]. Compared to this baseline, 8-bit LSB truncation ($M[12, 8, 0]$) produces a similarly low energy minimization curve, whereas 12-bit truncation ($M[8, 12, 0]$) results in a higher minimum energy curve. According to our experiments, 8-bit precision with less than 11-bit LSB truncation (i.e., $M[8, 8, 4]$, $M[8, 9, 3]$, and $M[8, 10, 2]$) achieves comparable energy minimiza-

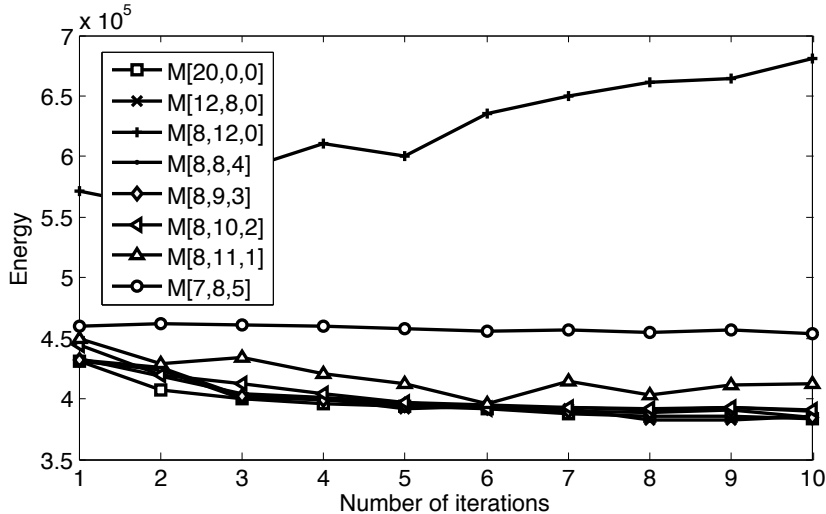


Figure 5.3: Energy minimization performance against various precision in computation of STRM-TRWS.

tion performance to the baseline precision (M[20, 0, 0]). However, other precisions, such as M[8, 11, 1] or M[7, 8, 5], perform worse due to large magnitude quantization or saturation errors.

5.2 Simulation Methodology and System Evaluation Setup

A cycle accurate software simulator has been implemented to simulate STRM-TRWS. We employ this simulator in the experiments we describe in the following section. The simulation methodology is summarized in Fig. 5.4(a). We evaluate error resiliency of STRM-TRWS by injecting errors in the simulator. For accurate error resiliency simulation, input dependent timing based error statistics are obtained via gate level simulations using an HDL simulator. First, the gate delay is characterized with respect to supply voltage for basic gates such as a full adder and XOR using circuit level simulators with a commercial 45 nm process library. Then a structural HDL implementation of the REPARAM and MSG_UPD block is simulated via an HDL simulator using the pre-characterized delay values. By choosing the delay values that correspond to various supply voltages, HDL simulation is effectively run at different voltages. For a supply voltage (V_{dd}) less than the critical voltage, errors can be observed in the output. Through characterization of these errors, error statistics are

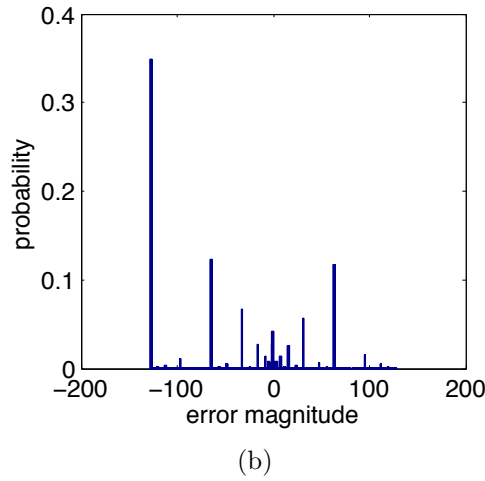
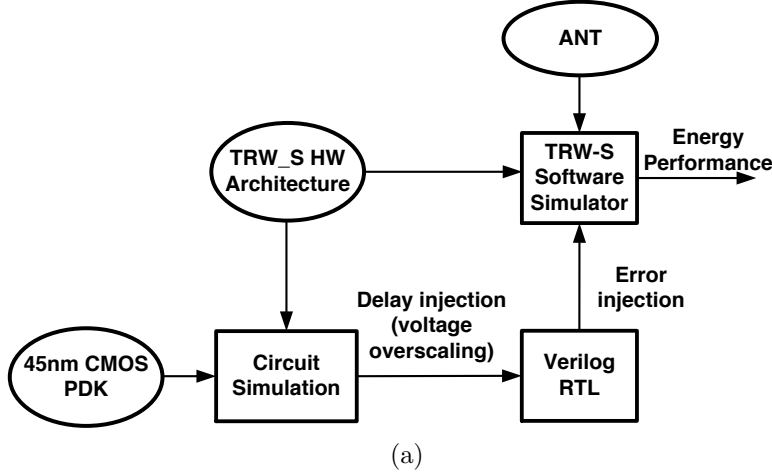


Figure 5.4: TRWS-HW simulation: (a) methodology, and (b) error statistics for AS at $V_{dd} = 0.78$ V.

obtained and used to inject errors in the AS and CS block of the software simulator.

The message passing unit was designed for $V_{dd,crit} = 1.2$ V at $f_{CLK,crit} = 270$ MHz. Figure 5.4(b) shows error statistics obtained for the AS block at $V_{dd} = 0.78$ V. Large magnitude errors can be seen to occur with high probability, which is expected due to the nature of LSB first computation. The CS block did not exhibit any errors under these operating conditions, due to its short critical path. Thus, ANT was only applied to AS operations in REPARAM and MSG_UPD units of STRM-TRWS (Fig. 3.5(b) and (c)), which are vulnerable to large magnitude errors. For the estimator, a reduced precision replica (RPR) of the main block has been used. This particular estimator has a shorter critical path than the original block and thus is not subject to timing errors but suffers from estimation errors.

5.3 Results

As discussed in Section 5.1, the message passing inference such as TRW-S has intrinsic robustness to small magnitude errors but is vulnerable to large magnitude errors. Since ANT converts large hardware errors into small estimation errors, it can effectively compensate large magnitude errors and significantly enhance error resiliency of the message passing inference. To evaluate error compensation of ANT on the message passing inference, we run TRW-S with ANT (the estimator with precision $E[4,12,4]$ is used) applied for Tsukuba stereo matching task. Figure 5.5(a) shows the energy minimization performance of ANT when uniform errors with different magnitude, as in Section 5.1.2, were injected at various error rates. Only the tail (where $e_{max} > 32$) is shown. For small magnitude errors ($e < 1024$), performance of ANT is mostly dictated by the main block. When large magnitude errors ($e > 4096$) are injected, the estimator becomes active and successfully compensates for the errors. The slight degradation in performance is observed when errors of ($1024 \leq e \leq 8192$) are injected; this is because hardware and estimator errors have similar magnitude and thus ANT is not able to compensate for the errors. To avoid this ambiguity, the estimator should be designed to have estimation errors distinct from the hardware errors ; this is true in our case where the timing errors have large magnitude but the RPR has low magnitude quantization errors.

Next, we explore the optimal precision for the estimator. We run the Tsukuba task at various V_{dd} and compare the energy minimization performance. A precision of $M[8, 8, 4]$ has been used for the precision optimized main block, and an estimator of precision $E[a, b, c]$ is used for ANT. Figure 5.5(a) shows the energy minimization performance of ANT with different estimators at various V_{dd} . It is evident that energy minimization performance is drastically degraded in the conventional case (no error protection), while ANT introduces tolerance to a significant amount of errors. Compared to 2-bit and 3-bit precision estimators, estimators with precision higher than 4-bit show much lower minimum energy. At $V_{dd} = 1\text{ V}$, where the error rate of the AS block is $p_{e,AS} = 0.8\%$, the minimum energy of $E[4, 12, 4]$ is almost the same as the minimum energy of the error free case.

To further establish the effectiveness of error compensation capability of ANT for STRM-

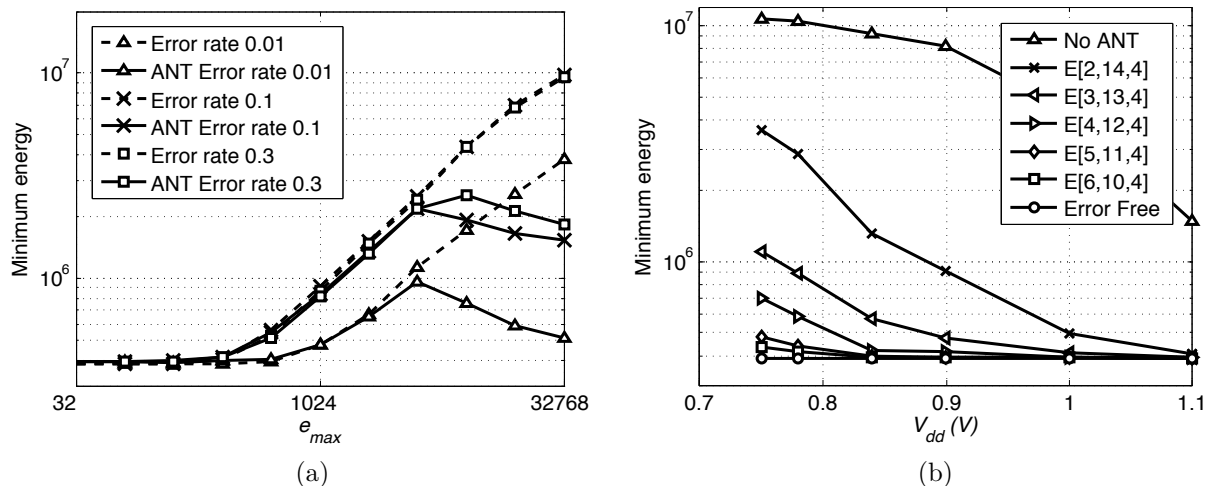










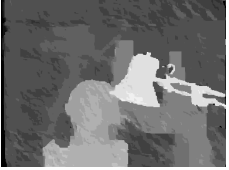


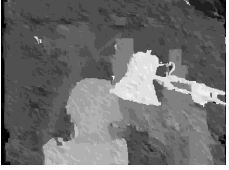
Figure 5.5: Simulation results: (a) performance of ANT with injection of uniform errors with different magnitude using $M[20, 0, 0]$ and $E[4, 12, 4]$, and (b) error injection rate vs. energy minimization for different ANT estimators with $M[8, 8, 4]$.

TRWS, the disparity maps are compared for three cases: error free, conventional, and ANT. The estimator precision is set to $E[4, 12, 4]$. Table 5.1 shows the disparity map of each case at different V_{dd} . The disparity map for the conventional case becomes corrupted even at a low error rate of 1%. In contrast, the disparity map when ANT is applied is comparable to the error free case when the same 1% error rate is applied. Furthermore, at low V_{dd} , where the error rate is 5% to 25%, the disparity map of ANT is still close to the disparity map of the error free case, which demonstrates the outstanding error compensation of ANT.

To evaluate the accuracy of the disparity maps, bad pixel ratio (BPR) is employed [5] as the system level metric. BPR is calculated by comparing the depth label of each pixel in the non-occlusion region to the true disparity map (ground truth) and counting a pixel to be bad if the label differs by more than a threshold κ ($\kappa = 1$ in our case). As shown in Table 5.1, BPR of the conventional case rises drastically from 10.4% to 70.4% as V_{dd} is scaled down. In contrast, BPR of ANT is robust to errors; BPR is at most 6.27%, at a high error rate of 21.3%.

Such enhancement in error robustness of ANT can be exploited to achieve power savings in circuit implementation. The power consumption of the message passing unit is estimated via switching activity based RTL synthesis in a commercial 45 nm synthesis library. Switching

Table 5.1: disparity map and BPR comparison for error-free, conventional and ANT at various V_{dd} .

V_{dd}	Error-Free	Conventional	ANT
1.0V $p_{e,AS} = 0.8\%$	 2.7%	 10.4%	 2.51%
0.9V $p_{e,AS} = 2.9\%$	 2.7%	 60.7%	 2.31%
0.84V $p_{e,AS} = 5.1\%$	 2.7%	 70.3%	 3.93%
0.75V $p_{e,AS} = 21.3\%$	 2.7%	 70.4%	 6.27%

activity is obtained from Verilog simulations running stereo matching of Tsukuba task. This switching activity is then annotated in RTL synthesis to estimate accurate power consumption. Table 5.2 summarizes the synthesis results with cell utilization and estimated power consumption including leakage and dynamic power. As can be seen, the cell overhead of ANT is approximately 97.4%, which is significant. However, at an extreme supply voltage of $V_{dd} = 0.72\text{V}$, ANT is able to achieve 39.7% power savings while maintaining the system performance. This result shows that significant energy savings is possible via voltage scaling, even when additional complexity has to be added for the system to perform correctly.

Table 5.2: Estimated compensation overhead and power consumption of ANT obtained via synthesis in a commercial 45 nm CMOS process.

	M [8,8,4] only	ANT (M[8,8,4] + E[4,8,4])	
Cell	40,790	80,534	81,366
V_{dd} (V)	1.10	1.10	0.72
Power (mW) (leak, dyn)	11.82 (2.03; 9.79)	17.81 (3.39; 14.42)	7.136 (0.23; 6.91)

5.4 Summary

SEC has been applied to a Markov random field (MRF) based stereo image matching architecture. Analysis and simulations show that for a 20-bit architecture, small errors ($e \leq 1024$) are tolerable, while large errors ($e \geq 4096$) degrade the performance significantly. By applying ANT, experimental results show that the proposed ANT based hardware can tolerate an error rate of 21.3%, with performance degradation of only 3.5% at an overhead of 97.4%, compared to an error-free full precision hardware with an energy savings of 39.7%.

Chapter 6

Toward a General Purpose Probabilistic Inference System

So far, we have discussed TRW-S based MRF inference hardware covering issues from our high throughput architectures and associated system level optimizations to a proposed energy efficient implementation, but our scope of application was still limited to stereo matching. In this chapter, we exploit MRF inference as a general framework for machine learning to extend our scope to a variety of computer vision problems. To this end, we propose a novel hardware architecture that can support various MRF configurations for solving computer vision problems. Furthermore, we extend our understanding of the error resiliency of TRW-S inference and the impact of ANT toward more general MRF settings.

6.1 High Performance TRW-S Accelerator for Computer Vision

6.1.1 Challenges for high performance computer vision solver

As described in Chapter 2, MRF inference solves MAP problems in the process of minimizing the energy function (2.4), where the smoothness cost can be defined as follows:

$$\theta_{st}(x_s, x_t) = \lambda \cdot \omega_{st} \cdot \min \left\{ |x_s - x_t|^k, \theta_{max} \right\}, k = 1 \text{ or } 2. \quad (6.1)$$

λ and ω_{st} represent coupling strength of entire edges and a specific one, respectively, and θ_{max} is the maximum smoothness cost. (6.1) can represent linear/quadratic costs, truncated linear/quadratic costs, and the Potts model [10], which are widely used in computer vision applications. Some examples of the popular choices of $\{\lambda, \omega_{st}, k, \theta_{max}\}$ along with the number of labels ($|\chi|$) for different applications are shown in Table 6.1.

In Chapter 3 and 4, we have demonstrated that TRW-S inference can be implemented in

Table 6.1: MRF parameter configurations for target computer vision applications

Benchmarks	Tasks	Size	$ \chi $	λ	ω_{st}	k	θ_{max}
Stereo matching	Tsukuba	384x288	16	20	2	1	2
	Venus	434x383	20	50	1	2	7
	Teddy	450x375	60	10	3	1	1
Image denoising	House	256x256	256	5	1	2	∞
	Penguin	122x179		25	1	2	200
Object segmentation	Plane	320x213	4	1	Potts	1	1
	Building		7				
	Car		8				

FPGA to achieve significant speedup in stereo matching without sacrifice in the quality of inference. One of the factors that enable such speedup is application specific customization of data and control paths. For example, our streaming TRW-S hardware (STRM-TRWS) is optimized for the tasks that use multiple of 16 labels (up to 64 labels with folding) and truncated smoothness costs. This aggressive customization results in limitation in usability of STRM-TRWS; it cannot run applications that require the non-truncated quadratic cost, or it sacrifices significant speed if the number of labels is not a multiple of 16. Therefore, in order to enhance the usability of the TRW-S hardware to the wider applications, it is necessary to design a hardware architecture that can support all these applications and achieve high performance at the same time.

6.1.2 Accelerator architecture

In this section, we propose a novel block-parallel streaming accelerator architecture for high performance TRW-S inference on various computer vision problems. The proposed architecture consists of a novel block-parallel memory interface and multiple streaming processing elements (PEs). The goal of this architecture is to achieve maximum throughput by fully utilizing the available memory bandwidth, which can be represented as follows:

$$MemBWUtil = ParallelFactor(P) \times BlockSize(B).$$

The parallel factor and the block size are related to inter- and intra-node parallelization, respectively. The block-parallel memory interface enables parallel processing of multiple nodes in an MRF, and the streaming PE performs a block of message computation in parallel. More detailed description follows in the next section.

6.1.2.1 Block-parallel memory interface

The block-parallel memory interface is illustrated in Fig. 6.1. This memory interface is equipped with P sets of Read / Feedback / Writeback buffers and PEs. The buffers work as FIFOs, and Read Buffers handle not only node data but also data for horizontal and vertical edges (EHOR and EVER, respectively), while Feedback and Writeback buffers only handle edge data. The node data consists of data cost (a vector of values in multiple blocks) and control signals (e.g., edge connectivity information) obtained from the graph structure. The edge data consists of a message (also a vector in multiple blocks) and optional coefficients such as the per-edge coupling strength (ω_{st}). As the same memory read interface is shared to fetch node and edge data into the Read buffers, the node data is first loaded, followed by the edge data. On the other hand, Writeback buffers solely occupy the memory write interface.

In the beginning of inference, a look-up table inside each PE for smoothness cost is initialized according to the pre-defined MRF setup. Then, P streams of node/edge data of an MRF stored in the external memory are fetched via Read buffers, and along with the edge streams stored in Feedback buffers, they are processed by P PEs to compute new messages. (The control signals in the node data determine which edge stream each PE needs to wait.) The newly computed messages are temporarily stored in Feedback buffers to be used in the later message updates. They are also stored in Writeback buffers to be dumped back to the external memory. Each stream consists of a block of data that is a portion of a full vector of length $|\chi|$, and thus it takes multiple cycles to fetch a full vector if $B < |\chi|$.

The goal of parallel memory interface is to process multiple nodes of an MRF in parallel. The diagonal ordering used in Chapter 3 can be simply decomposed into multiple sub-ordering using a modulo operation (e.g., $subIndex = index \text{ modulo } P$), so that each sub-

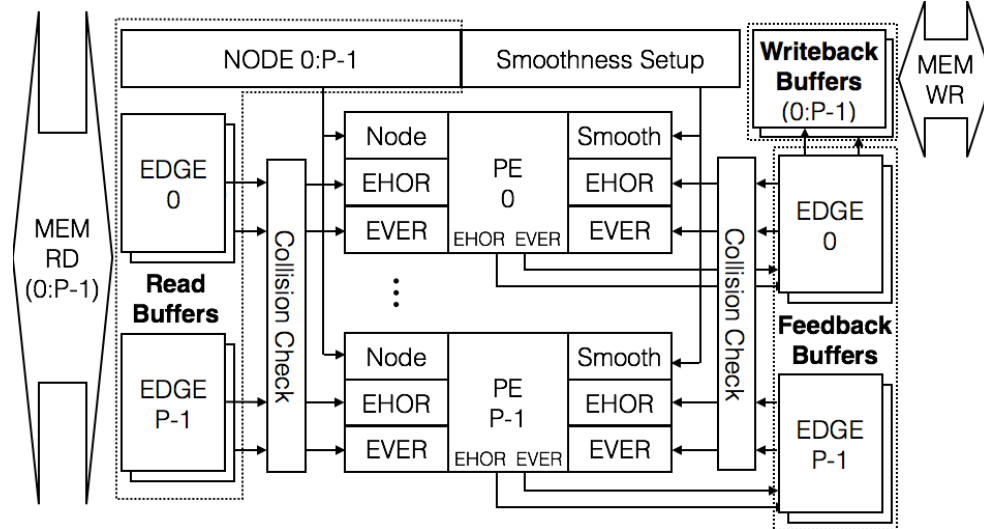


Figure 6.1: Overall architecture of block-parallel memory interface.

ordering can be fetched to corresponding Read buffers. Although the streams for node and edge data can be fetched independently, they should be executed in a way that all the diagonal rows are strictly ordered. Therefore, we check collision at each PE to process the stream with a lower node index first.

The following pseudo code shows how to check collision for each PE. There is no collision between the node streams since each node stream is uniquely associated with each PE. Whereas, collision check is necessary for edge streams since an edge stream can be processed by any PE based on the structure of the graph. Thus, for each PE, we check collision for accessing edge data by comparing its edge buffer index with the other PEs. For example, if $PE[i]$ and $PE[j]$ attempt to access edges in the same edge buffer, signals $collision[i][j]$ and $collision[j][i]$ become 1. At the same time, we also check which PE has a priority in accessing the buffer by comparing their node indices, e.g., $PE[i]$ has a priority over $PE[j]$ if $node_idx[i] < node_idx[j]$. To save computation, we only compare PE pairs corresponding to the entries of the upper triangle of a P by P matrix; the rest of the entries can be derived from them. Then, for each (i, j) pair, we can check if a stall is required; i.e., in case of $PE[i]$, if $stall[i][j] = 1$ for any j , $PE[i]$ needs to be stalled, and thus we set $collision_check[i]$ as 1. In this way, the diagonal ordering appropriate for parallel processing can be enforced.

As an example, Fig. 6.2 illustrates how an MRF is processed by the block-parallel memory

```

// Collision check for PE[i]
For (i=0; i<P; i++) {
  For (j=i+1; j<P; j++) {
    // Compare buffer indices for collision check
    collision[i][j] = (buffer_idx[i] == buffer_idx[j]);

    // The lower node index, the higher priority
    priority [i][j] = (node_idx[i] < node_idx[j]);

    // Collision check for the lower triangle
    collision[j][i] = collision[i][j];
    priority [j][i] = ~priority[i][j];
  }
  // Collision never occurs for the same PE
  collision[i][i] = 1'b0;
  priority [i][i] = 1'b1;

  For (j=0; j<P; j++) {
    // check stall for each (i,j) pair
    stall[i][j] = (collision[i][j] & ~priority[i][j]);
  }

  collision_check[i] = |(stall[i]);
}

```

interface. Data loading for Read buffers is shown in Fig. 6.2(a). We assume that $P = 2$ and $|\chi| = 4B$ (i.e., each node and edge data consists of 4 blocks). A sample 3x3 MRF is depicted on the left, where the indices for nodes and horizontal/vertical edges are assigned based on the diagonal ordering. The diagram on the right shows the node and edge data loaded into the corresponding Read buffers. As $P = 2$, there are two sets of Read buffers for nodes (RD_NODE[i]), horizontal (RD_EHOR[i]) and vertical (RD_EVER[i]) edges, which are associated with two memory read ports (MemRD[i]). Node and edge data in the external memory are grouped into streams via the modulo operation so that each memory port can feed each stream to the corresponding Read buffer (e.g., a node stream $\{0,2,4,6\}$ goes to RD_NODE[0]). Note that the same memory port is shared; thus Read buffers for nodes and edges take turns fetching data. The loaded messages are later temporarily stored in Feedback buffers (FB_EHOR[i], FB_EVER[i]) for proper message computation; e.g., the new messages *EHOR0* and *EVER0* computed from *NODE0* are stored in FB_EHOR[0] and FB_EVER[0], respectively, to be used for processing *NODE1* and *NODE2*, respectively.

The loaded streams are used to compute messages. Fig. 6.2(b) specifies the node indices that go in and out of each PE at each cycle, where each node takes edge data from Read and Feedback buffers to compute two messages in horizontal and vertical directions. We assume that the PEs have two cycles of latency for message computation, and data in Read buffers is pre-loaded and ready for use. At first, *PE*[0] starts processing *NODE0* while *PE*[1] waits for a new horizontal message from it. Since *NODE0* is a node in the left-top corner of the MRF, only data from Read buffers are required for computing new messages. At cycle 3, the first block of the new messages *EHOR0* and *EVER0* are computed and stored in Feedback buffers (i.e., FB_EHOR[0] and FB_EVER[0]), initiating the process for *NODE1* by *PE*[1]. After that, every PE can process the next node in its stream as soon as its current node is processed, since the newly computed messages are ready for use in the Feedback buffers. At the end, it takes 22 cycles to process 36 blocks of nodes (i.e., 9 nodes), achieving throughput of 0.82 block per cycle per PE.

However, as data loading is done independently for each Read buffer and its latency depends solely on the associated memory port, there can be unexpected delay in fetching data to Read buffers, inserting bubbles in the pipeline and causing collisions. Fig. 6.2(c)

Table 6.2: Relative resource overhead of memory interface for different parallel factors.

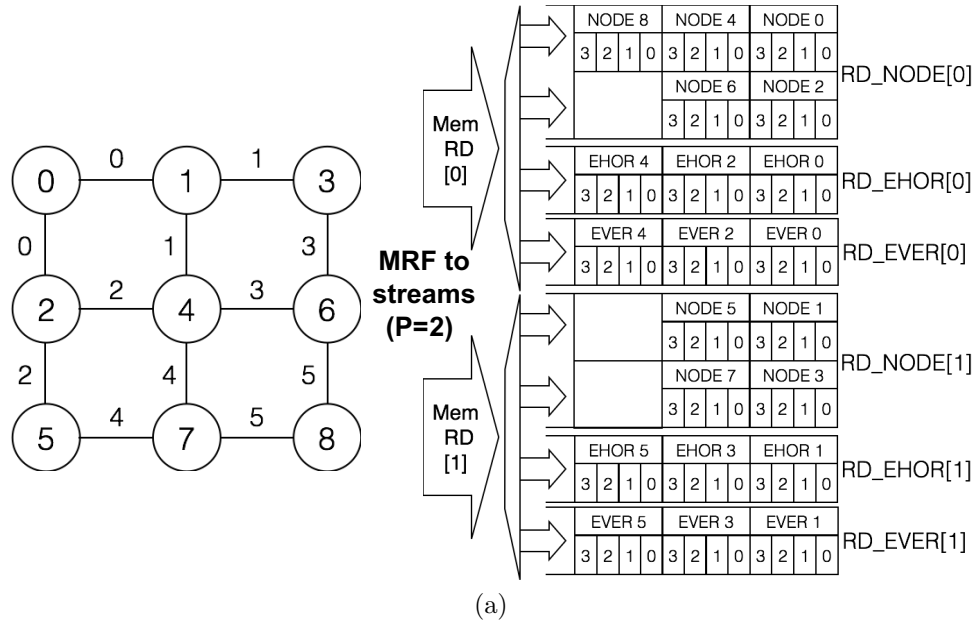
FPGA resources	P=4	P=8
Slice registers	100%	100%
Slice LUTs	208%	321%

illustrates this scenario, where 3 cycles of delay occur in data loading of `RD_NODE[0]` at cycle 5. In this case, $PE[0]$ stalls with three bubbles in the pipeline and resumes its processing for `NODE2` at cycle 8, while $PE[1]$ processes `NODE1` and `NODE3`. Then, at cycle 11, while $PE[0]$ is processing the last block of `NODE2`, $PE[1]$ attempts to process `NODE5` that takes `EVER2` in `FB_EVER[0]`. Since $PE[0]$ also accesses `FB_EVER[0]` to fetch `EVER0`, a collision occurs at this point, asserting `collision_check[1] = 1`. Thus, $PE[1]$ stalls one cycle.

The proposed block-parallel memory interface have two design parameters to configure, the parallel factor P and the block size B , and the choice of P and B affects overall performance as well as the resource utilization. If P is high, we can keep small block size, and thus we can avoid wasting the memory bandwidth for applications with a large range of number of labels. As an example, in case of STRM-TRWS in Chapter 3, $MemBWUtil = 16$, $P = 1$, and $B = 16$, and thus it wastes 12 labels of memory bandwidth for the Venus stereo matching task since it takes two blocks to convey data for 20 labels. If $P = 4$, there is no waste in memory bandwidth. However, the higher P results in the more complex memory interface, causing higher usage in hardware resources. Table 6.2 shows that FPGA resource utilization for the memory interface more than doubles as P doubles. Therefore, it is important to find the best option given the platform.

6.1.2.2 Streaming Jump Flooding for fast approximate message computation

For high performance TRW-S inference, it is important to quickly and accurately update the messages. In general, message computation has complexity of $O(|\chi|^2)$. STRM-TRWS in Chapter 3 reduces this complexity to $O(|\chi|)$ by limiting its capability to the truncated cost functions, which are common in stereo matching. [43] proposes a $O(|\chi|)$ message update



(a)

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
PE 0	i	0	0	0	0	2	2	2	2	4	4	4	4	6	6	6	6	8	8	8	8				
	o			0	0	0	0	2	2	2	2	4	4	4	4	6	6	6	6	8	8	8	8		
PE 1	i			1	1	1	1	3	3	3	3	5	5	5	7	7	7	7							
	o				1	1	1	1	3	3	3	3	5	5	5	5	7	7	7	7					

(b)

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
PE 0	i	0	0	0	0	B	B	B	2	2	2	2	4	4	4	4	6	6	6	6	8	8	8	8	
	o			0	0	0	0	B	B	B	2	2	2	2	4	4	4	4	6	6	6	6	8	8	8
PE 1	i			1	1	1	1	3	3	3	3	C	5	5	5	5	7	7	7	7					
	o				1	1	1	1	3	3	3	3	C	5	5	5	5	7	7	7	7				

Figure 6.2: Example of processing 3x3 MRF using two PEs. (a) Node and edge data for a 3x3 MRF mapped to two sets of streams for nodes ($\{0,2,4,6,8\}$, $\{1,3,5,7\}$), horizontal edges ($\{0,2,4\}$, $\{1,3,5\}$), and vertical edges ($\{0,2,4\}$, $\{1,3,5\}$) are loaded to the corresponding Read buffers. (b) The loaded streams are used by two PEs to compute messages for each node. The diagram shows indices of nodes that go in and out of each PE. Assume that Read buffers are preloaded. (c) Processing nodes using two PEs with three cycles of unexpected delay in Read buffer of PE0 at cycle 5.

algorithm (i.e., distance transform) for linear/quadratic cost functions, but this algorithm is strictly sequential. As a promising alternative, [50] proposes a parallel $O(|\chi| \log |\chi|/P)$ message update scheme based on the Jump Flooding algorithm, initially proposed for GPU implementation of distance transform in [51].

The message computation in (2.14) and (2.15) can be thought as a distance transform from $H(x_p) = \frac{1}{2}(\theta_p(x_p) + \sum_{s \in Nb(p)} M_{sp}(x_p)) - M_{qp}(x_p)$ to $M_{pq}(x_q)$ via distance measure $\theta_{pq}(x_p, x_q)$, which can be rewritten as

$$M(x_q) = \min_{x_p} \{H(x_p) + \theta_{pq}(x_p, x_q)\}. \quad (6.2)$$

The optimal $M(x_q)$ can be found by full search over x_p and x_q , which takes $O(|\chi|^2)$ computation. However, the Jump Flooding algorithm converts this computation into search over a trellis. Fig. 6.3(a) illustrates the concept of Jump Flooding. At level i , each label searches for the locally best choice among the candidate labels with a stride $d = (|\chi|/2^i)$, written as follows:

$$S_{i+1}(x_q) = \arg \min_{x_p \in \{S_i(x_q+d), S_i(x_q-d), S_i(x_q)\}} H(x_p) + \theta_{pq}(x_p, x_q), \quad (6.3)$$

where the initial label is set as $S_0(x_q) = x_q$.

As $d \rightarrow 1$, each label refers to all the other labels at least once, leading to a locally best label $\tilde{x}_q^* = S_i(x_q)$ that is close to the true optimal label $x_q^* = \arg \min_{x_p} \{H(x_p) + \theta_{pq}(x_p, x_q)\}$. Then, \tilde{x}_q^* can be used to reconstruct an approximate message as

$$\tilde{M}(x_q) = H(\tilde{x}_q^*) + \theta_{pq}(\tilde{x}_q^*, x_q) \simeq M(x_q^*).$$

Since the number of iterations is at most $\log |\chi|$, the total complexity is reduced to $O(|\chi| \log |\chi|)$. Also, note that for each i , computation of (6.3) can be parallelized across the labels. Therefore, complexity of the parallel implementation of Jump Flooding becomes $O(|\chi| \log |\chi|/P)$.

Although Jump Flooding significantly reduces the complexity of message update, it does not always find the optimal labeling due to its greedy search scheme. However, it is shown in [50] and [51] that the results of Jump Flooding are very close to the optimal solutions. In

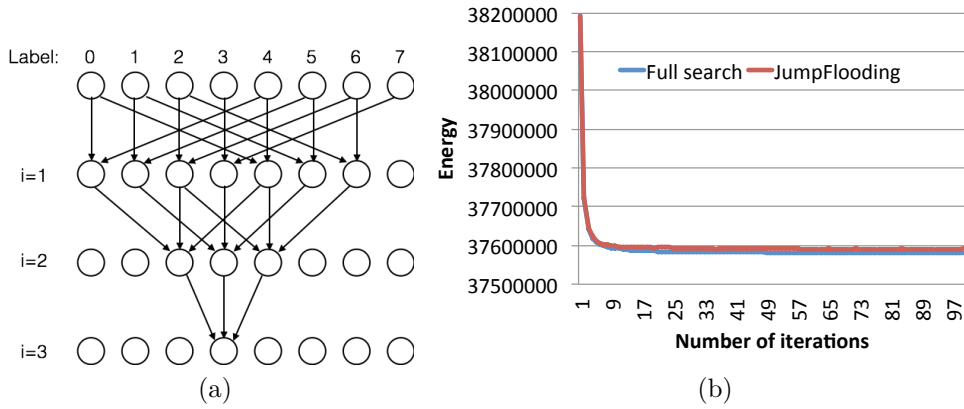


Figure 6.3: (a) Illustration of Jump Flooding algorithm. (b) Comparison of minimum energy for an image denoising task (House): Full search vs. Jump Flooding.

particular, if the smoothness cost is linear, it always finds the optimum messages. Even for non-linear smoothness costs, Jump Flooding finds the messages that are similar to the accurate ones since the candidates with different message values are highly penalized throughout the search. Thus, Jump Flooding achieves good empirical performance, as demonstrated in Fig. 6.3(b) that the full search and Jump Flooding produce almost identical energy curves for an image denoising task (House) which utilizes the quadratic smoothness cost.

To support various smoothness costs for message computation and achieve high throughput, we designed a streaming architecture based on Jump Flooding for each PE in Fig. 6.1, which is shown in Fig. 6.4. The JF-based message passing unit computes the messages in horizontal and vertical directions, and as shown in 6.4(a), the overall architecture of JF-based message passing is similar to the message passing unit in Chapter 3 (Fig. 3.5); the reparameterize unit (REPARAM) takes node and edge data from Read and Feedback buffers, respectively, to compute $H(x_p)$ in (6.2), and two Jump Flooding units (JF-UNITs) take this $H(x_p)$ as input to compute approximate messages $\tilde{M}(x_q)$ in horizontal and vertical directions via Jump Flooding (6.3).

It is important that the message update operation, like the message update unit (MSG_UPD) in our prior work, be fully pipelined for high throughput. Thus, we design a novel streaming architecture for Jump Flooding. We start from rewriting the original Jump Flooding operation (6.3) as a two-step update rule for the pipelined implementation as follows:

$$S_{i+1}(x_q) = \arg \min_{x_p \in \{x_q \pm d, x_q\}} H_i(x_p) + \theta_{pq}(S_i(x_p), x_q), \quad (6.4)$$

$$H_{i+1}(x_q) = H_i(S_{i+1}(x_q)).$$

This update rule highlights the fact that $S_{i+1}(x_q)$ can be found by considering the input vectors $S_i(x_q)$ and $H_i(x_q)$ together with their delayed version, i.e., $S_i(x_q \pm d)$ and $H_i(x_q \pm d)$, respectively. This motivates the FIR-filter-like streaming architecture for Jump Flooding. To illustrate this idea, Fig. 6.4(b) shows an example of 8-label message update. The solid and dashed arrows represent the label chosen and not chosen at level 1, respectively, and the numbers in the circles indicate the locally best label at level i , $S_i(x_q)$. As before, the initial label is set as $S_0(x_q) = x_q$, and H_0 is also initialized as $H_0(x_q) = H(x_q)$. At level 1, we compute (6.4) to find $S_1(x_q)$. Different from (6.3), we also update $H_1(x_q)$ based on $S_1(x_q)$ so that we do not need to keep the original vector $H(x_q)$. This results in FIR-filter-like feed-forward execution of Jump Flooding, since output of each stage can be computed using only its input passed from the previous stage. For example, to approximate for $M(0)$ at level 3, it is not necessary to maintain the input vector H throughout the stages as (6.3) does, since it can be obtained from $H_3(0)$ which is passed from the previous stage.

Motivated by the above discussion, the Jump Flooding scheme is implemented as a fully pipelined streaming architecture, as shown in Fig. 6.4(c). Each stage in JF-UNIT performs computation for (6.4) at each level. The buffers take a block of data at every cycle, and the depth of the buffer is set as d/B to allow necessary delay (i.e., the varying strides, d) for comparison of $S_i(x_q)$ and $H_i(x_q)$ with $S_i(x_q \pm d)$ and $H_i(x_q \pm d)$. The computation unit (COMP) compares three candidates (corresponding to $x_p \in \{x_q \pm d, x_q\}$ in (6.4)) through a 4-stage pipeline and passes the locally best candidate S_i along with the updated H_i and the corresponding message value M_i to the next stage. The smoothness cost (6.1) is implemented as a look-up table, where the data is indexed by the difference between two labels $|x_p - x_q|$. To save hardware resources, the smoothness cost table is decomposed according to the levels in (6.3), and thus the look-up table of the i th level contains 2^{i-1} entries. Due to its $\log |\chi|$ pipelined stages, this streaming architecture takes $O(\log |\chi|)$ COMPs, $O(|\chi|)$ storage and

latency, and a throughput of B per cycle for each PE. As a concrete example, the architecture shown in 6.4(c) compares the labels with varying strides (starting from 4) throughout three stages of COMPs with $B = 1$, and thus it takes 3 COMPs, 14 buffer storage, 7 entries for the smoothness cost table and latency of 19 cycles (7 cycles from the center buffers and 12 cycles from COMPs), achieving throughput of one block per cycle.

Furthermore, the proposed streaming architecture of JF-UNIT can be configured to support various smoothness costs. At the end of each stage, there are three options for output to be passed to the next stage: the bypassed input, the output of the center buffer, and the output of COMP. By configuring the choice of the output, we can realize various types of smoothness cost functions. For non-truncated linear/quadratic smoothness costs, the input data stream goes through all the stages to find the approximate messages. For truncated costs, the stream can skip the first few COMP stages since the search space is reduced by truncation. In case of the Potts model (i.e., $\theta_{max} = 1$), the stream can skip all the COMPs since we only need to find $\min H(x)$. To compensate the latency to find $\min H(x)$, we configure the data path to make the stream bypass some stages. In other words, the buffers in JF-UNIT can be configured to work as a configurable delay registers with the delay varying from 0 (bypass all the stages) to $2^{\log|x|-1}$ (go through all the stages). Thus, JF-UNIT can effectively support various smoothness costs used for a variety of computer vision applications.

6.1.3 Performance

In this section, we provide the performance of the proposed streaming Jump Flooding-based TRW-S accelerator for computer vision (STRM-TRWS-CV) in terms of speed and quality of inference. Table 6.4 shows comparison of execution time (1 iteration) of STRM-TRWS-CV with the software execution time (TRWS-SW) for the various computer vision tasks from Middlebury and OpenGM benchmarks [10, 24]. Stereo matching is to find the best disparity label given horizontally shifted stereo images, as explained in detail in Section 2.1. Image denoising is to restore a noisy image by inferring proper color intensities (in 8-bit grayscale) of all the pixels based on a noisy image. The labels are intensities (0-255), and the data cost

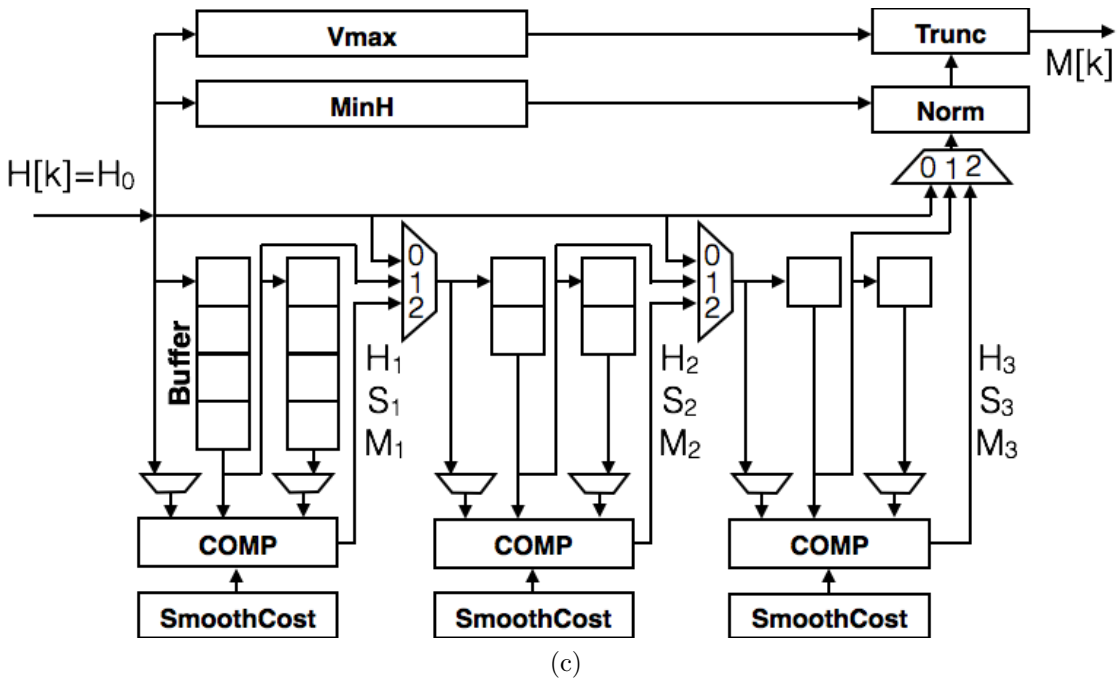
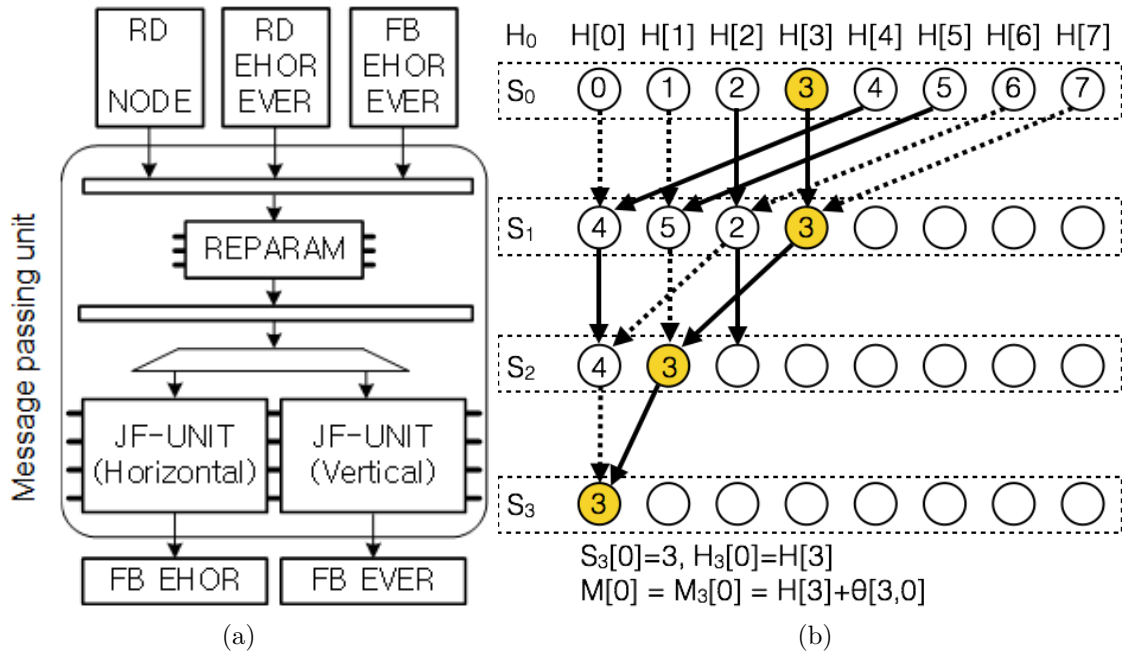


Figure 6.4: Streaming architecture for Jump Flooding-based message computation (JF-UNIT). (a) Message passing unit as a PE. (b) An example for operation of Jump Flooding. Solid and dashed arrows represent labels chosen and not chosen at each level, respectively. (c) Architecture for Jump Flooding-based message update.

Table 6.3: Summary for FPGA utilization and specification of PE.

XC5VLX330	Slice Reg	Slice LUT	BRAM	DSP48E
PE	11.4%	9.2%	0.0%	1.0%
MemIF	9.1%	9.3%	74.0%	1.6%

Specification of PE (P=4, B=4)				
Num of stages	Number of buffer entries	Block size (bits)	Depth of pipeline	Throughput (blocks/cycle)
6	34	96	70	4

for each pixel is the squared difference between the label and the observed intensity [10]. Object segmentation is to detect and segment object classes in images [24]. Each pixel of an image is associated with a discrete random variable that takes labels like grass, tree, or sky. The distribution over these variables is modeled as a likelihood of the MRF formulation, where we can assign smoothness prior to solve the problem as energy minimization. All of these benchmarks represent fundamental applications in computer vision. The detail MRF setting for each task is summarized in Table 6.1.

STRM-TRWS-CV is implemented in the same Xilinx Virtex-5 (LX330) FPGA on Convey HC-1, and the final version is configured as $P = 4$, exploiting most of the available FPGA resources (e.g., 80% of Slice Registers and 81% of Slices LUTs in total). Table 6.3 shows a summary of the FPGA utilization and the specification of our PE for $P = 4$ and $B = 4$.

The benchmark data and the execution time of TRWS-SW are from [24], in which an Intel Xeon W3550, 3.07GHz 12GB RAM machine has been used. For the stereo matching tasks, we also include the execution time of our TRW-S stereo matching system (STRM-TRWS) for comparison.

As shown in the table, STRM-TRWS-CV achieves significant speedup ($11 - 41\times$) across the tasks. Furthermore, in case of stereo matching, the execution times of STRM-TRWS-CV are comparable to STRM-TRWS, even though STRM-TRWS is highly optimized for the stereo matching application. This is due to the fact that STRM-TRWS-CV is less likely to waste the memory bandwidth than STRM-TRWS. For example, in case of Venus task, which uses 20 labels, STRM-TRWS-CV outperforms STRM-TRWS since STRM-TRWS suffers from wasting the memory bandwidth of 12 out of 32 labels due to its 16-label block size.

Table 6.4: Comparison between execution time (in second) of 1 iteration of TRWS SW, STRM-TRWS and STRM-TRWS-CV for various computer vision tasks.

Benchmarks [10, 24]	Tasks	TRWS-SW	STRM-TRWS	STRM-TRWS-CV	Speedup
Stereo Matching	Tsukuba	0.085	0.0032	0.0046	18.5
	Venus	0.352	0.0096	0.0086	40.7
	Teddy	0.514	0.0194	0.0263	19.5
Image Denoising	House	1.208	N/A	0.0436	27.7
	Penguin	0.435	N/A	0.0145	30.0
Object Segmentation	Plane	0.011	N/A	0.0007	15.5
	Building	0.016	N/A	0.0013	11.3
	Car	0.023	N/A	0.0013	16.2

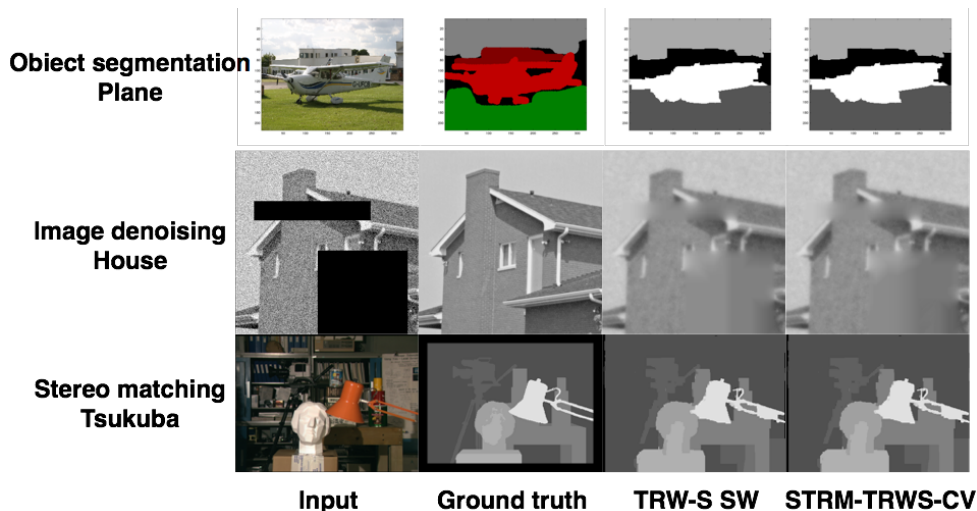


Figure 6.5: Comparison of inference quality between TRWS-SW and STRM-TRWS-CV for various computer vision tasks.

This demonstrates that our STRM-TRWS-CV architecture can support a wide variety of computer vision tasks more effectively than the previous architecture.

We further compare the output of inference between TRWS-SW and STRM-TRWS-CV for various computer vision tasks. As shown in Fig. 6.5, the outputs of TRWS-SW and STRM-TRWS-CV are almost identical, indicating that our Jump Flooding-based hardware architecture achieves significant speedup with little compromise in the output quality.

Next, we evaluate our hardware with the taxonomy of probabilistic inference platforms summarized in Table 1.1. We focus on stereo matching as it provides standardized performance comparison through the well-known benchmarks suite [10]. Table 6.5 shows various

performance results of stereo matching for the different belief propagation implementations along with our TRW-S implementations (STRM-TRWS and STRM-TRWS-CV). For fair comparison, we consider the same Tsukuba stereo matching task and measure the performance in terms of “Million Disparity Estimations per second” (MDE/s) that achieves the similar energy values. MDE/s is defined as (Number of pixels \times Disparity range (i.e., number of labels) \times Frame rate), and it reflects how large a problem a hardware can handle at what speed (the higher, the better). The reported execution times from the literature are used. Also, the numbers of iterations are chosen to achieve similar energy. The energy values for the tile-based BP and OpenGM are from [1] and [24], respectively, and we obtain the energy value for PATRA by SW simulation. The energy value for GraphGen is assumed to be equal to the STRM-TRWS result since they use the same TRW-S algorithm.

As shown in Table 6.5, our TRW-S implementation customized for stereo matching (STRM-TRWS) achieves the highest MDE/s compared to the other belief propagation implementations based on ASIC, FPGA, GPU and CPU platforms. In particular, our hardware outperforms PATRA since its lower throughput is compensated by faster convergence of the TRW-S algorithm over TRW. Even the configurable TRW-S implementation (STRM-TRWS-CV) shows compelling performance (achieving higher MDE/s than others except PATRA), demonstrating superiority of our hardware architectures.

As MDE/s is a very popular measure used in stereo matching literature, we can also use it to compare our hardware with the other non-BP based stereo matching implementations. The stereo matching implementations are grouped as local, semi-global, and global, based on the scope of search for the best disparity label. The local methods such as sum of difference (SAD) or adaptive support weight (ADSW) focus on local windows to find the best label corresponding to the disparity of matching pixels in the stereo images. On the other hand, the global methods such as dynamic programming (DP) and belief propagation consider all the pixels in the image for search (i.e., the window size is equal to the image size). The semiglobal method (i.e., semiglobal matching, SGM) is an alternative between the two, which limits search space from the entire image to a few streaks (e.g., 8 streaks from left, right, up, down, and four diagonal directions) in order to reduce the computational complexity. As the search window becomes larger, the greater the chance of finding the correct disparity

Table 6.5: Performance evaluation of streaming TRW-S hardwares within the taxonomy of probabilistic inference platforms.

Taxonomy	Application specific			Configurable		General purpose
HW type	ASIC	FPGA	GPU	FPGA		CPU
Name	Tile-based BP[1]	PATRA [26]	Tile-based BP [1]	STRM-TRWS-CV	GraphGen[31]	OpenGM[24]
Platform	UMC 90nm (185MHz)	Maxeler Max Workstation Xilinx Virtex-6 SX475T (100MHz)	NVIDIA GeForce 8800 GTS	Convey HC-1 Xilinx Virtex-5 LX330 (150MHz)	Terasic DE4 (150Mhz)	Intel Xeon W3550 (3.07GHz)
Task	Tsukuba (384x288, 16 labels)					
Iterations	5 (TI=20)	18	5 (TI=20)	5	5	5
Energy	396953	400192	396953	394434	393434	393434
Time (s) per iteration	N/A	0.0012	0.0195	0.0032	0.0070	0.0850
MDE/s	35.8	81.9	18.2	76.9	50.6	4.2

for each pixel, but the greater computational complexity is required [52].

Table 6.6 summarizes various stereo matching implementations using different hardware platforms including multi-core CPUs, GPU, ASIC, and FPGA. As shown in Table 6.6, the global methods (e.g., Wang 2006 [53]) achieve lower MDE/s than the semiglobal or local methods in general. However, our streaming hardware experiments (STRM-TRWS and STRM-TRWS-CV) show promising MDE/s compared to the other implementations, suggesting that our hardware can achieve compelling speed while maintaining the global search space for better inference quality.

6.2 Error Resilient TRW-S for Computer Vision via ANT

In Chapter 5, we discussed error resiliency of TRW-S message passing hardware and implication of ANT for it. However, the discussion was mainly focused on empirical results rather than thorough theoretical understanding. In particular, the previous error analysis of TRW-S was limited to a very simple MRF setting associated with the Potts model and binary labels. To better understand the behavior of erroneous inference applied to various machine learning applications, error resiliency analysis of message passing inference with a more general MRF setting is required. To this end, we discuss theoretical reasoning about error resiliency of TRW-S under general MRF setting. In particular, we discuss the relationship between the error resiliency and the coupling strength imposed by the smoothness cost of the message passing inference. Furthermore, we discuss the implication of ANT for this theoretical analysis, which is strongly supported by empirical results.

6.2.1 Error resiliency of message passing inference under varying coupling strength

As discussed in Chapter 5, message passing inference based on Sum-Product or Max-Product type belief propagation has exhibited intrinsic error resiliency [47, 34]. In [59], Ihler et al. have shown a contraction bound of Sum-Product BP that implies its inherent error resiliency, but the scope of their study was limited by the labels in the continuous domain. As this

Table 6.6: Performance evaluation of streaming TRW-S hardwares with other non-BP stereo matching implementations. *MDE/s is calculated for one iteration of TRW-S inference in order to compensate the fact that the other methods (e.g., SGM, DP and ADSW) perform one-time search for each pixel.

	Work	MDE/s	Type	Method	Platform
CPU	Gehrig 2010 [54]	72.7	Semiglobal	SGM	Intel Core i7-975ex, 3GHz
GPU	Wang 2006 [53]	53.0	Global	ADSW + DP	ATI Radeon XL1800
	Ernst 2008 [55]	63.9	Semiglobal	SGM	NVIDIA GeForce 8800 ULTRA
ASIC	Chang 2010 [56]	272.5	Local	ADSW	UMC 90nm (95MHz)
	Hirschmuller 2012 [57]	328.3	Semiglobal	SGM	Xilinx Virtex-5 (125MHz)
FPGA	Ttofs 2015 [58]	1179.0	Local	ADSW	Kintex-7 FPGA (200MHz)
	STRM-TRWS	553.0*	Global	TRW-S	Xilinx Virtex-5 (150MHz)
	STRM-TRWS-CV	384.7*			

bound sheds light on the understanding of error resiliency of belief propagation, we start from their framework to analyze the error resiliency of Max-Product type inference.

We first define the message update rule for the Max-Product inference as

$$m_{st}(x_t) \propto \max_{x_s} \{ \phi(x_s, x_t) \cdot \phi_s(x_s) \cdot \prod_{u \in Nb(t) \setminus s} m_{us}(x_s) \} = \max_{x_s} \{ \phi(x_s, x_t) \cdot M_{st}(x_s) \}, \quad (6.5)$$

where $x_s \in \chi$ is a discrete label, ϕ_s and ϕ_{st} are the potentials corresponding to the likelihood and the prior (as described in Chapter 2), and $Nb(t) \setminus s$ is a set of nodes that are direct neighbors of a node t excluding s . Also, assume that all the entries of ϕ_s and ϕ_{st} have positive values and all the messages are initialized as a uniform distribution. Note that this Max-Product type inference is equivalent to $-\log$ of (6.2). We now consider the case when arithmetic errors occurred during the computation of (6.5), producing an erroneous message $\hat{m}_{st}(x_t)$. We define an error as a difference (in ratio) between the error-free and the erroneous messages, $e(x) \triangleq \hat{m}(x)/m(x)$. Then, we can write an erroneous message update as

$$\hat{m}_{st}(x_t) \propto \max_{x_s} \{ \phi(x_s, x_t) \cdot \phi_s(x_s) \cdot \prod_{u \in Nb(t) \setminus s} \hat{m}_{us}(x_s) \} = \max_{x_s} \{ \phi(x_s, x_t) \cdot M_{st}(x_s) \cdot E_{st}(x_s) \}. \quad (6.6)$$

To analyze the behavior of $e(x)$, we need to define a measure. In this work, we follow the motivation presented in [59] to use a *dynamic range* as our measure. The dynamic range of a vector $e(x)$ and a matrix $\phi(x, y)$ are defined as

$$d(e(x)) \triangleq \max_{a,b} \sqrt{\frac{e(a)}{e(b)}}, \quad d(\phi(x, y))^2 \triangleq \max_{a,b,c,d} \frac{\phi(a, b)}{\phi(c, d)}. \quad (6.7)$$

Therefore, $\log d(e)$ represents the largest difference among the entries of $e(x)$.

For Sum-Product BP (i.e., \int instead of \max in (6.5)), Ihler et al. have shown that the following bound holds [59]:

$$d(e_{ts}) \leq \frac{d(\phi_{ts})^2 d(E_{ts}) + 1}{d(\phi_{ts})^2 + d(E_{ts})}. \quad (6.8)$$

(6.8) indicates that the dynamic range of errors on the updated messages is bounded by both the dynamic range of ϕ_{ts} and E_{ts} . This can be seen by describing limiting behavior; $d(e_{ts}) \leq d(\phi_{ts})^2$ as $d(E_{ts}) \rightarrow \infty$, and $d(e_{ts}) \leq d(E_{ts})$ as $d(\phi_{ts})^2 \rightarrow \infty$. In other words, the increasing dynamic range of error in the incoming messages has limited effect on the dynamic range of error in the output message, implying error resiliency of the Sum-Product algorithm.

We expected that a similar bound for the Max-Product inference would exist. However, it was not tractable to extend the analytical approach in [59] to the Max-Product to claim a bound similar to (6.8), since non-linearity of the max operation hinders adopting the approach of [59] based on linear combination of the message update rule. As an alternative, we can evaluate the asymptotic behavior of (6.7).

We start from the definition of the dynamic range as follows:

$$d(e)^2 = d(\hat{m}/m)^2 = \max_a \left\{ \frac{\max_{x'} \{ \phi(x', a) \cdot M(x') \cdot E(x') \}}{\max_x \{ \phi(x, a) \cdot M(x) \}} \right\} \cdot \max_b \left\{ \frac{\max_x \{ \phi(x, b) \cdot M(x) \}}{\max_{x'} \{ \phi(x', b) \cdot M(x') \cdot E(x') \}} \right\}.$$

For simplicity, consider the following pairwise potential:

$$\phi(x, y) \triangleq \exp(-\lambda|x - y|^p), \quad p = 1 \text{ or } 2.$$

Note that this pairwise potential is equivalent to $-\log$ of (6.1) where $\omega_{st} = 1$ and $\theta_{max} = \infty$, which is widely used in our computer vision applications. We claim that the following asymptotic bounds hold:

$$\begin{aligned} \text{If } d(\phi)^2 \rightarrow \infty, \text{ then } d(e)^2 &= d(E)^2, \\ \text{If } d(E)^2 \rightarrow \infty, \text{ then } d(e)^2 &\leq d(\phi)^4. \end{aligned} \tag{6.9}$$

The proof is as follows. If $d(\phi)^2 \rightarrow \infty$ (i.e., $\lambda \rightarrow \infty$), since ϕ dominates, the maximizing argument should be one that maximizes ϕ , which is when $x = y$ for $\phi(x, y)$. Therefore,

$$\begin{aligned}
\max_{x'} \{\phi(x', a) \cdot M(x') \cdot E(x')\} &= M(a)E(a), \\
\max_x \{\phi(x, a) \cdot M(x)\} &= M(a), \\
\max_{x'} \{\phi(x', b) \cdot M(x') \cdot E(x')\} &= M(b)E(b), \\
\max_x \{\phi(x, b) \cdot M(x)\} &= M(b).
\end{aligned}$$

By dividing the common factors $M(a)$ and $M(b)$, we get

$$d(e)^2 = \max_a \{E(a)\} \cdot \max_b \{1/E(b)\} = d(E)^2.$$

Now consider the case when $\max_x E(x) \rightarrow \infty$ and thus $d(E)^2 \rightarrow \infty$. Assume $x^* = \arg \max_x E(x)$. Since $\max_x E(x)$ now dominates, the maximizing argument should be one that maximizes E . Therefore, the following holds:

$$\begin{aligned}
\max_{x'} \{\phi(x', a) \cdot M(x') \cdot E(x')\} &= \phi(x^*, a) \cdot M(x^*) \cdot E(x^*), \\
\max_{x'} \{\phi(x', b) \cdot M(x') \cdot E(x')\} &= \phi(x^*, b) \cdot M(x^*) \cdot E(x^*).
\end{aligned}$$

By plugging this into $d(e)^2$,

$$d(e)^2 = \max_a \left\{ \frac{\phi(x^*, a) \cdot M(x^*) \cdot E(x^*)}{\max_x \{\phi(x, a) \cdot M(x)\}} \right\} \cdot \max_b \left\{ \frac{\max_x \{\phi(x, b) \cdot M(x)\}}{\phi(x^*, b) \cdot M(x^*) \cdot E(x^*)} \right\}.$$

Note that $M(x^*)$ and $E(x^*)$ are constants. By dividing the common constants,

$$d(e)^2 = \max_a \left\{ \frac{\phi(x^*, a)}{\max_x \{\phi(x, a) \cdot M(x)\}} \right\} \cdot \max_b \left\{ \frac{\max_x \{\phi(x, b) \cdot M(x)\}}{\phi(x^*, b)} \right\}.$$

Define $\phi_{max} = 1$ and $\phi_{min} = \exp(-\lambda \cdot ||\chi| - 1|^p)$, then we can obtain the following upper bound,

$$\begin{aligned}
d(e)^2 &\leq \max_a \left\{ \frac{\phi_{max}}{\max_x \{\phi(x, a) \cdot M(x)\}} \right\} \cdot \max_b \left\{ \frac{\max_x \{\phi(x, b) \cdot M(x)\}}{\phi_{min}} \right\} \\
&\leq \left\{ \frac{\phi_{max}}{\phi_{min} \cdot \max_x \{M(x)\}} \right\} \cdot \left\{ \frac{\phi_{max} \cdot \max_x \{M(x)\}}{\phi_{min}} \right\} \\
&\leq \left\{ \frac{\phi_{max}}{\phi_{min} \cdot \max_x \{M(x)\}} \right\} \cdot \left\{ \frac{\phi_{max} \cdot \max_x \{M(x)\}}{\phi_{min}} \right\} \\
&= \frac{\phi_{max}^2}{\phi_{min}^2} = d(\phi)^4.
\end{aligned}$$

Therefore, we can find the asymptotic bounds for $d(e)^2$, which are equivalent to the asymptotic bounds of Sum-Product BP 6.8. These bounds imply that similar error resiliency can be expected for the Max-Product type inference such as TRW-S.

6.2.2 Empirical support for theoretical error bounds

In this section, we provide experimental results to support our theoretical analysis described in Section 6.2.1. To evaluate the impact of error injection to the inference quality, we employ the same software simulator used in Chapter 5, but this time, we also change the coupling strengths λ of the smoothness cost. We run TRW-S for the Tsukuba stereo matching task, but for simplicity in analysis, we use linear smoothness cost with λ . Errors with varying magnitudes ($-2^{15} \sim 2^{15}$) are injected to the message computation with 1% injection rate. Fig. 6.6 shows the impact of the injected errors on the energy minimization performance for the different coupling strength λ . We measure degradation of inference quality in terms of energy increment; to measure the energy increment solely due to the error injection, we offset the erroneous energy with the error free energy, i.e., $Energy_{erroneous}(\lambda) - Energy_{error-free}(\lambda)$.

Overall, an error resiliency trend similar to that observed in Chapter 5 can be found; the TRW-S inference is resilient to the small magnitude errors but vulnerable to the large magnitude ones. However, one thing to note is that, as the error magnitude becomes large, the more degradation of inference quality can be observed as the larger λ is used. This trend is consistent with our analysis in 6.2.1 that, as the dynamic range of input errors becomes dominant, its impact on the output error becomes more bounded by the dynamic range of the pairwise potential, i.e., the coupling strength λ .

Next, we explore the impact of ANT on the the quality of inference with the erroneous message computation. We use a simple LSB-truncated reduced precision replica (RPR) as our estimator of ANT. Fig. 6.7(a) shows the degradation of inference quality vs. error magnitude for different λ . As can be seen, RPRs with larger than 8-bit precision effectively compensate for the errors and maintain low degradation of inference quality. Also, as λ increases, the larger degradation of inference quality is observed for the case without ANT. However, note that the error compensation of RPRs with 8-bit or larger precision is

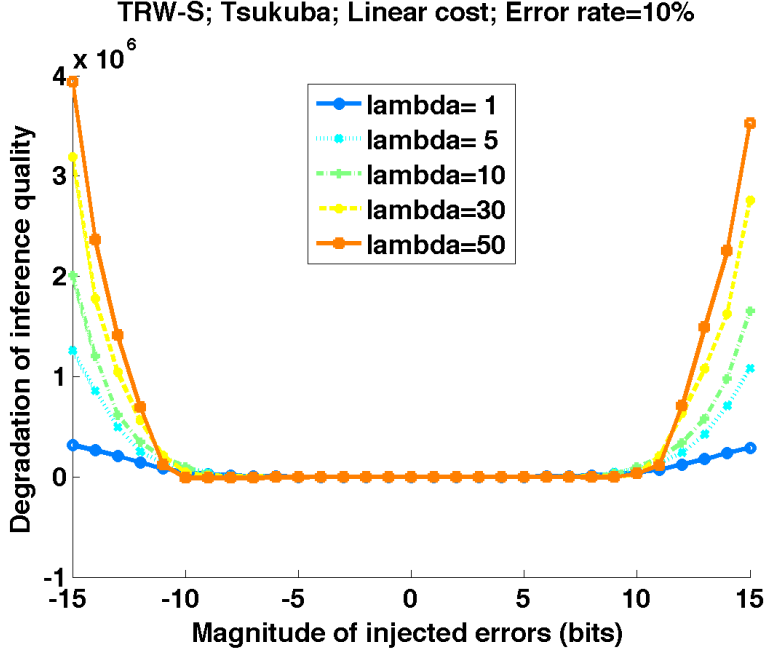


Figure 6.6: Impact of error injection on inference quality for different coupling strengths.

consistent regardless of λ . This unique trend is more distinct if we plot the same graph as the degradation of inference quality vs. λ . As shown in Fig. 6.7(b), as the error magnitude increases, the degradation of inference quality increases more drastically as λ increases. However, once protected by RPRs, the degradation of inference quality becomes less affected by λ . In other words, ANT enhances the error resiliency of the message passing inference regardless of the coupling strength.

We can explain this trend using our error analysis. From (2.18), one can show that the difference between the output of ANT y_{ANT} and the error-free output y_o is bounded by the estimation error ϵ and the ANT threshold τ as follows:

$$|y_{ANT} - y_o| \leq \epsilon + \tau.$$

In other words, ANT can replace a large magnitude error with a small magnitude one, decreasing the dynamic range of error. Therefore, from (6.9), the errors on the output that are once bounded by the coupling strength due to large magnitude input errors now become bounded by the input errors with low dynamic ranges thanks to error compensation by ANT. Therefore, the error resilience of the message passing inference can be greatly enhanced by

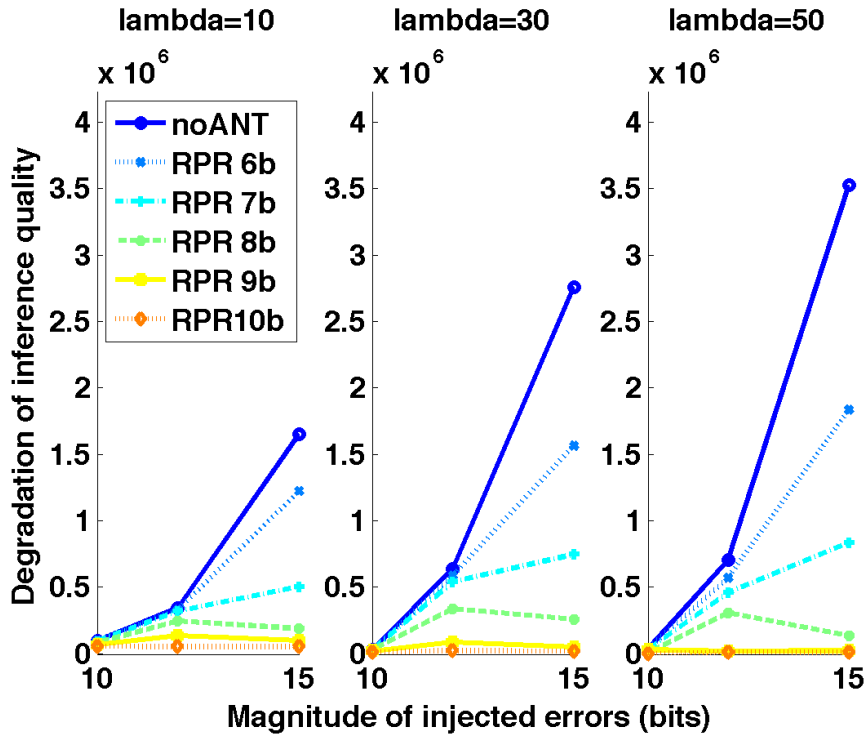
ANT regardless of the choice of coupling strength.

To verify our analysis, we perform stereo matching tasks with erroneous TRW-S under more realistic MRF settings; we run TRW-S for Tsukuba, Venus and Teddy using the different configurations specified in Table 6.1. The experimental results are shown in Fig. 6.8-6.10. Similar trends can be observed from all these tasks, supporting our analysis. This also implies that with help of ANT, one can realize a variety of computer vision applications using various coupling strengths under erroneous computational fabric, since ANT will effectively compensate for the errors regardless of the choice of the coupling strength.

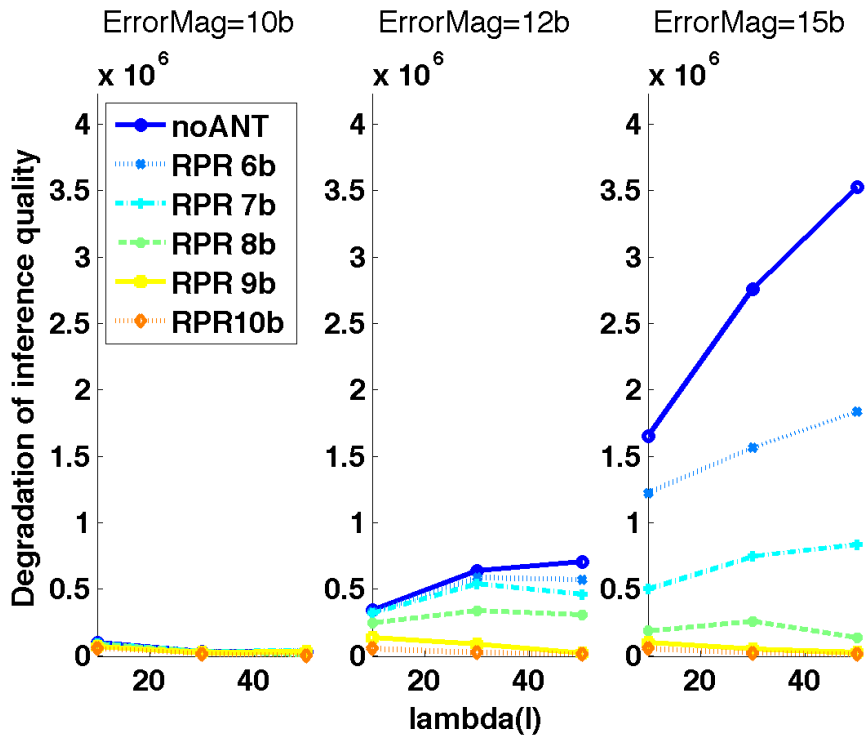
We further consider the impact of error characteristic of the message passing inference on the perceptual quality. Fig. 6.11 shows comparison of bad pixel ratio (BPR) for stereo matching of Teddy task using TRW-S with different λ s (i.e., $\lambda = 4$ or 10). When setting up an MRF for stereo matching, large λ is often preferred to motivate a smooth label assignment. The first row of Fig. 6.11 shows the disparity maps of Teddy task with different λ s, and as expected, the larger λ achieves the lower BPR. However, as we discovered in the previous sections, the larger λ is, the more the quality of inference is affected by errors, resulting in higher BPR, as shown in the second row of Fig. 6.11. Then the last row of Fig. 6.11 shows that ANT with 8-bit precision RPRs can effectively compensate for the errors and reduce BPR. This demonstrates that ANT can be used to enable error resilient processing of TRW-S for a wider variety of MRF settings.

6.3 Summary

In this chapter, we have discussed efforts to extend our high performance and error resilient probabilistic inference system toward more general purpose computing. We have proposed a novel block-parallel architecture to support three different MRF inference models, for stereo matching, for denoising, and for object segmentation, that are popular in computer vision and represent core components of the popular Middlebury and OpenGM benchmark suite [10, 24]. We then provided more thorough understanding about the error resiliency of the inference method, leading to a strong reasoning about effective ANT-based stochastic error compensation.



(a)



(b)

Figure 6.7: (a) Inference quality vs. error injection for different coupling strength and ANT, (b) Inference quality vs. coupling strength for different error magnitude and ANT.

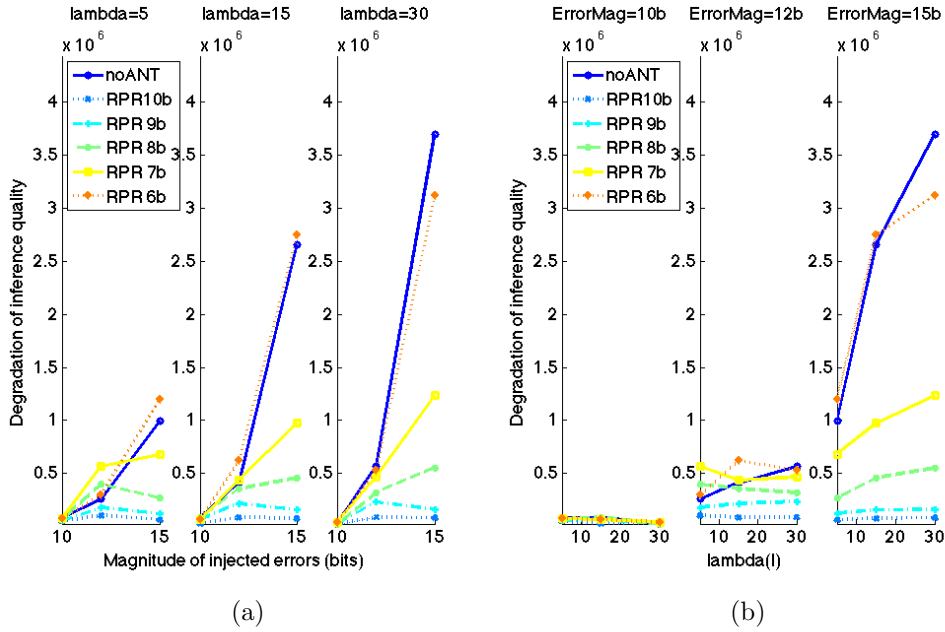


Figure 6.8: Tsukuba: (a) Inference quality vs. error injection for different coupling strength and ANT, (b) Inference quality vs. coupling strength for different error magnitude and ANT.

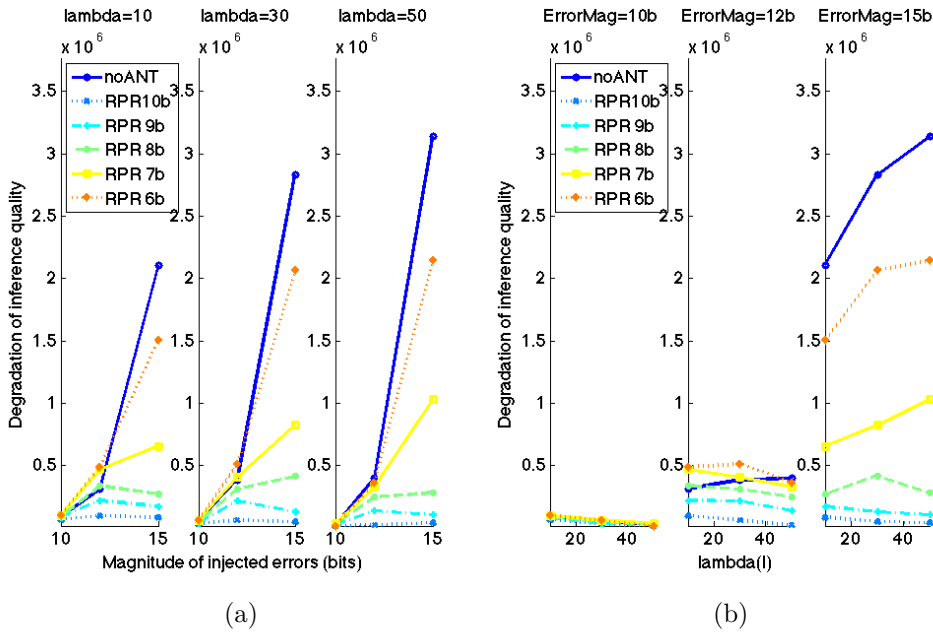


Figure 6.9: Venus: (a) Inference quality vs. error injection for different coupling strength and ANT, (b) Inference quality vs. coupling strength for different error magnitude and ANT.

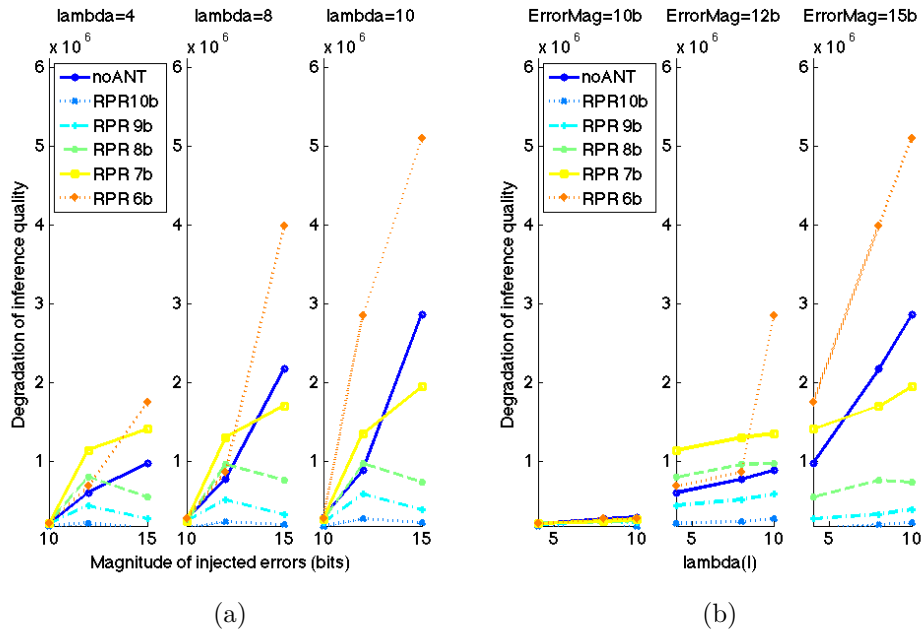


Figure 6.10: Teddy: (a) Inference quality vs. error injection for different coupling strength and ANT, (b) Inference quality vs. coupling strength for different error magnitude and ANT.

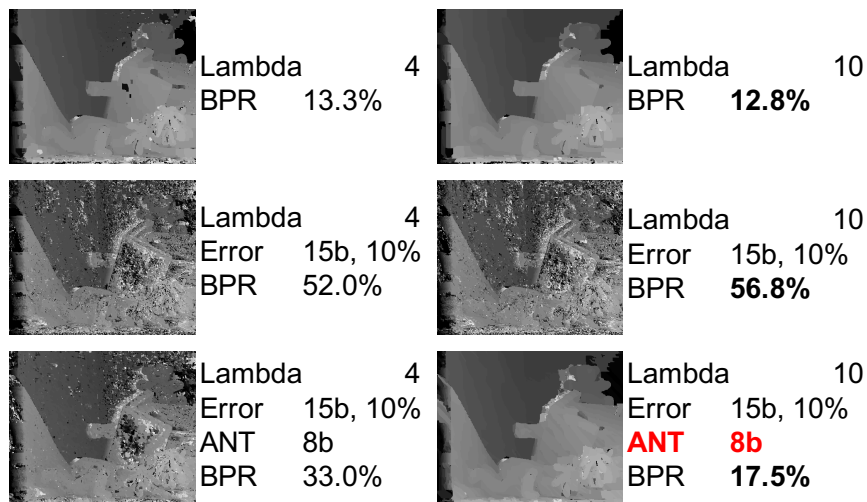


Figure 6.11: Comparison of bad pixel ratio (BPR) for stereo matching of Teddy task using TRW-S with different λ s.

Chapter 7

Conclusion

In this work, we designed and demonstrated a set of novel high performance and error resilient probabilistic inference systems. First, we implemented and benchmarked a video-rate stereo matching system implemented on a hybrid CPU+FPGA platform (Convey HC-1). We modeled the stereo task as a statistical inference on a Markov random field, which was reliably solved by the sequential tree-reweighted (TRW-S) algorithm. We proposed both algorithmic and frame level parallelization techniques for effective implementation of TRW-S inference on the hybrid platform. The pipelined streaming TRW-S architecture associated with three frame level optimization schemes (function level pipelining, frame level parallelization, and scene change detection based message reuse) achieved the goal of high quality video-rate stereo matching. Experimental results show that this system outperforms existing belief propagation based GPU/ASIC implementations in terms of accuracy and speed of inference.

Second, we studied the error propagation characteristics of an iterative message passing based stereo image matching application in depth. We observed that the message passing hardware has intrinsic robustness to small magnitude errors but is vulnerable to large magnitude errors. Our approach to apply a popular statistical error compensation technique (SEC) called algorithmic noise tolerance (ANT) exploited the error characteristic of the message passing hardware to remarkably enhance its error resiliency, which was further traded for significant energy savings.

Finally, we extended our high performance and error resilient probabilistic inference system toward more general purpose MRF computing. We proposed a novel block-parallel architecture to support various MRF inference models that are popular in computer vision, which achieved 10 – 40× speedup over a standard SW implementation for stereo matching,

image denoising, and object segmentation benchmarks. We then provided thorough understanding about the error resiliency of the inference method, leading to strong reasoning about effective ANT-based stochastic error compensation.

References

- [1] C.-K. Liang, C.-C. Cheng, Y.-C. Lai, L.-G. Chen, and H. H. Chen, “Hardware-efficient belief propagation,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 21, no. 5, pp. 525–537, 2011.
- [2] Q. Yang, L. Wang, R. Yang, S. Wang, M. Liao, and D. Nister, “Real-time global stereo matching using hierarchical belief propagation,” in *BMVC*, vol. 6, 2006, pp. 989–998.
- [3] 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.
- [4] E. Kim, D. Baker, S. Narayanan, D. Jones, and N. Shanbhag, “Low power and error resilient PN code acquisition filter via statistical error compensation,” in *Proc. Custom Integ. Circuits Conf. (CICC)*, Sep. 2011.
- [5] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *International Journal of Computer Vision*, vol. 47, no. 1-3, pp. 7–42, 2002.
- [6] Convey Computer, “Convey Reference Manual,” Online: <http://www.conveycomputer.com>, Sep. 2009.
- [7] “Stereo movie sample,” Online: <http://www.stereomaker.net/sample/index.html>.
- [8] D. Che, M. Safran, and Z. Peng, “From big data to big data mining: challenges, issues, and opportunities,” in *Database Systems for Advanced Applications*. Springer, 2013, pp. 1–15.
- [9] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. The MIT Press, 2009.
- [10] 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,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 30, no. 6, pp. 1068–1080, 2008.
- [11] M. Kim and P. Smaragdis, “Single channel source separation using smooth nonnegative matrix factorization with markov random fields,” in *Machine Learning for Signal Processing (MLSP), 2013 IEEE International Workshop on*. IEEE, 2013, pp. 1–6.

- [12] S. Bauer, R. Wiest, L.-P. Nolte, and M. Reyes, “A survey of mri-based medical image analysis for brain tumor studies,” *Physics in Medicine and Biology*, vol. 58, no. 13, p. R97, 2013.
- [13] R. Hameed, W. Qadeer, M. Wachs, O. Azizi, A. Solomatnikov, B. C. Lee, S. Richardson, C. Kozyrakis, and M. Horowitz, “Understanding sources of inefficiency in general-purpose chips,” in *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3. ACM, 2010, pp. 37–47.
- [14] J. Hauswald, M. A. Laurenzano, Y. Zhang, C. Li, A. Rovinski, A. Khurana, R. G. Dreslinski, T. Mudge, V. Petrucci, L. Tang et al., “Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers,” in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2015, pp. 223–238.
- [15] Y. Boykov, O. Veksler, and R. Zabih, “Fast approximate energy minimization via graph cuts,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, no. 11, pp. 1222–1239, 2001.
- [16] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
- [17] V. Kolmogorov, “Convergent tree-reweighted message passing for energy minimization,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 10, pp. 1568–1583, 2006.
- [18] M. Miranda, “The threat of semiconductor variability,” *IEEE Spectrum*, 2012.
- [19] C. G. Almudever and A. Rubio, “Carbon nanotube growth process-related variability in cnfets,” in *2011 11th IEEE International Conference on Nanotechnology*, 2011.
- [20] N. R. Shanbhag, “Reliable and efficient system-on-a-chip design,” *IEEE Computer*, vol. 37, no. 3, pp. 42–50, Mar. 2004.
- [21] N. R. Shanbhag, R. A. Abdallah, R. Kumar, and D. L. Jones, “Stochastic computation,” in *Proc. 47th Design Automation Conf. (DAC)*, 2010, pp. 859–864.
- [22] J. I. Woodfill, G. Gordon, and R. Buck, “Tyzx deepsea high speed stereo vision system,” in *Computer Vision and Pattern Recognition Workshop, 2004. CVPRW’04. Conference on*. IEEE, 2004, pp. 41–41.
- [23] W. Van Der Mark and D. M. Gavrila, “Real-time dense stereo for intelligent vehicles,” *Intelligent Transportation Systems, IEEE Transactions on*, vol. 7, no. 1, pp. 38–50, 2006.
- [24] J. H. Kappes, B. Andres, F. Hamprecht, C. Schnorr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, J. Lellmann, N. Komodakis et al., “A comparative study of modern inference techniques for discrete energy minimization problems,” in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*. IEEE, 2013, pp. 1328–1335.

- [25] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, “Graphlab: A new framework for parallel machine learning,” in *UAI 2010, Proceedings of the Twenty-Sixth Conference on Uncertainty in Artificial Intelligence, Catalina Island, CA, USA, July 8-11, 2010*, 2010. [Online]. Available: https://dslpitt.org/uai/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=2126&proceeding_id=26 pp. 340–349.
- [26] W. Zhao, H. Fu, G. Yang, and W. Luk, “Patra: Parallel tree-reweighted message passing architecture,” in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*. IEEE, 2014, pp. 1–6.
- [27] 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.
- [28] J. Park, S. Lee, and H.-J. Yoo, “A 30fps stereo matching processor based on belief propagation with disparity-parallel PE array architecture,” in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 453–456.
- [29] M. Lin, I. Lebedev, and J. Wawrzyniek, “High-throughput bayesian computing machine with reconfigurable hardware,” in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*. ACM, 2010, pp. 73–82.
- [30] H. Kroll, S. Zwicky, R. Odermatt, L. Bruderer, A. Burg, and Q. Huang, “A signal processor for Gaussian message passing,” in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 1969–1972.
- [31] E. Nurvitadhi, G. Weisz, Y. Wang, S. Hurkat, M. Nguyen, J. C. Hoe, J. F. Martínez, and C. Guestrin, “GraphGen: An FPGA framework for vertex-centric graph computation,” in *Field-Programmable Custom Computing Machines (FCCM), 2014 IEEE 22nd Annual International Symposium on*. IEEE, 2014, pp. 25–28.
- [32] S. Hershey, J. Bernstein, B. Bradley, A. Schweitzer, N. Stein, T. Weber, and B. Vigoda, “Accelerating inference: towards a full language, compiler and hardware stack,” *arXiv preprint arXiv:1212.2991*, 2012.
- [33] R. Abdallah and N. R. Shanbhag, “Error-resilient low power Viterbi decoders,” in *Int. Symp. on Low Power Elect. and Design (ISLPED)*, 2008, pp. 111–116.
- [34] E. P. Kim and N. R. Shanbhag, “Energy-efficient LDPC decoders based on error-resiliency,” in *IEEE Workshop on Signal Process. Syst. (SiPS)*, 2012, pp. 149–154.
- [35] J. Sun, N.-N. Zheng, and H.-Y. Shum, “Stereo matching using belief propagation,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 25, no. 7, pp. 787–800, 2003.

- [36] S. Birchfield and C. Tomasi, “A pixel dissimilarity measure that is insensitive to image sampling,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 4, pp. 401–406, 1998.
- [37] H. Hirschmuller and D. Scharstein, “Evaluation of stereo matching costs on images with radiometric differences,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 9, pp. 1582–1599, 2009.
- [38] Y. Weiss, “Correctness of local probability propagation in graphical models with loops,” *Neural Computation*, vol. 12, no. 1, pp. 1–41, 2000.
- [39] R. Hegde and N. R. Shanbhag, “A voltage overscaled low-power digital filter IC,” *IEEE Journal of Solid-State Circuits*, vol. 39, no. 2, pp. 388–391, Feb. 2004.
- [40] J. D. Bakos, “High-performance heterogeneous computing with the convey hc-1,” *Computing in Science & Engineering*, vol. 12, no. 6, pp. 80–87, 2010.
- [41] K. K. Parhi, C.-Y. Wang, and A. P. Brown, “Synthesis of control circuits in folded pipelined DSP architectures,” *Solid-State Circuits, IEEE Journal of*, vol. 27, no. 1, pp. 29–43, 1992.
- [42] Xilinx, “Virtex-5 FPGA Xtreme DSP Design Considerations,” Online: <http://www.xilinx.com>, January 2012.
- [43] P. F. Felzenszwalb and D. P. Huttenlocher, “Efficient belief propagation for early vision,” *International journal of computer vision*, vol. 70, no. 1, pp. 41–54, 2006.
- [44] S.-J. Kang, S. I. Cho, S. Yoo, and Y. H. Kim, “Multi-histogram based scene change detection for frame rate up-conversion,” in *Consumer Electronics (ICCE), 2013 IEEE International Conference on*. IEEE, 2013, pp. 332–333.
- [45] K. Alahari, P. Kohli, and P. H. Torr, “Reduce, reuse & recycle: Efficiently solving multi-label MRFs,” in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.
- [46] M. Lam, “Software pipelining: An effective scheduling technique for vliw machines,” in *ACM Sigplan Notices*, vol. 23, no. 7. ACM, 1988, pp. 318–328.
- [47] J. Choi, E. P. Kim, R. A. Rutenbar, and N. R. Shanbhag, “Error resilient MRF message passing architecture for stereo matching,” in *IEEE Workshop on Signal Process. Syst. (SiPS)*, 2013.
- [48] M. Tappen and W. Freeman, “Comparison of graph cuts with belief propagation for stereo, using identical MRF parameters,” in *IEEE Int. Conf. on Comp. Vision*, 2003, pp. 900–906.

- [49] J. Choi and R. A. Rutenbar, “Video-rate stereo matching using Markov random field TRW-S inference on a hybrid CPU+FPGA computing platform,” in *Proc. of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. ACM, 2013, pp. 63–72.
- [50] S. Alchatzidis, A. Sotiras, and N. Paragios, “Efficient parallel message computation for map inference,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1379–1386.
- [51] G. Rong and T.-S. Tan, “Jump flooding in GPU with applications to Voronoi diagram and distance transform,” in *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. ACM, 2006, pp. 109–116.
- [52] M. Bleyer and C. Breiteneder, “Stereo matching—state-of-the-art and research challenges,” in *Advanced Topics in Computer Vision*. Springer, 2013, pp. 143–179.
- [53] L. Wang, M. Liao, M. Gong, R. Yang, and D. Nister, “High-quality real-time stereo using adaptive cost aggregation and dynamic programming,” in *3D Data Processing, Visualization, and Transmission, Third International Symposium on*. IEEE, 2006, pp. 798–805.
- [54] S. K. Gehrig and C. Rabe, “Real-time semi-global matching on the CPU,” in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2010 IEEE Computer Society Conference on*. IEEE, 2010, pp. 85–92.
- [55] I. Ernst and H. Hirschmüller, “Mutual information based semi-global stereo matching on the GPU,” in *Advances in Visual Computing*. Springer, 2008, pp. 228–239.
- [56] N. Y.-C. Chang, T.-H. Tsai, B.-H. Hsu, Y.-C. Chen, and T.-S. Chang, “Algorithm and architecture of disparity estimation with mini-census adaptive support weight,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 20, no. 6, pp. 792–805, 2010.
- [57] H. Hirschmüller, M. Buder, and I. Ernst, “Memory efficient semi-global matching,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 3, pp. 371–376, 2012.
- [58] C. Ttofis, C. Kyrkou, and T. Theocharides, “A hardware-efficient architecture for accurate real-time disparity map estimation,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 2, p. 36, 2015.
- [59] A. T. Ihler, J. W. Fisher III, and A. S. Willsky, “Loopy belief propagation: Convergence and effects of message errors,” *Journal of Machine Learning Research*, pp. 905–936, 2005.