Accelerating Probabilistic Computing with a Stochastic Processing Unit

by

Xiangyu Zhang

Department of Electrical and Computer Engineering
Duke University

Date: _____

Approved:

_____
Alvin R. Lebeck, Advisor

_____
Hai Li

_____
Sayan Mukherjee

_____
Daniel J. Sorin

_____
Lisa Wu Wills

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Electrical and Computer Engineering
in the Graduate School of Duke University
2020

Abstract

Accelerating Probabilistic Computing with a Stochastic Processing Unit

by

Xiangyu Zhang

Department of Electrical and Computer Engineering
Duke University

Date: _____

Approved:

_____
Alvin R. Lebeck, Advisor

_____
Hai Li

_____
Sayan Mukherjee

_____
Daniel J. Sorin

_____
Lisa Wu Wills

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering
in the Graduate School of Duke University
2020

Abstract

Statistical machine learning becomes a more important workload for computing systems than ever before. Probabilistic computing is a popular approach in statistical machine learning, which solves problems by iteratively generating samples from parameterized distributions. As an alternative to Deep Neural Networks, probabilistic computing provides conceptually simple, compositional, and interpretable models. However, probabilistic algorithms are often considered too slow on the conventional processors due to sampling overhead to 1) computing the parameters of a distribution and 2) generating samples from the parameterized distribution. A specialized architecture is needed to address both the above aspects.

In this dissertation, we claim *a specialized architecture is necessary and feasible to efficiently support various probabilistic computing problems in statistical machine learning, while providing high-quality and robust results.*

We start with exploring a probabilistic architecture to accelerate Markov Random Field (MRF) Gibbs Sampling by utilizing the quantum randomness of optical-molecular devices—Resonance Energy Transfer (RET) networks. We provide a macro-scale prototype, the first such system to our knowledge, to experimentally demonstrate the capability of RET devices to parameterize a distribution and run a real application. By doing a quantitative result quality analysis, we further reveal the design issues of an existing RET-based probabilistic computing unit (1st-gen RSU-G) that lead to unsatisfactory result quality in some applications. By explor-

ing the design space, we propose a new RSU-G microarchitecture that empirically achieves the same result quality as 64-bit floating-point software, with the same area and modest power overheads compared with 1st-gen RSU-G. An efficient stochastic probabilistic unit can be fulfilled using RET devices.

The RSU-G provides high-quality true Random Number Generation (RNG). We further explore how quality of an RNG is related to application end-point result quality. Unexpectedly, we discover the target applications do not necessarily require high-quality RNGs—a simple 19-bit Linear-Feedback Shift Register (LFSR) does not degrade end-point result quality in the tested applications. Therefore, we propose a Stochastic Processing Unit (SPU) with a simple pseudo RNG that achieves equivalent function to RSU-G but maintains the benefit of a CMOS digital circuit.

The above results bring up a subsequent question: are we confident to use a probabilistic accelerator with various approximation techniques, even though the end-point result quality ("accuracy") is good in tested benchmarks? We found current methodologies for evaluating correctness of probabilistic accelerators are often incomplete, mostly focusing only on end-point result quality ("accuracy") but omitting other important statistical properties. Therefore, we claim a probabilistic architecture should provide some measure (or guarantee) of statistical robustness. We take a first step toward defining metrics and a methodology for quantitatively evaluating correctness of probabilistic accelerators. We propose three pillars of statistical robustness: 1) sampling quality, 2) convergence diagnostic, and 3) goodness of fit. We apply our framework to a representative MCMC accelerator (SPU) and surface design issues that cannot be exposed using only application end-point result quality. Finally, we demonstrate the benefits of this framework to guide design space exploration in a case study showing that statistical robustness comparable to floating-point software can be achieved with limited precision, avoiding floating-point hardware overheads.

To my beloved parents and wife Zhan Zhang.

Contents

List of Tables

List of Figures

Acknowledgements

Pursuing Ph.D. is a six-year journey to adventure, mostly with excitement, enrichment, and satisfaction, but occasionally with anxiety and disappointment as well. I could not imagine completing this academic expedition without the steady stream of help and encouragement from my advisor, collaborators, instructors, friends, family, and many more others.

To begin with, I would like to express my deepest gratitude to my advisor, Alvin Lebeck, for his tireless guidance and generous support on my research and beyond. His word "think as a researcher" becomes my motto as a Ph.D. student, guiding me to appropriately approach research problems. His visionary thinking helps me clear countless road blockers throughout the way I am chasing advanced academic knowledge. His prompt and considerate mentoring, starting with "Hey Mike" in emails or messages, motivates me to cope with numerous thorny situations beyond research. Most importantly, his passion for research and rigorous attitude toward teaching inspire me to be a responsible person.

I would also like to sincerely thank my committee members Hai Li, Sayan Mukherjee, Daniel Sorin, and Lisa Wu Wills, for their valuable comments and constant help throughout the process. In particular, I am thankful to Sayan Mukherjee for his thoughtful insights as a statistician that make the statistical robustness work possible. I am also grateful for the constructive suggestions Daniel Sorin provided on my paper summaries in Fault-tolerant Systems class when I was in the first year as

a graduate student, the most unforgettable and rewarding class I have ever taken at Duke.

Plentiful thanks to many mentors whose great advice drives me on track. I would like to wholeheartedly thank Chris Dwyer for his great guidance on the macro-scale prototype work. I want to thank Omesh Tickoo, Richard Dorrance, Mahesh Subedar, Srikanth Srinivasan, and other Intel researchers for financially and intellectually supporting my research. I would like to thank Rong Ge, David Page, and Aubrey Barnard for their insightful inputs as domain experts. I sincerely thank Benjamin Lee and Andrew Hilton for serving on my milestone examination committees and providing valuable feedback.

I cannot imagine finishing this dissertation without the countless support from my awesome colleagues and peers. I would like to express my thank to my friend and greatest collaborator Ramin Bashizade. We have collaborated on many works since 2015, none of which would be successful without his supportive efforts. I want to thank Siyang Wang, Craig Laboda, and Yuxuan Li for their hands-on guidance when I was a junior graduate student. I would like to thank Pulkit Misra, John Snyder, Yicheng Wang, Chris Shin, Ceyu Xu, Cheng Lyu, and all other my classmates and friends.

Finally but most importantly, this journey cannot conclude without concrete support from my family. I am deeply grateful to my parents for their unconditional love. A phone call with them is always one of the most joyful moments. I would like to show my greatest thank to my beloved wife, Zhan Zhang, for all of her support, company, tolerance, inspiration, and encouragement. My journey will reach a place called "home" that she and I have created.

1

Introduction

Over the past decades, computing systems have been evolving rapidly with the constant battle between the trend of providing general-purpose processors (standardization) and application-specific accelerators (specialization). Starting from 1947, the cyclical 10-year alternation of standardization and specialization in the semiconductor industry is known as Makimoto's Wave [85]—another influential law beyond Moore's Law. The pendulum is swinging toward specialization in the recent decade due to the end of CMOS scaling and the wave of Artificial Intelligence (AI) and machine learning [50], bringing both enormous challenges and opportunities on computing systems to support large scale applications. In particular, most of the specialized accelerators target deep learning applications (a.k.a., Deep Neural Networks). Examples of tide players include TPU [58], NVDLA [110], DianNao [20], and an NPU [29].

Despite great efforts on Deep Neural Networks accelerators, insufficient attention has been drawn on accelerating other statistical machine learning approaches, especially on probabilistic computing (or probabilistic algorithms). Statistical machine learning often utilizes probabilistic algorithms to solve various high-dimensional prob-

lems, such as computer vision [35], robotics [45], natural language processing [31], global health [44], and wireless communications [46], by iteratively generating samples from parameterized distributions. Compared with other deterministic algorithms which usually require complex mathematical gymnastics, probabilistic algorithms are conceptually simple and enable the potential to provide generalized frameworks to solve wider types of problems, and sometimes are the only viable approach, such as when a problem dimension is high or when a deterministic solution is intractable. As an alternative or complement to Deep Neural Networks, the probabilistic approach provides easier access to interpreting why a given result is obtained [39] through their model transparency and statistical properties. Industry envisions that "probabilistic computing will lead to significant improvements in the reliability, security, serviceability and performance of AI systems" [96].

Unfortunately, probabilistic algorithms can be inefficient on conventional processors. Probabilistic algorithms usually require repeatedly generating samples. For example, Markov Chain Monte Carlo method (MCMC) takes hundreds or thousands of iterations to converge to an acceptable result for complex problems, each of which involves generating many samples in the inner loop [121]. The sampling process includes two steps: 1) computing the parameters of a distribution to sample from (e.g., the decay rate of an exponential distribution, or entries of a discrete distribution); 2) generating samples from the parameterized distribution. Both steps are inner-loop computation, involve complicated arithmetic and transcendental functions, and need to be addressed to efficiently support probabilistic algorithms. A general-purpose CPU takes at least hundreds of cycles to generate a simple distribution (e.g., 167 cycles for an exponential distribution, step-2 only), thus not efficient for sampling. Although a general-purpose GPU has the potential to exploit the native parallelisms in some probabilistic models, its performance and power efficiency remains to be evaluated.

Therefore, we claim the following: *a specialized architecture is necessary and feasible to efficiently support various probabilistic computing problems in statistical machine learning, while providing high-quality and robust results.* To verify this statement, we ask and answer two major questions in four subsequent works:

- Question I: *what is the appropriate architecture of a stochastic processing unit to efficiently support probabilistic computing?* The stochastic nature of probabilistic algorithms requires efficient sample generation from target distributions, whereas current CMOS digital circuits are intrinsically deterministic. The first work explores the feasibility and design space of a stochastic processing unit using a type of optical-molecular device called Resonance Energy Transfer (RET) networks [141]. A RET network has the potential to generate high-quality quantum randomness from various distributions. We find an acceleration unit (the new RSU-G [149]) is feasible and competitive to accelerate 1st-Order Markov Random Field (MRF) Gibbs Sampling, one of the popular Markov Chain Monte Carlo (MCMC) methods. The second work further investigates the necessity of using high-quality Random Number Generation (RNG) by quantitatively evaluating end-point result quality of tested applications. Surprisingly, we discover the RET circuits can be replaced with a simple CMOS RNG without reducing result quality. A CMOS stochastic processing unit (SPU) is available to provide similar efficiency as RSU-G with the benefit of a pure-CMOS digital circuit.

- Question II: *what methodology should we use to evaluate correctness of a probabilistic accelerator?* A probabilistic accelerator often uses approximation techniques (e.g., reduced precision, truncation, and simple RNGs) to maximize efficiency. A methodology is needed to evaluate whether those approximations jeopardize correctness of results. The third work points out the limitations

of using current methodologies that mostly only focus on the application end-point results and omit other important statistical properties for a probabilistic architecture. We propose a three-pillar framework that collectively and comprehensively evaluates statistical robustness of a probabilistic accelerator beyond end-point result quality. The framework surfaces the design issues of the previous SPU that cannot be found only using end-point result quality. Finally, the fourth work shows a case study on using the three-pillar framework to guide robust hardware design.

The following sections summarize each work and its contributions.

1.1 An MRF Gibbs Sampling Unit Using Emerging Technology

The increasing difficulty in leveraging CMOS scaling for improved performance places importance on exploring alternative technologies. A promising technique is to exploit the physical properties of devices to specialize in certain computations. A recently proposed approach uses molecular-scale optical devices (RET networks) [141] to construct a RET-based Sampling Unit (RSU-G) [142] to accelerate sampling from parameterized probability distributions. Sampling is an important component of many algorithms, including statistical machine learning. The previously proposed RSU-G (1st-gen RSU-G) focuses on Gibbs Sampling using MCMC solvers for Markov Random Field (MRF) Bayesian Inference, providing 21-84× speedups as a discrete accelerator over a Titan X GPU [142].

An experimental demonstration is needed to show the operation of a theoretically promising RSU-G. This work first introduces a macro-scale prototype to demonstrate a RET network's ability to parameterize a distribution and to solve real a probabilistic computing problem. The macro-scale prototype is a close-looped system consisting of a PC, an FPGA, laser settings, and single chromophore RET network samples.

The prototype shows: 1) the ability to parameterize pairwise relative probabilities within 10% error when the ratio is below 30, and 24% for higher ratios up to 255; 2) the ability to run a simple foreground-background image segmentation using MRF Gibbs Sampling. Setting up the prototype as a true RNG without post-processing passes 165/188 items in NIST statistical randomness test. The work further explores the relationship between application result quality and RSU-G designs. By quantitatively analyzing the result quality across three computer vision applications (stereo vision with 3 inputs, motion estimation with 3 inputs, and image segmentation with 30 inputs), we find 1st-gen RSU-G [142] lacks both sufficient precision and dynamic range in key design parameters, which limits the overall result quality compared to software-only MCMC implementations. Naively scaling the problematic parameters to increase precision and dynamic range consumes too much area and power. Using our developed RSU-G simulator, we identify four key RSU-G design parameters and explore how each of these parameters influences the result quality. We arrive at a new RSU-G design with four major circuit/microarchitecture changes: 1) improved dynamic range, 2) a new RET circuit and peripheral circuits, 3) supporting multiple energy functions for more applications, and 4) efficient probability conversion. The new RSU-G keeps the same architectural interface as the previous design except for an additional support for simulated annealing and achieves the comparable result quality to 64-bit floating-point (FP64) results. The new design incurs 1.27× power and equivalent area, while maintaining the significant speedups of 1st-gen RSU-G and supporting a wider set of applications.

1.2 A CMOS Stochastic Processing Unit

The new RSU-G is by all means promising except the optical-molecular device requires an additional back end of line process during CMOS fabrication, increasing manufacturing costs. Therefore, we explore the feasibility of a pure CMOS sampling

unit equivalent to RSU-G. One favorable property of RSU-G is the high-quality true randomness from quantum states of RET networks, which in theory guarantees unrepeatable and unpredictable samples. Without RET circuits, the randomness needs to be provided by a CMOS RNG. A deterministic CMOS digital circuit only provides pseudo randomness without an external entropy source. The key question is do we actually need a true RNG for our target applications? If not, what RNGs are enough to provide good result quality?

We evaluate six different RNGs (8-bit, 16-bit, 19-bit Linear-Feedback Shift Register, Mersenne-Twister 19937, and Intel DRNG with pseudo and true randomness) on motion estimation and stereo vision in floating-point precision. Unexpectedly, we discover a simple 19-bit Linear-Feedback Shift Register (LFSR) is enough to provide good application end-point result quality, and using more complicated RNGs does not further improve the results. We observe notable drops in quality if using lower quality RNGs than a 19-bit LFSR. The result indicates designing an efficient CMOS sampling unit equivalent to RSU-G is feasible. Therefore, we propose a CMOS Stochastic Processing Unit (SPU) by replacing the molecular-optical device with a CMOS discrete sampler. The CMOS SPU design provides flexibility to be deployed on an FPGA or fabricated in an ASIC. The SPU optimized for FPGA achieves at least 3× faster in performance and 33.7× less in memory compared with an HLS baseline with FP32 probability, indicating a human-designed architecture is needed to improve efficiency. The ASIC SPU design with a simple 19-bit LFSR avoids area/power overhead of a complex RNG and saves 33% in area and 57% in power, compared with RSU-G. Note that the SPU results do not preclude other potential benefits of true RNG in RSU-G such as unpredictable seeds, which is beyond the scope of this dissertation.

1.3 Statistical Robustness

The above two works indicate our proposed probabilistic architectures can achieve good application end-point result quality even with aggressive hardware approximations. Can we safely conclude the architectures are correct and robust?

Many specialized probabilistic accelerators utilize approximation techniques to improve the hardware efficiency [9, 16, 67, 79, 84, 87, 100], such as reducing bit representation, truncating small values to zero, or simplifying RNGs. Understanding the influence of these approximations on the application results is crucial to provide correct execution of target algorithms. A common approach is comparing the end-point result quality ("accuracy") against accurately-measured or hand-labeled ground-truth data using community-standard benchmarks and metrics: the hardware execution is considered to be correct if it provides comparable "accuracy" to the software-only implementations that do not have these approximations. We use this approach in our first and second works like most other previous architecture works.

However, we further figure out in the domain of probabilistic algorithms correctness is defined by more than the end-point result of executing the algorithm, and includes additional statistical properties that convey uncertainty and interpretability about the end-point results. End-point "accuracy" is necessary but not sufficient to claim correctness. Current methodologies for evaluating probabilistic accelerators are often incomplete or adhoc in evaluating correctness, focusing only on end-point result quality ("accuracy") or limited statistical properties. Failure to adequately account for domain-defined correctness can have adverse or catastrophic outcomes, such as a surgeon failing to completely remove a tumor due to incorrect uncertainty in a segmented image [22, 97]. It is important for hardware designers and domain experts to look beyond end-point "accuracy" and be aware of the impact of archi-

tectural optimizations on other statistical properties.

This work takes a first step toward defining metrics and a methodology for quantitatively evaluating correctness of probabilistic accelerators beyond end-point result quality. We propose three pillars of statistical robustness: 1) *sampling quality*, 2) *convergence diagnostic*, and 3) *goodness of fit*. Each pillar has at least one quantitative empirical metric: Effective Sample Size (ESS) for sampling quality; Gelman-Rubin's $\hat{R}$ and convergence percentage for convergence diagnostic; Root Mean Squared Error (RMSE) and Jensen-Shannon Divergence (JSD) for goodness of fit. These pillars do not require ground-truth data and collectively enable comparison of specialized hardware to 64-bit floating-point (FP64) software. We expose several challenges with naively applying existing popular metrics for our purposes, including: high dimensionality of the target applications, and random variables with zero empirical variance. Therefore, we modify the existing methodologies for sampling quality and convergence diagnostic, and propose a new metric (convergence percentage) for convergence diagnostic.

The three pillars can inform end-users by characterizing existing hardware. As a case study, we demonstrate the framework in a representative probabilistic MCMC accelerator—the SPU proposed in our second work—and show that 1) end-point result quality alone is insufficient, particularly for predicting outcome for previously unseen inputs; 2) FP64 is insufficient as ground-truth since in some cases more limited precision can produce more accurate end-point results based on labeled data; 3) the accelerator achieves the same application end-point result quality as the FP64 software, confirming the previous work, but has compromised ESS and convergence percentage results. The analysis reveals that applications need to run 2× more iterations on the accelerator to achieve the same statistical robustness as FP64, reducing the accelerator's effective speedup by 2×.

1.4 Design Space Exploration with Statistical Robustness

The above work shows that architectural optimizations might have a negative influence on the statistical robustness, even though producing comparable end-point results to FP64 software. Can we achieve desirable end-point result quality and statistical robustness without the commensurate overhead of FP64?

To find an answer, we demonstrate the benefits of using the proposed framework to guide design space exploration on the SPU. The design space exploration exposes the design trade-offs between statistical robustness and area/power with the following key results: 1) a simple 19-bit LFSR with 12-bit RNG outputs does not reduce the statistical robustness or result quality across all design points; 2) considerable improvement in statistical robustness, comparable to the FP64 software, can be achieved by slightly increasing the bit precision from 4 to 6 and removing an approximation technique, with only 1.20× area and 1.10× power overhead. The expected 21-84× speedups can therefore be achieved with no additional iterations.

1.5 Organization of Dissertation

The rest of this dissertation is organized as follows. Chapter 2 introduces motivation and necessary background. Chapter 3 presents the experimental demonstration and architectural exploration on an efficient sampling unit with emerging technology (RSU-G). Chapter 4 evaluates RNGs' influence on the application end-point result quality and presents a CMOS stochastic processing unit (SPU). Chapter 5 provides a methodology for statistical robustness and presents a case study on characterizing the SPU using the proposed methodology. Chapter 6 performs design space exploration using the proposed methodology. Chapter 7 reviews related work. Chapter 8 concludes the dissertation and provides future directions. The main content of this dissertation is also available elsewhere: [142] (Section 7), [149], and [150].

2

Background and Motivation

This chapter introduces background in probabilistic statistical machine learning and our motivation for providing specialized architectures to accelerate probabilistic algorithms. We first introduce basic concepts in probabilistic statistical machine learning, which includes probabilistic algorithms, methods to solve problems by iteratively sampling from distributions, and probabilistic models, methods to represent problems as probability distributions. In this dissertation, we focus on First-Order Markov Random Field (1st-Order MRF) as the Probabilistic Graphical Model and Gibbs Sampling as the probabilistic algorithm. We introduce three computer vision applications that can be solved by 1st-Order MRF Gibbs Sampling: image segmentation, motion estimation, and stereo vision. We use these applications as evaluation benchmarks in the following chapters. By profiling sample generation using C++ STL library and a C++ image segmentation application, we found a specialized architecture needs to accelerate both steps of the sampling process—parameterizing a distribution and drawing samples from the parameterized distribution—to provide notable speedups.

2.1 Probabilistic Statistical Machine Learning

2.1.1 Probabilistic Algorithms

Probabilistic algorithms are used in many statistical machine learning applications for the simplicity and generality [102, 104]. Many of those applications use Bayesian Inference as their framework, which learns information from the combination of prior beliefs and observed data. Suppose θ is the latent random variable of interest in an application. In Bayesian Inference, we are interested in estimating θ given the observed data X. Both θ and X can be scalars or vectors. The posterior distribution $p(\theta|X)$ is given by Bayes' rule in Equation 2.1, where $p(\theta)$ is prior distribution, corresponding to the prior beliefs, and $p(X|\theta)$ is likelihood, corresponding to the observed data. The denominator is considered as a normalization constant to make $p(\theta|X)$ a valid probability.

$$p(\theta|X) = \frac{p(X|\theta)p(\theta)}{\sum_\theta p(X|\theta)p(\theta)} \tag{2.1}$$

In most cases, the posterior distribution is primarily used to evaluate the expectation of a function $f(\theta)$ with respect to a posterior distribution (Equation 2.2), where the integral becomes a summation in discrete cases. Analytically calculating the exact result of this expectation can be intractable or complex.

$$\mathbb{E}[f(\theta)] = \int f(\theta)p(\theta|X)d\theta \tag{2.2}$$

An alternative approach is using sampling method. By drawing multiple samples of θ from $p(\theta|X)$, the expectation can be estimated by Equation 2.3, where the estimation gets close to the true expectation when N is large.

$$\hat{f}(\theta) = \frac{1}{N}\sum_{l=1}^{N} f(\theta^l) \tag{2.3}$$

Several basic sampling methods, such as rejection sampling and importance sampling, work well for low-dimensional problems, but become significantly inefficient when the problem dimensionality grows due to "the curse of dimensionality" [13].

One approach to address this issue is called Markov Chain Monte Carlo (MCMC) method. The basic idea is constructing a Markov Chain with the stationary distribution that is equivalent to the desired distribution and iteratively generating samples from this Markov Chain. A Markov Chain consists a state space $\mathcal{S} = \{s_1, s_2, ..., s_L\}$, where each state s_i corresponds to a parameter value of θ_i in Bayesian Inference. The transition matrix $\boldsymbol{P}$ contains the probabilities of one state moving to another state at any time. The probability vector $\nu = [\nu_1, \nu_2, ..., \nu_L]$ denotes the probabilities of current state is in s_i respectively. If the detailed balance is satisfied, the probability vector converges to a unique stationary distribution v^* such that $\lim_{T\to\infty} \boldsymbol{P}^T \nu = \nu^*, \forall \nu$. Sampling on the Markov Chain is equivalent to sampling on the stationary distribution, regardless of the initial state [103].

One of the most commonly used MCMC methods is Metropolis Hastings algorithm. To draw a sample from the desired distribution, Metropolis Hastings starts from an initial valid state and iteratively moves between the states. In each iteration, it randomly generates a proposed move from the current state. This proposal is randomly accepted by a probability called acceptance ratio, which is determined by the current state and the input data. After adequate iterations, it outputs all or a subset of accepted states as samples, depending on sample quality requirements from different applications.

We can apply a special case of Metropolis Hastings algorithm, called Gibbs Sampling (Algorithm 2.1), when analytical forms of full-conditional distributions are available. Consider in Metropolis Hastings the generated samples are drawn from the joint distribution of $\theta = \{\theta_1, \theta_2, ..., \theta_L\}$. If analytical forms of full-conditional $p(\theta_i|\{\theta_{j\neq i}\}, X)$ are available, we can effectively sample from the joint distribution by

Algorithm 2.1: Gibbs Sampling

Input: data X; initial state $\theta^0 = \{\theta_1^0, \theta_2^0, ..., \theta_L^0\}$
for *each iteration* $t \in \{1, 2, ..., T\}$ **do**
  for *each* i **do**
    Compute full-conditional $p(\theta_i^t | \{\theta_{j \neq i}^{t-1}\}, X)$
    Draw sample $\theta_i^t \sim p(\theta_i^t | \{\theta_{j \neq i}^{t-1}\}, X)$
    Update $\theta_i^{t-1} = \theta_i^t$
  end
  Record θ^t
end
Output: all or a subset of $\{\theta^1, \theta^2, ..., \theta^T\}$

iteratively sampling from the full-conditional distributions using the following steps: 1) set a random initial state θ^0; 2) for each i and iteration t, draw samples θ_i^t from $p(\theta_i^t | \{\theta_{j \neq i}^{t-1}\}, X)$; 3) update the state of θ_i to the latest sample value ($\theta_i^{t-1} = \theta_i^t$); 4) iterate step 2 and 3 until converged or the sufficient number of samples obtained. An output sample θ^t is the combination of samples θ_i^t for all i. Unlike Metropolis Hastings, Gibbs Sampling accepts every proposed sample (i.e., acceptance ratio is always 1). Note that keeping θ_i most updated in step 3 is crucial to maintain Markov property and good sampling quality, although creating strict dependencies between samples.

2.1.2 Probabilistic Graphical Models

Recall probabilistic algorithms solve problems by sampling on the target distributions. The key to applying probabilistic algorithms then is representing problems as probability distributions. This is, however, not usually straightforward especially for problems with high dimensions or complicated structures, such as computer vision [35] and medical record mining [38]. Probabilistic Graphical Models [68] provide a convenient way to represent probability distributions by describing the structural relationship between random variables via graphs. The vertices of a graph represent random variables and the edges represent the relationship between random variables.

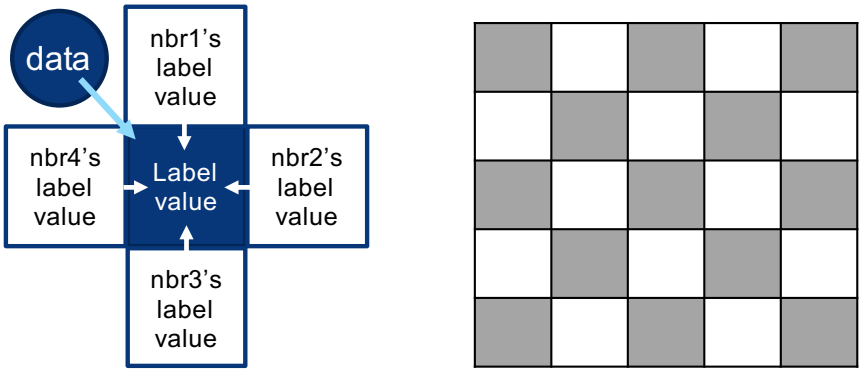


FIGURE 2.1: First-Order Markov Random Field (left) and conditional independence (right).

Vertices are connected by edges if random variables are dependent and disconnected otherwise. The probabilistic graphs can be directed such as Bayesian Networks, or undirected such as Markov Random Fields (a.k.a., Markov Networks).

This dissertation focuses on First-Order Markov Random Field (1st-Order MRF), an undirected probabilistic model that assumes the value of a random variable is only dependent on its four direct neighbors (left, right, up, and down) in a two-dimensional grid and is conditionally independent on all other random variables. Consider each random variable is discrete and randomly picked from several possible label values. Figure 2.1 demonstrates the dependency of a random variable in 1st-Order MRF. The model provides embarrassingly high parallelism since independent labels can be simultaneously evaluated in a checkerboard manner. Grey and white labels can be updated in parallel respectively. Since full-conditional distributions are available, the model can be solved by Gibbs Sampling. Many applications can apply 1st-MRF Gibbs Sampling [36, 106], such as computer vision [35, 72], statistical physics [99], and signal processing [67]. Section 2.2 describes three representative computer vision applications solved by 1st-Order MRF Gibbs Sampling.

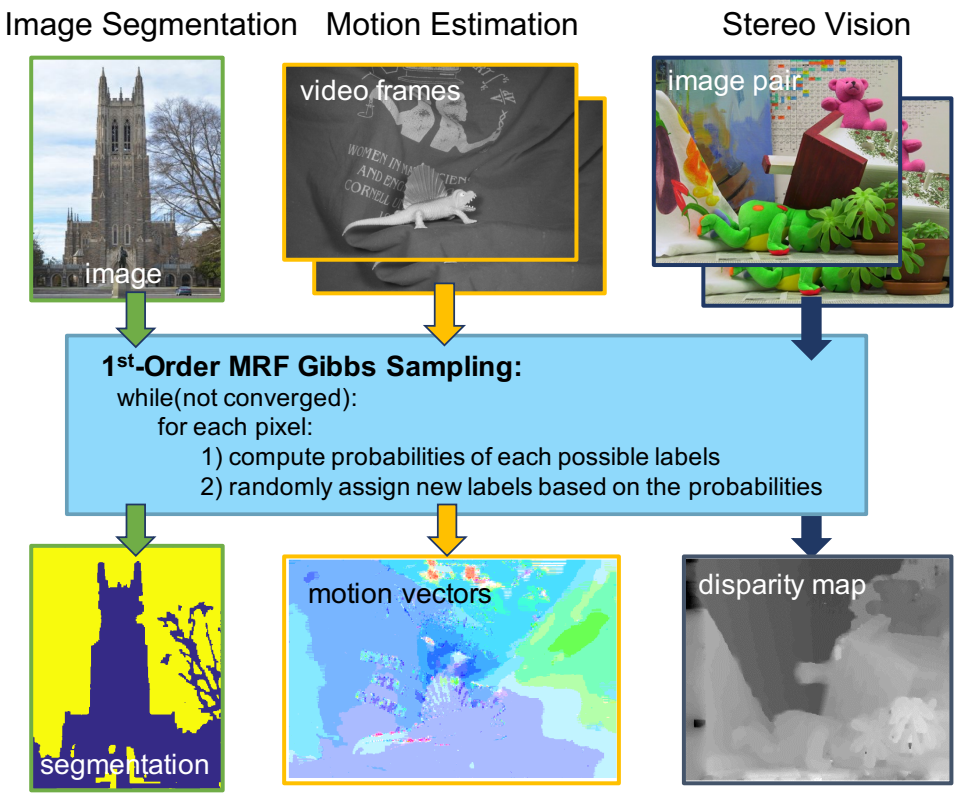


FIGURE 2.2: Computer vision applications (image segmentation, motion estimation, and stereo vision) using 1st-Order MRF Gibbs Sampling

2.2 Representative Applications

This section introduces three computer vision applications that can be solved by 1st-Order MRF Gibbs Sampling: image segmentation [35, 130], motion estimation [69], and stereo vision [11]. We use these applications as evaluation benchmarks in the following chapters. Figure 2.2 shows the high-level steps (1 and 2) to solve these problems. The applications share the same process in step-2, but have different energy functions in step-1 and different temperature annealing schedule if using simulated annealing—a technique to converge to an exact result faster. The label of each application represents different semantics. The result of motion estimation (motion vectors) is color-coded. The rest of this section describes these applications in detail.

Algorithm 2.2: Image segmentation using 1st-Order MRF Gibbs Sampling with simulated annealing

Data: image I, each pixel $I_{x,y} \in \{0, 1, ..., 255\}$
Input: n labels l to segment I; Initial label values L^0; initial temperature T_0; Simulated annealing rate k;
for *each iteration* $t < MAX\_ITERATION$ **do**
  for *each pixel* (x, y) **do**
    for *each possible label* l_i **do**
      Calculate energy: $E(l_i) = \alpha(I_{x,y} - l_i)^2 + \beta \sum_r \delta(L_r^t, l_i)$

$$\delta(L_r^t, l_i) = \begin{cases} 0 & L_r^t = l_i \\ 1 & otherwise \end{cases}$$

      $p_s(l_i) = \exp\left(\frac{-E(l_i)}{T^t}\right)$
    end
    Generate a sample l_s from $\{l_0, l_1, ..., l_{n-1}\}$ following the discrete distribution: $l_s \sim \{p_s(l_0), p_s(l_1), ..., p_s(l_{n-1})\}$
    Update label $L_{x,y}^t = l_s$
  end
  Update temperature: $T^{t+1} = kT^t$
end
Output: final labels $L^{MAX\_ITERATION}$

Image segmentation partitions a single image into multiple segments for further analysis. Each of segment usually has common features such as color, grayscale, or textures. We can solve image segmentation using 1st-Order MRF Gibbs Sampling [35, 130] by considering the segment of each pixel as a random variable. The detailed procedure is provided in Algorithm 2.2. Figure 2.3 shows applying 1st-Order MRF on a two-label image segmentation. The image is partitioned into the foreground (the chapel) and background (the sky). Given the image data, initial label values, and application parameters, the algorithm iteratively processes the image pixel by pixel. For each pixel (x, y), it evaluates the probabilities of picking each possible labels. A possible label l_i represents a segment in grayscale that the current pixel possibly belongs to (e.g., foreground or background). As shown in Figure 2.3, the

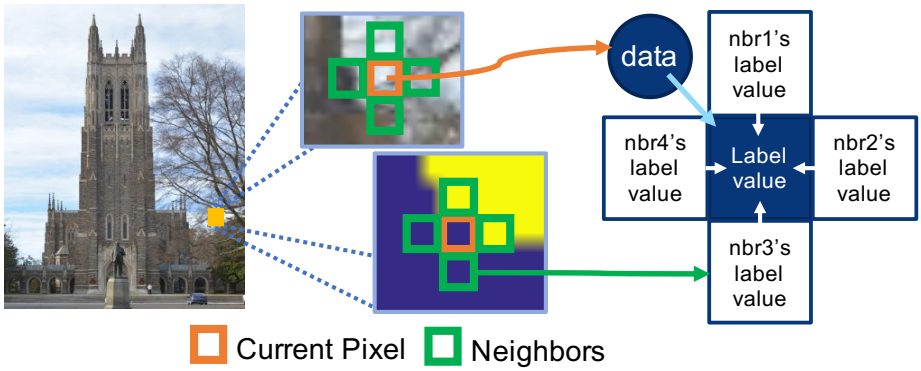


FIGURE 2.3: Image segmentation on 1st-Order MRF

label value of current pixel only depends on the data (singleton)—the image grayscale of current pixel $I_{x,y}$—and the label values L_r^t of its four direct neighbors (doubleton): $r \in \{(x-1,y),(x,y-1),(x+1,y),(x,y+1)\}$. A standard-form full-conditional energy function is then obtained by adding singleton, the squared difference between pixel grayscale and the possible label, to doubleton, the sum of binary differences between the possible label and neighbor labels. α and β are the weights to decide whether the model pays more attention to singleton (image grayscales) or doubleton (smoothness among neighbors), learned during a training process. The energy $E(l_i)$ is then scaled by a temperature T and exponentiated to a scaled probability $p_s(l_i)$. The above process corresponds to step-1 of the high-level sampling procedure. Once the probabilities of all possible labels are obtained, step-2 draws a sample from the computed discrete distribution and updates the label value of current pixel to the assigned sample. The outer loop applies simulated annealing (if available) at the end of each iteration and continues to proceed the image until convergence or a pre-defined maximum iteration is reached.

Note that the original mathematical representation of energy is written as Equation 2.4 [130] where $\alpha = 1/(2\sigma^2)$. We can omit the constant term $\ln(\sqrt{2\pi\sigma^2})$ in actual energy computation since it cancels out in the later sampling process. It also applies

to motion estimation and stereo vision.

$$E(l_i) = \ln(\sqrt{2\pi\sigma^2}) + \frac{(I_{x,y} - l_i)^2}{2\sigma^2} + \beta \sum_r \delta(L_r^t, l_i) \tag{2.4}$$

We can optionally apply simulated annealing to converge to an exact result faster by strategically decreasing temperature T [35]. T is initially high so that every label has a similar probability of being selected. As T decreases, labels with the lowest energy are likely to be selected, eventually leading to convergence. Choosing the appropriate annealing schedule is critical and determined case-by-case [106]. The final labels after the last iteration are directly the output results. Alternatively, we can use a constant T throughout all iterations, where T is considered as a parameter obtained during model training. This pure-sampling method consistently generates Gibbs samples from the targeted distribution after a burn-in period. In the end, we can estimate the entire distribution of a pixel picking possible labels by collecting the latest N samples. An exact result can be obtained from the mode of the estimated distribution, the most frequent label. Overall, simulated annealing usually converges faster but pure-sampling provides an estimated distribution, containing important information on uncertainty and diagnostic. Our work in chapters 3 and 4 uses simulated annealing only in evaluation, and uses both simulated annealing and pure-sampling methods in chapters 5 and 6.

We use a subset of Berkeley Segmentation Database (BSD300) [90] as the evaluation dataset. We evaluate the result quality using BISIP [146], a widely-used evaluation package that provides four quantitative quality metrics: 1) Probabilistic Rand Index (PRI); 2) Variation of Information (VoI); 3) Global Consistency Error (GCE); 4) Boundary Displacement Error (BDE). Details of these 4 metrics can be found elsewhere [146].

Algorithm 2.3: Motion estimation using 1st-Order MRF Gibbs Sampling with simulated annealing

Data: Motion preceding frame I^- and following frame I^+, each pixel $I_{x,y} \in \{0, 1, ..., 255\}$.

Input: n labels $l_i = (\Delta x, \Delta y)$ to represent the displacement vector for each pixel from I^- to I^+; initial label values L^0; initial temperature T_0; Simulated annealing rate k;

for *each iteration* $t < MAX\_ITERATION$ **do**

  for *each pixel* (x, y) **do**

    for *each possible label* l_i **do**

      Calculate energy: $E(l_i) = \alpha(I^+_{x,y} - I^-_{l_i})^2 + \beta \sum_r ||L^t_r - l_i||^2$

      $p_s(l_i) = \exp\left(\frac{-E(l_i)}{T^t}\right)$

    end

    Generate a sample l_s from $\{l_0, l_1, ..., l_{n-1}\}$ following the discrete distribution: $l_s \sim \{p_s(l_0), p_s(l_1), ..., p_s(l_{n-1})\}$

    Update label $L^t_{x,y} = l_s$

  end

  Update temperature: $T^{t+1} = kT^t$

end

Output: final labels $L^{MAX\_ITERATION}$

2.2.2 Motion Estimation

Motion estimation (optical flow) estimates how objects move between two frames in a video. The data input is usually two consecutive frames of images: preceding frame I^- and following frame I^+. The output result is a map of 2D displacement vectors, indicating how pixels move from a preceding frame to a following frame.

Algorithm 2.3 solves motion estimation using 1st-Order MRF Gibbs Sampling. The high-level procedure is similar to image segmentation but different in input data (a pair of frames instead of a single image), label representation, and energy function [69]. Figure 2.4 demonstrates how to apply 1st-Order MRF on motion input frames. In the inner loop of Gibbs Sampling, the algorithm evaluates every possible pixel location within a search window $r \times c$ in the following frame to which the current pixel in the preceding frame could move. The red pixel presents the current pixel and yellow pixels represent possible locations in Figure 2.4. A possible label

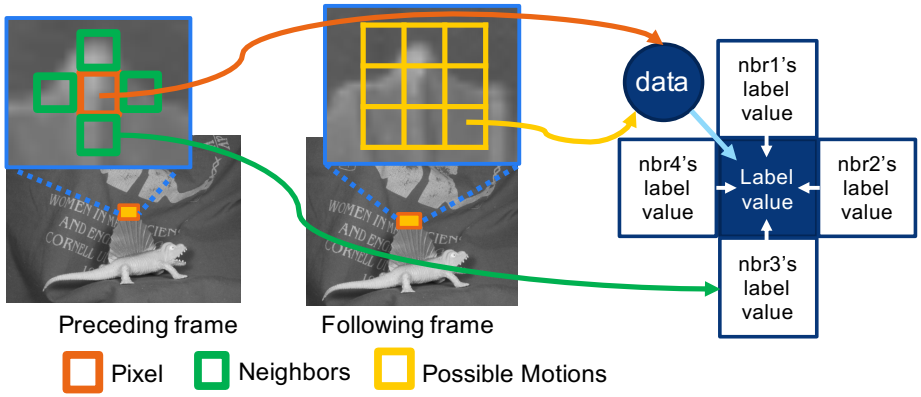


FIGURE 2.4: Motion estimation on 1st-Order MRF

$l_i = (\Delta x, \Delta y)$ is a vector of signed integers that a pixel could move relative to its current location. The label value of current pixel depends on the data values of both frames and the label values of four direct neighbors. The energy function accumulates singleton, the squared grayscale difference between the current pixel and the possible destination pixel, and doubleton, the sum of squared distance between the possible label and neighbor labels. Similar to image segmentation, we draw a sample from the computed distribution and update to the latest label. The simulated annealing schedule is the same as image segmentation with a different k value. The outer loop processes the frames until convergence or the maximum iteration is reached.

We use a subset of Middlebury motion estimation benchmarks [7] as our evaluation dataset and a common end-point error [7] as our result quality metric.

2.2.3 Stereo Vision

Stereo vision reconstructs the depth information of objects in an image pair by matching the corresponding pixels that represent the same objects. The input is a pair of images, left image I^L and right image I^R, taken by a stereo camera. The output is a disparity map, indicating the depth of the corresponding objects in the image—a larger disparity indicates a closer object.

Algorithm 2.4 solves stereo vision by 1st-Order MRF Gibbs Sampling following a

Algorithm 2.4: Stereo vision using 1st-Order MRF Gibbs Sampling with simulated annealing

Data: Stereo left image I^L and right image I^R, each pixel $I_{x,y} \in \{0, 1, ..., 255\}$.

Input: n labels $l_i = \Delta x$ to represent distance between a pixel in I^L and its corresponding pixel on I^R; initial label values L^0; initial temperature T_0; Simulated annealing rate k;

for *each iteration* $t < MAX\_ITERATION$ **do**

  for *each pixel* (x, y) **do**

    for *each possible label* l_i **do**

      Calculate energy: $E(l_i) = \alpha|I^L_{x,y} - I^R_{x-l_i,y}| + \beta \sum_r |L^t_r - l_i|$

      $p_s(l_i) = \exp\left(\frac{-E(l_i)}{T^t}\right)$

    end

    Generate a sample l_s from $\{l_0, l_1, ..., l_{n-1}\}$ following the discrete distribution: $l_s \sim \{p_s(l_0), p_s(l_1), ..., p_s(l_{n-1})\}$

    Update label $L^t_{x,y} = l_s$

  end

  Update temperature: $k = k + \Delta k$, $T^{t+1} = T^0/k$

end

Output: final labels $L^{MAX\_ITERATION}$

similar procedure to image segmentation and motion estimation, with different label representation, energy function, and annealing schedule. Figure 2.5 shows using 1st-Order MRF on stereo vision. The inner-loop computation evaluates probabilities of each possible matching from the current pixel in the left image (red) to the candidate pixels in the right image (yellow). Each label l_i corresponds to a possible disparity of the pixel location from one image to the other. Theoretically, disparities are 2D-vectors that represent the relative horizontal and vertical location of corresponding pixels. In practice, algorithms typically use image pairs that are calibrated in a standard form such that correspondence of pixels is constrained to only the horizontal line and the disparity reduces to scalars. The label value of a pixel depends on data value of the image pair—the greyscale differences between the current and the possible matching pixel—and label values of four direct neighbors. Multiple types of energy functions are available for solving stereo vision using different algorithms

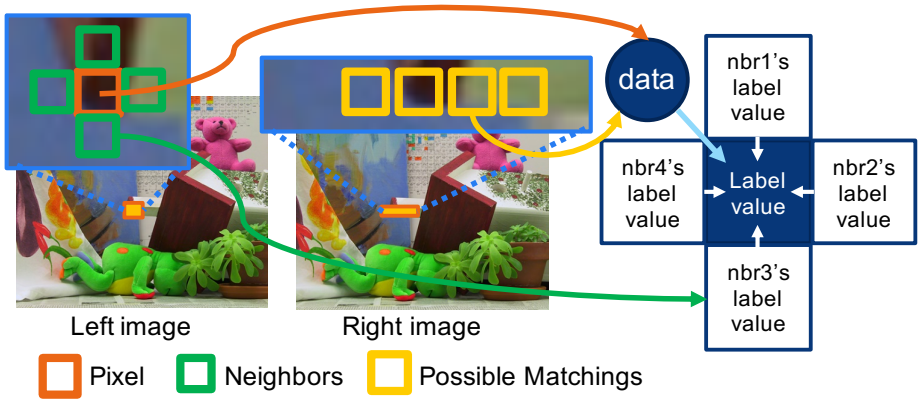


FIGURE 2.5: Stereo vision on 1st-Order MRF

[60, 125]. We experimentally find that absolute difference works best in our case, shown in Algorithm 2.4. We replace temperature annealing schedule to $T^{t+1} = T^0/k$, where k is additive to a constant Δk at the end of each iterations ($k = k + \Delta k$).

We use Middlebury Stereo [125], a commonly used stereo benchmark in the computer vision community as our test dataset. We use the common bad-pixel percentage (BP) and root-mean squared (RMS) error as quality evaluation metrics, and set the bad-pixel threshold to 1, as in the previous work [125]. This means a pixel is considered mislabeled if the distance between calculated disparity and ground-truth disparity is larger than 1. More detailed evaluations can distinguish the disparity map for subregions such as *occluded* and *textureless*; however, for simplicity, we provide overall result quality where all subregions are included.

2.3 Sampling Overhead

The sampling process in probabilistic algorithms includes two steps: 1) parameterizing a distribution (e.g., calculating the probabilities of each possible outcomes in a discrete distribution); 2) drawing samples from the parameterized distribution. The first step requires mapping application data values to probability parameters, which is application and model dependent. The second step usually requires con-

Table 2.1: Average cycles per sample for different distributions from 10^9 samples per run

Distributions	Exponential	Normal	Discrete	Gamma	Binomial	Poisson
Parameters	$\mu = 3.5$	$\mu = 5$,$\sigma = 2$	K=10	$\alpha = 2$,$\beta = 2$	$t = 4$,$p = 0.5$	$\mu = 4$
Cycles/sample	167	121	465	277	523	209

verting uniform random numbers generated by a Random Number Generator (RNG) to targeted distributions using conversion algorithms such as inverse transformation or rejection sampling [13]. An overview of computational techniques for drawing samples is provided elsewhere [27]. Both steps are in the inner loop of probabilistic algorithms, and thus critical to the overall performance.

To understand the sampling overhead, we collect the average cycles per sample for 7 different distributions on Intel E5-2640 processor using C++11 STL library. The experiment configuration is the same as the previous work [142] except the average cycles are collected from 10^9 samples per run to amortize the overheads of performance counter and allow more optimized execution in CPU. Table 2.1 shows the distribution parameter configurations and average cycles per sample. Similar to the previous work [142], generating a sample from a simple distribution takes hundreds of CPU cycles. The profiling results in Intel VTune suggest that in most distributions, 80% of the total CPU cycles are spent on converting uniform random numbers to the targeted distribution. On the other hand, merely accelerating drawing samples is not enough to provide substantial speedups for applications. Preliminary profiling results on an image segmentation application indicate that drawing samples takes around 50% of total execution time. The exponential calculation, one step in parameterizing distributions, takes 30% execution time. These preliminary results lead to a motivation to accelerate both steps in sampling.

2.4 Summary

This chapter introduces probabilistic statistical machine learning, popular probabilistic algorithms and models, three computer vision applications solved by 1st-Order MRF Gibbs Sampling, and a preliminary study on CPU sampling overhead. We found both steps of the sampling process contribute to sampling overhead and need to be accelerated. The following chapters provide hardware solutions to address sampling overhead and a methodology to comprehensively evaluate the correctness of those probabilistic hardware solutions.

3

An MRF Gibbs Sampling Unit using Emerging Technology

The impending halt in CMOS scaling places increasing importance on finding alternative approaches to improve computational efficiency. Architectural innovation, such as specialization, remains critical to overcoming the challenges faced today. Equally important is the need to explore new device technology that can augment CMOS specialization.

Recent work investigates emerging technologies that enable specialization by exploiting physical device properties. Memristors [76], strain-switched magneto-tunneling junctions [64], and fluorescent molecules [105,142] have the potential to accelerate certain machine learning algorithms, such as Deep Neural Networks (DNNs) and Bayesian Inference. Wang et al. [142] recently proposed RSU-G—a molecular optical Gibbs sampler that exploits Resonance Energy Transfer (RET) to efficiently sample from parameterized probability distributions [141, 142]. An RSU-G is a hybrid CMOS-RET functional unit for accelerating Markov Random Field Bayesian Inference problems. It operates by first computing an energy value that is used to

obtain a decay rate for an exponential distribution for each possible value a random variable may take on. Samples from the commensurate exponential distributions are used to probabilistically choose a value for the random variable.

This chapter demonstrates the feasibility and design trade-offs of RSU-G. We first develop a macro-scale prototype to experimentally demonstrate the operation of theoretically promising RSU-G and its ability to solve probabilistic algorithms. The macro-scale prototype is a close-looped system consisting of a PC, an FPGA, laser settings, and single chromophore RET network samples. The prototype shows: 1) the ability to parameterize pairwise relative probabilities within 10% error when the ratio is below 30, and 24% for higher ratios up to 255; and 2) the ability to run a simple foreground-background image segmentation using 1st-Order MRF Gibbs Sampling. Setting up the prototype as a true Random Number Generator (true RNG) without post-processing, we found the prototype with RET networks passes 165/188 items in NIST statistical randomness test compared with 19/188 when substituting RET networks with silica scattering, confirming RET network's ability to generate relatively high-quality randomness.

An important criterion for machine learning accelerators, CMOS and non-CMOS, is the relationship between precision and result quality. We further ask, and answer, several questions about application result quality and the impact of precision on the integrated RSU-G design.

Question 1: What is the impact of precision on result quality? We use standard benchmarking methods to compare results for three applications with widely-used data sets: 1) stereo vision with 3 inputs, 2) motion estimation with 3 inputs, and 3) image segmentation with 30 inputs. Our results show that the previously proposed RSU-G fails to match software-only result quality. We identify four key RSU-G design parameters that require a specific precision or value: 1) energy computation, 2)

exponential decay rate, 3) time measurement, and 4) distribution truncation. Using a functional simulator of an RSU-G, we explore how each of these four design parameters affects quality.

Our results show that overall result quality in the previous RSU-G design degrades due to lack of both precision and dynamic range on certain parameters. Thus, we introduce 1) decay rate scaling and 2) probability cut-off to maximize the dynamic range at the early and late optimization iterations respectively, which significantly improve the overall result quality. Surprisingly, we found that high result quality can be preserved using very few unique decay rates while preserving precision and dynamic range. We also expose a relationship between distribution truncation—treating all samples beyond a specified threshold as infinity—and time measurement precision that can be exploited to improve the RSU-G design.

Question 2: What, if any, changes must be made to RSU-G microarchitecture to support the desired precision and achieve result quality comparable to software-only implementations? Based on the required design parameters, we find that using light source intensity to control exponential decay rates in the previous RSU-G design is not an effective technique since area/power scale with the required dynamic range. Thus, we introduce a new RSU-G design that exploits an alternative design parameter, molecular concentration [141], along with other techniques to achieve the desired precision (and result quality) while keeping the same area and increasing power by only 1.27×, relative to the previous RSU-G design. We also propose a design where multiple RSU-Gs share optical resources, which further reduces power/area overheads and allows the use of conventional light sources.

Question 3: Do changes in RSU-G microarchitecture require any architectural changes? Radical changes in RSU-G design may impose commensurate changes at the ar-

chitectural level since extensive delays or modified interfaces may require software changes. Fortunately, the RSU-G proposed in this chapter is mostly a microarchitectural change and can be used in place of the previously proposed unit with minimal architectural modifications. This preserves the sizable performance improvements demonstrated previously [142].

The remainder of this chapter is organized as follows. Section 3.1 provides an overview of the recently proposed RSU-G and the underlying technology. Section 3.2 present a macro-scale RSU-G prototype. We perform an extensive analysis of result quality and the impact of RSU-G design parameters in Section 3.3. Alternative RSU-G designs are presented and evaluated in Section 3.4. Section 3.5 elucidates limitations and future work. Section 3.6 summarizes this chapter.

3.1 Background

3.1.1 Enabling Technology

Previous work provides a theoretical foundation for constructing physical samplers based on molecular-scale Resonance Energy Transfer (RET) networks [141]. RET is the probabilistic transfer of energy between two optically active molecules, called chromophores, through non-radiative dipole-dipole coupling [136]. When a donor and acceptor chromophore pair are placed a few nanometers apart and their emission and excitation spectra overlap, energy transfer can occur between them. A RET network is constructed by placing multiple chromophores in a physical geometry where chromophores interact through RET. A fully specified RET network can be conveniently and economically fabricated with sub-nanometer precision using hierarchical DNA self-assembly [70, 119].

RET networks are integrated with an on-chip light source, e.g., quantum-dot LEDs (QDLEDs), waveguide, and single photon avalanche detector (SPAD) to create a RET circuit, shown in Figure 3.1. Each RET circuit can contain an ensemble

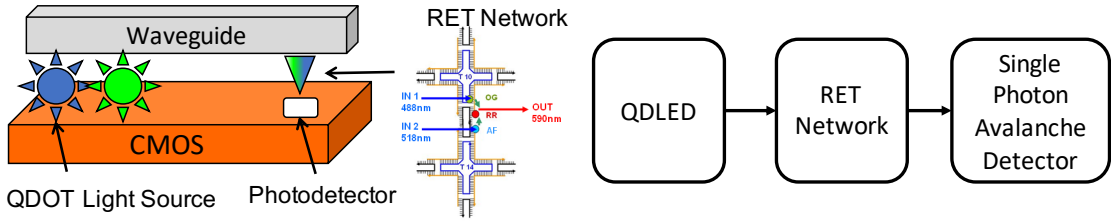


FIGURE 3.1: Structure of RET Circuit [141]

of RET networks. RET circuits can then be integrated with hybrid electro-optical CMOS using spin coating [18], polymer doping [42], or various other back-end-of-line processing techniques [81, 82]. SPAD arrays fabrication is demonstrated elsewhere [30, 108]. Dark count rate of SPADs (∼KHz [5, 86, 113]) has negligible effects given RSU-G frequency (1GHz).

3.1.2 RET-Based Sampling Units

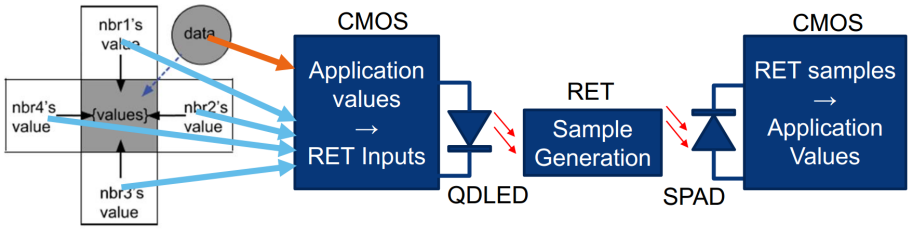
Recent work [142] presents RET-based samplers for 1st-Order MRF Gibbs Sampling that utilize exponential samplers. Probabilistic functional units—called RET Sampling Units for Gibbs Sampling (RSU-G)—are constructed using CMOS specialization to accelerate distribution parameterization and RET networks to accelerate obtaining a sample from the parameterized distribution, the whole inner loop of 1st-Order Markov Random Field Gibbs Sampling (recall Figure 2.2).

RSU-G utilizes the *first-to-fire* design based on the property of competing exponential random variables [141]. RET circuits generate samples from exponential distributions ($\lambda e^{-\lambda t}$) parameterized by the decay rate (λ) and record the time to fluorescence (TTF) for each exponential sample. The decay rate is determined for each possible label of a random variable, which depends on the neighboring random variables' current labels and the singleton energy. The label that produces the shortest TTF is chosen as the result label. *First-to-fire* equivalently draws samples from discrete distributions $\{\lambda_1, \lambda_2, ..., \theta_n\}$, where the decay rate λ_i corresponds to a

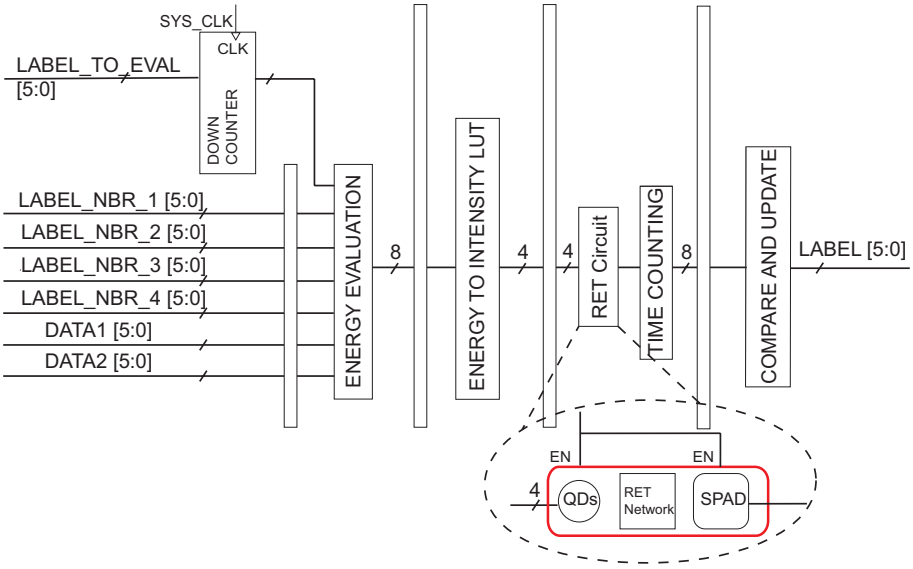
scaled probability of picking ith random variable. The decay rate λ is physically determined by various parameters $\lambda = NP_eP_{fl}P_d\lambda_0$ when using single chromophores as one-node RET networks. N is the number of chromophore particles in the unit area (concentration). P_e is the probability of a single chromophore being excited by photons, proportional to the QDLED emission intensity. P_{fl} refers to the probability that an excited chromophore fluoresces a photon. P_d refers to the probability that a fluoresced photon is detected by the SPAD. λ_0 is the intrinsic decay rate determined by chromophore types. Therefore, the decay rate λ can be tuned by changing the concentration of RET networks, the QDLED emission intensity, the specific chromophore, or a combination of these. The previously proposed RSU-G uses the QDLED intensity to change λ for each label evaluated.

RSU-G inputs and outputs are integers that correspond to values of interest depending on the application (e.g., image segment labels, pixel motion vectors, etc.). A block diagram of an RSU-G is shown in Figure 3.2a. An RSU-G performs a series of three operations: 1) map application values to RET inputs, 2) generate samples, 3) map RET output to application value. Steps 1 and 3 are implemented using conventional CMOS specialization, whereas step 2 is a RET circuit that exploits the probabilistic behavior of RET networks.

Figure 3.2b shows how the three abstract steps map to 5 stages in a pipeline that evaluate one possible label (out of M) per cycle. These stages correspond to 1) label decrement/input, 2) energy computation, 3) energy to intensity mapping (exponential decay rate), 4) sampling (using RET circuits), and 5) selection. Label decrement is used to iterate over all M possible 6-bit labels. The energy computation obtains an 8-bit value, which maps through a look-up-table (LUT) to a 4-bit exponential decay rate (QDLED intensity). The RET circuit samples the exponential distribution by measuring the time it takes to observe an output photon after illuminating the RET network with the appropriate QDLED intensity. The measured time for the sample,



(a) RSU-G block diagram.



(b) RSU-G pipeline.

FIGURE 3.2: RET-based Gibbs Sampling Unit (RSU-G) [142]

represented by a 5-bit integer, is used in the selection block to choose the lowest from all M possible labels. The total latency is 7+(M-1) for M possible labels since stage 3 (RET sampling) requires 4 cycles. Replicated RET circuits are used to avoid structural hazards caused by this multicycle stage. The four replicated RET circuits expand the window of time for observing samples (due to the tail of the exponential) such that 99.6% of the samples are covered; this truncates the last 0.4% of samples for the lowest decay rate (this number is less than 0.4% for higher decay rates) and assumes they occur at infinity (i.e., no sample is generated).

When added to a GPU, these units achieve 3× and 16× speedups for image segmentation (5 labels) and motion estimation (49 labels), respectively for HD images.

A discrete accelerator with 336 RSU-G units achieves 21× and 54× speedups assuming a 336GB/s memory bandwidth limitation for HD images and up to 84× for 320×320 images in motion estimation. Each unit consumes low power and occupies a very small area. Using $15nm$ CMOS, a single RSU-G (CMOS+RET) consumes 3.91 mW, occupies 0.0029 mm^2, and generates entropy at 2.89Gb/s. Compared to 6.4Gb/s Intel DRNG [98], RSU-G only consumes 13% of the power in similar area while also providing programmability to parameterize distributions.

The previous results are encouraging for accelerating probabilistic algorithms. However, important works remain: 1) a working example of a RET-based sampling unit is needed to demonstrate proof-of-concept; 2) the relationship between application result quality and RSU-G design are to be explored. The following sections address these two aspects.

3.2 A Macro-scale RSU-G Prototype

Although the theoretical proposal of RSU-G is promising, a complete demonstration on the integrated circuit is ideal but difficult. We instead propose a macro-scale RSU-G prototype to demonstrate the ability to parameterize a distribution and run a real application. This section presents the prototype setup and experimental results.

3.2.1 Prototype Setup

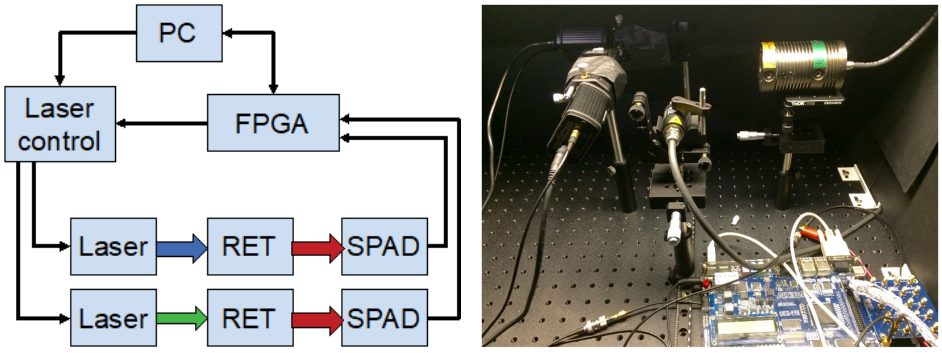
The prototype follows the basic design of a generic RSU-G in Figure 3.2a, with parameterization and RET circuits for sampling the distribution, but is different from the exact RSU-G microarchitecture in Figure 3.2b due to the scale difference and experimental limitation. Figure 3.3 shows the system diagram and the real experimental setup. The un-pipelined prototype consists: a PC to parameterize distributions and run the outer loop of Gibbs Sampling algorithm, an FPGA to trigger laser pulses and implement an approximation of *first-to-fire*, a pair of intensity

programmable laser settings as light sources, two cuvettes of single chromophores as RET networks, and a pair of SPADs as photodetectors.

The prototype contains two channels of RET circuits and thus can evaluate two possible labels simultaneously. A straightforward *first-to-fire* design for sampling from a two-entry discrete distribution $\{\lambda_1, \lambda_2\}$ is to simultaneously illuminate two RET circuits and record which channel detects the RET excited photon first. Experimentally, our FPGA implementation achieves sufficient precision to resolve 250ps differences in photon arrival times. However, *first-to-fire* is able to parameterize decay rate λ only if both channels detect at least one photon per laser pulse, that is, "collection efficiency" $\eta = NP_eP_{fl}P_d > 1$. Our prototype system reaches at most $\eta = 0.179$ limited by the experimental environment such as the optical settings, maximum laser power, and re-absorption between chromophore particles. The RET networks in both channels always present their intrinsic decay rates λ_0 in this case, regardless of intensity settings. Our experiment verifies this hypothesis, where the probability of choosing either label is always 50%. Therefore, we instead use Bernoulli process to approximate "first-to-fire". Given $\eta < 1$, we can also interpret η as the probability of detecting a photon given a laser pulse. The sampling process can be described as a sequence of Bernoulli trials: 1) we emit a light pulse in both channels at $t = 0$; 2) during a sufficient time window (1us in our case), if only one channel detects a photon, we choose this channel as the winner; otherwise, we redo the sample until one channel wins. Suppose two channels have collection efficiency η_1 and η_2 where $\eta_1, \eta_2 \ll 1$, we can represent the probability of channel 1 fires first as:

$$P(\text{channel 1 fires first}) = \frac{\eta_1}{1-\eta_1} \cdot \frac{1}{\eta_1 + \eta_2 - \eta_1\eta_2} \approx \frac{\eta_1}{\eta_1 + \eta_2} \tag{3.1}$$

The value of η can also be parameterized by light intensity and thus we can approximate a two-entry discrete distribution $\{\lambda_1, \lambda_2\}$ by $\{\eta_1, \eta_2\}$.



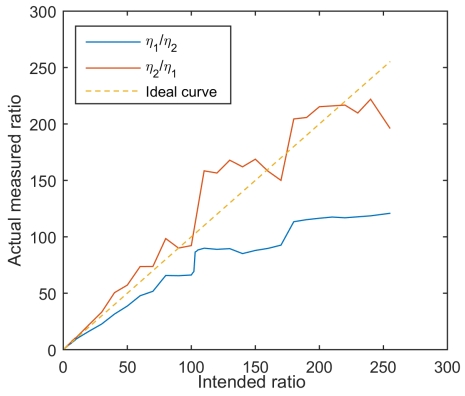
(a) Functional block diagram (b) Photo of prototype

FIGURE 3.3: Macro-scale RSU-G prototype

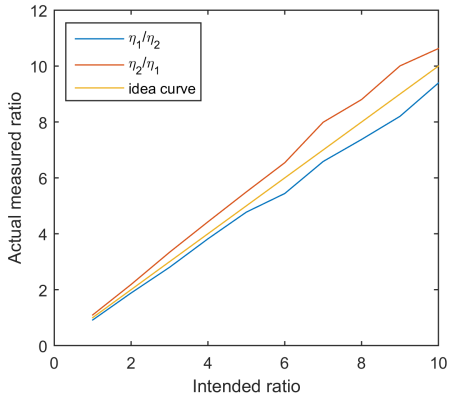
A complete sampling process using prototype can be described as follows: 1) a PC software parameterizes the distribution and sets laser intensities via laser control; 2) once intensity setting is done, the FPGA triggers both lasers to emit a light pulse in both channels; FPGA circuit is open to detect photons; 3) if either of SPADs exclusively detects a photon in a time limit, FPGA circuit records the winning channel and sends back to the PC; otherwise, the FPGA re-triggers the lasers until a channel exclusively detects a photon; 4) the PC updates the label. The operation time to generate a sample is dominated by setting intensities via proprietary laser control (∼85ms). Communication between FPGA and PC can therefore use simple RS232 without notably additional delays.

3.2.2 Experimental Results

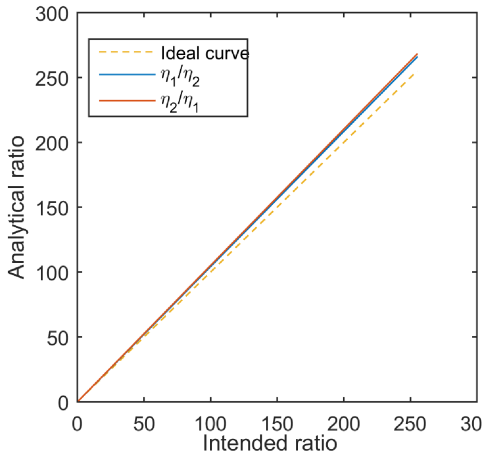
Our first experiment demonstrates the ability to parameterize a two-entry discrete distribution $\{P_1, P_2\}$ by changing the ratio of collection efficiency η_1/η_2 (i.e., changing light pulse intensity). We characterize the curve of laser intensity vs. collection efficiency η for accurate control. η is calibrated on FPGA by collecting the total times of success (i.e., a photon detected with a time limit given a light pulse) in 2^{24} Bernoulli trails. The laser controller (PicoQuant PDL 828) has 1000 intensity steps between 0 to the maximum power. Figure 3.4a shows our intended ratios η_1/η_2 (blue)



(a) Intended vs. actual ratios of relative probabilities from 1 to 255



(b) Intended vs. actual ratios of relative probabilities from 1 to 10



(c) Intended vs. analytical ratios with approximation

FIGURE 3.4: Results on parameterizing pairwise relative probabilities

of pairwise relative probabilities from 1 to 255 vs. the actual measured ratios P_1/P_2. The results are collected from 2^{16} samples. We switch the ratio settings η_2/η_1 (red) to test the symmetry of the prototype system. The prototype can achieve desired pairwise relative probabilities (η_2/η_1, red) within 10% errors when ratio is below 30, partly shown in Figure 3.4b. For higher ratios, η_2/η_1 achieves desired ratios within 24% errors. Figure 3.4c shows the errors introduced by the Bernoulli process approximation which slightly overestimates intended ratios but with negligible errors.

High errors at large ratios and the asymmetry in the system are partially caused by some time-relevant properties of RET networks, such as evaporation and photobleaching [136]. These problems may be solved or alleviated during the fabrication process [18,42,82]. A finer characterization and control of the prototype components could further improve the accuracy.

We also run an image segmentation problem with only two possible labels (e.g., foreground and background) on the prototype. We use the same procedure as Algorithm 2.2 (Chapter 2) but a different simulated annealing schedule $T = \log(100/(k + \Delta k))$ with initial $k = 100$. The PC executes the outer loop of 1st-Order MRF Gibbs Sampling, energy computation, and intensity mapping. The FPGA and RET circuits sample from the output label distribution. An image is initialized by simply binarizing pixels to the closest labels. Figure 3.5 shows the segmentation results of a 50×67 image by software algorithm vs. the RET-based prototype. The initial labels (0 iterations) include noise pixels in the bottom of the image. The RET functional unit successfully reduces the noise in the bottom and segments the image into the foreground and background after 10 iterations. Asymmetry on the two boundaries of the "chapel" is presumably explained by asymmetry on the prototype system shown in Figure 3.4a, which may be improved by a finer system.

Our next experiment evaluates the capability of using RSU-G prototype as a true Random Number Generator (RNG) without post-processing. As the simplest application, the prototype can be treated as a uniform RNG if $\eta_1 = \eta_2$. Since the randomness comes from the quantum states of RET networks [59,136], the prototype has the capability as a high-quality true RNG. We use NIST statistical test [12], a popular statistical test on the quality of randomness. The test suite reads a sequence of random bit inputs and comprehensively detects a variety of possible non-randomness sources from the inputs via 15 types (188 items) of tests. A good RNG for cryptography should pass all items of tests. A commercial RNG usually has complex

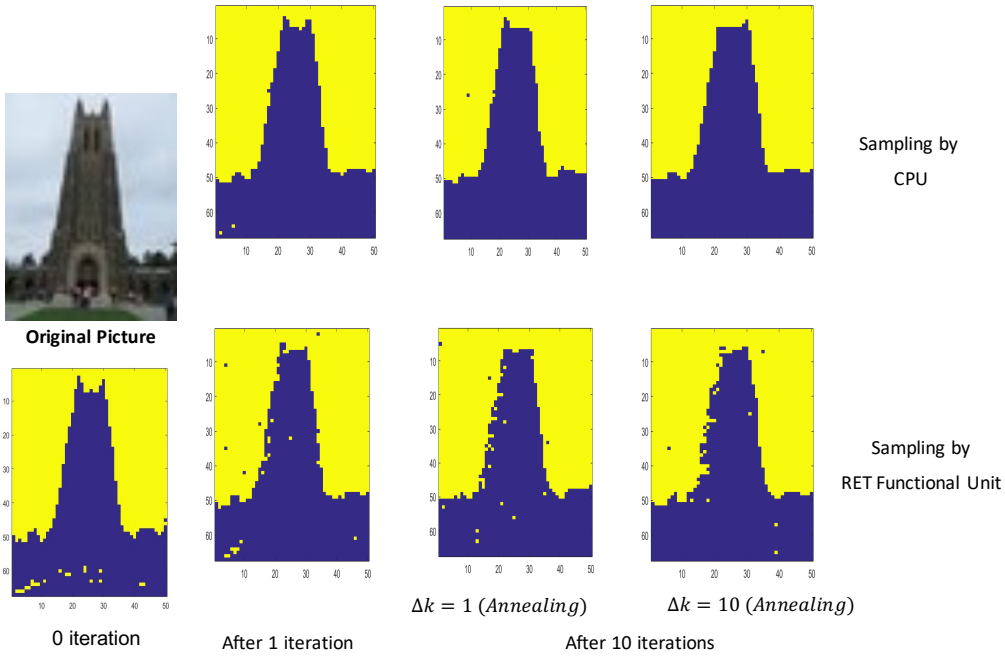


FIGURE 3.5: Prototype image segmentation results

post-processing steps to provide a high quality of randomness [129]. Note that the prototype does not have any post-processing other than recording the winning labels (i.e., 0 or 1 random bits). Randomness directly comes from the entropy source—the RET networks.

We wonder how many tests the prototype can pass without post-processing. We send 10 sequences of 10^6 random bits for the test, each with the proportion of '1' between 0.499 to 0.501 by precisely controlling laser intensity. We follow the test suite guide to set up input parameters, shown in Table 3.1. According to the test suite guide, for all items except random excursions, at least 8 out of 10 sequences passing the test leads to a pass in that particular item or fail otherwise. A pass or fail in random excursions is decided by the test suite based on input random numbers.

We replace the RET networks with silica scattering samples as a comparison group. Silica scattering per se reflects photons emitted by a laser source to a SPAD without a stochastic process. However, the SPAD still exists a probability to detect a

Table 3.1: NIST statistical test input parameters

Parameters	**Values**
Block Frequency Test - block length	128
NonOverlapping Template Test - block length	10
Overlapping Template Test - block length	10
Approximate Entropy Test - block length	14
Serial Test - block length	17
Linear Complexity Test - block length	500

photon from the laser given a light pulse and a time limit. Therefore, the prototype with silica scattering replacement can be also considered as an RNG. Comparing randomness results with silica scattering helps us extract the randomness directly from the RET networks.

Table 3.2 shows the NIST test results. The prototype with RET networks passes 165 items out of 188 without post-processing, compared with 19 out of 188 if replacing the RET networks with silica scattering. The results indicate the randomness mostly comes from the RET network. Some non-randomness and bias from optical settings, the FPGA, or data communication wires potentially reduce the number of passes and could be alleviated or completely removed by post-processing. Nevertheless, using RET networks can generate relatively high-quality random numbers without post-processing. A commercial integrated circuit true RNG—Intel DRNG [98]—passes all tests, but have complex post-processing steps.

In summary, we propose a macro-scale RSU-G prototype, the first such system to our knowledge, that demonstrates the capability of a RET network to parameterize a distribution and run a real application. The NIST statistical test confirms the RET network produces relatively high-quality randomness without post-processing. The following sections explore the quality/precision relationship in the integrated RSU-G microarchitecture.

Table 3.2: NIST statistical test result on prototype as a true RNG without post-processing

NIST Statistical Test	**RET Network**	**Silica Scattering**
1. Frequency	Pass	Pass
1. Block Frequency	Pass	Pass
2. Cumulative Sums (2 items)	Pass 2/2	Pass 2/2
4. Runs	Fail	Fail
3. Longest Runs of Ones	Pass	Fail
5. Rank	Pass	Fail
6. Spectral	Pass	Fail
7. Non-overlapping Template (148 items)	Pass 128/148	Pass 11/148
8. Overlapping Template Matchings	Pass	Fail
9. Universal Statistical	Pass	Fail
12. Approximate Entropy	Fail	Fail
10. Random Excursions (8 items)	Pass 8/8	Fail
11. Random Excursion Variant (18 items)	Pass 18/18	Pass 3/18
13. Serial (2 items)	Pass 1/2	Fail
15. Linear Complexity	Pass	Pass
Total Passes	**165/188**	**19/188**

3.3 RSU-G Precision vs. Quality

Recall RSU-G accelerates statistical machine learning problems using Markov Chain Monte Carlo (MCMC) for MRF Bayesian Inference, specifically 1st-Order MRF Gibbs Sampling. Many factors influence result quality for statistical machine learning algorithms. For the problems we study in this chapter, a domain expert develops a model that includes clique definitions, number of labels, conditional probabilities, etc. This model clearly has a profound effect on the final results, but refining specific models is beyond the scope of this work. In this chapter we focus on the impact of RSU-G circuit and microarchitectural decisions on end-point result quality for given models. Chapters 5 and 6 discuss the impact of these decisions on statistical robustness beyond end-point result quality.

3.3.1 Methodology

Our overall goal is to ensure that RET-based samplers achieve results comparable to software-only implementations that use commodity processors or GPUs with IEEE floating point, which theoretically generate the highest result quality. Recall that the software-only method incurs high computation overhead to generate samples from parameterized distributions. RSU-G accelerates this computation, but it must do so without consuming excessive area or power.

We develop a functional simulator of an RSU-G in MATLAB that enables us to explore the various aspects of RSU-G design with respect to result quality. This allows running a baseline software version that uses the appropriate routines in MATLAB to obtain samples and update labels in MCMC solvers for MRF problems. The RSU-G functional simulator replaces the appropriate code sections with the RSU-G equivalent functionality. The functional simulator uses MATLAB default Mersenne-Twister RNG (*mt19937ar*) [93].

We evaluate the result quality using three applications: 1) image segmentation, 2) motion estimation (optical flow), 3) stereo vision, which are good representations of computer vision, and can all be solved using MCMC with an MRF model. We performed the same analysis for all three applications; for brevity, we use stereo vision as a running example to illustrate the impact of precision on quality, since this application has the highest bit precision requirements based on our experiments. We use standard metrics for evaluating quality that are specific to each application (e.g., end point error for motion estimation, and PRI, etc. for image segmentation). Exploring result quality for applications in other areas is part of our ongoing work.

The previous RSU-G supports only squared distance energy function, which fits well with motion estimation [69]. However, as described in Section 2.2, the other two applications require different distance functions: binary distance for image seg-

mentation [130], and absolute distance for stereo vision [60, 125]. Supporting these distance functions requires modest changes to the energy calculation stage, incurring additional area and power, but not influencing precision. Our new design adds support for all three distance functions. For our quality vs. precision analysis, we add this same support to the previously proposed RSU-G [142].

Stereo vision reconstructs the depth information of objects taken from two cameras by matching the corresponding pixels between two images. Previous work demonstrated various MCMC sampling techniques for stereo vision [11, 21, 125], and we use the version based on a 1st-Order MRF model [11], which the RSU-G directly supports. Recall 1st-Order MRF Gibbs Sampling stereo vision iterates pixel by pixel and samples from possible labels. Each label corresponds to a possible disparity of the pixel location from one image to the other. Disparity values directly reflect the depth information of objects in the images: foreground objects have larger disparities than background objects. We use Middlebury Stereo [125] as our test dataset. Since the previous RSU-G design can support up to 64 labels, we randomly selected three datasets *teddy* (56 labels), *poster* (30 labels), and *art* (28 labels) that all meet this constraint. We use bad-pixel percentage (BP) with the threshold to 1 and root-mean squared (RMS) error as quality evaluation metrics. A pixel is considered mislabeled if the distance of calculated disparity vs. ground-truth is larger than 1.

Similar to previous work [11, 14], we use simulated annealing for stereo vision to converge to a stable result. Recall simulated annealing divides the energy by a decreasing temperature after each iteration so that every label has a similar probability to be chosen at the beginning, but gradually labels with lower energy are more likely to be chosen, until finally converging to the optimal results. In the RSU-G, simulated annealing can be implemented by updating the energy-to-intensity LUT in Figure 3.2b at the end of each iteration. Note that this method provides equivalent functionality to software-only simulated annealing, but causes stalls in the RSU-G pipeline.

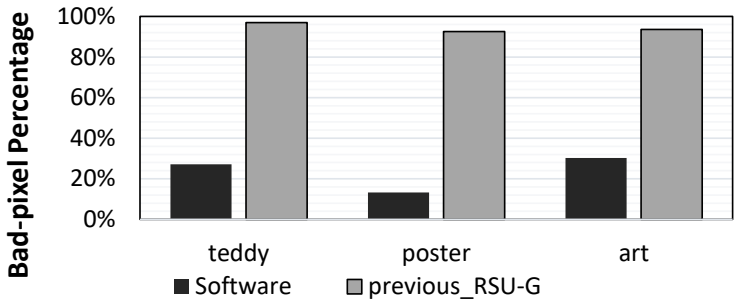


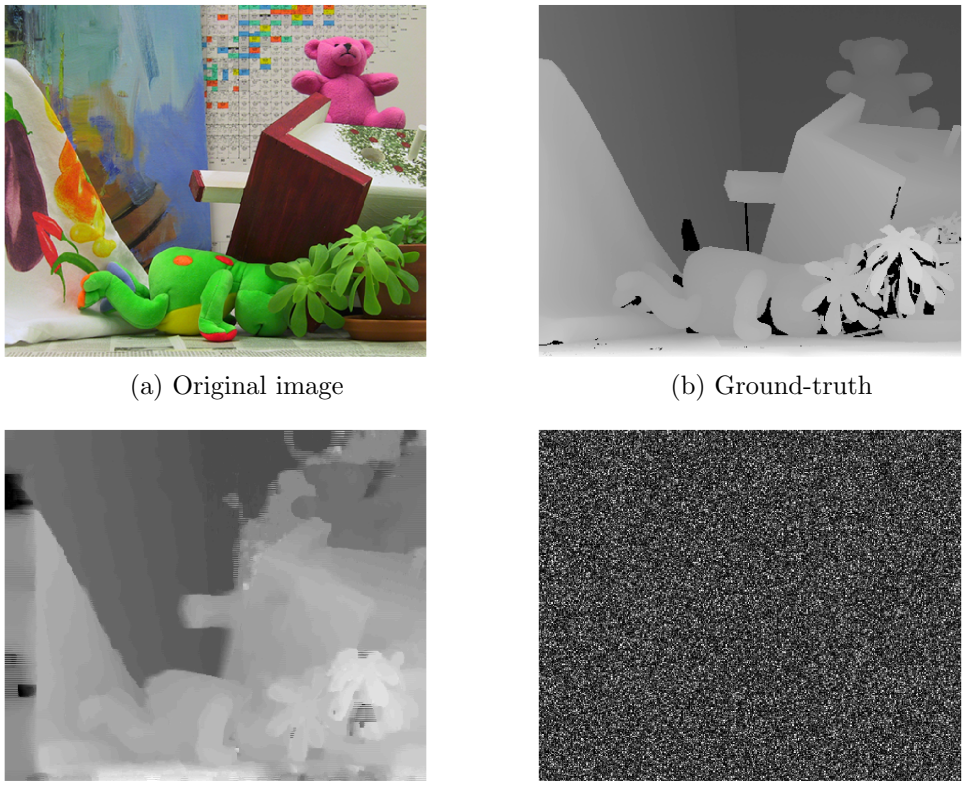
FIGURE 3.6: Software-only vs. previous RSU-G result quality

Our new RSU-G design eliminates these stalls with a new energy-to-intensity conversion scheme described in Section 3.4. We tune simulated annealing parameters (e.g., *temperature* and *annealing rate*) such that it can provide result quality with acceptable simulation time. We run *teddy* and *art* with 1,000 iterations and *poster* with 500 iterations. All three datasets converge.

3.3.2 RSU-G vs. Software-only Quality

We begin by first comparing the previously proposed RSU-G as defined by Wang et al. [142] to the software-only implementation. We compare the results using both BP and RMS error. Since they are consistent, we only show BP in Figure 3.6. Based on *teddy* results [125], the only dataset of the three that has both the public-accessible ground-truth and evaluation scores, MCMC software-only (BP 27%) can reach very close to the quality of Graph Cuts algorithms [14] (BP 25%). Better MCMC-software results can be obtained by fine-tuning the application parameters and running more iterations, which is beyond the scope of this chapter (although compared to the software implementation, RSU-G acceleration allows executing more iterations in the same amount of time). We perform a best-effort optimization for MCMC algorithm parameters (e.g., *energy weights*) and apply them throughout the evaluation.

Figure 3.7a and 3.7b show the left image of the original image pair and the ground truth disparity, respectively. Disparity is color coded in the gray-level scale:



(a) Original image (b) Ground-truth

(c) Software disparity map (d) RSU-G disparity map

FIGURE 3.7: Software-only vs. previous RSU-G stereo images

light pixels represent high disparity values, indicating they are closer to the camera. Darker pixels represent low disparity values and farther from the camera, except for the black area, which means no correspondence between the image pair due to occlusion. We conservatively consider all software and RSU-G results in those areas as mislabeled pixels, making our result quality pessimistic. Clearly, the previous RSU-G does not perform well with the three datasets, producing BP >90%, mislabeling nearly all pixels. Figure 3.7c and 3.7d show the disparity maps generated by the software only algorithm and the RSU-G. The striking difference between those two disparity maps indicates the result quality of RSU-G is unacceptable.

3.3.3 RSU-G Design Parameters and Quality

Given the unacceptable results for the previous design, the next step is to determine which RSU-G design parameters cause the result quality loss. We identify three primary design parameters where bit precision may influence overall results: 1) energy computation ($Energy_{bits}$), 2) exponential decay rate ($Lambda_{bits}$), and 3) time measurement ($Time_{bits}$). These three parameters correspond directly to components in the equations from the MRF model and how it relates to RET-based sampling [141]. A fourth parameter of interest is distribution truncation—the fraction of samples in the exponential tail rounded up to infinity.

$$E = \alpha E_{singleton} + \beta \sum E_{neighborhood} \tag{3.2}$$

$$\lambda = e^{-E/T} \tag{3.3}$$

$$p(t) = \lambda e^{-\lambda t} \tag{3.4}$$

For the RSU-G pipeline shown in Figure 3.2b, the first stage of the pipeline computes the total energy of a label based on Equation 3.2. α and β are application parameters. $Energy_{bits}$ refers to the precision of E, the output of this stage. The second pipeline stage obtains the decay rate by implementing Equation 3.3 in a LUT. Temperature T can be folded into the calculation of LUT entries. $Lambda_{bits}$ refers to the number of bits used to represent λ, which also indicates the maximum unique λ values the RSU-G can support. In stage three, the RET circuit samples from an exponential distribution parameterized by λ, shown in Equation 3.4, to obtain the time to fluorescence (TTF). Theoretically, TTF has no upper-bound, but to ensure forward progress, RSU-G has a maximum TTF it can detect and rounds up to infinity for any TTF beyond this bound. $Time_{bits}$ determines the finest time resolution within RSU-G's detection window.

An RSU-G that uses full IEEE floating point is theoretically possible; however, it is impractical due to area, power, and timing limitations. Therefore, any RSU-G design must rely on limited integer precision. *The key architectural challenge is to determine the specific value required for each RSU-G design parameter to achieve acceptable result quality while minimizing area and power.*

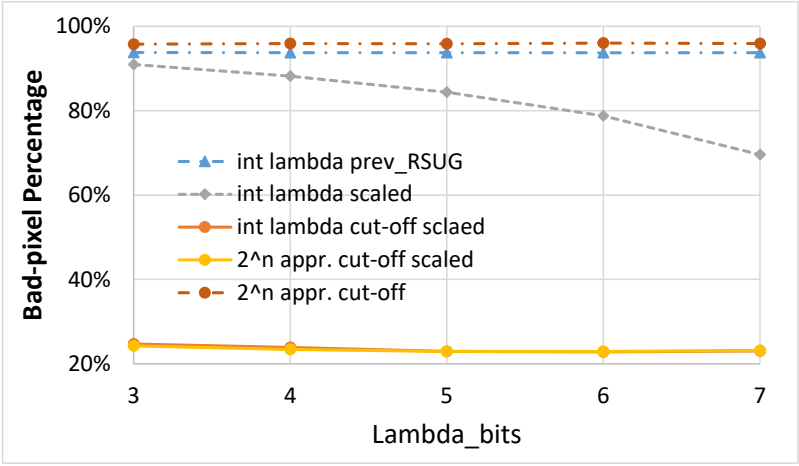
To answer this question, we use a sequential evaluation approach that finds the best limited precision for only the earliest RSU-G pipeline stage, energy computation ($Energy_{bits}$), while allowing the remaining parameters ($Lambda_{bits}$ and $Time_{bits}$) to use IEEE FP precision. Based on this result, we fix the $Energy_{bits}$ to the best value for all remaining experiments. Next, we vary the exponential decay ($Lambda_{bits}$) to find the lowest precision while keeping the time measurement as floating point. Finally, we fix the energy and exponential decay rate while exploring timing precision.

Energy Computation

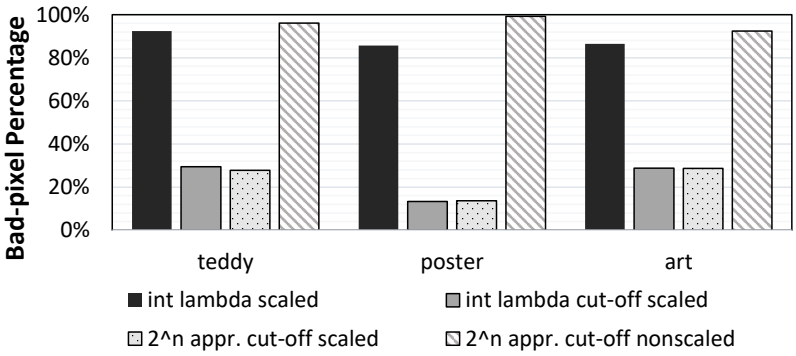
Our experiments confirm previous work [87] that shows 8-bit energy is sufficient for stochastic algorithms. Fewer than 8 bits significantly degrades result quality. BP for the three data sets on RSU-G (8-bits) and software (floats) are 27.0% vs. 27.1%, 12.6% vs. 13.3%, and 27.3% vs. 30.3% for *teddy*, *poster*, and *art*, respectively.

Exponential Decay Rate

Given $Energy_{bits} = 8$ we explore the next RSU-G design parameter, $Lambda_{bits}$. Recall, the exponential decay rate is used to create relative probabilities for possible label values. That is, the ratio of probabilities for any two possible labels is equal to the ratio of the corresponding exponential decay rates ($P(M_i)/P(M_j) = \lambda_i/\lambda_j$). RSU-G obtains the decay rate using a LUT indexed by the energy value for a given possible label, and this value is used to control a set of QDLEDs that illuminate the RET network. The area and power scale with the number of unique decay rates



(a) Results for configurations in exponential decay rate



(b) Results for $Lambda_{bits} = 4$

FIGURE 3.8: Result quality vs. exponential decay rate precision

since it requires either more QDLEDs or introducing DACs. Naively scaling the design with $Lambda_{bits} = 7$ requires 128 unique decay rates, expanding the RET circuit area by $8\times$ to $12,800um^2$. Therefore, it is desirable to minimize the number of decay rates, and thus $Lambda_{bits}$.

The line *int lambda prev_RSUG* in Figure 3.8a shows the average BP results across three stereo vision data sets for RSU-G when varying $Lambda_{bits}$ from 3 to 7. We observe BP is above 90% even with $Lambda_{bits} = 7$, which means we can not achieve our quality goal by naively increasing $Lambda_{bits}$. Further analysis indicates that high result quality requires a larger dynamic range for λ and a way to reduce the

accumulated error caused by limited precision. We meet these needs by introducing decay rate scaling[1] and probability cut-off, as described below.

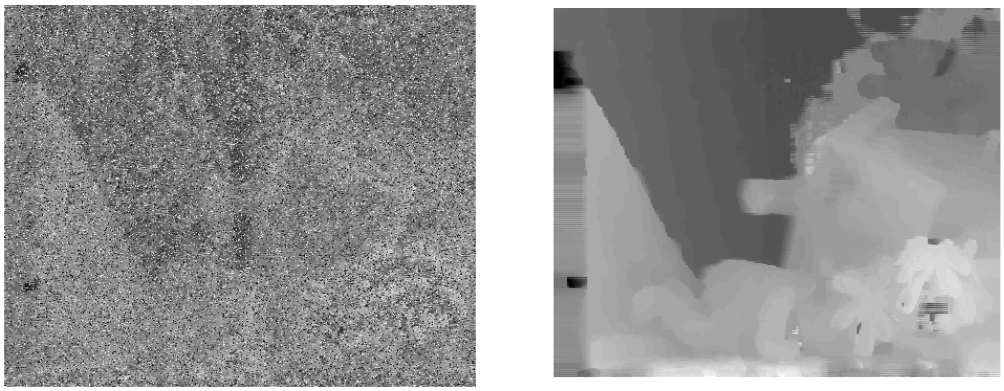
Decay Rate Scaling Recall the key property in utilizing RET for *first-to-fire* is the ratio of decay rates λ_i/λ_j for the competing exponential distributions. Theoretically the absolute value of λ_i and λ_j does not matter. However, given limited integer precision, small λ_i and λ_j will be rounded to the same value even when their ratio λ_i/λ_j is high (e.g., 0.4 and 0.005 are both rounded to 0, whereas their ratio is 80). This problem becomes crucial in the early part of simulated annealing when temperature, T, is high. Thus, we want λ as large as possible to minimize the precision loss. Given the decay rate for each possible label ($\lambda_i;\ i = 0\ to\ M-1$ see Equation 3.3), we scale all decay rates λ_i by a factor k, such that $\max_i \lambda_i$ is equal to the maximum λ we can support in RSU-G. From Equation 3.3, we get maximum λ when $E = 0$. Thus, we can maximize the dynamic range of λ_i while maintaining the invariant of constant λ_i/λ_j ratios shown in Equation 3.5.

$$\frac{\lambda_i}{\lambda_j} = \frac{k\lambda_i}{k\lambda_j} = \frac{e^{-(E_i - E_{min})}}{e^{-(E_j - E_{min})}}. \tag{3.5}$$

It also indicates that we can convert λ multiplication to energy subtraction, thereby simplifying the implementation. This produces a new set of decay rates $\lambda'_i = k\lambda_i$ which has a dynamic range from 1 to the maximum supported λ. Line *int lambda scaled* in Figure 3.8a indicates that with decay rate scaling, BP decreases with increasing $Lambda_{bits}$ and reaches about 70% when $Lambda_{bits} = 7$, however it still does not meet the quality requirement (see Figure 3.9a).

Probability Cut-off The previous RSU-G design limits the minimum probability to λ_0 ($\lambda_{min} = \lambda_0$), which corresponds to the minimum unique decay rate in a RET

[1] The idea of decay rate scaling is proposed by Ramin Bashizade.



(a) 7 lambda bits, decay rate scaling only

(b) 4 Lambda bits, with cut-off, decay rate scaling and 2^n truncation.

FIGURE 3.9: Stereo vision *teddy*: scaled decay rates and probability cut-off

circuit. Lower probabilities are rounded up to λ_0. Although this design keeps all labels active during the entire execution, it introduces noise during the later iterations due to limited precision in $Lambda_{bits}$, especially for applications with many labels. Consider a given pixel in a late iteration of *teddy* with 56 possible labels. Label 0 has a 0.98 probability to be chosen based on Equation 3.3 while the other 55 labels have a combined 0.02 probability to be chosen. With $Lambda_{bits} = 7$ in RSU-G, label 0 is mapped to the maximum supported $\lambda = 128\lambda_0$, while each of the other labels is mapped to the minimum λ_0. These minimum λ_0s introduce inaccuracy due to rounding since now the probability to choose label 0 becomes $128/(128 + 55) = 0.7$, leaving 0.3 probability to choose other labels, and thus preventing convergence.

To address this problem, we use a probability cut-off (approximation) policy when the calculated probability is small enough to ignore. The threshold is implicitly applied during LUT value generation. $E = 0$ is mapped to the largest λ. Other values for each energy entry are calculated by Equation 3.3, multiplied by a scale $2^{Lambda_{bits}}$, and truncated to the nearest integer. The probability is ignored (set to 0) when the calculated value is less than one, which means this probability is not large enough to use λ_0. Using a probability cut off dramatically improves result quality, as

shown in Figure 3.8a. $Lambda_{bits} = 3, 4$ produce average BPs of 24.7% and 23.8%, respectively. $Lambda_{bits} = 4$ enables a practical RSU-G implementation in terms of area and power.

Truncating λs to the nearest 2^n integer value reduces the unique number of λs needed from $2^{Lambda_{bits}}$ to $Lambda_{bits}$, without reducing quality (see Figure 3.8a), and thus can reduce area and power. Probability cut-off must be incorporated with decay rate scaling, otherwise all probabilities are cut off in the early annealing iterations, leading to poor result quality, as shown in Figure 3.8a. Figure 3.8b shows BP results across all three datasets, indicating that RSU-G can achieve quality comparable to the software-only implementation using the above techniques, with BP of 27.8% for *teddy*, 13.7% for *poster*, and 28.6% for *art*. Figure 3.9b shows the disparity map of *teddy*.

Timing Precision and Distribution Truncation

The last component we analyze is timing precision. Recall in *first-to-fire*, a sample is generated by parameterizing the decay rate λ of an exponential distribution and choosing the label (corresponding to a λ_i) that has the shortest TTF among all possible labels [141]. The key is to differentiate the TTF for each label and choose the label with the shortest TTF as the new value for the random variable.

Timing precision addresses two aspects: 1) timing resolution and 2) maximum detection time. Theoretically higher timing resolution achieves better results. However, transistor physics, such as gate and wire delays, limits the fastest detection speed and an ultra-high clock frequency would consume unacceptable power. Furthermore, TTF for exponential distribution has no upper bound and waiting too long to cover a high percentage of TTF can cause a structural hazard in the pipeline for TTF longer than one clock cycle. The previously proposed RSU-G uses 4 RET circuit replicas to avoid this hazard, but does not scale well when the maximum de-

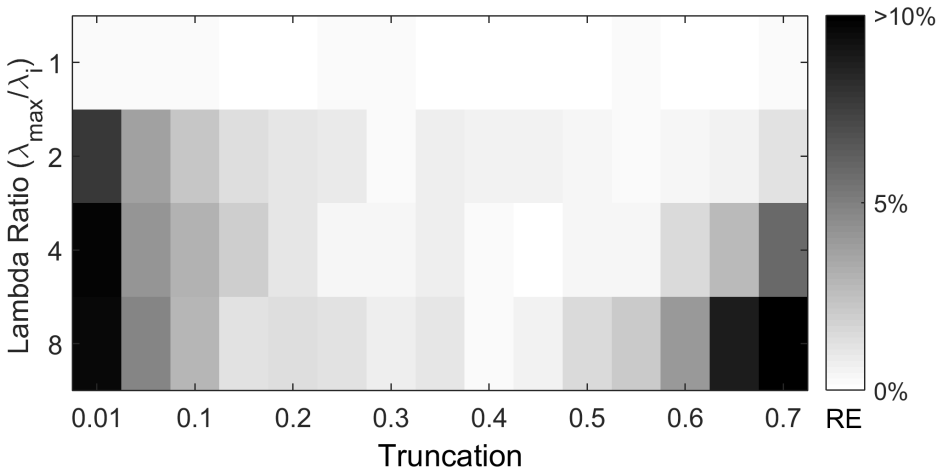


FIGURE 3.10: Relative error (RE) between actual probability ratios and intended lambda ratios under different $Truncation$s, given $Time_{bits} = 5$

tection time is high. Mitigating this requires a deeper analysis of timing precision's impact on result quality.

To analyze the time precision, we define the finest timing resolution of RSU-G as a unit time bin (duration 1 unit). TTF within a time bin cannot be differentiated. In the simulation, we randomly pick a label if two labels are tied in TTF, which in the RSU-G architecture, is one bit of additional random input from any source of RNG. We treat the sampling stage of RSU-G as an exponential sampler that, given a positive input decay rate λ, generates output at time $t; 1 \leqslant t \leqslant t_{max}$ following an exponential distribution. $Time_{bits}$ is the number of integer bits used to represent t. Since TTF is theoretically infinite, we define a new parameter $Truncation$ to provide a detection boundary from a probability perspective. Distribution $Truncation$ refers to the probability that TTF is longer than t_{max} given λ_0, the lowest decay rate RSU-G can support ($Truncation = p(TTF > t_{max}|\lambda_0) = exp(-\lambda_0 t_{max})$). TTF beyond t_{max} is numerically rounded to t_{max}.

$Time_{bits}$ influences result quality since fewer $Time_{bits}$ indicates two detected TTF are more likely to be in the same bin—a tie. However, simulation results indicate

that given a fixed $Time_{bits}$, $Truncation$ also significantly influences result quality. Recall, in the ideal case, the ratio of probabilities for choosing any two labels is proportional to the ratio of their calculated decay rates = λ_i/λ_j. Figure 3.10 shows the relative error between actual probability ratios and intended lambda ratios given $Time_{bits} = 5$ and various $Truncation$ values. Each data point is acquired by going through the last two RSU-G stages (sampling and comparison) and collecting 10^6 samples from 2 possible labels with the corresponding λ_{max} and λ_i, where λ_{max} is the largest possible decay rate, which is $8\lambda_0$ when $Lambda_{bits} = 4$. With 2^n lambda approximation, the ratio of two lambdas can only be 1, 2, 4, and 8. We vary λ_i for one label and keep the other label constant at λ_{max} to achieve the intended λ ratio, which is how the previously described decay rate scaling works.

The results show that the divergence between the computed probability ratio and the intended ratio is large when $Truncation$ is low (below 0.1 in this case) or too high (above 0.6), while the divergence is small when $Truncation$ is in the middle range. $Truncation$ has little impact when the lambda ratio is 1. Since $\lambda_0 \propto -\ln(Truncation)$, small $Truncation$ leads to a larger λ_0, and consequently larger λ_is. According to Equation 3.4, large λ_i compresses the TTF into a small range early in time, thus placing more samples into the same time bin. This causes divergence from true probability ratios in the left side of Figure 3.10. Conversely, a large $Truncation$ value over-truncates the distribution for both λs and severely changes the distribution, causing divergence in the right side of Figure 3.10. Finding a reasonable $Truncation$ value is critical to balance the information loss.

We explore the space of $Time_{bits}$ and $Truncation$ to examine their impact on result quality. Figure 3.11 shows the BP results for the stereo vision dataset *poster*. Darker colors indicate a high BP (lower quality, lower cost) and white indicates a low BP (higher quality, higher cost). These results show that we can improve quality by either increasing $Time_{bits}$ or increasing $Truncation$ up to a point for fixed $Time_{bits}$.

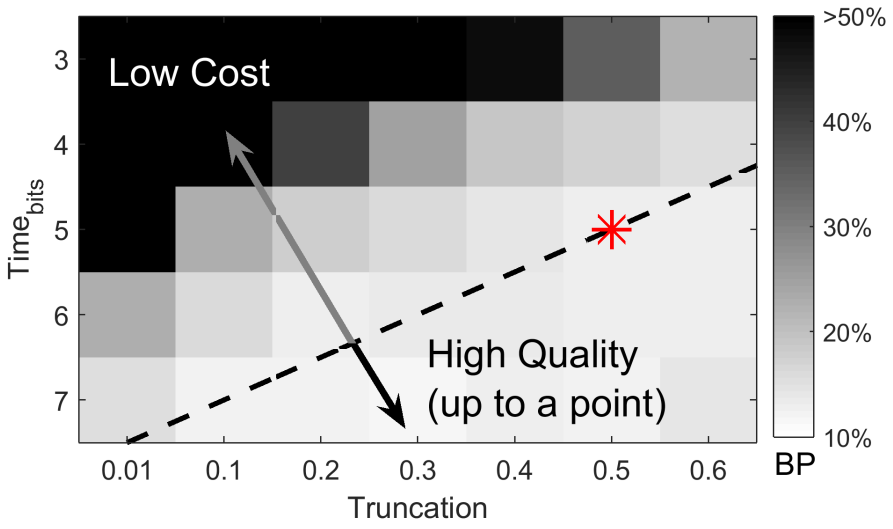
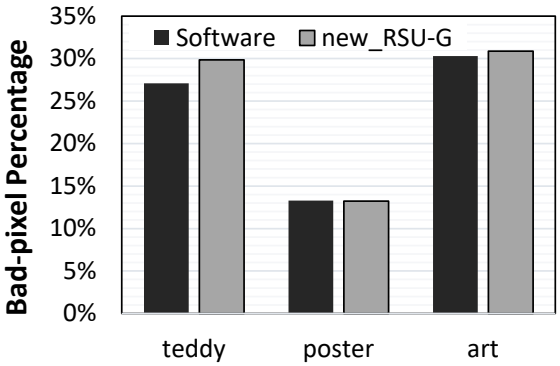
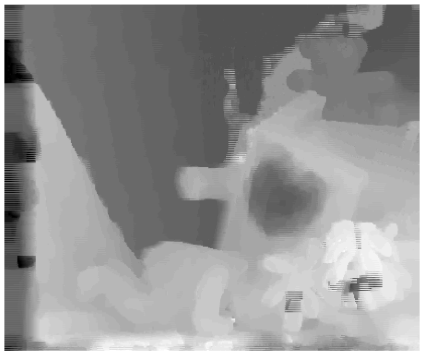


FIGURE 3.11: Result quality of $Time_{bits}$ vs. $Truncation$ in *poster*

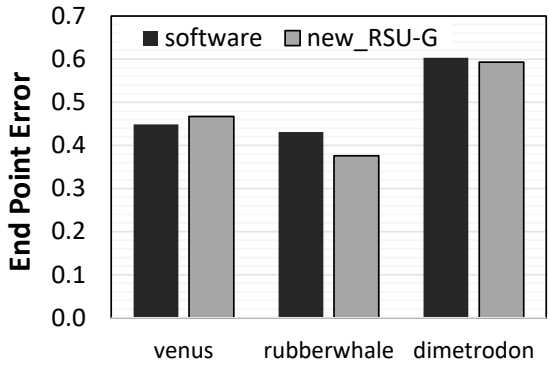
Configurations on the dashed line produce the same result quality (we verify this across all three applications). However, a designer has the freedom to move along this line (up-right or down-left) to optimize the RSU-G implementation. Section 3.4 discusses the implementation trade-offs involved in moving along this line. Qualitatively, increasing $Time_{bits}$ requires either increasing clock frequency or replicating RET circuits to provide the necessary number of timing bins, thus increasing area and power. Reducing $Time_{bits}$ requires increasing $Truncation$ to maintain result quality. Unfortunately, for high $Truncation$ values, it is more likely that photons generated by one sample influence the next sample(s) since the RET network remains excited beyond the $Truncation$ threshold. To avoid this excitation "bleed through", $Truncation$ requires RET network replicas. The red star in Figure 3.11 corresponds to our chosen point ($Time_{bits} = 5$ with $Truncation = 0.5$) which provides a good balance. In contrast, the previous RSU-G design requires very low $Truncation$ (< 0.01) due to using a single RET network in the RET circuit.



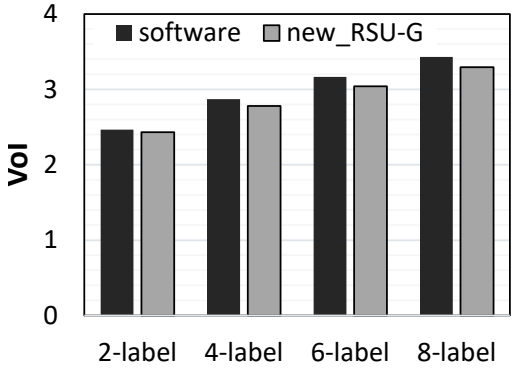
(a) Stereo vision



(b) *teddy* by new RSU-G



(c) Motion estimation



(d) Image segmentation

FIGURE 3.12: RSU-G result quality for $Time_{bits} = 5$ and $Truncation = 0.5$ across applications

3.3.4 Result Quality for new RSU-G

We run all three applications on our new RSU-G design with $Energy_{bits} = 8$, $Lambda_{bits} = 4$, $Time_{bits} = 5$, $Truncation = 0.5$ and compare with software-only results using standard benchmarking metrics. We believe these benchmarks represent the characteristics of most workloads.

Stereo Vision Figure 3.12a shows the BP across three datasets in stereo vision. The new RSU-G design achieves comparable BP with only small variations (3% BP differences in *teddy*, 0.1% in *poster*, and 0.5% in *art*). Figure 3.12b shows *teddy* disparity map in new RSU-G.

Table 3.3: Standard deviation of VoI across 30 tested images

	2-label	4-label	6-label	8-label
Software-only	0.63	0.71	0.71	0.79
New-RSUG	0.63	0.69	0.68	0.76

Motion Estimation Motion estimation finds pixel correspondence between two temporally sequenced frames, with a 2D search space. This differs from stereo vision in two important aspects: 1) the method of energy calculation [69] and 2) more labels (N^2 labels for the $N \times N$ search window). Due to the maximum-label limit, we make the common assumption that motion is relatively small compared to whole images [69]. Larger search windows can be obtained using an image pyramid method [69]. We use Middlebury motion estimation benchmarks [7] and randomly pick 3 datasets (*venus*, *rubberwhale*, and *dimetrodon*) that can fit in our search window. We use the common end-point error as the quality metric [7]. From Figure 3.12c, the new RSU-G produces results comparable in quality to the software implementation.

Image Segmentation We use the Berkeley Segmentation Database (BSD300) [90], and randomly select 30 images from among 300 images. We run multiple instances for each input with a different number of image segments (labels). We run 2, 4, 6, and 8 segmentations across these selected images with 30 iterations for each. Result quality is obtained using BISIP [146], a widely-used evaluation package that provides four result quality metrics. We only present Variation of Information ($VoI \in [0, \infty)$, lower VoI is better) as an example. Figure 3.12d shows RSU-G achieves result quality comparable to the software. We also show the standard deviation of VoI in Table 3.3. It is possible that in some cases RSU-G outperforms software-only results due to the stochastic property of MCMC algorithms, but overall they produce the same result quality.

3.4 A High Quality RSU-G

The previous quality analysis shows that an RSU-G with $Energy_{bits} = 8$, $Lambda_{bits} = 4$, and scaling decay rates with probability cut-off provides high result quality. From the several possible combinations of $Time_{bits}$ and distribution truncation, we choose $Time_{bits} = 5, Truncation = 0.5$ based on preliminary analysis. Importantly, these changes are mostly microarchitectural with minimal impact on the overall pipeline design or the architectural interface to the RSU-G, therefore the new design retains the sizable performance improvements of the previous work [142], as summarized in Section 3.1.2. The challenge is to design a new microarchitecture that meets the specifications with minimum area and power overheads.

3.4.1 Qualitative Design Trade-offs

Section 3.3 shows that simply scaling the previous RSU-G design cannot achieve the desired precision for high result quality. First, the existing design lacks support for decay rate scaling and probability cut-off, and thus, fails to provide sufficient dynamic range for λ. Simply increasing $Lambda_{bits}$ precision cannot improve result quality without applying decay rate scaling. Second, the previous RSU-G does not utilize 2^n lambda approximation which can reduce area and power. Using light intensity to achieve decay rate scaling requires excessive QDLEDs for large λ values.

Furthermore, the previous design does not exploit distribution truncation. Figure 3.11 exposed a trade-off between distribution truncation and time measurement precision, where points along the diagonal line produce similar high result quality while minimizing cost. Finding an optimal point along this line requires evaluating full implementations of each point. Qualitatively, increasing distribution truncation increases the number of RET networks required for some decay rates. We cannot re-use RET networks for consecutive label evaluations since there is a significant

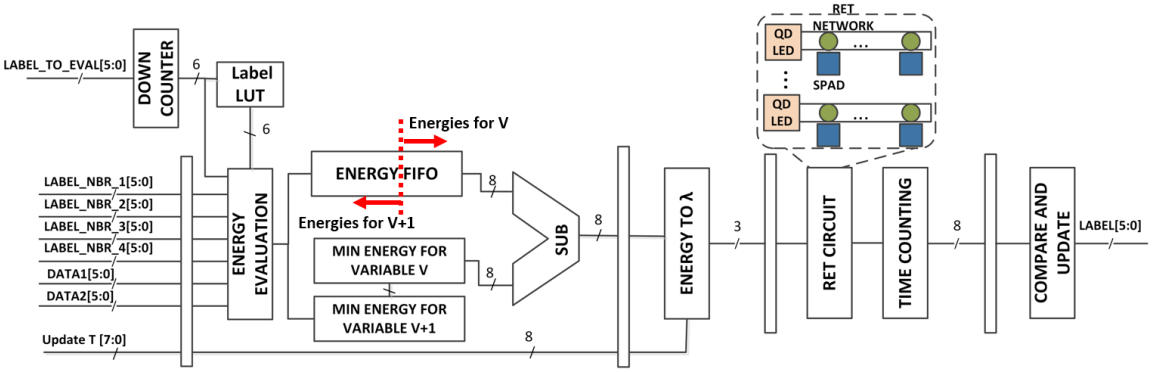


FIGURE 3.13: High quality RSU-G pipeline

probability that the RET network will produce an unwanted sample during the next label evaluation: a large number of chromophores may remain excited and are likely to emit photons in a subsequent cycle. The previous RSU-G design used 4 replicas to provide a distribution truncation of 0.004, each RET network had only a 0.004 probability of producing an unwanted sample. Naively scaling replicas in the current design would require even more QDLEDs and additional SPADs, resulting in excessive area and power.

From this discussion we conclude that we need a new RSU-G design that 1) efficiently provides decay rate scaling and probability cut-off, 2) takes advantage of fewer unique decay rates, and 3) balances distribution truncation with timing precision. The remainder of this chapter describes techniques to achieve this goal.

3.4.2 A New RSU-G Design

Figure 3.13 shows our new RSU-G pipeline. This design satisfies all the criteria outlined above while maintaining nearly the same external (architectural) interface as the previous design, the only addition is to allow updating the temperature dynamically each iteration without adding additional latency or pipeline stalls, thus achieving the same steady-state performance as the previous design. The latency for a single pixel evaluation increases since there are more pipeline stages, but the

throughput remains identical to the previous design at one label evaluation completed per cycle. However, the microarchitecture is substantially different since almost all stages in the pipeline, except for comparison, differ from the previous RSU-G. The important differences are in the energy evaluation, the efficient energy to λ conversion, the RET circuit, and a decoupling of the pipeline into two sections such that the back-end operates on the variable (e.g., pixel) v while the front-end processes variable $v + 1$. The remainder of this section describes the techniques we utilize to achieve high result quality with no or modest area and power increase over the previous design.

Support for Multiple Distances

To support a broader set of applications we add new distance calculations in the energy calculation stage of the previous RSU-G, which only supports squared distance. Specifically, we add binary and absolute distance in the doubleton calculation and absolute distance in the singleton calculation, which covers the majority of MCMC MRF applications in computer vision [71]. This additional flexibility requires a LUT to store all possible label values and additional combinational logic. We modify the architectural interface such that users can configure the energy calculation at the beginning of the application.

Decay Rate Scaling

Decay rate scaling is unique for each pixel evaluation and is performed by subtracting the energy for each label from the minimum energy, $E'_i = E_i - E_{min}$. This requires observing all label energies (E_i) to find E_{min} and then performing the subtraction. To implement this, we introduce a FIFO queue in the existing RSU-G pipeline for storing all label energies. This queue decouples the pipeline stages before λ look-up from the rest of the pipeline, enabling us to find E_{min} and perform scaling. We

use two registers to support decay rate scaling. One register stores the minimum observed energy as new energy values are inserted into the FIFO for variable $v + 1$. A second register stores the minimum energy to use for scaling the decay rates while processing variable v. Each cycle an energy is read from the FIFO and subtracted from the value in the second register to perform scaling. This scaled energy is used as the input value for energy to λ conversion. Note that at any given time during the steady state, energies of two different variables reside in the queue (except for a single cycle when the energy for the last label of variable $v + 1$ enters, and all the energies in the FIFO belong to that variable); one variable is going through energy computation and thus, is constantly updating its minimum energy, and the other has its minimum energy determined and is going through the energy to λ conversion stage.

Efficient Energy to Lambda Conversion

Since a relatively small number of unique λ is needed due to 2^n approximation, we re-evaluate the previous LUT design for energy to λ conversion. Recall that energy to λ conversion is based on Equation 3.3. This can be implemented in two ways: 1) using a LUT with energy as an index to load precalculated values, or 2) storing the boundary values for each λ value and performing comparisons to determine the interval the energy belongs to. The LUT design is more efficient when many λs are needed since comparisons do not scale well when the number of intervals is large, and more memory is needed to store the boundary values. However, since 4 unique λs only generate five intervals $\lambda = 0, 1, 2, 4, 8$, at most 4 comparisons need to be performed for any energy value.

The comparison-based design outperforms a LUT in two ways: first, it significantly reduces the total memory needed from 1024 bits to 32 bits. Second, updating a 32-bit register is much faster than updating a 1K-bit LUT when updating the tem-

perature. With an 8-bit interface, it only takes four cycles to update all boundaries, which would add 3 additional stall cycles in the pipeline at end of each iteration. To eliminate these 3 additional cycles, we add additional 32-bit registers to buffer the new boundary values so that temperature updates can occur simultaneously with sampling. Although a wider 32-bit interface can hide this additional latency, we choose an 8-bit interface with buffers to keep the total interface small in size. Energy to Lambda conversion outputs a 3-bit value, using the MSB to indicate probability cut-off and 2 other bits to indicate the unique λ used by the RET circuit. Our area/power evaluation indicates a 0.46× area and 0.22× power relative to the LUT implementation.

Implementing Decay Rates

Recall (Section 3.1.2) that RET network decay rate depends on input light intensity and chromophore concentration [141]. These parameters lead us to the following design alternatives: 1) keep using QDLEDs to provide four unique intensities and have one RET network and one SPAD; 2) have one QDLED, whose voltage (and thus intensity) is controlled by a digital-to-analog converter (DAC), and have one RET network and one SPAD; 3) have one QDLED and add RET networks with different chromophore concentrations, one for each desired decay rate.

As shown previously, intensity-based decay rates cannot benefit from fewer unique λs since DACs are power hungry [19]. Instead, we use a unique concentration for each decay rate, where a waveguide couples to 4 RET networks with concentrations of 1×, 2×, 4×, and 8× of that corresponding to λ_0, and a single QDLED. The appropriate SPAD output corresponding to the RET network with the desired decay rate (concentration) is selected as input to the timing circuit using a multiplexer (see Figure 3.14). Without considering truncation, a single QDLED coupling 4 RET networks covers all unique λs. However, as described later, distribution truncation

requires replicated RET circuits due to the long exponential tail.

Timing Precision

The previous quality analysis shows that we need between 4 and 8 time bits (16-256 time bins). We use a clock multiplier and a shift register to read the SPAD output which generates the TTF for a given RET network. Assuming a 1GHz clock and an 8× multiplier, the finest resolution is 125ps for a time bin, any lower resolution becomes impractical due to the clock multipliers. The SPAD output is sent to an 8-bit shift register to obtain a unary encoded value for the sample, with all zeros indicating no photon observed in this 1ns cycle. Wire delay from SPADs to shift register is negligible compared to 125ps [55]. This design provides $Time_{bits} = 3$ (8-bit unary = 3-bit binary). This is clearly not sufficient to provide high result quality.

To increase timing precision, we extend the window for observing fluorescence to more than one clock cycle. The number of clock cycles required for a specific time precision is $Cycles = 2^{Time_{bits}}/8$ and ranges from 2 to 32 cycles for $4 \leqslant Time_{bits} \leqslant 8$. The number of cycles directly influences RSU-G design since this is the window within which to potentially observe fluorescence, and replicated RET circuits are required to avoid a structural hazard and sustain one label per cycle evaluation. Our chosen point, $Time_{bits} = 5$ (32 bins), requires 4 replicas since our observation window is 4 cycles (32/8).

Distribution Truncation

To truncate the tail of the distribution, we simply stop looking for fluorescence of a given RET network and assume it never occurs ($TTF = \infty$). Unfortunately, the RET network may still have excited chromophores that fluoresce at a later time, therefore the RET network cannot be re-used until there is a sufficiently low prob-

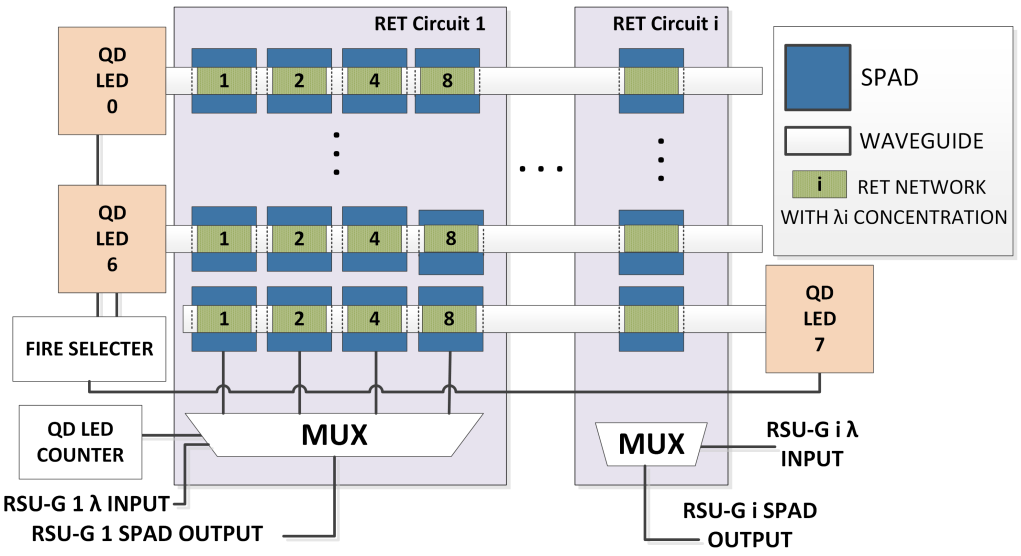


FIGURE 3.14: RSU-G RET circuit components

ability of fluorescence. To sustain one label evaluation per cycle, we replicate the QDLED+RET network in each RET circuit.

The previous RSU-G achieves 99.6% probability that a previous sample does not affect a later sample; however, higher truncation rates require more replicas to achieve this same goal, and lower truncation is preferred to minimize design overhead. For example, with $Truncation = 0.5$, we need 8 RET network replicas to achieve 99.6%. RET network replication is separate from, and can be combined with, RET circuit replication required to obtain a specific timing precision, as described above. Replicating RET networks necessitates multiple light sources since exciting two RET networks with the same light source is equivalent to having no replicas, and multiple RET circuits within a single RSU-G cannot share the light sources and waveguide for the same reason. However, multiple RSU-Gs can share the same waveguide as long as each RET network is not reused within the minimum interval time to reach 99.6% probability of fluorescence.

Our design, shown in Figure 3.14, allows multiple RET circuits to share the same light source and waveguide while satisfying the minimum interval time constraint.

This design is based on $Time_{bits} = 5$ and $Truncation = 0.5$, where our preliminary analysis shows a good balance. Other design points incur either 1) more RET circuit replicas to achieve higher time precision, or 2) more RET network replicas and larger select logic to satisfy the minimum interval time constraint. Finding the optimal design point requires synthesizing results of all points on the line.

Within a RET circuit, four RET networks with unique concentrations share a waveguide and eight sets of RET network replicas are located on different waveguides for concurrent operation. A QDLED counter increases by 1 every four cycles, indicating which waveguide is in use. A 32-to-1 MUX selects the SPAD signal from the desired RET network for the subsequent timing circuit. The QDLED counter and λ input specify the RET network row and column respectively. Multiple RET circuits from different RSU-Gs can be placed on the same waveguide as long as the light source provides sufficient intensity to drive all RET network replicas. With a proper layout, overheads caused by the light source and the RET networks can be amortized without incurring significant interconnection overhead. The new RSU-G design also opens the possibility to use a different light source, such as thin-film edge emitting lasers or VCSELs [115], since intensity control is no longer needed. Moreover, placing multiple RET network replicas from multiple RSU-Gs on one waveguide enables the potential to use external light sources across all RSU-Gs without a need to integrate on-chip light sources. Previous work demonstrates selectively coupling optics from a single light source to a desired waveguide using ring resonators [57,114,116]. Finding the proper design and layout for a multiple RSU-G architecture is ongoing work.

3.4.3 Evaluation

The new RSU-G design described above preserves nearly the same architectural interface but adds an interface for temperature updates. Since our design guarantees no additional latency during temperature updates, speedups from the previous RSU-

Table 3.4: Stereo vision execution time (seconds)

	320x320 SD		1920x1080 HD	
	10-label	64-label	10-label	64-label
GPU_float	0.078	0.401	0.894	6.522
GPU_int8	0.070	0.378	0.784	5.870
RSUG_aug	0.025	0.071	0.220	1.067
Speedup_flt	3.125	5.652	4.058	6.115
Speedup_int8	2.828	5.323	3.561	5.504

G still hold [142]. However, we introduce a new application in this chapter, stereo vision, thus we implement two GPU versions of stereo vision with float precision energy and 8-bit integer energy implementation, and compare results with an RSU-G augmented GPU (RSU-G_1) using the previous methodology [142] (i.e., best-effort GPU implementations with packed inputs as the baseline). Table 3.4 shows the execution time and speedup. RSU-G provides speedups similar to image segmentation in SD images, but provides higher speedup in HD images.

We estimate the area/power of the CMOS portion of the new RSU-G using Cacti [133] and a Verilog implementation synthesized using a predictive 15nm process [92]. First principles are used to calculate the area/power for QDLEDs [47,127], RET networks, and SPADs [5, 86, 113], as in the previous work [142]. Conservatively, area/power analysis uses one RSU-G per light source plus waveguides, i.e., each RSU-G has 8 independent QDLEDs. Waveguides are straight, with pitch equal to half width of a QDLED, and no circuitry in the spare area. Table 3.5 summarizes our evaluation. The new RSU-G design consumes a slightly higher (1.27×) power but keeps the same area compared to the previously proposed RSU-G. Most power increase is introduced by supporting more functionality in energy calculation. However, a single RET circuit alone consumes 0.7× area and 0.5× power compared to the previous design. Sharing light sources and waveguide can further reduce area/power. Most importantly, the new design achieves the goal of providing high result quality

Table 3.5: New RSU-G area and power consumption

Component	Area(μm^2)	Power (mW)
RET Circuit	1120	0.08
CMOS Circuitry	1128	3.49
LUT	655	1.42
RSU-G Total	2903	4.99

and significant speedups for MCMC MRF models.

To further compare RSU-G with pure-CMOS designs, we estimate the area of equivalent CMOS designs by replacing the sampling portion of RSU-G with an alternative true RNG (Intel DRNG [98]) and more aggressive pseudo-RNG designs (LFSR and mt19937 [94]). These RNGs lack programmability. As a result, generating parameterized distributions requires a LUT to store the target cumulative distribution function (e.g., store {1,3,6,7} for the discrete probability distribution {1,2,3,1}). The LUT size is proportional to the maximum number of supported labels.

Table 3.6 shows the estimated area comparison. To evaluate the potential benefits of RSU-G, we estimate RSU-G area in 1) a design where 4 RSU-Gs share a light source set (*RSUG_4share*), and 2) an optimistic design in which many RSU-Gs share a light source set with negligible amortized area, and CMOS circuits can reside underneath the waveguides (*RSUG_optimistic*). Mt19937 RNG area is obtained from the previous work [144] and scaled to 15nm technology [128]. We estimate the area when using one RNG per sampling unit (*mt19937_noshare*), per 2 units (*mt19937_4share*), and per 208 units (*mt19937_208share*, maximum value in [144]). We consider only AES-256 [1] area, which is one of three stages in Intel DRNG. One Intel DRNG can only support one sampling unit given the throughput limitation [98]. The 19-bit LFSR design is the most aggressive herein. Our preliminary quality analysis shows that the design provides result quality as good as mt19937 and RSU-G for the selected benchmarks (stereo vision and motion estimation). However, the

Table 3.6: Area comparison with alternative designs

True-RNG	Area(μm^2)	Pseudo-RNG	Area(μm^2)
RSUG_noshare	2903	19-bit LFSR	2186
RSUG_4share	2303	mt19937_noshare	19269
RSUG_optimistic	1867	mt19937_4share	6507
Intel DRNG (part)	3721	mt19937_208share	2336

result quality for other benchmarks and applications remains to be evaluated given the relatively short period of LFSR. Our work in the following chapter explores the possibility of a pure-CMOS design using a pseudo RNG. Previous work [117] uses true-RNG similar to [98] for neuron firing. Unlike Gibbs Sampling Unit in [117], RSU-G is a full functional unit. Overall, RSU-G can provide true-RNG using area comparable to LFSR designs and the power/area benefit [142] remains.

3.5 Limitations and Future Work

This work evaluates the result quality of three popular applications in computer vision, which we view as good representations of other applications in this field. The new RSU-G design keeps the same maximum number of labels that can be supported as the previous work, which provides sufficient support for most applications in this area. Nonetheless, providing support for more than 64 labels would expand the applications that can benefit from our approach. A deeper analysis of distribution truncation vs. timing precision is also needed to determine the optimal design parameters. Finally, although fabrication of RET technology [118,142], QDLED [47] and SPAD [109] are individually demonstrated elsewhere, a fully integrated RSU-G is yet to be demonstrated. We are also evaluating possible designs that use other types of light sources, which may further simplify fabrication. Photo-bleaching, which can degrade RET circuits, can be mitigated using known techniques [111].

Additional future work includes, but is not limited to, extending the samplers to support more than Gibbs Sampling, support for a wider application domain, and

exploring sampling from phase-type distributions.

3.6 Summary

The recent advances in statistical machine learning create new opportunities and challenges for improving overall computational efficiency. Direct support for probabilistic algorithms is an intriguing path to help alleviate the challenges due to the slowing of CMOS scaling. Several approaches that utilize emerging technologies are being explored in the community. This chapter builds on the recent technique of utilizing molecular-scale optical devices to construct efficient samplers that exploit the physical property of resonance energy transfer (RET). The previously proposed RET sampling unit for Gibbs Sampling (RSU-G) can be added to commodity processors or used to create a discrete accelerator and provide significant speedups.

We first propose a macro-scale RSU-G prototype, the first such system to our knowledge, that demonstrates the capability of a RET network to parameterize a distribution and run a real application. The NIST statistical test results confirm the RET network produces relatively high-quality randomness without post-processing.

We further ask and answer several questions related to the implementation of the RSU-G and how certain design decisions affect the overall application result quality. Using community standard metrics for three represented applications, we find that the previously proposed RSU-G does not provide adequate result quality. We identify four primary RSU-G design parameters and explore their impact on result quality. We present a new RSU-G design with minimal architectural interface changes that maintains the performance improvements of the previous design, and provides high result quality with a negligible area overhead and modest power increases. We also enable opportunities to further reduce power, area, and fabrication costs with a shared light source and waveguide design.

This work takes one more step on the path toward finding methods to accelerate

probabilistic algorithms. Next, we explore the feasibility of a pure CMOS stochastic processing unit that can maintain the performance, area, and power benefits of the RSU-G.

4

A CMOS Stochastic Processing Unit

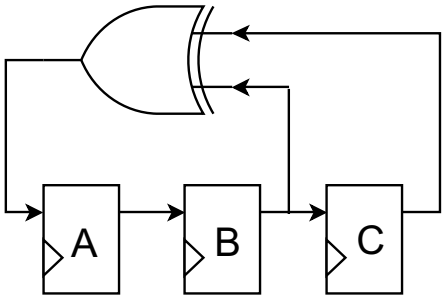
The new RSU-G is promising to accelerate 1st-Order MRF Gibbs Sampling algorithms. Fabrication of RSU-G in an integrated circuit, however, requires an additional back-end-of-line process to integrate RET networks and optical devices onto traditional CMOS circuits, increasing manufacturing costs. One favorable property of RSU-G is the high-quality true randomness from the quantum states of RET networks, which in theory guarantees unrepeatable and unpredictable samples. Without RET circuits, the randomness needs to be provided by CMOS RNGs. A deterministic CMOS digital circuit only provides pseudo randomness without an external entropy source. A CMOS true RNG, such as Intel DRNG [98], consumes too much area/power. The key question is *do we actually need a true RNG for our target 1st-Order MRF Gibbs Sampling applications? If not, what RNGs are enough to provide good result quality?*

In this chapter, we first explore the feasibility of a pure CMOS sampling unit equivalent to RSU-G. We evaluate six different RNGs (8-bit, 16-bit, 19-bit LFSR, Mersenne-Twister 19937 [94], and Intel DRNG with pseudo and true randomness) on motion estimation and stereo vision applications with 64-bit floating-point (FP64)

precision elsewhere. Unexpectedly, we discover a simple 19-bit LFSR is sufficient to provide good application end-point result quality. Using more complicated RNGs does not further improve result quality. We observe notable drops in quality if using lower quality RNGs than a 19-bit LFSR. The results indicate *designing an efficient CMOS sampling unit providing functional equivalence to RSU-G is feasible.*

Therefore, we propose a CMOS Stochastic Processing Unit (SPU) by replacing the RET-based sampler with a CMOS discrete sampler. The design uses the RSU-G front-end pipeline for energy computation and probability conversion, and thus keeps the RSU-G optimization techniques to improve the efficiency while maintaining high result quality, including: 1) dynamic scaling, 2) probability truncation (cut-off), and 3) 2^n approximation. The discrete sampler implements inverse transform sampling fed by the least 12-bits of a 19-bit LFSR and uses mathematical approximations to maximally simplify arithmetic computation in hardware. The SPU can be deployed on an FPGA or fabricated in an ASIC. We further optimize a design targeted to an Intel Arria 10 FPGA by adding an additional internal stage and packing multiple computations into one DSP. Importantly, the SPU does not change the architectural interface and thus maintains the sizable RSU-G speedups.

Our quality analysis on three 1st-Order MRF applications shows the SPU with a simple 19-bit LFSR achieves the same result quality as FP64 software. An FPGA HLS baseline is implemented to assess the option of directly using FP32 after energy computation without a specialized architecture. The SPU optimized for FPGA achieves at least 3× faster in performance and 33.7× less in memory compared with the HLS baseline, indicating a human-designed architecture is needed to improve efficiency. The SPU avoids using a complex RNG and thus saves 33% in area and 57% in power compared with an RSU-G without light source sharing. Note that the SPU results do not preclude other potential benefits of the true RNG in RSU-G such as unpredictable seeds [129], which is beyond the scope of this dissertation.



Cycle	A	B	C	RN
0	0	0	1	1
1	1	0	0	4
2	0	1	0	2
3	1	0	1	5
4	1	1	0	6
5	1	1	1	7
6	0	1	1	3
7	0	0	1	1

FIGURE 4.1: A 3-bit Linear-Feedback Shift Register (LFSR) and its output random numbers (RN) [148]

The remainder of this chapter is organized as follows. Section 4.1 explores the relationship between RNGs and application result quality. Section 4.2 describes the SPU pipeline and optimization details. The evaluation is provided in Section 4.3. Section 4.4 describes limitations and future work. Section 4.5 summarizes the chapter.

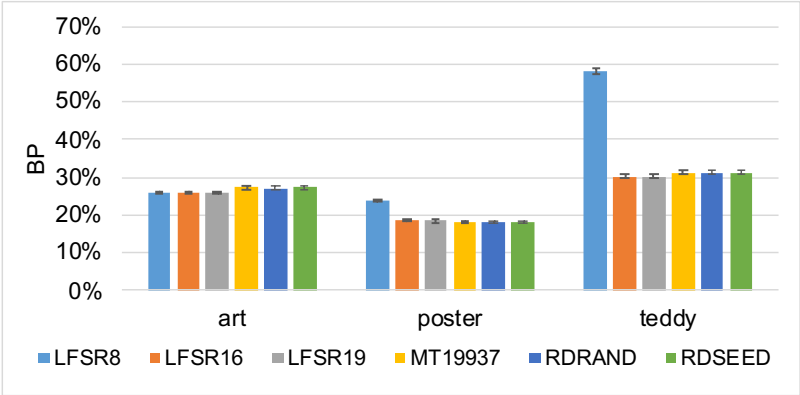
4.1 RNGs vs. Application Result Quality

The key element of probabilistic algorithms is generating random samples using an RNG. The previously discussed RSU-G uses RET networks and optical devices to provide high-quality quantum randomness. Another example of an optical quantum RNG is a quantum photon RNG [139]. Although quantum randomness offers "a possibility for scientific proof of randomness" [129], those RNGs require optical elements in addition to conventional CMOS when fabricated as integrated circuits, increasing manufacturing costs.

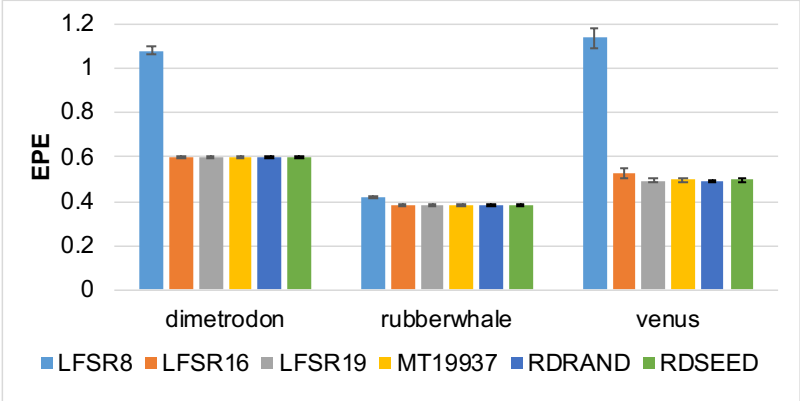
Alternatively, we can use conventional CMOS technology for pseudo or true RNG without a quantum randomness guarantee. A straightforward approach is using pseudo RNG—pure digital logic and mathematical computation to produce a sequence of bits that seem to be "random". One of the simplest, yet popular, pseudo

RNG is a Linear-Feedback Shift Register (LFSR). Figure 4.1 shows an example of a 3-bit LFSR that can generate pseudo-random numbers between one and seven. The key elements are a chain of shift registers, and XOR gates that feed into the beginning of the chain. The locations of the XORs are determined by the maximal-length polynomials so that every number—in this case, from one to seven—can be picked. Note that the initial state cannot be all zeros. A 3-bit LFSR has only a seven-cycle period: the pattern of random numbers will repeat after the seventh cycle. A more practical 19-bit LFSR has a period of $2^{19} - 1 = 524,287$. A more complex pseudo RNG, Mersenne Twister introduced by Matsumoto and Nishimura [94], has a period of $2^{19937} - 1$ and is usually used as the default pseudo RNG in many systems and IDEs, including MATLAB. CMOS-based true RNGs are available by utilizing thermal noise [98], telegraph noise [51], free-running oscillator [53], etc. In theory, a low-quality RNG can lead to biased results given a repeated pattern in a short period. Previous experiments show that bad RNGs can lead to significantly different results in Monte Carlo simulation [23]. More complex RNGs produce higher quality randomness, but typically consume more hardware resources: a Mersenne Twister needs 2.5KB of memory to buffer states; Intel DRNG [98] involves two stages of post-processing. Chapter 3 (Table 3.6) presented a preliminary area comparison by replacing RET circuits with other CMOS RNGs. The question is *what RNGs are good enough in our targeted applications?*

We evaluate application end-point result quality of C++ stereo vision and motion estimation using six different RNGs: 8, 16, 19-bit LFSRs, Mersenne Twister (*mt19937*), Intel DRNG with pseudo-random output (*RDRAND*), and Intel DRNG with true-random output (*RDSEED*). The C++ software uses FP64 precision elsewhere. Figure 4.2 shows the mean and standard deviation of result quality in bad-pixel percentage (BP) for stereo vision and end-point result (EPE) for motion estimation. Each bar is collected from 50 MCMC runs. Theoretically, RNGs



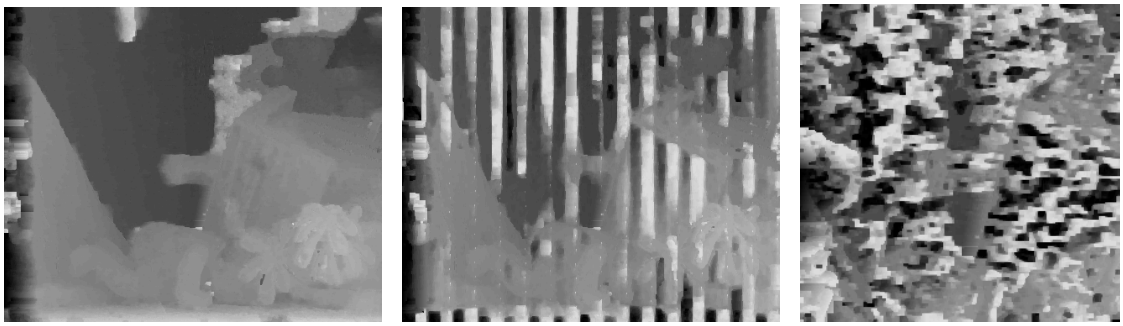
(a) Stereo vision result quality over RNGs (lower is better).



(b) Motion estimation result quality over RNGs (lower is better).

FIGURE 4.2: Result quality analysis over RNGs with floating-point elsewhere

from left to right provide better randomness quality. Unexpectedly, a simple 19-bit LFSR provides the same application result quality as more complex RNGs (*mt19937*, *RDRAND*, and *RDSEED*). An example *teddy* result with 19-bit LFSR is provided in Figure 4.3a. The 8-bit LFSR has a repeated pattern every 255 cycles, which significantly degrades the result quality in 4 out of 6 benchmarks. An example result with a stripe pattern shows in Figure 4.3b. The 16-bit LFSR produces good result quality in 5 out of 6 benchmarks and slightly worse quality in motion estimation *venus*. However, since the period $65,535 = 255 \times 257$ is not a prime number, for the cases where the size of image inputs is aligned with the factors of the period, the effective period for these inputs is reduced and the result quality can drop. Figure 4.3c shows



(a) *Teddy* with 19-bit LFSR (BP: 29.2%).

(b) *Teddy* with 8-bit LFSR (BP: 59.5%).

(c) 257 × 255 cropped *teddy* with 16-bit LFSR (BP: 81.5%).

FIGURE 4.3: Examples of *teddy* RNG results

an extreme example where *teddy* is cropped to 257 by 255 pixels. The result quality significantly drops compared with the original input with a size of 450 by 375. To avoid these cases, the LFSR period needs to be a prime, such as 524,287 in a 19-bit LFSR. These results indicate that *a CMOS sampling unit providing functional equivalence to RSU-G is feasible using a pseudo RNG*. The next section explores such a design.

4.2 Exploring a CMOS Stochastic Processing Unit

In this section, we explore a CMOS Stochastic Processing Unit (SPU) by replacing the RET circuit sampler with a CMOS discrete sampler. The design provides equivalent functionality to RSU-G and flexibility to be deployed on an FPGA or fabricated in an ASIC. We demonstrate a couple of design optimization techniques for an Intel Arria 10 FPGA.

4.2.1 SPU Pipeline

Figure 4.4 shows the block diagram of the Stochastic Processing Unit (SPU) pipeline. It is divided into four main stages (9 internal stages) with two internal decoupling FIFOs and an inverse transform method is used for the discrete sampler. The front-end

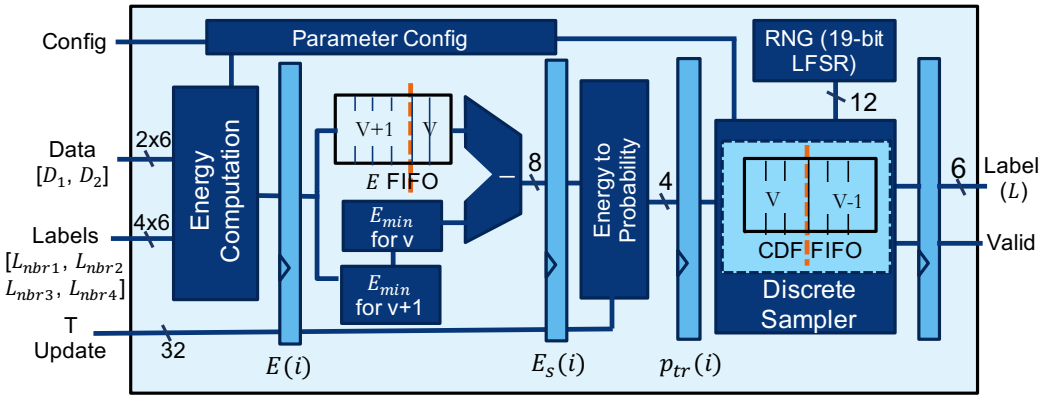
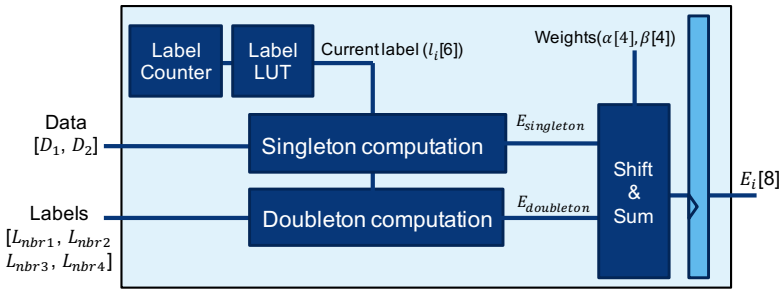


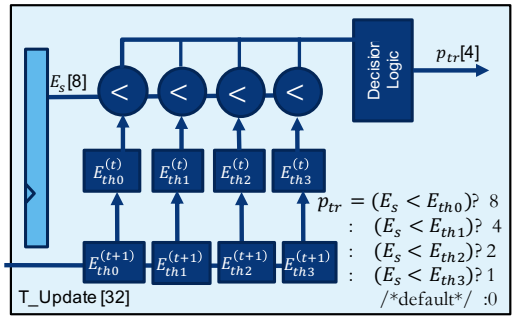
FIGURE 4.4: SPU pipeline

stages in the RSU-G before RET sampling stage is transferable to SPU. Techniques to improve result quality in these stages also apply to SPU and the previous design point still holds (recall $Energy_{bits} = 8$ and $Lambda_{bits} = 4$ in Chapter 3). The term "decay rate" λ is changed to "scaled probability" p_s for accuracy. A discrete sampler with a CMOS RNG replaces the RET sampling and comparison stages. The SPU supports two operating modes: 1) pure-sampling and 2) optimization (simulated annealing). Recall pure-sampling iteratively generates Gibbs samples using constant temperature T and simulated annealing strategically decreases the temperature per iteration for faster convergence. Compared with the related work [66, 87], the SPU exploits the discoveries from a comprehensive design space exploration by using: 1) full-custom precision; 2) optimization techniques including dynamic scaling (a.k.a., decay rate scaling), probability truncation, and 2^n approximation; 3) a simple 19-bit LFSR with the 12-bit LSB output. Below summarizes the operation of each stage.

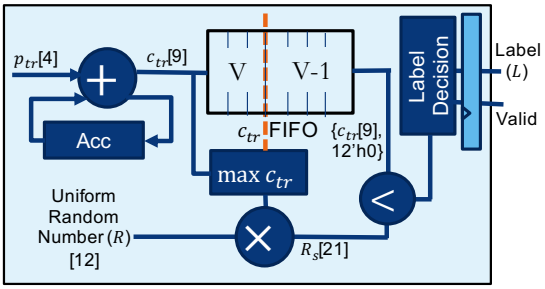
Energy Computation and Scaling Figure 4.5a presents the block diagram of energy computation, the first stage in the SPU. Given the data, current label, and neighbor labels, the stage computes the total energy of a possible label $E(i)$ per cycle. The energy $E(i)$ is a weighted sum of singleton energy from data and doubleton energy from neighbor labels (Equation 4.1). α and β are application parameters approximated by



(a) Energy computation



(b) Energy to scaled probability conversion



(c) CMOS discrete sampler

FIGURE 4.5: SPU design in each stage

shift operations. The SPU supports three types of energy function including binary, absolute, and squared distance. $E(i)$ is then dynamically scaled using subtraction to maximize the dynamic range (Equation 4.2). Both $E(i)$ and $E_s(i)$ are 8-bit unsigned integers.

$$E(i) = \alpha E_{singleton}(i) + \beta \sum E_{neighborhood} \tag{4.1}$$

$$E_s(i) = E(i) - E_{min} \tag{4.2}$$

Energy to Scaled Probability In the third stage, the scaled energy $E_s(i)$ is converted to a scaled probability represented in 4-bit unsigned integer. The original probability is computed by $exp(-E_s(i)/T)$ which is represented as a real number between 0 and 1 using floating-point in software, where T is a fixed parameter per outer iteration. However, the probability is scaled using Equation 4.3 and truncated using Equation 4.4 to match the unsigned integer representation, where $P_{bits} = 4$ ($Lambda_{bits} = 4$ in

Chapter 3) is the number of bits in the scaled probability output $p_{tr}(i)$. Additionally, probability truncation drives all scaled probabilities that are less than one to zero and 2^n approximation rounds all scaled probabilities down to the nearest 2^n integer value (Equation 4.4). The value of $p_{tr}(i)$ can be pre-computed and stored in a look-up table (LUT). The values in the LUT need updates if T changes between iterations.

$$p_s(i) = (2^{P_{bits}} - 1) \times exp(-E_s(i)/T) \tag{4.3}$$

$$p_{tr}(i) = \lfloor 2^{\lfloor \log_2 p_s(i) \rfloor} \rfloor \tag{4.4}$$

As discussed in Section 3.4.2, energy to scaled energy conversion can be implemented in two ways: 1) using a LUT with energy as an index to load pre-calculated values, or 2) storing the energy boundary values for each p_{tr} value and performing comparisons across the boundaries (or thresholds, E_{th}s). The former design is more efficient when p_{tr} has many unique values. However, since $p_{tr} \in \{0, 1, 2, 4, 8\}$ and p_{tr} is monotonic, we can benefit from the latter design to reduce the total memory needed from 1024 bits to 32 bits. Figure 4.5b shows the energy to scaled probability conversion hardware. Furthermore, simulated annealing requires updating memory values at the end of each MCMC iteration. Reducing the memory to 32 bits significantly simplifies the update. The latency caused by memory update can be easily hidden by double buffering ($E_{th}^{(t)}$ and $E_{th}^{(t+1)}$).

Discrete Sampling The final stage of SPU generates a discrete sample per variable based on the probabilities of all possible label values $\{p_{tr}(0), p_{tr}(1), ...\}$ using the least 12-bits of a 19-bit LFSR to implement the inverse transform sampling [27]. A 19-bit LFSR generates a 19-bit uniform random number per cycle and update its internal states from S^t to S^{t+1} by Equation 4.5, where s_i^t is the i-th bit of S^t starting from

the least significant bit s_0.

$$\begin{aligned} s_{18}^{t+1} &= s_0^t \text{ XOR } s_1^t \text{ XOR } s_2^t \text{ XOR } s_5^t \\ s_i^{t+1} &= s_{i+1}^t, i \in \{0, ..., 17\} \end{aligned} \tag{4.5}$$

Next, we take the least 12-bits of LFSR state S as the uniform random output R to generate a discrete random sample per variable. The logic behind picking 12 bits is provided later. Given the truncated scaled probability p_{tr} of all possible labels, we accumulate a scaled Cumulative Distribution Function (CDF) $c_{tr}(i) = \sum_{j=0}^{i} p_{tr}(j)$ for all possible labels and obtain the largest scaled CDF $\max(c_{tr})$ from the last possible label. The original inverse transform sampling picks label $i + 1$ if

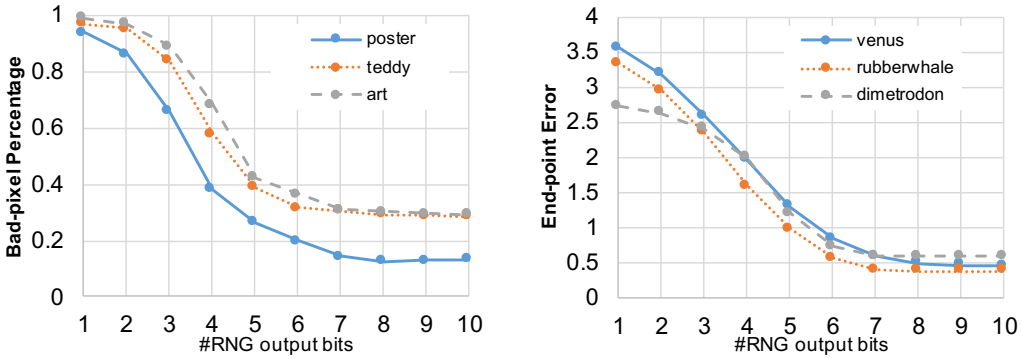
$$\frac{c_{tr}(i)}{\max(c_{tr})} < \frac{R}{R_{max}} \leqslant \frac{c_{tr}(i+1)}{\max(c_{tr})} \tag{4.6}$$

or picks label 0 if $R/R_{max} \leqslant c_{tr}(0)/\max(c_{tr})$. $R_{max} = 2^{12} - 1$ is the maximum possible value of a uniform random output. To maximally avoid expensive divisions and multiplications, our discrete sampler turns divisions into multiplications and approximates R_{max} to $R_{max} + 1 = 2^{12}$ so that a couple of multiplications become simple padding zeros. We pick label $i + 1$ if

$$c_{tr}(i)(R_{max} + 1) \leqslant \max(c_{tr})R < c_{tr}(i + 1)(R_{max} + 1) \tag{4.7}$$

or pick label 0 if $\max(c_{tr})R < c_{tr}(0)(R_{max} + 1))$. Only $\max(c_{tr})R$ requires an actual hardware multiplier. The comparison operators are deliberately switched to compensate approximation errors. Figure 4.5c shows the CMOS discrete sampler. It contains an internal decoupling by a CDF FIFO. The phase before the FIFO accumulates a scaled CDF and the phase after decides an output label based on the scaled CDF and a random number.

A full output of the LFSR is a 19-bit integer. Picking a full or a subset of RNG output is a design trade-off: a narrower output increases bias introduced by



(a) Stereo vision (b) Motion estimation

FIGURE 4.6: Result quality vs. RNG output bits in the discrete sampler

quantization and $R_{max}+1$ approximation whereas a wider output raises the hardware cost of the subsequent logics. Figure 4.6 shows stereo vision and motion estimation result quality vs. RNG output bits in the discrete sampler. A narrower output significantly degrades result quality. Result quality plateaus after 9-bits of RNG output. We conservatively pick 12-bits since it theoretically improves the precision and only incurs $48\mu m^2$ area overhead in the discrete sampler compared with 9-bits in a preliminary 15nm synthesis. A design that outputs all 19-bits incurs an additional $120\mu m^2$ overhead and we consider it unnecessary given the theoretical quality improvement is very marginal.

4.2.2 Optimization for FPGA

Compared with RSU-G, the CMOS SPU provides flexibility to be deployed on an FPGA or fabricated in an ASIC. We further optimize the above design specifically for an Intel Arria 10 FPGA by adding an additional internal stage in energy computation and packing multiple "sum of square" computations into one DSP. Adding an internal stage (i.e., 10 internal stages in total) increases Fmax from 321MHz to 374MHz. A fully-registered DSP can further improve Fmax. An unverified design with 11 total internal stages brings Fmax to 408MHz, close to the DSP Fmax limit (459MHz) in

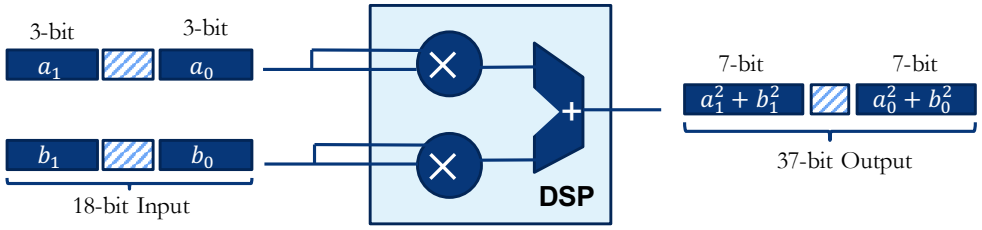


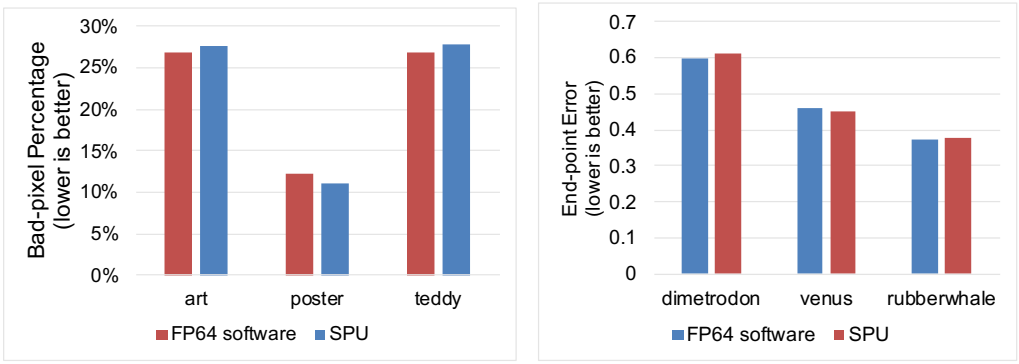
FIGURE 4.7: Packing two 3-bit by 3-bit "sum of square" into a 18-bit DSP

the specific FPGA model. Note that SPU operation frequency is jointly determined by SPU Fmax and a system-level architecture. A thorough optimization should incorporate with the system-level architecture and performing such an optimization is beyond the scope of this dissertation.

Placing more SPUs on an FPGA accelerator generally provides higher speedups. Without considering a system-level architecture, the maximum number of SPUs can be synthesized in the specific Arria 10 FPGA (10AX115N2F40E2LG) is limited by the total number of DSPs. Each SPU takes 6 DSPs: 4 for doubleton, 1 for singleton, and 1 for the discrete sampler. Each DSP can compute an at-most 18-bit by 18-bit "sum of square". The SPU implements four 3-bit by 3-bit "sum of square" for doubleton computation, each takes an entire DSP with most bits unutilized. We found it is mathematically possible to pack two of such computations into one DSP without interfering with each other by using the highest and lowest 3-bits of DSP inputs respectively, demonstrated in Figure 4.7. This technique reduces SPU DSP usage from 6 to 4 and thus creates the capability to place more SPUs into an FPGA accelerator, increasing the potential speedups. The next section shows the full synthesis results.

4.3 Evaluation

We implement the SPU in Verilog and Chisel, and High-Level Synthesis (HLS). The Verilog and Chisel implementation is verified in QuestaSim simulation and HLS



(a) Stereo vision (b) Motion estimation

FIGURE 4.8: Stereo vision and motion estimation result quality

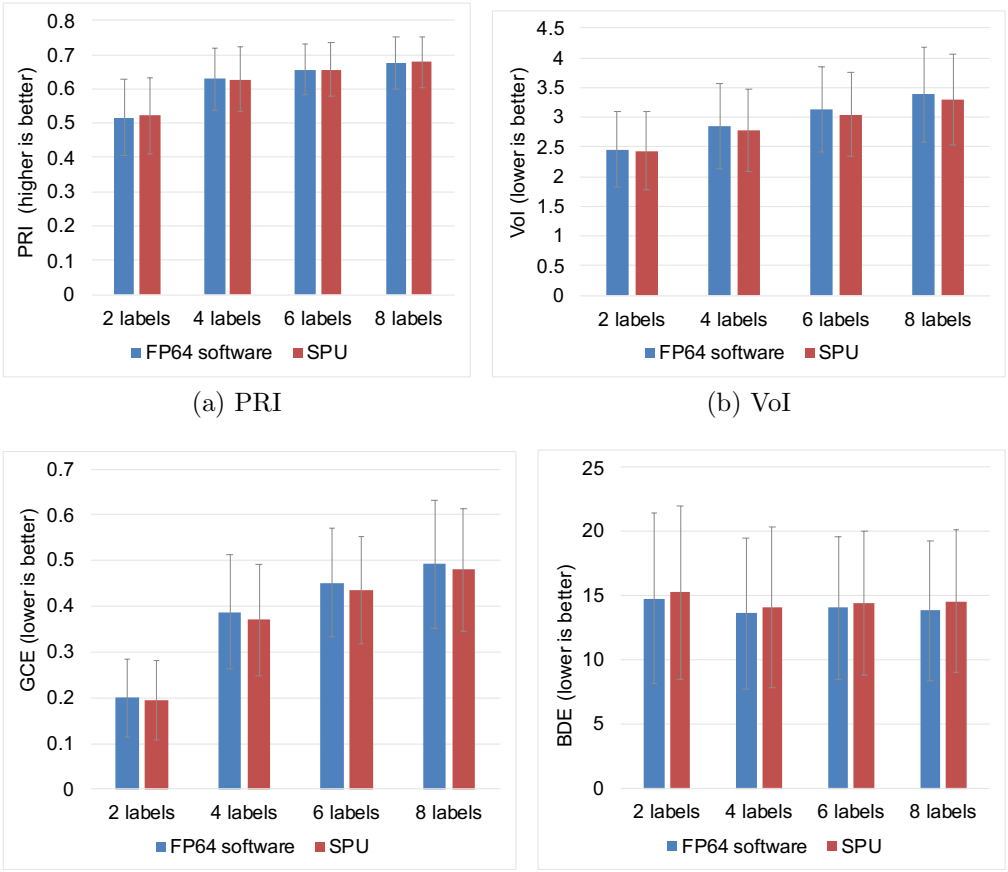
implementation is verified in an FPGA prototype. This section presents SPU result quality, FPGA synthesis results, and ASIC synthesis results.

4.3.1 Result Quality

We use QuestaSim simulation to evaluate the end-point result quality of Verilog SPU implementation. The applications and datasets are the same as those assessed in Chapter 3: image segmentation (30 images), motion estimation (3 datasets), and stereo vision (3 datasets). The FP64 software baseline is implemented in MATLAB. Figure 4.8 shows the result quality comparison between the FP64 software and the SPU for stereo vision and motion estimation. Figure 4.9 shows the mean and standard deviation of 30 image segmentation results in four quality metrics. Each result is collected by a single run per dataset in optimization mode (simulated annealing). We validate that the SPU with a simple 19-bit LFSR as its RNG achieves the same result quality as the FP64 software. We also obtain similar high-quality application results on an Intel Arria 10 FPGA prototype.

4.3.2 FPGA

We evaluate four different implementations of the SPU on an Intel Arria 10 FPGA: 1) a hand-written Verilog for ASIC (*verilog-asic*), 2) a hand-written Verilog optimized



(a) PRI (b) VoI

(c) GCE (d) BDE

FIGURE 4.9: Image segmentation result quality

for FPGA (*verilog-fpga*), 3) a High-Level Synthesis (HLS) implementation (*hls-int*) that matches the hand-written Verilog (but using HLS basic integer data-types), and 4) an HLS implementation with a 32-bit floating-point (FP32) back-end after energy computation (*hls-fp*), eliminating the energy scaling stage. *Hls-fp* is developed in order to assess the option of directly using FP32 representation inside the SPU for probability conversion and sampling. Table 4.1 shows the synthesis results. As discussed previously, *verilog-fpga* increases Fmax by adding an internal stage and reduces DSP usage by packing computations. *Hls-int* is close to *verilog-fpga* in terms of performance (frequency and initiation interval), but consumes more resources. The resource usage of *hls-int* can be further decreased by using reduced-precision

Table 4.1: Resource usage and performance of various SPU implementations on Arria 10 FPGA

Parameter	**Verilog-asic**	**Verilog-fpga**	**Hls-int**	**Hls-fp**
Frequency (MHz)	321	374	369	320
ALMs	313	321	1,189	4,407
Registers	620	680	2,551	7,932
Memory Bits	1,472	1,472	10,688	49,536
DSPs	6	4	10	25
Initiation Interval (Cycles)	1	1	1	3

Table 4.2: SPU area and power consumption

Component	Area(μm^2)	Power (mW)
Circuitry	1428	1.72
Memory	529	0.45
SPU Total	1957	2.17

integers. *Hls-fp* consumes 13.7× ALMs, 33.7× memory, 6.3× DSPs compared to *verilog-fpga* and most importantly performs remarkably worse due to its lower frequency and throughput (initiation interval) caused by the FP addition [54]. Clearly, naively implementing the SPU in FP32 consumes too many resources and significantly reduces the performance benefits. A human-designed architecture is needed to improve efficiency.

4.3.3 ASIC

We estimate the ASIC area/power for SPU implemented in Chisel. Compared with Verilog, Chisel provides easier access to design space exploration used for Chapter 6. Circuitry elements written in Chisel are synthesized in a predictive 15nm library [92] using Synopsys Design Compiler. Memory elements, including a 64×8-bit energy FIFO, a 64×9-bit CDF FIFO, and a 64×6-bit label LUT, are estimated using Cacti 7 [8] in 22nm technology, the smallest supported technology. Table 4.2 summarizes the results. Total area/power is the sum of 15nm circuitry and 22nm memory elements. Power is estimated at 1GHz. Since Cacti requires widths in multiples of bytes,

we estimate a 64-byte LUT and scale it to the label LUT ($97.5 \times 0.75\ \mu m^2$ and $0.8 \times 0.75\ mW$). Similarly, the CDF FIFO is estimated from a 64-byte FIFO ($215 \times 1.125\ \mu m^2$ and $0.18 \times 1.125\ mW$). The SPU saves 33% in area and 57% in power compared with an RSU-G without light source sharing (cf. Table 3.5). RSU-G requires a timing detection clock running 8× faster in RET circuits and notable power saving is observed in the SPU without those circuits. The SPU can run up to 3.3GHz, bounded by the SPU energy computation stage. Importantly, the SPU keeps the same architectural interface as RSU-G and thus maintains sizable performance benefits from RSU-G: 21-84× speedups as a discrete accelerator over a Titan X GPU [142].

4.4 Limitations and Future Work

Using CMOS technology enables the potential to address several limitations in the RSU-G, including 1) the maximum number of possible labels is limited to 64 per random variable; 2) the supported solver is limited to Gibbs Sampling, with the support of simulated annealing; 3) graphical model is limited to a 1st-order MRF. Addressing these limitations is our future work, which leads to a more generalized stochastic processing unit in a wider scope of probabilistic computing. Our work evaluates RNGs vs. application result quality using an empirical approach, as previous work does in the field of Monte Carlo simulation [23, 24, 126]. Related work is discussed in Chapter 7. An analytical approach is ideal, but difficult to our knowledge given the complexity of MCMC methods. The effect of a 19-bit LFSR on other applications remains to be assessed. Finally, like most previous work in hardware accelerators, our result quality analysis in this chapter focuses only on end-point result quality using application community-standard benchmarks and metrics, which omits other important statistical properties in probabilistic computing. The next chapter proposes a comprehensive methodology to address those statistical properties.

4.5 Summary

In this chapter, we evaluate the feasibility of a CMOS sampling unit to accelerate 1st-Order MRF Gibbs Sampling using a pseudo RNG. We empirically explore the relationship between six RNGs with different theoretical quality and end-point result quality in 1st-Order MRF applications. We unexpectedly found a simple 19-bit LFSR is sufficient to provide empirically good result quality in the tested applications. Using more complicated RNGs does not further improve the result quality. We propose a CMOS Stochastic Processing Unit (SPU) functionally equivalent to the RSU-G by replacing the RET-based sampler with a CMOS discrete sampler. The design produces the same end-point result quality as FP64 software in three assessed applications. The simple 19-bit LFSR avoids area/power overhead of a complex RNG and saves 33% in area and 57% in power, compared with RSU-G. The CMOS SPU design provides a starting point to explore a generalized architecture in a wider scope of probabilistic computing.

In the next chapter, we re-examine the methodology used in this chapter, as used in many previous works, for evaluating quality of a probabilistic accelerator. We expose limitations on the current methodology focusing only on end-point result quality and propose a new framework to address statistical robustness.

5

Statistical Robustness

The previous chapters present two MCMC acceleration units for 1st-Order MRF Gibbs Sampling. Many specialized accelerators, including RSU-G and SPU, are proposed to address the sampling inefficiency of probabilistic algorithms [9,16,67,79, 84,87,100], by utilizing approximation techniques to improve the hardware efficiency, such as reducing bit representation, truncating small values to zero, or simplifying the random number generator. Understanding the influence of these approximations on the application results is crucial to meet the quality requirement in real applications. A hardware accelerator should provide correct execution of target algorithms.

A common approach to evaluating correctness is to compare the end-point result quality ("accuracy") against accurately-measured or hand-labeled ground-truth data using community-standard benchmarks and metrics: the hardware execution is considered to be correct if it provides comparable "accuracy" to the software-only implementations that do not have these approximations. However, in the domain of probabilistic computing/algorithms, correctness is defined by more than the end-point result of executing the algorithm, and includes additional statistical properties that convey uncertainty and interpretability about the end-point result. End-point

"accuracy" is necessary but not sufficient to claim correctness: 1) given the uncertainty of input data, the observed end-point result quality has no indication of "accuracy" for unseen data, and thus just making statements on the observed "accuracy" is not enough; 2) many applications look beyond the end-point "accuracy" and consider uncertainty quantification; 3) measuring "accuracy" may not always be possible as ground-truth data is not always accessible. Current methodologies for evaluating probabilistic accelerators are often incomplete or adhoc in evaluating correctness, focusing only on end-point "accuracy" or limited statistical properties. Failure to adequately account for domain-defined correctness can have adverse or catastrophic outcomes, such as a surgeon failing to completely remove a tumor due to incorrect uncertainty in a segmented image [22, 97]. Therefore, *a probabilistic architecture should provide some measure (or guarantee) of statistical robustness.*

This chapter takes a first step toward defining metrics and a methodology for quantitatively evaluating correctness of probabilistic accelerators beyond end-point result quality. We propose three pillars of statistical robustness: 1) *sampling quality*, 2) *convergence diagnostic*, and 3) *goodness of fit*. Each pillar has at least one quantitative empirical metric, does not require ground-truth data, and collectively these pillars enable comparison of specialized hardware to 64-bit floating-point (FP64) software implementation. We expose several challenges with naively applying existing popular metrics for our purposes, including: high dimensionality of the target applications, and random variables with zero variance. Therefore, we modify the existing methodologies for sampling quality and convergence diagnostic, and propose a new metric for convergence diagnostic. Below is a summary of each pillar.

Pillar I) Sampling Quality. The intrinsic nature of MCMC methods creates dependency between samples. A sufficient number of independent samples are needed to converge and produce high-quality results. We use *Effective Sample Size* (ESS) [79, 132] to measure the number of independent samples drawn from an MCMC

run, and report the arithmetic mean as a scalar metric. The existing method does not consider a practically possible case that a random variable empirically produces zero variance. We modified the method to report "overall" and "active" ESS values separately to account for possible biases. Low ESS indicates that more iterations may be required to generate sufficient independent samples.

Pillar II) Convergence Diagnostic. The total running time of an MCMC run is determined by when it converges. Convergence can be measured by Gelman-Rubin's $\hat{R}$ [15], but this metric is undefined for variables with zero variance. Therefore, we propose a process to determine convergence that accounts for zero variance and a new metric—*convergence percentage*—based on $\hat{R}$, to measure the total percentage of converged results. Low convergence percentage indicates that more iterations are required for the model to converge.

Pillar III) Goodness of Fit. In the absence of ground-truth data (labeled data), it is important to understand the differences between the baseline FP64 and hardware end-point results to evaluate the overall quality of hardware. We provide two "goodness of fit" approaches: 1) Root Mean Squared Error (RMSE) on application specific data relative to a software reference, and 2) Jensen-Shannon Divergence (JSD) [73] to evaluate all possible data inputs in the binary label case and provide the worst-case distribution divergence.

The three pillars can inform end-users by characterizing existing hardware and inform hardware designers by using the pillars in design space exploration. This chapter presents the first scenario in a case study. We demonstrate the framework in a representative probabilistic accelerator—the SPU proposed in Chapter 4—and show that 1) end-point result quality alone is insufficient, particularly for predicting outcome for previously unseen inputs; 2) FP64 is insufficient as ground-truth since in some cases more limited precision can produce more accurate end-point results based on labeled data; and 3) the accelerator achieves the same application end-

point result quality as the FP64 software, confirming the previous chapter, but has compromised ESS and convergence percentage results. The analysis reveals that applications need to run 2× more iterations on the accelerator to achieve the same statistical robustness as FP64, reducing the accelerator's effective speedup. The next chapter demonstrates how the three pillars can be used for architectural design space exploration, using the SPU as a case study.

The remainder of this chapter is organized as follows. Section 5.1 introduces three pillars of statistical robustness and their insights. Section 5.2 describes the analysis of statistical robustness on existing probabilistic hardware. Section 5.3 discusses limitations and future work and Section 5.4 summarizes this chapter.

5.1 Three Pillars of Statistical Robustness

To identify appropriate measures of hardware statistical robustness, we draw on known techniques utilized by domain experts to evaluate their models and algorithms. Ideally, we could formally prove bounds on relevant metrics [33, 56]. Unfortunately, some hardware optimizations (e.g., truncation to zero) make formal proofs extremely difficult or impossible. A provable architecture introduces more complicated hardware. Therefore, we rely on existing empirical diagnostic tests for MCMC techniques, based on foundations in statistics, to establish three pillars for assessing a probabilistic accelerator's statistical robustness: 1) sampling quality [132], 2) convergence diagnostic [26], and 3) goodness of fit. Each pillar has at least one quantitative measure, and provides insight to application users and to hardware designers. Collectively these pillars help in understanding/explaining end-point results, and can indicate the performance of the MCMC execution, such as how many iterations are required to converge. Note that the statistical robustness is jointly affected by the algorithm and hardware architecture. Therefore, we compare hardware results with an FP64 software as the baseline to extract the impact of hardware optimizations. The

remainder of this section presents our proposed three pillars for evaluating statistical robustness of an MCMC accelerator.

5.1.1 Pillar 1: Sampling Quality

A sampling algorithm with perfect sampling quality generates independent samples. However, an MCMC sample is drawn based on the current values of random variables—the outcome of samples in the previous iteration. This dependency creates correlations between samples which is non-trivial until several subsequent samples are drawn, which can be represented as an autocorrelation time τ. This implies that by generating n samples from MCMC, only n/τ samples can be considered independent. A sufficient number of independent samples are required to derive meaningful statistical measures (e.g., mean and variance). Note that the sample dependency is an intrinsic property of MCMC algorithms and exists even with a perfect random number generator and FP64 precision.

Effective Sample Size (ESS) is commonly used as a sampling quality metric that represents how many independent samples are drawn among all the dependent samples. In general, higher ESS indicates the MCMC sampler is more efficient at generating independent samples. Unfortunately, there is no consensus on a single ESS definition [40]. We use the definition discussed by Kass et al. [61] based on autocorrelation. Since closed form expressions for ESS are difficult, we estimate ESS using the known initial positive sequence (IPS) methods [79, 132].

$$ESS = n/(1 + 2\sum_{k=1}^{K} \rho(k)) \tag{5.1}$$

We estimate ESS on a univariate random variable using Equation 5.1, where n is the number of dependent MCMC samples (iterations) and $\rho(k)$ is the autocorrelation function of the sample sequence. We sum up the first K contiguous lags where

$\rho(k) + \rho(k+1) \geqslant 0$. Theoretically, $ESS = n$ provides the best sampling quality where all samples are independent; however, Equation 5.1 is an estimate of ESS, and thus it is numerically possible that $ESS > n$.

The above ESS method cannot be directly applied to our evaluation for two reasons. First, many MCMC problems are high-dimensional (many random variables). For example, in stereo vision a 320×320 input image has 102,400 dimensions. The above ESS does not account for multidimensional problems. Furthermore, the above ESS has no definition when all collected samples have the same value (zero empirical variance), which is possible in practice as shown in Section 5.2. An ideal metric can report a scalar ESS value to account for both issues. While methods exist to report multivariate ESS [137], to our knowledge they are not practical in our case and they do not allow zero variance for any variable.

To address multi-dimensionality, we consider each dimension (each pixel in our applications) as a separate random variable (RV) to compute ESS per dimension separately and report a scalar value of mean ESS among all dimensions. To further address zero variance, we propose two metrics: 1) mean *"overall" ESS* that omits the random variables with zero variance in software and hardware implementations, respectively; and 2) mean *"active" ESS*, a paired metric that only includes the random variables with non-zero variance in both software and hardware. Section 5.2 suggests overall ESS is biased toward software due to small but non-zero variance. Active ESS omits small variance in software, which can potentially benefit hardware implementations. Algorithm 5.1 and Algorithm 5.2 outlines the procedure of computing the two metrics.

Pillar Insight. If ESS is low it may take more MCMC iterations to achieve an acceptable ESS. If a hardware accelerator produces substantially lower ESS than software, the additional iterations may reduce its effective speedup.

Algorithm 5.1: Overall ESS

Input: trace of multidimensional samples $\boldsymbol{X}$ from a MCMC run, either in software or hardware implementations.

sum_ESS = 0, num_valid_rvs = 0

for *x (trace of each univariable RV) in* $\boldsymbol{X}$ **do**

  if *variance(x)* $\neq$ *0* **then**

    sum_ESS += ESS(x)

    num_valid_rvs++

  end

end

Output: overall_ESS = sum_ESS/num_valid_rvs

Algorithm 5.2: Active ESS

Input: trace of multidimensional samples $\boldsymbol{X_{sw}}$ from a MCMC run in software, and $\boldsymbol{X_{hw}}$ in hardware implementations.

sum_ESS_sw = 0, sum_ESS_hw = 0

num_valid_rvs = 0

for x_{sw} *(trace of each univariable RV in software) in* $\boldsymbol{X_{sw}}$ *and* x_{hw} *(trace of corresponding RV in hardware) in* $\boldsymbol{X_{hw}}$ **do**

  if *variance(*x_{sw}*)* $\neq$ *0 and variance(*x_{hw}*)* $\neq$ *0* **then**

    sum_ESS_sw += ESS(x_{sw})

    sum_ESS_hw += ESS(x_{hw})

    num_valid_rvs++

  end

end

active_ESS_sw = sum_ESS_sw/num_valid_rvs

active_ESS_hw = sum_ESS_hw/num_valid_rvs

Output: active_ESS_sw, active_ESS_hw

5.1.2 Pillar 2: Convergence Diagnostic

An important question for an MCMC method is when to stop iterating, determined by when the MCMC is converged. Similar to ESS, the time to convergence is used to analyze algorithms and input data when using software even with FP64 and good random number generators. Multiple methods exist to measure the convergence. A comprehensive review is provided by Cowles et al. [26]. We use Gelman-Rubin's $\hat{R}$ [15], a popular quantitative method provided by many statistical packages, to measure whether a univariate random variable (e.g., a pixel in stereo vision) is converged

at a certain iteration.

Gelman-Rubin's $\hat{R}$ (potential scale reduction factor) estimates the convergence by comparing the between-chain variance (B) and within-chain variance (W) across multiple independent runs on the same MCMC instance[1]. Equations 5.2 to 5.5 show the computation to obtain an $\hat{R}$ given the sample trace x from m independent MCMC runs, each with n samples. σ_+^2 is an overestimate on the variance of a random variable. As a rule of thumb [15,34], a univariate random variable is considered converged when $\hat{R} < 1.1$. Typically larger $\hat{R}$ indicates that more iterations are needed to converge. Note that the $\hat{R}$ method requires a random value initialized from an overdispersed distribution. We meet this requirement by initializing random variables (i.e., initial labels of pixels) uniform-randomly.

$$B/n = \frac{1}{m-1}\sum_{j=1}^{m}(\bar{x}_{j\cdot} - \bar{x}_{\cdot\cdot})^2 \tag{5.2}$$

$$W = \frac{1}{m(n-1)}\sum_{j=1}^{m}\sum_{t=1}^{n}(x_{jt} - \bar{x}_{j\cdot})^2 \tag{5.3}$$

$$\hat{\sigma}_+^2 = (n-1)/n \times W + B/n \tag{5.4}$$

$$\hat{R}^2 = \frac{m+1}{m}\frac{\hat{\sigma}_+^2}{W} - \frac{n-1}{mn} \tag{5.5}$$

A scalar convergence diagnostic is preferred for multi-dimensional applications. Similar to ESS, handling high dimensions and the random variables with zero empirical variance ($W = 0$) is challenging using existing methods [15,138]. The original Gelman-Rubin's $\hat{R}$ metric has no definition at $W = 0$. Considering each dimension as a separate random variable (RV), we propose an extended procedure (shown in Figure 5.1) to consider a random variable converged when $B = 0$ and $W = 0$, which indicates all samples are the same value from different iterations and MCMC runs. A

[1] Instance refers to the same input data, model and configuration parameters.

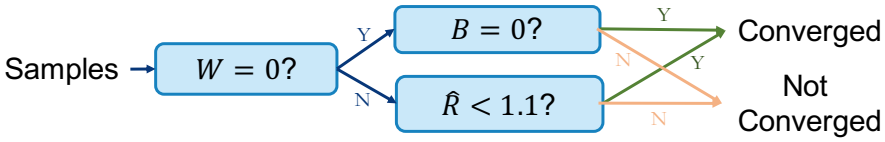


FIGURE 5.1: Determine convergence of a random variable

random variable is not considered converged when $B > 0$ and $W = 0$, which indicates samples are the same value within MCMC runs, but different across MCMC runs. We propose *convergence percentage*, the percentage of converged univariate random variables, as our new metric. Algorithm 5.3 outlines the procedure of computing convergence percentage.

Algorithm 5.3: Convergence percentage

Input: trace of multidimensional samples $\boldsymbol{X}$ from m MCMC runs, either in FP64 software or hardware implementations.

num_converged_rvs = 0, num_rvs = 0

for *x (trace of each univariable RV) in* $\boldsymbol{X}$ **do**

  if *($\hat{R}(x)<1.1$) or ($B = 0$ and $W = 0$)* **then**

    num_converged_rvs++

  end

  num_rvs++

end

convergence_percentage = num_converged_rvs/num_rvs*100 (in %)

Output: convergence_percentage

Pillar Insight. Low convergence percentage indicates that more iterations are needed for the model to converge. If a hardware accelerator takes substantially more iterations to converge than the software, the additional iterations may reduce its effective speedup.

5.1.3 Pillar 3: Goodness of Fit

Finally, understanding the "goodness of fit"—the difference between end-point results produced by the software and by the hardware accelerator—is critical to evaluating the overall quality of the hardware accelerator. A straightforward approach

is to compare the end-point result quality using community-standard benchmarks and metrics. However, ground-truth data are not always available. We provide two "goodness of fit" approaches: 1) using application specific data to measure how good the hardware results fit a reference software result, and 2) using a distribution divergence measurement to evaluate all possible data inputs and provide the worst-case divergence.

Application Data Analysis

We are interested in how close/different the results are between the software and hardware. Popular quantitative metrics for "goodness of fit" include Root Mean Squared Error (RMSE) and coefficient of determination (R^2). We choose RMSE given the value of R^2 can be misleading by the small variance of the software results. RMSE measures the root of average squared difference between the result from a hardware MCMC run and a reference software run, ranging from 0 to infinity where lower is better. Due to the stochastic nature of MCMC methods, each MCMC run can have different end-point results for either software or hardware. To account for this variation, we compute RMSE for both hardware and software with respect to a reference software result from multiple MCMC runs. The reference software result is obtained using the mode of multiple software runs to minimize the result variation in a single software reference run.

Data-independent Analysis

Recall the step-1 of sampling is computing the probability distribution to sample from. Hardware approximations in this step introduce divergence from the distribution obtained from FP64 software. Quantifying the distribution divergence of hardware from software provides 1) insights on why the results are good (or bad), 2) how the hardware may perform on unobserved data, and 3) the worst-case divergence

in arbitrary data inputs.

One popular divergence measurement is Kullback-Leibler (KL) divergence. Given the same input data, model, configuration parameters, and states of neighbors, the distribution of a given random variable is computed as P_{sw} from FP64 software and P_{hw} from a hardware implementation. KL divergence (D_{KL}) from P_{hw} to P_{sw} is defined in Equation 5.6. χ is a collection of all possible outcomes of the random variable and i is a possible outcome.

$$D_{KL}(P_{sw}||P_{hw}) = \sum_{i \in \chi} p_{sw}(i) \log \frac{p_{sw}(i)}{p_{hw}(i)} \tag{5.6}$$

One major drawback of KL-divergence is it goes to infinity when any entry of $P_{hw}(i)$ is zero while $P_{sw}(i)$ is non-zero, which is likely to happen under the hardware technique of truncating small probabilities to zero, and thus cannot be directly applied to our study. Therefore, we choose Jensen-Shannon Divergence (JSD) as our divergence measurement [73], shown in Equation 5.7. JSD is defined based on KL-devergence, where $M = (1/2)P_{sw} + (1/2)P_{hw}$. Note that KL-divergence is asymmetric: $D_{KL}(P_{sw}||P_{hw}) \neq D_{KL}(P_{hw}||P_{sw})$, but JSD is symmetric: $D_{JS}(P_{sw}||P_{hw}) = D_{JS}(P_{hw}||P_{sw})$. A lower JSD is preferable, showing distributions of a random variable computed from FP64 software and hardware implementations are close.

$$D_{JS}(P_{sw}||P_{hw}) = \frac{1}{2} D_{KL}(P_{sw}||M) + \frac{1}{2} D_{KL}(P_{hw}||M) \tag{5.7}$$

Evaluating JSD on arbitrary data inputs for a random variable with many possible labels, such as in stereo vision, is challenging in both analytical and empirical approaches given the complicated mathematical representation and the large parameter space. In this work, we evaluate the JSD in binary label cases, such as in a foreground-background image segmentation.

Pillar Insight. Substantially worse RMSE or JSD results for a hardware accelerator means it is likely producing low quality application end-point results and more iterations or model/hardware design changes may be required.

5.2 Analyzing Existing Hardware

We apply the three pillars of statistical robustness on an existing MCMC hardware, the Stochastic Processing Unit (SPU) described in Chapter 4. Recall the SPU uses approximation techniques including full-custom precision, truncating small scaled probability to zero, 2^n approximation, and a simple 19-bit LFSR as RNG. The SPU achieves the same result quality as FP64 software in three applications. However, we are left with the question: *How do these approximations influence the SPU statistical robustness?* This section answers this question.

5.2.1 Methodology

In this work, we consider a single SPU as it is sufficient to explore the statistical robustness questions. We primarily utilize MATLAB for both the FP64 software and for a functionally equivalent SPU simulator. We choose stereo vision and motion estimation as our test applications. Each disparity per pixel in stereo vision is treated as a random variable. Each 2D motion vector per pixel in motion estimation is considered as two random variables x and y. We pick the same three datasets from Middlebury [7, 125] for each application as the previous chapters. We use FP64 runs to find the application parameters (e.g., α and β). Motion estimation has one set of parameters for all datasets, and stereo vision has two sets for all datasets. Some parameters can be optimized in a training process, which is beyond the scope of this work. We also considered, but omit, image segmentation since it converges too fast (30 iterations for simulated annealing) to produce meaningful statistical measurements.

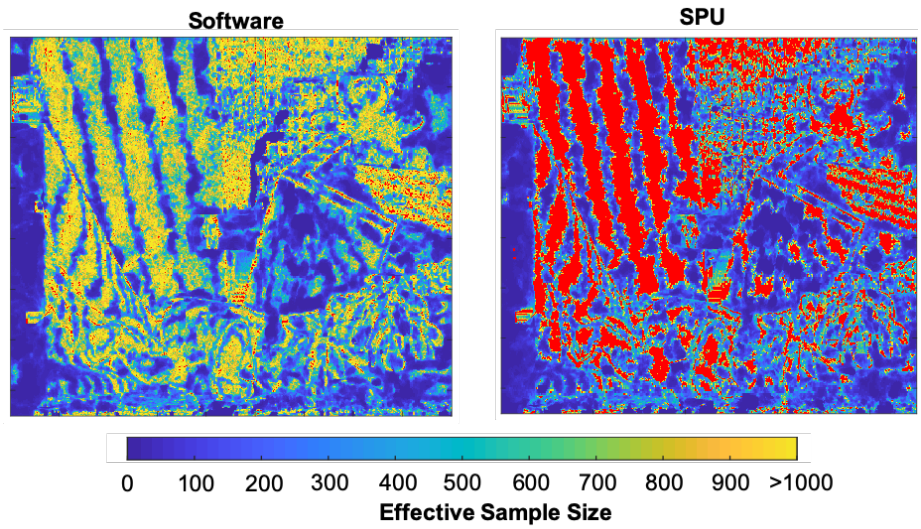


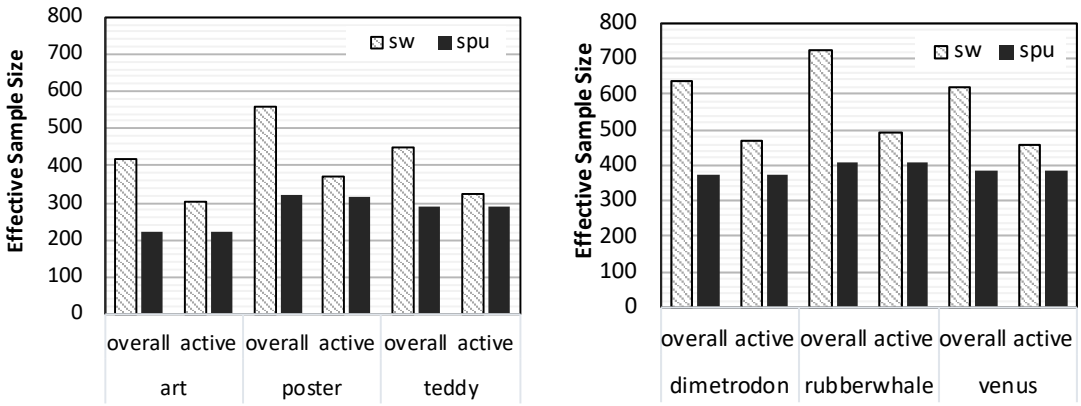
FIGURE 5.2: ESS per random variable in stereo vision *teddy*. Red regions correspond to zero variance.

Recall the SPU supports two operating modes: pure sampling that produces the full estimated distribution (sampling), and optimization using simulated-annealing (optimization) that converges quickly to an exact result. For optimization, measuring Effective Sample Size (ESS) and Gelman Rubin's $\hat{R}$ is not conceptually meaningful, we evaluate sampling quality and convergence diagnostic for sampling only and goodness of fit for both modes. Parameter settings for each dataset are the same in sampling as in optimization, except for a different, fixed temperature. Our empirical results show that all datasets converge after 1,000 iterations for optimization and 3,000 for sampling, except for *poster* in stereo vision which takes only 500 and 1,500 iterations, respectively.

5.2.2 Results Analysis

Sampling Quality

We analyze ESS on SPU compared with the FP64 software by collecting the last 1,000 iterations of MCMC runs in the two applications. We evaluate the ESS per random variable and report the arithmetic mean. Figure 5.2 shows an example

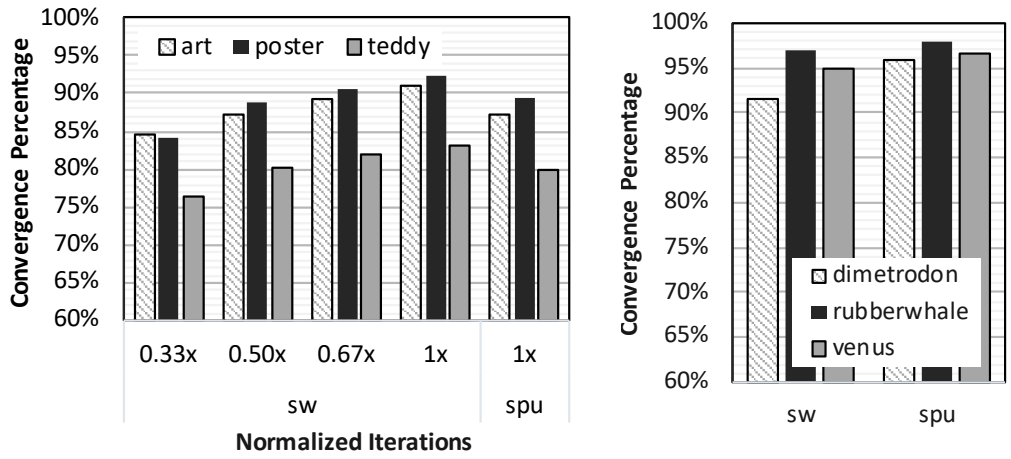


(a) Stereo vision (b) Motion estimation

FIGURE 5.3: Mean overall and active ESS (higher is better)

ESS per random variable in stereo vision *teddy* dataset. Red regions indicate the random variables with zero variance, and thus ESS cannot be calculated. Due to truncating small probabilities to zero, more random variables in the SPU have zero variance than in the software. We consider a random variable with zero variance inactive. The percentage of inactive random variables with respect to the total (a.k.a. inactive percentage) in three stereo vision datasets are 26.9% for *art*, 44.6% for *poster*, and 26.2% for *teddy* in the SPU, compared with 0.3% for *art*, 4.1% for *poster*, and 1.4% for *teddy* in the FP64 software. Motion estimation exhibits similar inactive percentages. Zero variance means the probability of a possible label is large enough that all random samples pick the same label empirically, which can indicate convergence. The variance of corresponding inactive random variables in the FP64 software is consistently small, indicating the random variable is likely to consistently pick the same label as well—a concentrated distribution. Therefore, a high inactive percentage does not necessarily imply bad result quality.

Figure 5.3 shows the ESS arithmetic mean for a single MCMC run per dataset. We verify that different runs have a small ESS difference (< 6 in stereo vision). The mean *"overall" ESS* eliminates the random variables with zero variance in the software and hardware, respectively. Figure 5.2 reveals that the inactive regions in



(a) Stereo vision (b) Motion estimation

FIGURE 5.4: Convergence percentage (higher is better) results

the SPU (red) correspond to the regions with high ESS in the FP64 software due to small but non-zero variance (yellow), and thus overall ESS is biased toward the software. Therefore, we also report the mean *"active" ESS* which only includes the regions with non-zero variance in both the software and the SPU, where ESS is more meaningful. As a consequence, the active ESS eliminates the regions with small variance in the software, which can potentially benefit the SPU. The importance of these small variance needs to be evaluated and we are actively looking for methods to account for these regions. The FP64 software has 1.1-1.4× higher active ESS than the SPU in stereo vision and around 1.2× in motion estimation. This implies *the SPU needs to run 1.1-1.4× iterations to reach the same active ESS as the software.*

Convergence Diagnostic

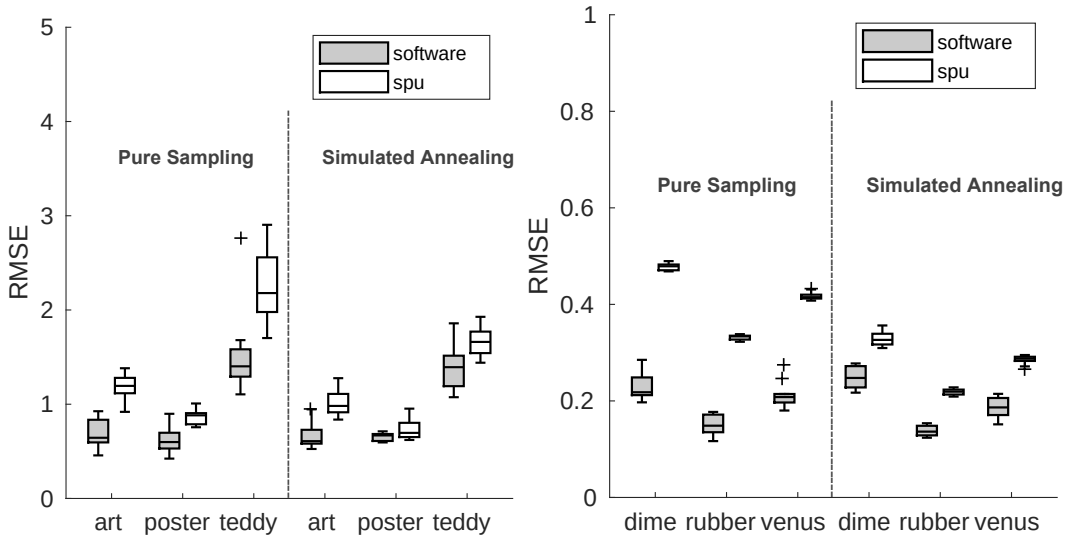
We evaluate the convergence diagnostic of SPU using the proposed convergence percentage metric. Each convergence percentage value is collected from 10 runs per dataset. Each run forfeits the first half of iterations as the burn-in period and only keeps the second half, as proposed by Gelman and Rubin [34]. Recall a random variable is considered converged if $\hat{R} < 1.1$ or both within-chain variance W and

between-chain variance B are zero. Figure 5.4 shows the results. The number of iterations is normalized with respect to SPU runs in stereo vision and are the same in motion estimation. Overall, convergence percentage is high in both the software and the SPU: more than 80% of random variables in stereo vision and more than 90% in motion estimation. More than 99.5% of random variables with $W = 0$ in the SPU are converged. In stereo vision, the SPU reaches the same or better convergence percentage than software with 2× iterations. This indicates *the SPU needs to be at least 2× faster in order to have a better overall performance in this application in terms of convergence percentage*. Previous work [142] as well as chapters 3 and 4 shows that RSU-G and the SPU provide the speedups of at least 2.8-5.5× and up to 84×. The SPU has higher convergence percentages than the FP64 software in motion estimation, indicating the SPU converges faster in this application. Note that converging to a distribution faster does not necessarily lead to a better end-point result. The goodness of fit should be evaluated.

Goodness of Fit

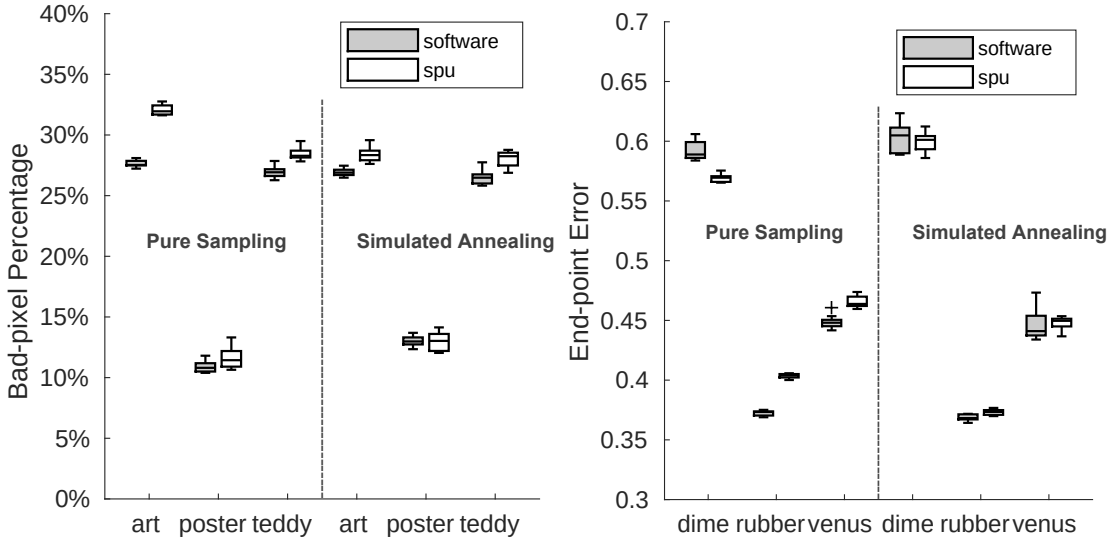
Figure 5.5 shows the RMSE box plots of 10 MCMC runs per dataset compared with a reference result obtained by the mode of 10 software runs per dataset. Solid boxes show the range from 25th to 75th percentile with the medians of data as the horizontal lines inside. The whiskers include the range of data that are not considered as outliers. We use 1.5× interquartile range as the rule to decide outliers, shown as pluses. Whiskers of the FP64 software and the SPU overlap in all stereo vision benchmarks, suggesting close results. RMSE results in motion estimation are visually different in Figure 5.5b. However, these differences are small considering the small scale of y-axis. The FP64 software and the SPU produce closer results in simulated annealing optimization mode.

Figure 5.6 shows the end-point result quality using ground-truth data and appli-



(a) Stereo vision (b) Motion estimation

FIGURE 5.5: Root Mean Squared Error (lower is better). Scales are different in (a) and (b) due to application differences.



(a) Stereo vision (b) Motion estimation

FIGURE 5.6: Application end-point result quality (lower is better)

cation metrics. Most whiskers of the FP64 software and the SPU overlap except for *art* in stereo vision and *rubberwhale* in motion estimation, both of which are in sampling mode. In optimization mode, software and SPU whiskers overlap, indicating the difference in end-point result quality is very small. This is consistent with the single-run results in Section 4.3. Note that no obvious differences between software and the SPU are visually observable in the stereo vision disparity maps and motion estimation flow maps.

It seems intuitive to assume that FP64 software should produce no worse results than hardware with limited precision, truncation, and a simplified RNG. We find this assumption holds in most but not all cases. We observe that in sampling mode of *dimetrodon*, the SPU has consistently lower end-point result error (Figure 5.6b) but higher RMSE (Figure 5.5b) than the FP64 software. To better understand this result, we examine per-pixel differences of end-point error between the software reference and the SPU, as shown in Figure 5.7. Blue regions correspond to lower end-point error in the SPU and yellow to lower end-point error in FP64 software. The figure suggests the FP64 software and the SPU have strengths in different regions, which potentially leads to a high RMSE compared to the software reference. This result indicates two insights: 1) software with higher precision does not necessarily produce better application end-point result quality, and 2) a higher RMSE compared to software does not always indicate worse application end-point result quality. Although bad pixel-percentage results are consistent with RMSE in stereo vision, the general link between the goodness of fit measure and the application end-point result quality needs to be further explored. This confirms *collectively applying all three pillars beyond end-point result is necessary to evaluate correctness.*

We analyze the Jensen-Shannon Divergence of the SPU relative to the software with FP64 probability representation. Our goal is to provide insights on why hardware exhibits good or bad application end-point results and how it may perform

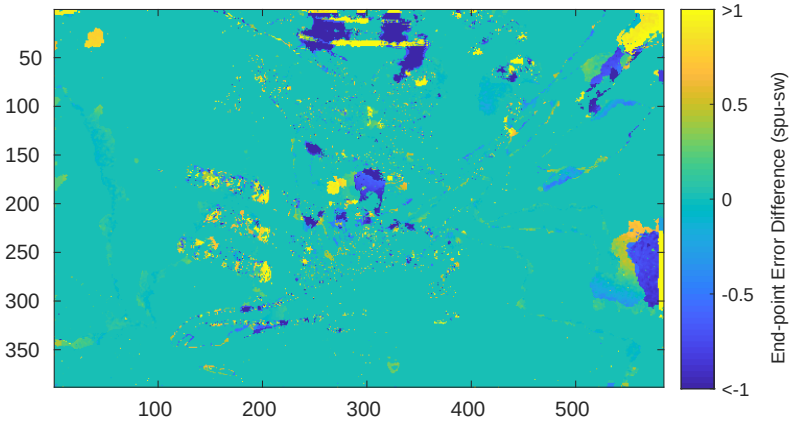


FIGURE 5.7: *Dimetrodon* end-point error difference ($spu - sw$) at pixel level. End-point error: 0.581 (software) vs. 0.567 (SPU).

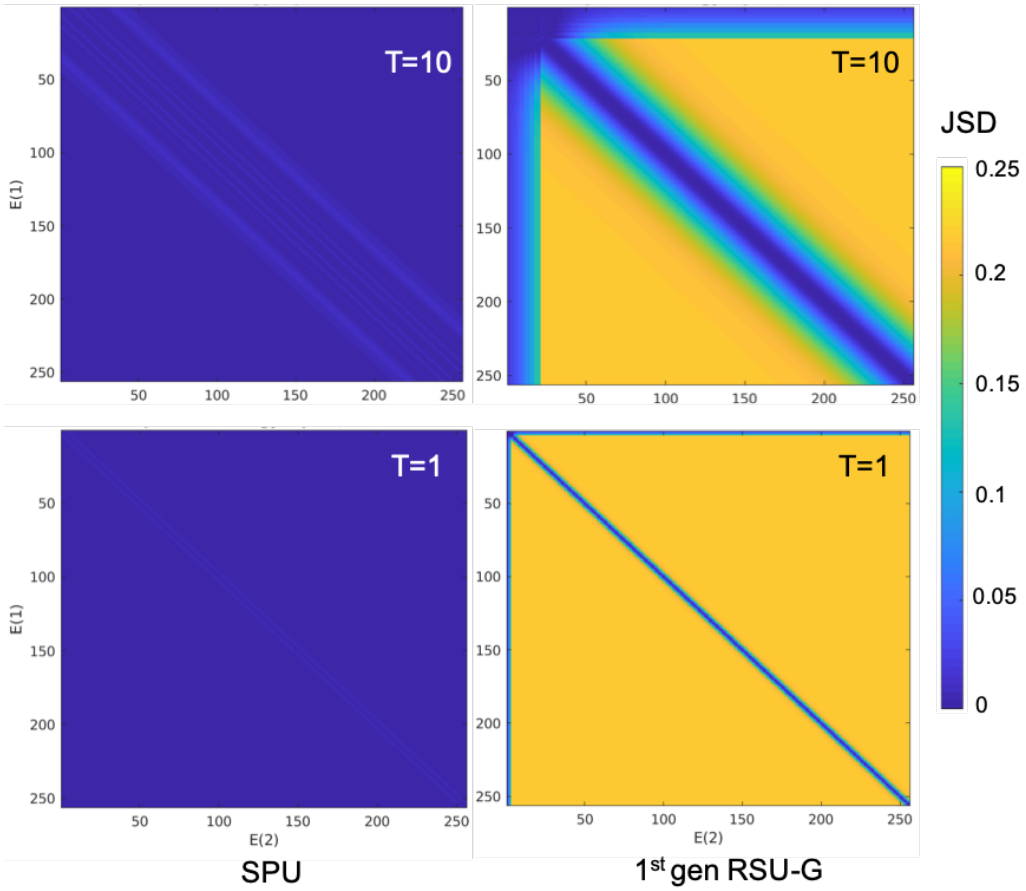


FIGURE 5.8: Jensen-Shannon Divergence comparison between designs: SPU vs. 1st-gen RSU-G [142]

with arbitrary input data. We assume each random variable has a binary distribution in this analysis. By sweeping a wide range of possible energy inputs $E(i)$ (refer to Equation 4.1) from 0 to 255 in integer, corresponding to arbitrary data inputs, Figure 5.8 plots JSD for two temperatures (1 and 10) and two different microarchitectures: 1) the SPU and 2) an early design [142]—1st-gen RSU-G—that was shown to lack sufficient precision and dynamic range as discussed in Chapter 3. These results clearly show the problems with the 1st-gen RSU-G. The more recent SPU has negligible JSDs in most energy inputs (blue regions), whereas the 1st-gen RSU-G has high JSD (>0.2, yellow) for many inputs and becomes worse when the temperature decreases, which explains the poor application result quality. A key difference between these two designs is dynamic scaling for energy values in the SPU, which is not present in the 1st-gen RSU-G.

5.3 Limitations and Future Work

Our proposed framework is an important starting point towards quantifying the statistical robustness of probabilistic accelerators. This work selects the most popular metrics and estimation approaches from many within each pillar. The analysis of other metrics and methods (e.g., MCMC standard error [32]) might help identify limitations of selected metrics. The challenges of naively applying existing metrics motivate us to propose modified processes and a new metric for reporting scalar measures for sampling quality and convergence diagnostic. Our proposals are conceptually straightforward, but could benefit from domain experts developing metrics with stronger theoretical foundations. Additionally, the adequateness of rule-of-thumb $\hat{R} < 1.1$ to determine convergence is under debate [138].

5.4 Summary

In probabilistic algorithms, statistical robustness is an important aspect of correctness defined by domain experts. Current methodologies often omit statistical robustness and thus lack a comprehensive definition of correctness. This chapter represents three pillars of statistical robustness: 1) sampling quality, 2) convergence diagnostic, and 3) goodness of fit. The framework, to our knowledge, is the first attempt at a comprehensive methodology for quantitatively evaluating correctness of probabilistic accelerators in considerations of both end-point result quality and statistical robustness. Previous work [41, 79, 87, 123] belongs to one of three proposed pillars and we argue all three pillars are needed to fully characterize statistical robustness of an MCMC accelerator. Related work is reviewed in Chapter 7.

The three pillars can inform end-users by characterizing existing hardware and inform hardware designers by guiding design space exploration. In this chapter, we apply our framework to a representative MCMC accelerator—the SPU—and surface design issues that cannot be exposed using only application end-point result quality. The SPU achieves the same application end-point result quality as the FP64 software, confirming the previous chapter, but has compromised ESS and convergence percentage that requires 2× more iterations on the SPU to achieve the same statistical robustness as FP64, reducing the SPU's effective speedup. The next chapter demonstrates the framework by using the pillars to guide design and overcome the above limitations.

6

Design Space Exploration with Statistical Robustness

Statistical robustness is an important characteristic of probabilistic computing and therefore should be considered during the hardware design process. The previous chapter shows that architectural optimizations might have a negative influence on the statistical robustness, even though they produce comparable end-point results as the FP64 software. A design space exploration is needed to find the design points in compliance with statistical robustness. The question is *can we achieve desirable end-point result quality and statistical robustness without the commensurate overhead of FP64?*

This chapter answers the above question by applying the proposed three pillars of 1) sampling quality, 2) convergence diagnostic, and 3) goodness of fit, to design space exploration. We use the SPU as a case study to expose the trade-offs between statistical robustness and area/power, with the following key results: 1) a simple 19-bit LFSR with a 12-bit RNG output does not reduce the statistical robustness or result quality across all design points; 2) considerable improvement in statistical robustness, comparable to the FP64 software, can be achieved by slightly increasing

the bit precision from 4 to 6 and removing 2^n approximation technique, with only 1.20× area and 1.10× power overhead. The expected 21-84× SPU speedups as a discrete accelerator compared with GPU [142] can therefore be achieved without additional iterations.

The remainder of this chapter is organized as the follows. Section 6.1 provides a case study on design space exploration in guidance of statistical robustness. Section 6.2 discusses limitations and future work. Section 6.3 summarizes the chapter.

6.1 A Case Study: SPU

We use the SPU as a case study to demonstrate design space exploration with statistical robustness. Recall the current SPU design produces good end-point result quality but compromised statistical robustness. The SPU pipeline (Figure 4.4) has several design parameters related to bit precision that potentially influence statistical robustness, including energy $E(i)$ and $E_s(i)$, scaled and truncated probability $p_{tr}(i)$, and RNG output bits. We fix energy $E(i)$ and $E_s(i)$ at 8 bits based on the previous work [87] and Chapter 3. The number of bits in $p_{tr}(i)$ considerably influences the size of the energy-to-probability converter and the discrete sampler. We evaluate three design points with 4-bit, 6-bit, and 8-bit $p_{tr}(i)$s. The influence of RNG output bits is small compared to $p_{tr}(i)$ and we confirm from our experiment that a 19-bit LFSR with 12-bit RNG outputs does not reduce the statistical robustness or result quality across all design points.

Recall the SPU truncates all $p_{tr}(i)$s to the nearest 2^n values, called 2^n approximation, enabling efficient energy-to-probability conversion by comparing the boundaries of energy values. Without 2^n approximation, a double-buffered 256-entry LUT is required to store the $p_{tr}(i)$ values to achieve a stall-free design. We evaluate the statistical robustness of each scaled probability design point with and without 2^n approximations. The above design parameters generally do not directly influence the

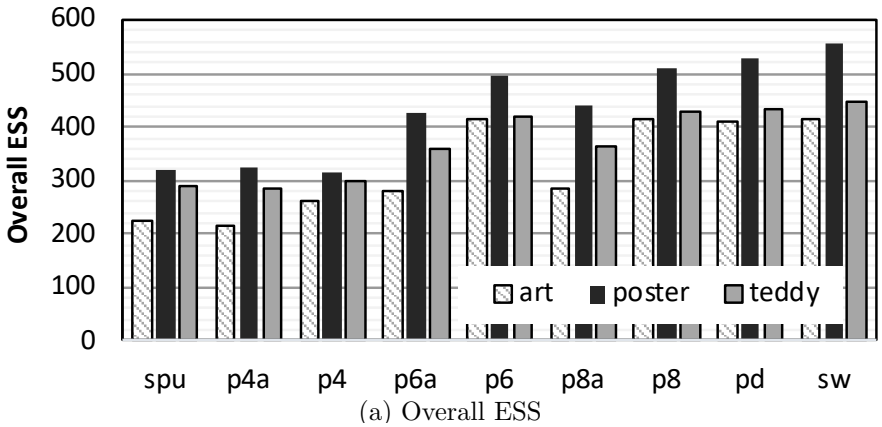
SPU per-iteration performance assuming the same interface at the same clock frequency. However, a design with lower precision may take more iterations to converge. On the other hand, higher precision requires more area and power affecting the number of SPU units in systems with limited area/power budget. Detailed system-level architecture investigations are beyond the scope of this dissertation.

6.1.1 Evaluating Design Parameters

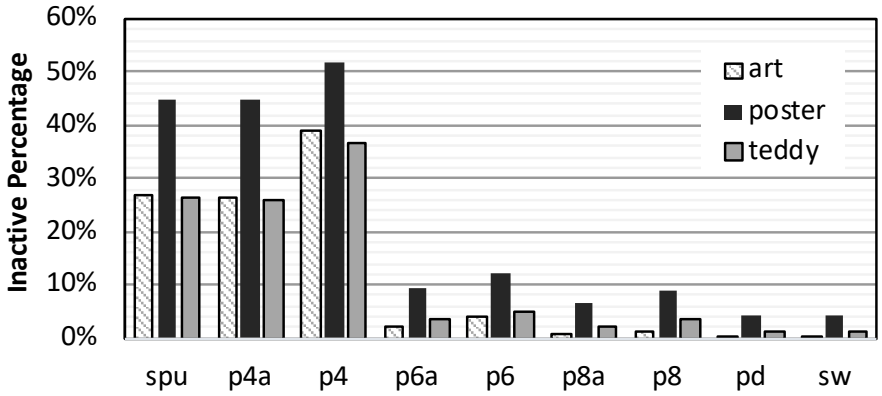
Figures 6.1-6.8 show our design space results. The results are obtained from the MATLAB SPU simulator discussed in Section 5.2. For brevity, we explain stereo vision results in detail and highlight motion estimation results where needed. Starting from the current SPU design ("spu"), we analyze the statistical robustness by gradually increasing the precision: 1) replace the 19-bit LFSR sampler with an FP64 Mersenne Twister sampler while keeping the front-end pipeline unchanged ("p4a"); 2) increase the bit width of $p_{tr}(i)$ to 6, 8-bit ("p6a" and "p8a"), with 2^n approximation; 3) remove 2^n approximation ("p4", "p6", and "p8"); and 4) keep front-end pipeline up to the scaled energy ($E_s(i)$) output unchanged, but has an FP64 back-end for probability conversion and discrete sampling ("pd").

Sampling Quality

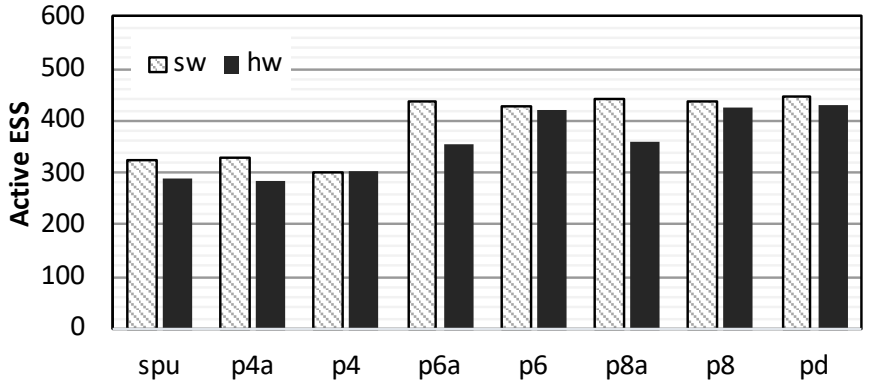
Figure 6.1a shows overall ESS for stereo vision, which omits random variables with zero variance for each design, respectively. Recall this metric can create biases that benefit the software for variables with small but non-zero variance. Overall ESSs increase when more bits are added, partly as a result of fewer random variables with zero variance. Recall the SPU truncates small scaled probabilities $p_{tr}(i) < 1$ to zero. Adding more bits keeps more possible labels with small probabilities available to be sampled. Figure 6.1b indicates inactive percentage drops significantly when increasing $p_{tr}(i)$ bit size from 4 to 6. Interestingly, 2^n approximation helps reduce



(a) Overall ESS



(b) Percentage of inactive random variables (inactive percentage).



(c) Active ESS in *teddy*

FIGURE 6.1: Stereo vision sampling quality in the design points

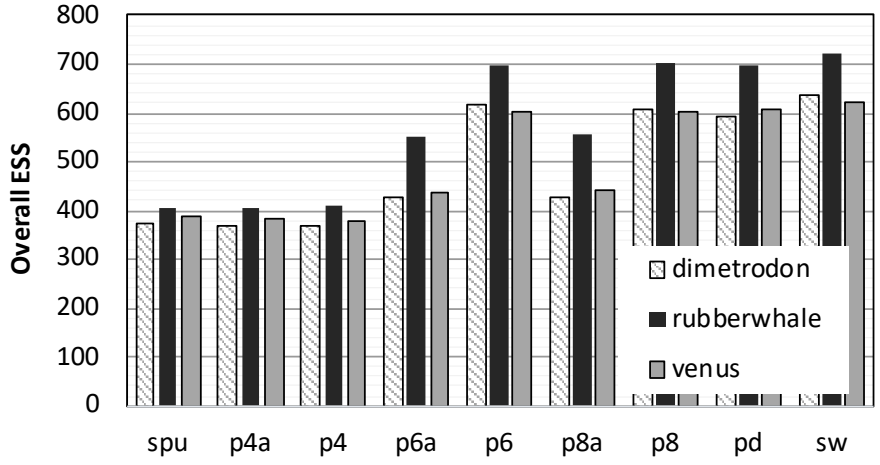
the inactive percentage under the same bit precision, but decreases ESS for 6-bit and 8-bit designs. Figure 6.1c shows the active ESS for the *teddy* dataset. Recall active ESS masks out the random variables inactive in either the software or SPU. With 2^n approximation, increasing bit precision does not close the gap in active ESS with the software. Without 2^n approximation, 6 or 8-bit $p_{tr}(i)$ have comparable overall and active ESS to software. As expected, increasing bit precision decreases the difference between overall and active ESS due to fewer inactive variables. Similar results for motion estimation are shown in Figure 6.2.

Convergence Diagnostic

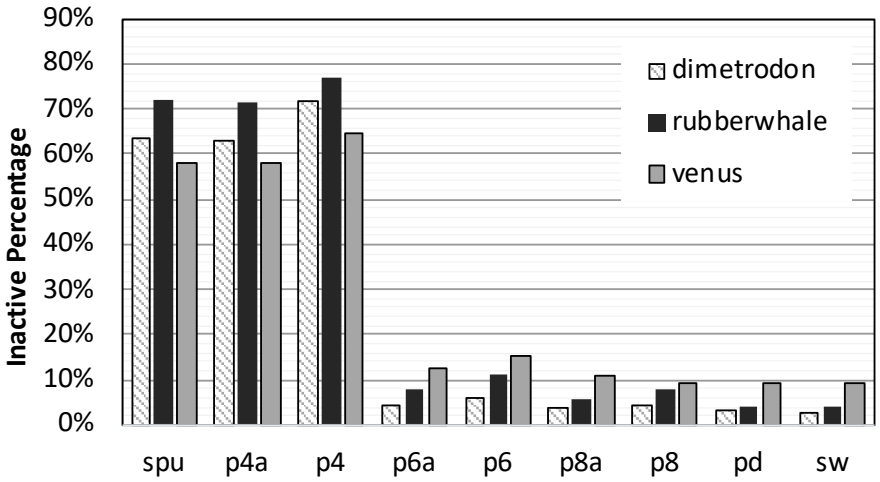
Figure 6.3 shows the convergence percentage for stereo vision increases with the increasing bit precision. In contrast to ESS, 2^n approximation improves the convergence percentage under the same bit precision. Hardware with 6-bit and 8-bit $p_{tr}(i)$ with and without 2^n approximation produces comparable convergence percentage to software. All designs except "p4" produce the same or higher convergence percentage for motion estimation, shown in Figure 6.4. All values of convergence percentage are high ($> 90\%$) in motion estimation.

Goodness of Fit

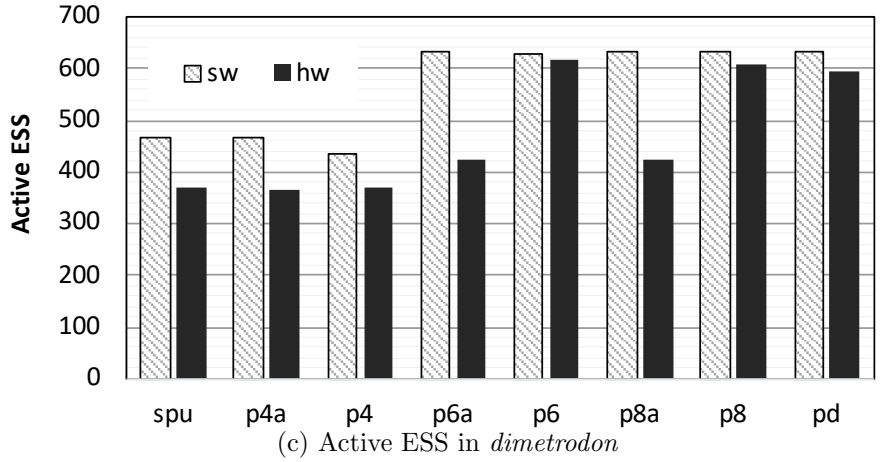
Figure 6.5 shows stereo vision RMSE results compared with software reference results. Observable lower RMSEs can be found in stereo vision *art* when increasing the bit precision from 4 to 6. Differences of RMSEs are hard to notice when further increasing the precision given whiskers largely overlap in most datasets. Stereo vision application end-point results in Figure 6.7 exhibit the same trends. All designs produce comparable result quality to the software in simulated annealing (optimization), consistent with the results discussed in Section 4.3. We highlight the following results for motion estimation (shown in figures 6.6 and 6.8): 1) the design parameters have



(a) Overall ESS



(b) Percentage of inactive random variables (inactive percentage)



(c) Active ESS in *dimetrodon*

FIGURE 6.2: Motion estimation sampling quality in the design points

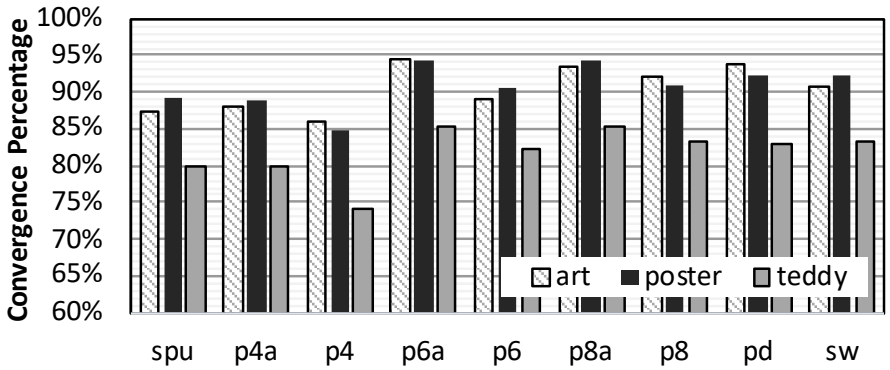


FIGURE 6.3: Stereo vision convergence percentage in the design points

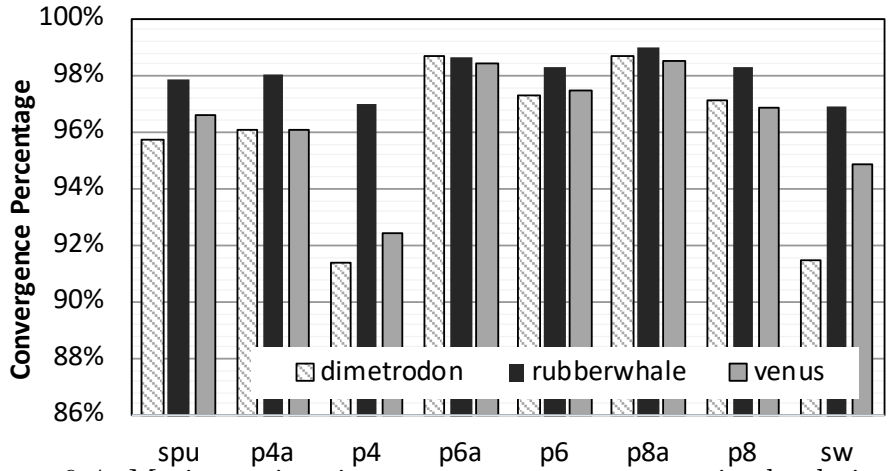
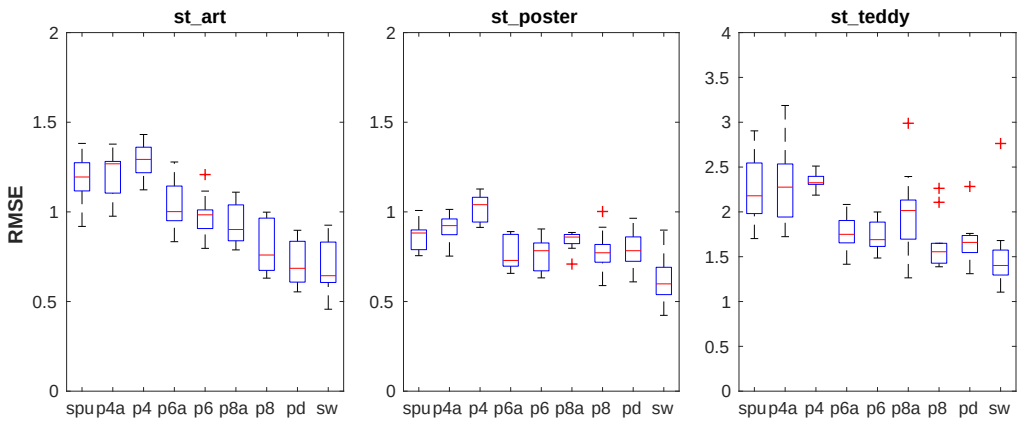
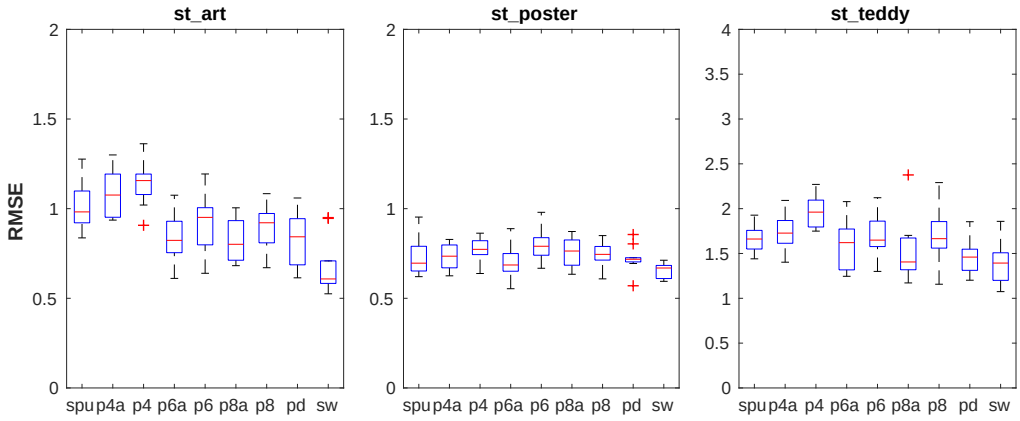


FIGURE 6.4: Motion estimation convergence percentage in the design points

negligible influence on application end-point result quality (end-point error) except "p4" in a couple of cases, which performs observably worse; 2) all designs except "p4" produce better end-point error than the FP64 software for *dimetrodon* with sampling; 3) all designs produce slightly worse end-point error than the software for *rubberwhale* with sampling; and 4) gaps exist between the software and all hardware designs including "pd" for RMSE, but not in end-point error. These results confirm the importance of using all three pillars. Overall, optimization is more robust than sampling at producing good result quality across various designs. For both modes, increasing the scaled probability to 6 bits produces comparable goodness of fit results



(a) Pure sampling (sampling)



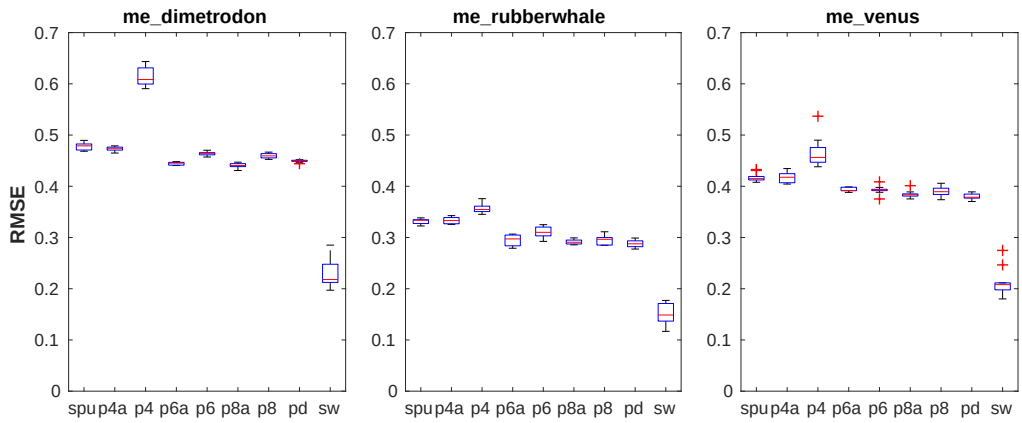
(b) Simulated annealing (optimization)

FIGURE 6.5: Stereo vision RMSE in the design points

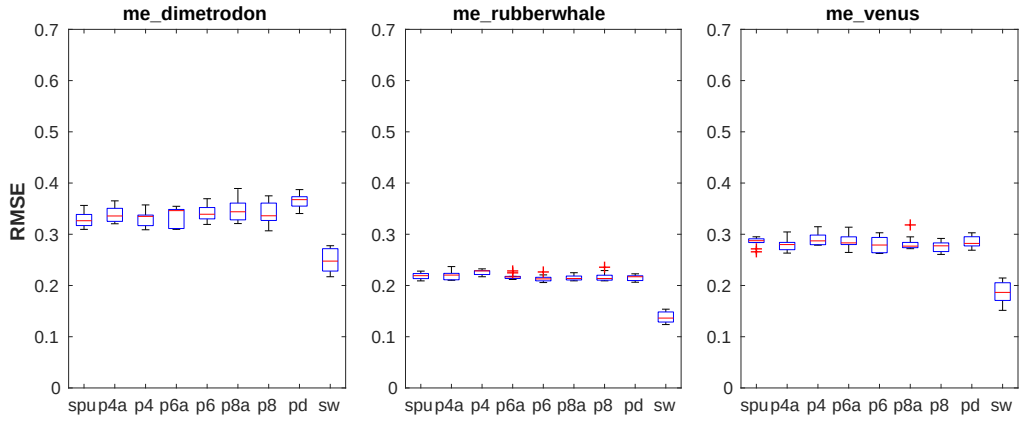
to the FP64 software.

6.1.2 Evaluating RNGs

We verify that using a 19-bit LFSR sampler with 12-bits of RNG output is sufficient to provide the same statistical robustness as MATLAB default RNG *mt19937ar* in any of the pillars. However, we are also interested in whether a bad RNG can be identified by the three pillars. We replace the 19-bit LFSR in the original SPU with an 8-bit LFSR (period of 255) and an 8-bit RNG output. We run the three stereo vision datasets to obtain end-point result quality, ESS, autocorrelation function, and convergence percentage. The bad-pixel percentage for pure-sampling is 36.9% for



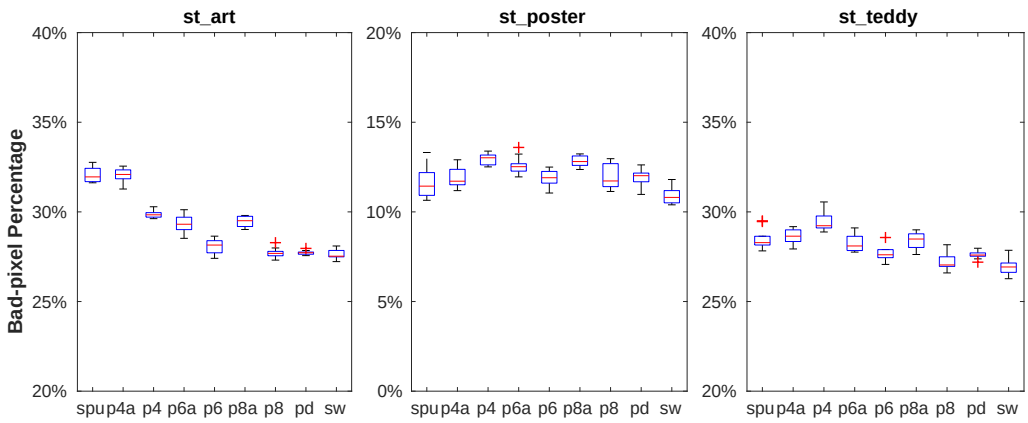
(a) Pure sampling (sampling)



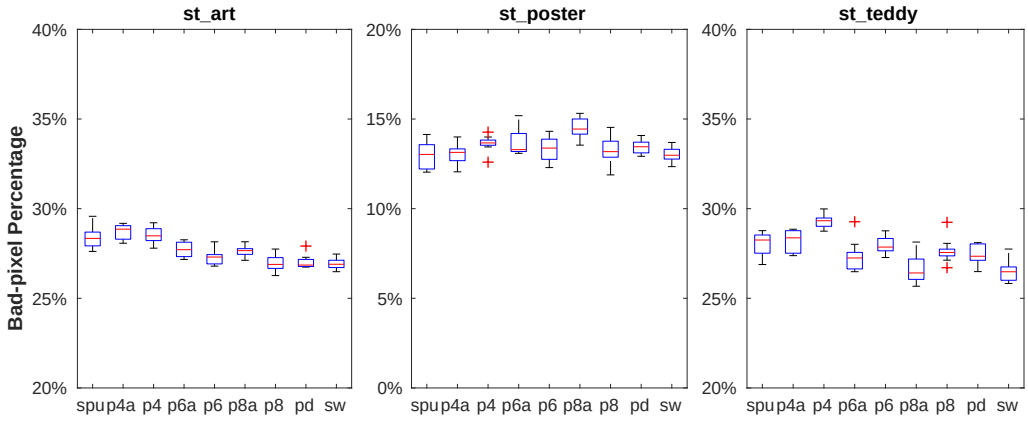
(b) Simulated annealing (optimization)

FIGURE 6.6: Motion estimation RMSE in the design points

art, 15.2% for *poster*, and 33.9% for *teddy*, all considerably worse than any of tested designs. We verify the quality degradation is not caused by fewer RNG output bits: a 19-bit LFSR with 8-bits of output has the results of 32.5% for *art*, 11.0% for *poster*, and 27.8% for *teddy*. Figure 6.9 shows the autocorrelation function comparison on a *poster* pixel location between the FP64 software with MATLAB default RNG, the SPU design with 19-bit LFSR, and an SPU design with 8-bit LFSR. Recall Equation 5.1 that autocorrelation function is a sub-step of estimating an ESS. The input to autocorrelation function is a trace of 1000 MCMC samples from each configuration. The autocorrelation function detects the repeating patterns by correlating the original input trace with its lagged copy. A peak in an autocorrelation function result



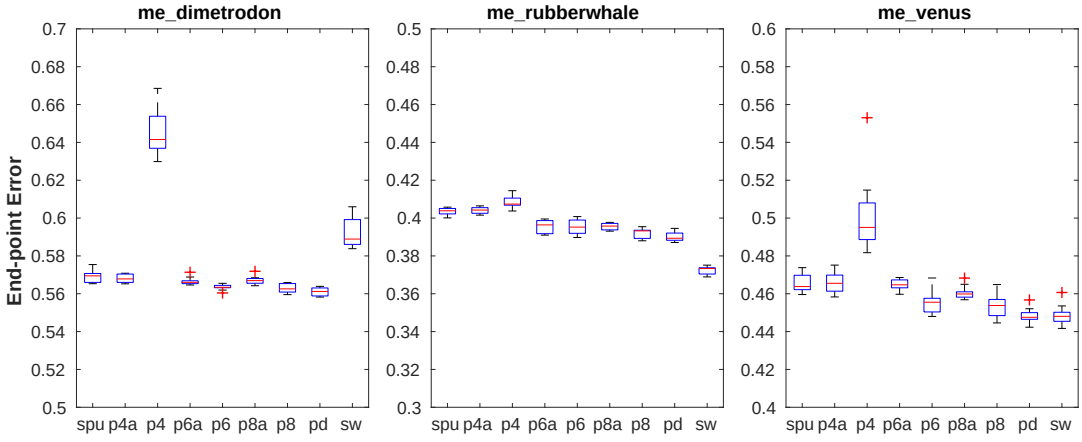
(a) Pure sampling (sampling)



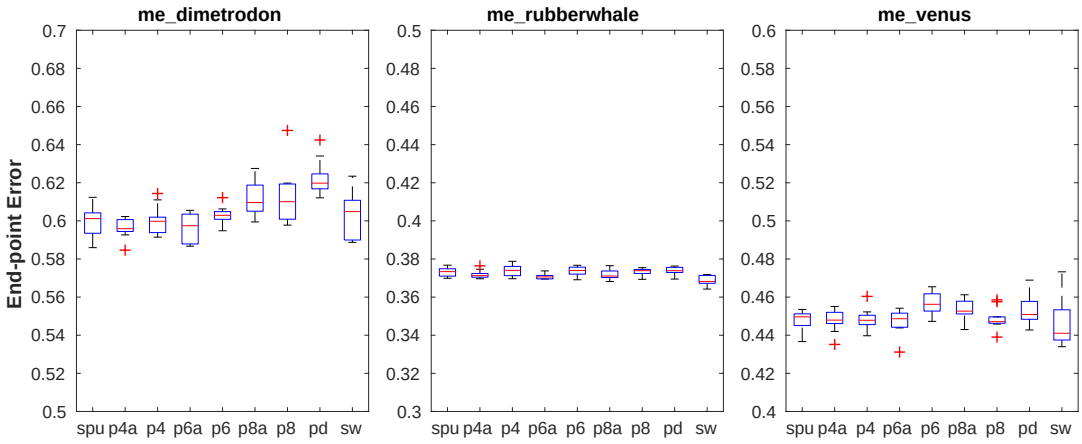
(b) Simulated annealing (optimization)

FIGURE 6.7: Stereo vision application end-point result quality in the design points

indicates a possible repeating pattern is found under that lag. The software and the SPU have similar results, whereas the design with 8-bit LFSR has strong autocorrelation, indicating the results have strong long-term dependency caused by the short period. Interestingly, the ESS in the 8-bit LFSR is higher than the software. A possible explanation is that ESS is designed for evaluating short-term dependencies in consecutive MCMC samples. The dependency on the 8-bit LFSR is respectively long-term compared with the consecutive samples. Meanwhile, the short period of 8-bit LFSR could potentially break the metric. The theory behind this phenomenon needs to be further explored. As expected, the converge percentage in the 8-bit LFSR design is significantly lower: 57.1% in *art*, 73.4% in *poster*, and 61.5% in *teddy*, in-



(a) Pure sampling (sampling)



(b) Simulated annealing (optimization)

FIGURE 6.8: Motion estimation application end-point result quality in the design points

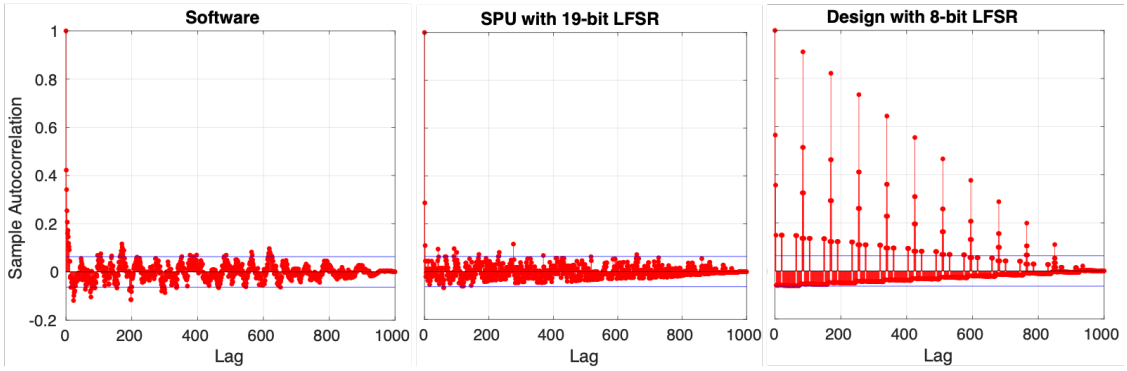


FIGURE 6.9: Autocorrelation function on *poster* pixel (x,y)=(250,200). The values between blue lines can be considered as noises.

Table 6.1: Area and power (@1GHz) analysis in ASIC

Design	**Area (μm^2)**	**Power (mW)**	**Design**	**Area**	**Power**
spu	1957	2.17	p4	2112	2.21
p6a	2134	2.31	p6	2356	2.38
p8a	2309	2.46	p8	2599	2.54

dicating the poor quality of the design. This again confirms *collectively applying all three pillars beyond end-point result is necessary to evaluate correctness.*

6.1.3 Area and Power

We estimate the ASIC area/power for various design points using the same method as presented in Section 4.3. Circuitry elements are written in Chisel and synthesized using Synopsys Design Compiler in 15nm library [92]. Memory elements (FIFOs and LUTs) are estimated using Cacti 7 [8] in 22nm technology. The designs are verified in stereo vision *art*. Table 6.1 summarizes the total area/power of design points. The numbers are the sum of 15nm circuitry and 22nm memory elements. Power is estimated at 1GHz. We estimate a double-buffered 2×256-byte LUT (537 μm^2 and 0.32 mW) and a 64-byte FIFO (215 μm^2 and 0.18 mW) with 8-bit ports, and linearly scale them to target widths of 4 and 6 bits. All designs can run up to 3.3GHz, limited by the SPU energy computation stage. Increasing the SPU $p_{tr}(i)$ from 4-bit to 6-bit precision while keeping the 2^n approximation ("p6a") incurs 1.09× area and 1.07× power overheads, but has considerably better statistical robustness. Removing 2^n approximation ("p6a") adds double-buffered LUTs for energy-to-probability conversion, thus incurs 1.20× area and 1.10× power overheads. Despite a 10% difference in area, we advocate the 6-bit designs without 2^n approximation in an ASIC for better sampling quality if area is not a major concern. The benefit from further increasing the bit-precision is marginal based on the previous analysis.

6.2 Limitations and Future Work

This chapter demonstrates a case study on applying statistical robustness to design space exploration of an MCMC accelerator. Applying the three pillars to other accelerators, applications, and models is our future work. Our proposed methodology applies to other MCMC accelerators and applications, especially for those when directly applying existing methods is difficult due to high dimensionality and potential random variables with zero empirical variance. The effects of hardware approximations are unknown for applications that require information from variables with very low variance, such as rare event simulation. However, the notion of bringing statistical robustness to the architecture design process applies to all types of accelerators, which needs more work and likely inputs from domain experts to find the right metrics and pillars. Ideally, formally proving bounds on the metrics for an accelerator could provide guarantees on statistical robustness, but is extremely difficult or impossible due to many hardware approximations techniques (e.g., truncation to zero).

6.3 Summary

Domain-defined correctness should be comprehensively evaluated when designing a specialized architecture. Statistical robustness is an essential element of probabilistic computing defined by domain experts and therefore should be considered throughout the hardware design process. This chapter explores the design space of a representative MCMC accelerator guided by the three pillars of statistical robustness: 1) sampling quality, 2) convergence diagnostic, and 3) goodness of fit. We solve the design issues in statistical robustness, surfaced in the previous chapter, by slightly increasing the bit precision and removing an approximation technique. The new design point incurs 1.20× area and 1.10× power overhead, but achieves compara-

ble statistical robustness to the FP64 software, maintaining the expected speedups without additional iterations.

7

Related Work

The previous chapters present our approaches to accelerating probabilistic computing and evaluating the correctness of a probabilistic architecture. This section reviews the related work and places our contributions in the field of probabilistic computing.

7.1 Accelerating Probabilistic Computing

Efficient support for probabilistic computing requires addressing sampling overhead in computing the parameters (step-1) and drawing samples from the parameterized distribution (step-2). One approach is to accelerate the convergence of probabilistic algorithms, such as MCMC, in an entirely algorithmic perspective. For example, Hamiltonian Monte Carlo [107] exploits the geometry to converge more efficiently. Stochastic Gradient Langevin Dynamics [145] uses sub-sampling to reduce computation for large datasets. Orthogonal MCMC [91] creates multiple Markov Chains running in parallel. The combination of multiple schemes is also available [83]. Other deterministic methods, such as Expectation Propagation and Variational Bayesian, are alternatives to probabilistic methods. Although these methods are often more efficient in the applied cases, domain experts use MCMC as a conceptually straight-

forward, mathematically simple, yet accurate framework. Other algorithmic works attempt to address step-2 of sampling by optimizing the conversion from uniform random numbers to different targeted distributions, such as discrete distribution [140], normal distribution [89], and gamma distribution [88].

Another approach to accelerate probabilistic algorithms, the one we take in this dissertation, is hardware specialization. The key trade-off is generalization vs. specialization. Some previous works address step-2 of sampling by accelerating drawing samples. For example, by using thermal noise as entropy, Probabilistic CMOS (PCMOS) [17] can build a discrete sampler parameterized by the supply voltage and noise across the chip. A Digital-to-Analog Converter (DAC) is needed to convert probability values to analog voltages. Some previous works utilize FPGA to generate some specific types of distributions, such as multivariate Gaussian distribution [131] and exponential distribution [6]. Recently, Tye et al. [135] uses transfer characteristics of Graphene Field-Effect Transistors (GFETs) to generate non-uniform univariate distributions. As discussed in Section 2.3, both steps of sampling need to be addressed to maximize efficiency.

Another type of hardware specialization is to in whole or in part accelerate both steps of sampling in specific algorithms and models. A compiler workflow is proposed to map probabilistic models into auto-generated accelerators [9]. A series of works propose FPGA accelerators for variations of Metropolis Hastings MCMC methods with algorithmic modifications, including Communication-Aware MCMC (CA-MCMC) [78], Custom-precision Firefly MCMC (CF-MCMC) [80], and the proposed Particle MCMC (ppMCMC) [100]. Mansinghka and Jonas [87] presents a Stochastic Transition Circuit and an FPGA implementation to efficiently update random variables within a provided graphical model. The design converts input parameters to normalized probabilities and generates samples from those probabilities. A similar accelerator in CPU-FPGA SoC is proposed by Ko et al. [66] using 32-bit

fixed-point precision. In an abstract concept, our proposed RSU-G and SPU are instances of Stochastic Transition Circuits. However, both RSU-G and SPU are full-custom, fully-functioned pipelines derived from a comprehensive architectural design space exploration for FPGA (SPU only) and ASIC. The RSU-G takes a different approach using emerging technology to efficient sampling from non-uniform distributions and providing high-quality randomness for probabilistic algorithms [142]. The SPU uses a customized CMOS discrete sampler with a simple 19-bit LFSR. For both RSU-G and SPU, a normalized probability is not needed and only the ratio of probabilities matters. The result quality analysis in Section 3.3 brings new information that using only a few unique values of scaled probability (or decay rate) is enough for the targeted applications, providing opportunities for design optimization to efficiently convert energy to scaled probability and efficiently support simulated annealing.

Other examples of specialized architecture for probabilistic computing include an FPGA Bayesian Neural Networks accelerator (VIBNN) [16], an FPGA acceleration framework for Stochastic Gradient Descent (Tabla) [84], and an ASIC accelerator for Bayesian Networks [62]. Other accelerators exist for deterministic Bayesian Inference, such as an FPGA Bayesian Computing Machine [74], and a Versatile Inference Processor for Belief Propagation [52].

The slowing down of Moore's Law also brings opportunities to explore new technologies for improving computational efficiency. Several recent works exploit the physical properties of emerging technologies to accelerate probabilistic computing or machine learning. The proposed RSU-G utilizes single-choromophore RET networks for efficient sampling using *first-to-fire* [142]. Furthermore, both the aforementioned GFET and multi-chromophore RET networks are claimed to have potential to sample from a general univariate distribution [135, 141] for sampling step-2 acceleration. Probabilistic CMOS can be used for Bayesian Inference [17, 112]. Quantum dots

can be used to solve decision making problems such as an intractable satisfiability problem (SAT) [4] and the "tug-of-war model" (TOW) [105]. Strain-switched magneto-tunneling junctions are proposed to support causal inference [63, 64]. Multiple memristor architectures are proposed to accelerate Neural Networks [3, 76].

For more details, a survey of stochastic computing provided by Alaghi and Hayes can be found elsewhere [2].

7.2 Evaluation Methodologies for Probabilistic Computing

Domain experts define metrics for statistical robustness of MCMC methods. A comprehensive introduction to MCMC diagnostics is provided by Robert and Casella [120] (Chapter 12). A comparison of autocorrelation time methods, one approach to defining Effective Sample Size (ESS), is provided by Thompson [132]. An alternative ESS definition is "in the custom of survey sampling" [40, 65]. A comparative review of MCMC convergence diagnostics is presented by Cowles and Carlin [26]. Multiple goodness of fit statistical tests exist, such as Kolmogorov–Smirnov test (KS-test), Analysis of Variance (ANOVA), a kernel two-sample test [41], a goodness-of-fit test based on Stochastic Rank Statistic (SRS) [123], etc. Previous work addresses some of these statistical metrics for MCMC accelerators. Mansinghka and Jonas [87] evaluates data input precision using KL-divergence and QQ plots. Liu et al. [79] argues using ESS/second as a performance metric for MCMC samplers. Mingas et al. [100] uses both ESS/second and KL-divergence. These metrics all belong to one of three pillars proposed in Chapter 5 and we argue all three pillars are needed to fully characterize the statistical robustness of an MCMC accelerator. The key is to find the appropriate metrics in pillars for different accelerators and applications.

Previous work provides analytical evaluation of approximation techniques. An analytical tool (Gappa++) for quantization error is proposed to help hardware design decision under limited precision [75], but does not address statistical robustness.

Theoretical studies provide error bounds for MCMC with algorithmic approximation techniques given mathematical assumptions [33, 56]. As mentioned in Section 5.1, some hardware approximations (e.g., truncation to zero) in the SPU design significantly complicate the formal proof, making it extremely difficult or impossible. Analytical and empirical studies have been done on evaluating limited precision in Neural Networks [25, 43, 49, 124].

Adopting a proper quality metric in approximate computing in general is crucial to both ensuring correctness and controlling the trade-off between quality of the results and the gains in the desired metric, e.g., performance, energy, or storage. These quality metrics include relative difference (e.g., in *MapReduce* and *n-body simulation*), peak signal-to-noise ratio and structural similarity (e.g., in *x264* and *image smoothing*), pixel difference (e.g., in *raytracer* and *bodytrack*), energy conservation across scenes in physics-based simulations (e.g., in *collision detection* and *constraint solving*), among others [101]. These quality metrics could be used to analyze probabilistic accelerators for different applications. Furthermore, like MLPerf [95], a comprehensive benchmark for probabilistic computing is desirable. A benchmark for Bayesian Inference models is proposed for performance evaluation [143].

Finally, understanding how the quality of RNGs influences the behavior of an application is important to make a quality statement. CMOS accelerators usually use simplified RNGs: a 19-bit LFSR for the SPU, a 128-bit LFSR for VIBNN [16], a 128-bit XORshift for the Stochastic Transition Circuit [87], and a combination of 43-bit LFSR and 37-bit CASR for a CPU-FPGA SoC [66]. A sharing LFSR scheme is proposed in Muller C-elements for Stochastic Bayesian Inference [48]. Evaluations of RNGs in these works are empirical and adhoc. Empirical researches have reported "bad" RNGs can lead to biased results in Monte Carlo simulations. For example, some Lagged Fibonacci Generators (LFG) perform poorly in 2D Ising model simulations [24]. The triplet correlations in a Generalized Feedback Shift Register (GFSR)

introduce systematic errors in Blume-Capel model Metropolis updating [126]. A Linear Congruential Generator (LCG) produces significant different simulation results in organic and biological systems [23]. The link between these observations and MCMC is unknown to our knowledge. A theoretical work [134] justifies to use the full output sequence of a small RNG for MCMC sampling, referred to as Markov Chain quasi-Monte Carlo (MCQMC). Overall, inputs from domain experts significantly help hardware designers for efficient and robust probabilistic architectures.

8

Conclusion

The impending halt in CMOS scaling and the tidal wave of Artificial Intelligence (AI) and machine learning bring both tremendous challenges and opportunities for computer architects to design efficient and robust computing systems. Statistical machine learning uses probabilistic computing as a conceptually simple, interpretable, and generalized framework to solve a wide range of problems by iteratively sampling from parameterized distributions. The sampling process is often considered too slow on conventional processors due to the overhead in computing distribution parameters and drawing samples. In this dissertation, we claim *a specialized architecture is necessary and feasible to efficiently support various probabilistic computing problems in statistical machine learning, while providing high-quality and robust results.* We ask two questions to approach the above statement:

- What is the appropriate architecture of a stochastic processing unit to efficiently support probabilistic computing?
- What methodology should we use to evaluate correctness of a probabilistic accelerator?

We provide our answer in four successive works:

- A macro-scale prototype of the previously proposed RSU-G and a new RSU-G microarchitecture to account for issues in the previous design.
- An efficient CMOS Stochastic Processing Unit (SPU) derived from RSU-G using a simple 19-bit LFSR as the RNG.
- An evaluation framework with three pillars of statistical robustness to evaluate correctness of an MCMC accelerator.
- A demonstration on using the three-pillar framework to guide design and overcome hardware limitations.

Below summarizes the contributions of this dissertation.

8.1 Summary of Contributions

In light of a promising technique to natively generate non-uniform distributions, Wang et al. [142] proposed a Resonance Energy based Sampling Unit (RSU-G) to accelerate both steps of sampling for 1st-Order Markov Random Field Gibbs Sampling. Our work starts with building a macro-scale prototype to experimentally demonstrate the RSU-G's ability to parameterize pairwise relative probabilities and conduct a simple foreground-background image segmentation. Setting up the prototype system as a true RNG without post-processing passes 165/188 items in NIST statistical randomness test. We further explore the relationship between application result quality and RSU-G design, finding the previous RSU-G design (1st-gen RSU-G) lacks both sufficient precision and dynamic range in key design parameters to provide acceptable result quality in three computer vision applications (image segmentation, motion estimation, and stereo vision). Naively scaling the problematic parameters to increase precision and dynamic range consumes too much area and

power. By performing a design space exploration on four identified design parameters, we arrive at a new RSU-G design that achieves the comparable result quality as the 64-bit floating-point (FP64) software. The new design contains four major circuit/microarchitecture changes incurring 1.27× power overhead and equivalent area: 1) improved dynamic range, 2) a new RET circuit and peripheral circuits, 3) supporting multiple energy functions for more applications, and 4) efficient probability conversion. The new RSU-G retains the same architectural interface except for an additional support for simulated annealing and therefore maintains the sizable speedups of 1st-gen RSU-G: 21-84× as a discrete accelerator over a Titan X GPU.

The promising RSU-G could lead to a higher manufacturing cost due to an additional back-end-of-line process during fabrication. We explore the feasibility of replacing the RSU-G high-quality RET-based RNG with a conventional CMOS pseudo RNG. By evaluating six different CMOS RNGs, we unexpectedly discover that a simple 19-bit LFSR provides good end-point result quality in FP64 motion estimation and stereo vision. Using more complicated RNGs does not further improve the results while using lower quality RNGs can notably degrade result quality. Therefore, we propose a CMOS Stochastic Processing Unit (SPU) by replacing the molecular-optical device with a CMOS discrete sampler with multiple optimization techniques. The SPU design produces the same result quality as the FP64 software in the three computer vision applications. The design is flexible to be deployed on an FPGA or fabricated in an ASIC. The SPU optimized for FPGA achieves at least 3× faster in performance and 33.7× less in memory compared with an HLS baseline with FP32 probability, indicating a human-designed architecture is needed to improve efficiency. The SPU for ASIC using a 19-bit LFSR avoids area/power overhead of a complex RNG and saves 33% of area and 57% of power, compared with RSU-G.

The fact that the SPU achieves good result quality even with aggressive hardware approximations motivates us to evaluate its other important statistical properties

defined by domain experts. Current methodologies for evaluating probabilistic accelerators are often incomplete or adhoc in evaluating correctness, focusing only on end-point results ("accuracy") or limited statistical properties. Failure to adequately account for domain-defined correctness can have adverse or catastrophic outcomes. Therefore, we claim a probabilistic architecture should provide some measure (or guarantee) of statistical robustness. We propose three pillars of statistical robustness: 1) *sampling quality*, 2) *convergence diagnostic*, and 3) *goodness of fit*. Each pillar has at least one quantitative empirical metric: Effective Sample Size (ESS) for sampling quality; Gelman-Rubin's $\hat{R}$ and convergence percentage for convergence diagnostic; Root Mean Squared Error (RMSE) and Jensen-Shannon Divergence (JSD) for goodness of fit. These pillars do not require ground-truth data and collectively compare specialized hardware with FP64 software. Naively applying existing popular metrics for our purposes is challenging due to high dimensionality of the target applications and random variables with zero empirical variance. Therefore, we modify the existing methodologies for sampling quality and convergence diagnostic, and propose a new metric (convergence percentage) for convergence diagnostic. The three pillars take a first step toward defining metrics and a methodology for quantitatively evaluating correctness of probabilistic accelerators. As a case study, we demonstrate the framework in a representative probabilistic MCMC accelerator—the SPU—and discover design issues that cannot be exposed using only application result quality. The SPU provides good end-point result quality but compromised statistical robustness, reducing its effective speedup by a factor of $2\times$.

Finally, we demonstrate the benefits of using the proposed three pillars to guide design space exploration and conquer the above limitations. By investigating the design trade-offs between statistical robustness and area/power, we reveal: 1) a simple 19-bit LFSR with 12-bits of RNG output does not degrade the statistical robustness or result quality across all design points; 2) considerable improvement in

statistical robustness, comparable to the FP64 software, can be achieved by slightly increasing the bit precision from 4 to 6 and removing an approximation technique, incurring only 1.20× in area and 1.10× in power without the commensurate FP64 overhead. The SPU expected speedup of 21-84× is therefore accessible.

8.2 What's Next?

This dissertation provides four subsequent works to explore specialized architecture and methodology in accelerating probabilistic computing. The extensions and future directions of each individual work are provided in the previous chapters. As discussed before, the key trade-off for a specialized architecture is generalization for flexibility vs. specialization for efficiency. Our work has explored architecture designs to accelerate 1st-Order MRF Bayesian Inference using Gibbs Sampling, leaning to the side of specialization. Starting from the current SPU, we provide potential directions for a generalized stochastic processing unit.

Supporting more graphical models Although 1st-Order MRF supported in the current SPU is widely used, providing more flexibility on models supports an even wider range of applications. For example, a two-layer MRF model is used for range sensing [28]. Variations of Markov Networks are used to model adverse drug events in electronic health records [10]. A Field-of-Experts (FoE) model is used for multiple image applications [122]. A flat-fading frequency-domain system model is applied to an MCMC Multiple-Input Multiple-Output (MIMO) detector [46]. The challenges are to find the appropriate granularity of functional units and maintain efficiency while providing flexibility.

Supporting more algorithms/solvers The current SPU focuses on Gibbs Sampling with the support of simulated annealing for inference. A future direction can address

model training, which is learning the parameters of models (e.g., α and β in Equation 4.1) to be later used for inference. A 1st-Order MRF model can be trained via Stochastic Gradient Descent (SGD) [147]. A Binary Pairwise Markov Network can be learned by Stochastic Proximal Gradient [37] or Contrastive Divergence [77]. An FoE model can be learned via Contrastive Divergence [122]. The inner loops of these learning algorithms usually use MCMC to estimate expectations. Deciding the granularity of acceleration, inner loop sampling vs. outer loop optimization, is an important trade-off.

With generalization, new supported models, algorithms, and applications require additional quality analysis and new metrics. The metrics in the pillars of statistical robustness may need to be revisited.

Bibliography

[1] Verilog implementation of aes as specified in nist fips 197. `https://github.com/secworks/aes/`.

[2] Armin Alaghi and John P. Hayes. Survey of stochastic computing. *ACM Trans. Embed. Comput. Syst.*, 12(2s):92:1–92:19, May 2013.

[3] Aayush Ankit, Izzat El Hajj, Sai Rahul Chalamalasetti, Geoffrey Ndu, Martin Foltin, R Stanley Williams, Paolo Faraboschi, Wen-mei W Hwu, John Paul Strachan, Kaushik Roy, et al. Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 715–731, 2019.

[4] Masashi Aono, Makoto Naruse, Song-Ju Kim, Masamitsu Wakabayashi, Hirokazu Hori, Motoichi Ohtsu, and Masahiko Hara. Amoeba-inspired nanoarchitectonic computing: solving intractable computational problems using nanoscale photoexcitation transfer dynamics. *Langmuir*, 29(24):7557–7564, 2013.

[5] Solomon Assefa, Fengnian Xia, and Yurii A Vlasov. Reinventing germanium avalanche photodetector for nanophotonic on-chip optical interconnects. *Nature*, 464(7285):80–84, 2010.

[6] Tarek Ould Bachir, Mohamad Sawan, and Jean-Jules Brault. A new hardware architecture for sampling the exponential distribution. In *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*, pages 001393–001396. IEEE, 2008.

[7] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, Mar 2011.

[8] Rajeev Balasubramonian, Andrew B Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. Cacti 7: New tools for interconnect explo-

ration in innovative off-chip memories. *ACM Transactions on Architecture and Code Optimization (TACO)*, 14(2):14, 2017.

[9] Subho S. Banerjee, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. Acmc 2 : Accelerating markov chain monte carlo algorithms for probabilistic models. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, pages 515–528, New York, NY, USA, 2019. ACM.

[10] Aubrey Barnard. *Causal Discovery of Adverse Drug Events in Observational Data*. PhD thesis, University of Wisconsin–Madison, 2019.

[11] Stephen T. Barnard. Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1):17–32, May 1989.

[12] Lawrence E Bassham III, Andrew L Rukhin, Juan Soto, James R Nechvatal, Miles E Smid, Elaine B Barker, Stefan D Leigh, Mark Levenson, Mark Vangel, David L Banks, et al. *Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications*. National Institute of Standards & Technology, 2010.

[13] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[14] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[15] Stephen P. Brooks and Andrew Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, 1998.

[16] Ruizhe Cai, Ao Ren, Ning Liu, Caiwen Ding, Luhao Wang, Xuehai Qian, Massoud Pedram, and Yanzhi Wang. Vibnn: Hardware acceleration of bayesian neural networks. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 476–488. ACM, 2018.

[17] Lakshmi N Chakrapani, Bilge ES Akgul, Suresh Cheemalavagu, Pinar Korkmaz, Krishna V Palem, and Balasubramanian Seshasayee. Ultra-efficient (embedded) soc architectures based on probabilistic cmos (pcmos) technology. In *Proceedings of the conference on Design, automation and test in Europe:*

Proceedings, pages 1110–1115. European Design and Automation Association, 2006.

[18] David A. Chang-Yen and Bruce K. Gale. An integrated optical oxygen sensor fabricated using rapid-prototyping techniques. *Lab Chip*, 3:297–301, 2003.

[19] Ren-Li Chen and Soon-Jyh Chang. A 6-bit current-steering dac with compound current cells for both communication and rail-to-rail voltage-source applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(11):746–750, 2012.

[20] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, page 269–284, New York, NY, USA, 2014. Association for Computing Machinery.

[21] Li Cheng and Terry Caelli. Bayesian stereo matching. *Computer Vision and Image Understanding*, 106(1):85–96, 2007.

[22] Wenjun Cheng, Luyao Ma, Tiejun Yang, Jiali Liang, and Yan Zhang. Joint lung ct image segmentation: a hierarchical bayesian approach. *PloS one*, 11(9), 2016.

[23] Timothy H Click, Aibing Liu, and George A Kaminski. Quality of random number generators significantly affects results of monte carlo simulations for organic and biological systems. *Journal of computational chemistry*, 32(3):513–524, 2011.

[24] Paul D Coddington. Analysis of random number generators using monte carlo simulation. *International Journal of Modern Physics C*, 5(03):547–560, 1994.

[25] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc., 2015.

[26] Mary Kathryn Cowles and Bradley P Carlin. Markov chain monte carlo convergence diagnostics: a comparative review. *Journal of the American Statistical Association*, 91(434):883–904, 1996.

[27] Luc Devroye. Chapter 4 nonuniform random variate generation. In Shane G. Henderson and Barry L. Nelson, editors, *Simulation*, volume 13 of *Handbooks in Operations Research and Management Science*, pages 83 – 121. Elsevier, 2006.

[28] James Diebel and Sebastian Thrun. An application of markov random fields to range sensing. In *Advances in neural information processing systems*, pages 291–298, 2006.

[29] Hadi Esmaeilzadeh, Adrian Sampson, Luis Ceze, and Doug Burger. Neural acceleration for general-purpose approximate programs. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 449–460. IEEE, 2012.

[30] Ryan M Field, Simeon Realov, and Kenneth L Shepard. A 100 fps, time-correlated single-photon-counting-based fluorescence-lifetime imager in 130 nm cmos. *IEEE Journal of Solid-State Circuits*, 49(4):867–880, 2014.

[31] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 363–370. Association for Computational Linguistics, 2005.

[32] James M Flegal, Murali Haran, and Galin L Jones. Markov chain monte carlo: Can we trust the third significant figure? *Statistical Science*, pages 250–260, 2008.

[33] Rong Ge, Holden Lee, and Andrej Risteski. Simulated tempering langevin monte carlo ii: An improved proof using soft markov chain decomposition. *arXiv preprint arXiv:1812.00793*, 2018.

[34] Andrew Gelman and Donald B Rubin. Inference from iterative simulation using multiple sequences. *Statistical science*, 7(4):457–472, 1992.

[35] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, PAMI-6(6):721–741, 1984.

[36] Stuart Geman and Christine Graffigne. Markov random field image models and their applications to computer vision. In *Proceedings of the International Congress of Mathematicians*, volume 1, page 2, 1986.

[37] Sinong Geng, Zhaobin Kuang, Jie Liu, Stephen Wright, and David Page. Stochastic learning for sparse discrete Markov random fields with controlled gradient approximation error. *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, 1:156–166, 2018.

[38] Sinong Geng, Zhaobin Kuang, and David Page. An efficient pseudo-likelihood method for sparse binary pairwise markov network estimation. *arXiv preprint arXiv:1702.08320*, 2017.

[39] Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.

[40] Lei Gong and James M Flegal. A practical sequential stopping rule for high-dimensional markov chain monte carlo. *Journal of Computational and Graphical Statistics*, 25(3):684–700, 2016.

[41] Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

[42] Christos Grivas and Markus Pollnau. Organic solid-state integrated amplifiers and lasers. *Laser & photonics reviews*, 6(4):419–462, 2012.

[43] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. *CoRR*, abs/1502.02551, 2015.

[44] Ghassan Hamra, Richard MacLehose, and David Richardson. Markov chain monte carlo: an introduction for epidemiologists. *International journal of epidemiology*, 42(2):627–634, 2013.

[45] Marcel Häselich, Simon Eggert, and Dietrich Paulus. Parallelized energy minimization for real-time markov random field terrain classification in natural environments. In *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1823–1828. IEEE, 2012.

[46] Jonathan C Hedstrom, Chung Him Yuen, Rong-Rong Chen, and Behrouz Farhang-Boroujeny. Achieving near map performance with an excited markov chain monte carlo mimo detector. *IEEE Transactions on Wireless Communications*, 16(12):7718–7732, 2017.

[47] Martin T Hill and Malte C Gather. Advances in small lasers. *Nature Photonics*, 8(12):908–918, 2014.

[48] David HK Hoe and Chet Pajardo II. Implementing stochastic bayesian inference: Design of the stochastic number generators. In *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1105–1109. IEEE, 2019.

[49] Jordan L Holi and Jenq-Neng Hwang. Finite precision error analysis of neural network hardware implementations. *IEEE Transactions on Computers*, 42(3):281–290, 1993.

[50] Joel Hruska. How makimoto's wave explains the tsunami of new ai processors. `https://www.extremetech.com/computing/287137-how-makimotos-wave-explains-the-tsunami-of-specialized-ai-processors-headed-for-market`, Apr 2020.

[51] Chien-Yuan Huang, Wen Chao Shen, Yuan-Heng Tseng, Ya-Chin King, and Chrong-Jung Lin. A contact-resistive random-access-memory-based true random number generator. *IEEE Electron Device Letters*, 33(8):1108–1110, 2012.

[52] Skand Hurkat and José F Martínez. Vip: A versatile inference processor. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 345–358. IEEE, 2019.

[53] Cryptography Research Inc. Evaluation of via c3 "nehemiah" random number generator. `https://www.rambus.com/wp-content/uploads/2015/08/VIA_rng.pdf`, 2003.

[54] Intel®. Floating-point ip cores user guide. `https://www.intel.com/content/www/us/en/programmable/documentation/eis1410764818924.html`, 2019.

[55] Yehea I Ismail and Eby G Friedman. Effects of inductance on the propagation delay and repeater insertion in vlsi circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(2):195–206, 2000.

[56] James E Johndrow, Jonathan C Mattingly, Sayan Mukherjee, and David Dunson. Optimal approximating markov chains for bayesian inference. *arXiv preprint arXiv:1508.03387*, 2015.

[57] Ajay Joshi, Christopher Batten, Yong-Jin Kwon, Scott Beamer, Imran Shamim, Krste Asanovic, and Vladimir Stojanovic. Silicon-photonic clos networks for global on-chip communication. In *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '09, pages 124–133, Washington, DC, USA, 2009. IEEE Computer Society.

[58] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, page 1–12, New York, NY, USA, 2017. Association for Computing Machinery.

[59] Gediminas Juzeliunas and David L Andrews. Quantum electrodynamics of resonance energy transfer. *Advances in Chemical Physics*, 112:357–410, 2000.

[60] Takeo Kanade, Atsushi Yoshida, Kazuo Oda, Hiroshi Kano, and Masaya Tanaka. A stereo machine for video-rate dense depth mapping and its new applications. In *Proceedings CVPR IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 196–202. IEEE, 1996.

[61] Robert E Kass, Bradley P Carlin, Andrew Gelman, and Radford M Neal. Markov chain monte carlo in practice: a roundtable discussion. *The American Statistician*, 52(2):93–100, 1998.

[62] Osama U Khan and David D Wentzloff. Hardware accelerator for probabilistic inference in 65-nm cmos. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 24(3):837–845, 2016.

[63] Santosh Khasanvis, Mingyu Li, Mostafizur Rahman, Ayan K Biswas, Mohammad Salehi-Fashami, Jayasimha Atulasimha, Supriyo Bandyopadhyay, and Csaba Andras Moritz. Architecting for causal intelligence at nanoscale. *Computer*, 48(12):54–64, 2015.

[64] Santosh Khasanvis, Mingyu Li, Mostafizur Rahman, Mohammad Salehi-Fashami, Ayan K Biswas, Jayasimha Atulasimha, Supriyo Bandyopadhyay,

and Csaba Andras Moritz. Self-similar magneto-electric nanocircuit technology for probabilistic inference engines. *IEEE Transactions on Nanotechnology*, 14(6):980–991, 2015.

[65] Leslie Kish. *Survey sampling*. New York: John Wiley & Sons, 1965.

[66] Glenn G Ko, Yuji Chai, Rob A Rutenbar, David Brooks, and Gu-Yeon Wei. Accelerating bayesian inference on structured graphs using parallel gibbs sampling. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, pages 159–165. IEEE, 2019.

[67] Glenn G Ko and Rob A Rutenbar. A case study of machine learning hardware: Real-time source separation using markov random fields via sampling-based inference. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2477–2481. IEEE, 2017.

[68] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[69] Janusz Konrad and Eric Dubois. Bayesian estimation of motion vector fields. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 14(9):910–927, 1992.

[70] Craig LaBoda, Heather Duschl, and Chris L Dwyer. Dna-enabled integrated molecular systems for computation and sensing. *Accounts of chemical research*, 47(6):1816–1824, 2014.

[71] Stan Z Li. Markov random field models in computer vision. In *European conference on computer vision*, pages 361–370. Springer, 1994.

[72] Stan Z Li. *Markov random field modeling in image analysis*. Springer Science & Business Media, 2009.

[73] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37(1):145–151, 1991.

[74] Mingjie Lin, Ilia Lebedev, and John Wawrzynek. High-throughput bayesian computing machine with reconfigurable hardware. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, FPGA '10, pages 73–82, New York, NY, USA, 2010. ACM.

[75] Michael D. Linderman, Matthew Ho, David L. Dill, Teresa H. Meng, and Garry P. Nolan. Towards program optimization through automated analysis

of numerical precision. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '10, pages 230–237, New York, NY, USA, 2010. ACM.

[76] Chenchen Liu, Bonan Yan, Chaofei Yang, Linghao Song, Zheng Li, Beiye Liu, Yiran Chen, Hai Li, Qing Wu, and Hao Jiang. A spiking neuromorphic design with resistive crossbar. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, pages 14:1–14:6, New York, NY, USA, 2015. ACM.

[77] Jie Liu and David Page. Structure learning of undirected graphical models with contrastive divergence. In *ICML 2013 Workshop on Structured Learning: Inferring Graphs from Structured and Unstructured Inputs*, 2013.

[78] Shuanglong Liu and Christos-Savvas Bouganis. Communication-aware mcmc method for big data applications on fpgas. In *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*, pages 9–16. IEEE, 2017.

[79] Shuanglong Liu, Grigorios Mingas, and Christos-Savvas Bouganis. An exact mcmc accelerator under custom precision regimes. In *2015 International Conference on Field Programmable Technology (FPT)*, pages 120–127. IEEE, 2015.

[80] Shuanglong Liu, Grigorios Mingas, and Christos-Savvas Bouganis. An unbiased mcmc fpga-based accelerator in the land of custom precision arithmetic. *IEEE Transactions on Computers*, 66(5):745–758, 2017.

[81] Lin Luan, Randall D Evans, Nan M Jokerst, and Richard B Fair. Integrated optical sensor in a digital microfluidic platform. *IEEE Sensors Journal*, 8(5):628–635, 2008.

[82] Lin Luan, Matthew W Royal, Randall Evans, Richard B Fair, and Nan M Jokerst. Chip scale optical microresonator sensors integrated with embedded thin film photodetectors on electrowetting digital microfluidics platforms. *IEEE Sensors Journal*, 12(6):1794–1800, 2012.

[83] Yi-An Ma, Tianqi Chen, and Emily Fox. A complete recipe for stochastic gradient mcmc. In *Advances in Neural Information Processing Systems*, pages 2917–2925, 2015.

[84] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kyung Kim, and Hadi Esmaeilzadeh. Tabla: A unified template-based framework for accelerating statistical machine learning. In *High*

Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on, pages 14–26. IEEE, 2016.

[85] Tsugio Makimoto. Implications of makimoto's wave. *Computer*, 46(12):32–37, 2013.

[86] Shingo Mandai, Matthew W Fishburn, Yuki Maruyama, and Edoardo Charbon. A wide spectral range single-photon avalanche diode fabricated in an advanced 180 nm cmos technology. *Optics express*, 20(6):5849–5857, 2012.

[87] Vikash Mansinghka and Eric Jonas. Building fast bayesian computing machines out of intentionally stochastic, digital parts. *arXiv preprint arXiv:1402.4914*, 2014.

[88] George Marsaglia and Wai Wan Tsang. A simple method for generating gamma variables. *ACM Transactions on Mathematical Software (TOMS)*, 26(3):363–372, 2000.

[89] George Marsaglia and Wai Wan Tsang. The ziggurat method for generating random variables. *Journal of statistical software*, 5(8):1–7, 2000.

[90] David Martin, Charless Fowlkes, Doron Tal, and Jitendra Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423. IEEE, 2001.

[91] Luca Martino, Víctor Elvira, David Luengo, Jukka Corander, and Francisco Louzada. Orthogonal parallel mcmc methods for sampling and optimization. *Digital Signal Processing*, 58:64–84, 2016.

[92] Mayler Martins, Jody Maick Matos, Renato P. Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. Open cell library in 15nm freepdk technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ISPD '15, pages 171–178, New York, NY, USA, 2015. ACM.

[93] Makoto Matsumoto. mt19937ar: Mersenne twister with improved initialization. `www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html`, 2002.

[94] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator.

ACM Transactions on Modeling and Computer Simulation (TOMACS), 8(1):3–30, 1998.

[95] Peter Mattson, Vijay Janapa Reddi, Christine Cheng, Cody Coleman, Greg Diamos, David Kanter, Paulius Micikevicius, David Patterson, Guenther Schmuelling, Hanlin Tang, Gu-Yeon Wei, and Carole-Jean Wu. Mlperf: An industry standard benchmark suite for machine learning performance. *IEEE Micro*, 40(2):8–16, 2020.

[96] Michael Mayberry. Probabilistic computing takes artificial intelligence to the next step. `https://newsroom.intel.com/editorials/probabilistic-computing-takes-artificial-intelligence-next-step/`, May 2018.

[97] Patrick McClure, Nao Rho, John A. Lee, Jakub R. Kaczmarzyk, Charles Y. Zheng, Satrajit S. Ghosh, Dylan M. Nielson, Adam G. Thomas, Peter Bandettini, and Francisco Pereira. Knowing what you know in brain segmentation using bayesian deep neural networks. *Frontiers in Neuroinformatics*, 13:67, 2019.

[98] John P Mechalas. Intel® digital random number generator (drng) software implementation guide. `https://software.intel.com/en-us/articles/intel-digital-random-number-generator-drng-software-implementation-guide`, 2014.

[99] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[100] Grigorios Mingas, Leonardo Bottolo, and Christos-Savvas Bouganis. Particle mcmc algorithms and architectures for accelerating inference in state-space models. *International Journal of Approximate Reasoning*, 83:413–433, 2017.

[101] Sparsh Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, March 2016.

[102] Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

[103] Sayan Mukherjee. Probabilistic machine learning. `http://www2.stat.duke.edu/~sayan/561/2015/stat_ml.pdf`, 2015.

[104] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[105] Makoto Naruse, Masashi Aono, and Song-Ju Kim. Nanoscale photonic network for solution searching and decision making problems. *IEICE transactions on communications*, 96(11):2724–2732, 2013.

[106] Radford M Neal. *Probabilistic inference using Markov chain Monte Carlo methods*. Department of Computer Science, University of Toronto Toronto, Ontario, Canada, 1993.

[107] Radford M Neal. Mcmc using hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.

[108] Cristiano Niclass, Claudio Favi, Theo Kluter, Marek Gersbach, and Edoardo Charbon. A 128× 128 single-photon imager with on-chip column-level 10b time-to-digital converter array capable of 97ps resolution. In *2008 IEEE International Solid-State Circuits Conference-Digest of Technical Papers*, pages 44–594. IEEE, 2008.

[109] Cristiano Niclass, Alexis Rochas, P-A Besse, and Edoardo Charbon. Design and characterization of a cmos 3-d image sensor based on single photon avalanche diodes. *IEEE Journal of Solid-State Circuits*, 40(9):1847–1854, 2005.

[110] Nvidia. Nvdla primer. `http://nvdla.org/primer.html`, 2018.

[111] Hooisweng Ow, Daniel R Larson, Mamta Srivastava, Barbara A Baird, Watt W Webb, and Ulrich Wiesner. Bright and stable core- shell fluorescent silica nanoparticles. *Nano letters*, 5(1):113–117, 2005.

[112] Krishna V Palem. Energy aware computing through probabilistic switching: A study of limits. *IEEE Transactions on Computers*, 54(9):1123–1137, 2005.

[113] Darek Palubiak, Munir M El-Desouki, Ognian Marinov, M Jamal Deen, and Qiyin Fang. High-speed, single-photon avalanche-photodiode imager for biomedical applications. *IEEE Sensors Journal*, 11(10):2401–2412, 2011.

[114] Yan Pan, John Kim, and Gokhan Memik. Flexishare: Channel sharing for an energy-efficient nanophotonic crossbar. In *HPCA-16 2010 The Sixteenth International Symposium on High-Performance Computer Architecture*, pages 1–12. IEEE, 2010.

[115] Jun Pang, Chris Dwyer, and Alvin R. Lebeck. More is less, less is more: Molecular-scale photonic noc power topologies. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages*

and Operating Systems, ASPLOS '15, pages 283–296, New York, NY, USA, 2015. ACM.

[116] Jun Pang, Christopher Dwyer, and Alvin R. Lebeck. mnoc: Large nanophotonic network-on-chip crossbars with molecular scale devices. *J. Emerg. Technol. Comput. Syst.*, 12(1):1:1–1:25, August 2015.

[117] Seongwook Park, Kyeongryeol Bong, Dongjoo Shin, Jinmook Lee, Sungpill Choi, and Hoi-Jun Yoo. 4.6 a1. 93tops/w scalable deep learning/inference processor with tetra-parallel mimd architecture for big-data applications. In *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*, pages 1–3. IEEE, 2015.

[118] Constantin Pistol, Wutichai Chongchitmate, Christopher Dwyer, and Alvin R Lebeck. Architectural implications of nanoscale integrated sensing and computing. In *Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*, pages 13–24, 2009.

[119] Constantin Pistol and Chris Dwyer. Scalable, low-cost, hierarchical assembly of programmable dna nanostructures. *Nanotechnology*, 18(12):125305, 2007.

[120] Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

[121] Christian P. Robert, Víctor Elvira, Nick Tawn, and Changye Wu. Accelerating mcmc algorithms. *Wiley Interdisciplinary Reviews: Computational Statistics*, 10(5):e1435, 2018.

[122] Stefan Roth and Michael J Black. Fields of experts: A framework for learning image priors. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 860–867. IEEE, 2005.

[123] Feras A Saad, Cameron E Freer, Nathanael L Ackerman, and Vikash K Mansinghka. A family of exact goodness-of-fit tests for high-dimensional discrete distributions. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1640–1649, 2019.

[124] Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. Analytical guarantees on numerical precision of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3007–3016, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[125] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1-3):7–42, 2002.

[126] Friederike Schmid and Nigel B Wilding. Errors in monte carlo simulations using shift register random number generators. *International Journal of Modern Physics C*, 6(06):781–787, 1996.

[127] Gary Shambat, Bryan Ellis, Jan Petykiewicz, Marie A Mayer, Arka Majumdar, Tomas Sarmiento, James S Harris, Eugene E Haller, and Jelena Vuckovic. Electrically driven photonic crystal nanocavity devices. *IEEE Journal of Selected Topics in Quantum Electronics*, 18(6):1700–1710, 2012.

[128] Taejoong Song, Woojin Rim, Sunghyun Park, Yongho Kim, Giyong Yang, Hoonki Kim, Sanghoon Baek, Jonghoon Jung, Bongjae Kwon, Sungwee Cho, Hyuntaek Jung, Yongjae Choo, and Jaeseung Choi. A 10 nm finfet 128 mb sram with assist adjustment system for power, performance, and area optimization. *IEEE Journal of Solid-State Circuits*, 52(1):240–249, 2016.

[129] Mario Stipčević and Çetin Kaya Koç. True random number generators. In *Open Problems in Mathematics and Computational Science*, pages 275–315. Springer, 2014.

[130] Tamás Szirányi, Josiane Zerubia, László Czúni, David Geldreich, and Zoltán Kato. Image segmentation using markov random field model in fully parallel cellular network architectures. *Real-Time Imaging*, 6(3):195–211, 2000.

[131] David B Thomas and Wayne Luk. Using fpga resources for direct generation of multivariate gaussian random numbers. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*, pages 344–347. IEEE, 2009.

[132] Madeleine B Thompson. A comparison of methods for computing autocorrelation time. *arXiv preprint arXiv:1011.0175*, 2010.

[133] Shyamkumar Thoziyoor, Naveen Muralimanohar, Jung Ho Ahn, and Norman P Jouppi. Cacti 5.3. *HP Laboratories, Palo Alto, CA*, 2008.

[134] Seth D Tribble. *Markov chain Monte Carlo algorithms using completely uniformly distributed driving sequences*. PhD thesis, Stanford University, 2007.

[135] Nathaniel Joseph Tye, James Timothy Meech, Bilgesu Arif Bilgin, and Phillip Stanley-Marbell. A system for generating non-uniform random variates using graphene field-effect transistors. *arXiv preprint arXiv:2004.14111*, 2020.

[136] Bernard Valeur and Mário Nuno Berberan-Santos. *Molecular fluorescence: principles and applications*. John Wiley & Sons, 2012.

[137] Dootika Vats, James M. Flegal, and Galin L. Jones. Multivariate output analysis for markov chain monte carlo, 2015.

[138] Dootika Vats and Christina Knudson. Revisiting the gelman-rubin diagnostic. *arXiv preprint arXiv:1812.09384*, 2018.

[139] Michael Wahl, Matthias Leifgen, Michael Berlin, Tino Röhlicke, Hans-Jürgen Rahn, and Oliver Benson. An ultrafast quantum random number generator with provably bounded output bias based on photon arrival time measurements. *Applied Physics Letters*, 98(17):171105, 2011.

[140] Alastair J Walker. An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):253–256, 1977.

[141] Siyang Wang, Alvin R Lebeck, and Chris Dwyer. Nanoscale resonance energy transfer-based devices for probabilistic computing. *IEEE Micro*, 35(5):72–84, 2015.

[142] Siyang Wang, Xiangyu Zhang, Yuxuan Li, Ramin Bashizade, Song Yang, Chris Dwyer, and Alvin R. Lebeck. Accelerating markov random field inference using molecular optical gibbs sampling units. In *Proceedings of the 43rd International Symposium on Computer Architecture*, ISCA '16, pages 558–569, Piscataway, NJ, USA, 2016. IEEE Press.

[143] Yu Emma Wang, Yuhao Zhu, Glenn G Ko, Brandon Reagen, Gu-Yeon Wei, and David Brooks. Demystifying bayesian inference workloads. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 177–189. IEEE, 2019.

[144] Shingo Watanabe and Koki Abe. A vlsi design of mersenne twister. *IPSJ SIG Notes*, 2005(41):13–18, may 2005.

[145] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 681–688, 2011.

[146] Allen Y. Yang, John Wright, Yi Ma, and S. Shankar Sastry. Unsupervised segmentation of natural images via lossy data compression. *Computer Vision and Image Understanding*, 110(2):212 – 225, 2008.

[147] Laurent Younes. Stochastic gradient estimation strategies for markov random fields. In *Bayesian inference for inverse problems*, volume 3459, pages 315–325. International Society for Optics and Photonics, 1998.

[148] Xiangyu Zhang and Ramin Bashizade. Rolling dice at the nanoscale. *XRDS*, 26(1):18–22, September 2019.

[149] Xiangyu Zhang, Ramin Bashizade, Craig LaBoda, Chris Dwyer, and Alvin R Lebeck. Architecting a stochastic computing unit with molecular optical devices. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 301–314. IEEE, 2018.

[150] Xiangyu Zhang, Ramin Bashizade, Yicheng Wang, Cheng Lyu, Sayan Mukherjee, and Alvin R Lebeck. Beyond application end-point results: Quantifying statistical robustness of mcmc accelerators. *arXiv preprint arXiv:2003.04223*, 2020.

Biography

Xiangyu Zhang received his Ph.D. in Electrical and Computer Engineering at Duke University in 2020, advised by Alvin R. Lebeck. Zhang's research focuses on computer architecture and systems in support of AI/machine learning, probabilistic computing, and statistics. He is a recipient of ECE Graduate Teaching Award for Outstanding Course Administration in the academic year of 2017-2018. Prior to Duke, he received his B.S. in 2014 at South China Agricultural University, advised by Yongyao Li.