Statistical Robustness of MCMC Accelerators

Xiangyu Zhang, Ramin Bashizade, Yicheng Wang, Cheng Lyu, Sayan Mukherjee, Alvin R. Lebeck

Duke University

{*xiangyu.zhang, ramin.bashizade, yicheng.wang, cheng.lu3*}*@duke.edu, sayan@stat.duke.edu, alvy@cs.duke.edu*

***Abstract*—Statistical machine learning often uses probabilistic algorithms, such as Markov Chain Monte Carlo (MCMC), to solve a wide range of problems. Probabilistic computations, often considered too slow on conventional processors, can be accelerated with specialized hardware by exploiting parallelism and optimizing the design using various approximation techniques. Current methodologies for evaluating correctness of probabilistic accelerators are often incomplete, mostly focusing only on end-point result quality ("accuracy"). It is important for hardware designers and domain experts to look beyond end-point "accuracy" and be aware of how hardware optimizations impact statistical properties.**

This work takes 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 and surface design issues that cannot be exposed using only application end-point result quality. 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.

I. Introduction

Statistical machine learning, like other methods in artificial intelligence, has become an important workload for computing systems. Such workloads often utilize probabilistic algorithms, such as Markov Chain Monte Carlo (MCMC) methods, which enable the potential to provide generalized frameworks to solve a wide range of problems. As alternatives to Deep Neural Networks, these algorithms provide easier access to interpreting why a given result is obtained through their model transparency and statistical properties. Many specialized accelerators are proposed to address the sampling inefficiency of probabilistic algorithms, by utilizing approximation techniques to improve the hardware efficiency, such as reducing bit representation, truncating small values to zero, or simplifying random number generators.

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, and 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 [4]. Therefore, *a probabilistic architecture should provide some measure (or guarantee) of statistical robustness.*

This work takes a first step towards 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* **(Contribution 1)**. Each pillar has at least one quantitative empirical metric: Effective Sample Size (ESS) in sampling quality; Gelman-Rubin's $\hat{R}$ and convergence percentage in convergence diagnostic; and Root Mean Squared Error (RMSE) and Jensen-Shannon Divergence (JSD) in 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 variance. Therefore, we modify the existing methodologies for sampling quality and convergence diagnostic, and propose a new metric (convergence percentage) for convergence diagnostic **(Contribution 2)**. We call on domain experts to develop metrics with stronger theoretical foundations to account for common hardware optimizations.

As a case study, we demonstrate the framework in a representative probabilistic accelerator [20] and show that 1) end-point "accuracy" alone is insufficient, particularly for predicting outcome for previously unseen inputs, and 2) that FP64 is insufficient as ground truth since in some cases more

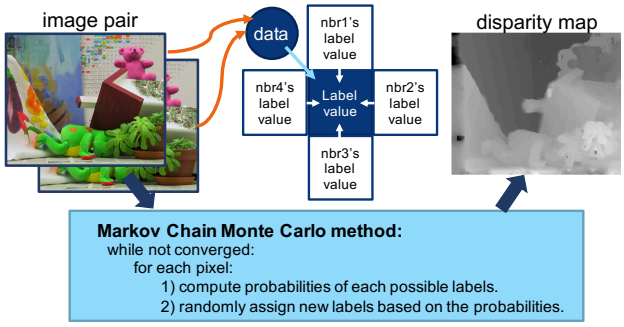


Fig. 1. Stereo vision using Markov Chain Monte Carlo (Markov Random Field Gibbs Sampling). Note that Sampling is performed in the inner loop.

limited precision can produce more accurate end-point results based on labeled data **(Contribution 3)**. The analysis shows that the accelerator achieves the same application end-point result quality as FP64 software, confirming the previous work. However, we show the accelerator differs with FP64 in ESS and convergence percentage, and reveal that applications need to run $2\times$ more iterations on the accelerator to achieve the same statistical robustness as FP64, reducing the accelerator's effective speedup. We also explore the design space of the above accelerator to expose the trade-offs between statistical robustness and area/power **(Contribution 4)**. We find that 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 $1.20\times$ area and $1.10\times$ power overhead.

II. Background and Motivation

A. Probabilistic Algorithms

Probabilistic (stochastic) algorithms are a powerful approach used in many applications (e.g., image analysis, robotics, natural language processing, global health, and wireless communications). Probabilistic algorithms solve problems by randomly inferring the parameters in the probabilistic models to explain observed data, which create opportunities to provide generalized frameworks and are the only practical approach to solve certain problems, such as high-dimensional inference. Compared with deep neural networks, probabilistic models are "conceptually simple, compositional, and interpretable" [8]. Bayesian Inference is an important framework in probabilistic models, which updates the probability estimate (a.k.a., posterior distribution) for a hypothesis by combining information from prior beliefs and observed data. As the dimension of data and probability distribution increases, analytically or numerically deriving the exact result of a posterior distribution becomes complicated and intractable. One approach to break "the curse of dimensionality" is Markov Chain Monte Carlo (MCMC), methods that solve the problems by iteratively generating samples on random variables and eventually converge to a result regardless of the initial stage.

Fig. 1 shows an example of using an MCMC method, Markov Random Field (MRF) Gibbs Sampling, in 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 results are presented in a disparity map, indicating the depth of the corresponding objects in the image (lighter is closer). The MCMC method iterates the image pixels by considering the disparity of each pixel as a random variable. For each pixel, it evaluates probabilities of each possible label (disparity outcome) and draws a sample as the output label. Each probability is determined by the neighboring pixels label values and the initial pixel data values of the image pair, defined by First-Order MRF graphical model. The outer loop (a.k.a., iteration) iterates on the whole image until convergence. MCMC methods rely on efficiently generating samples from a parameterized distribution, which involves step 1) efficiently computing the parameters of the distribution to sample from based on observed data, and step 2) efficiently generating samples from the parameterized distribution. Unfortunately, as described in previous work [19], sampling overhead can be notably high.

B. Specialized Probabilistic Accelerators

Meeting the needs of domain experts may be achieved by accelerating sampling through hardware specialization. Specialized architectures are proposed to accelerate specific algorithms and models, such as a Stochastic Transition Circuit [13], dedicated MCMC models [10], [12], CMOS-hybrid MRF Gibbs Sampling Unit [19], [20], and compiler workflows [2]. Many of these accelerators use various forms of approximation (e.g., limited bit precision, pseudo random number generators, etc.) to reduce area/power, allowing more individual units on a single chip and thus improving overall performance. As described in Sec. I, it is important to measure (or guarantee) the correctness of these accelerators, including both end-point result quality and statistical robustness. Sec. III presents our proposed metrics and framework and we use an accelerator from Zhang, et al. [20] (described below) as a case study to demonstrate how to analyze an existing accelerator and to perform design space exploration.

C. A Representative Probabilistic Accelerator

As a case study, we consider the Gibbs Sampling accelerator design described by Zhang, et al. [20] implemented entirely in CMOS using pseudo random number generation instead of molecular optical devices (cf. [20] Sec. IV.C). Fig. 2 shows the baseline pipeline design, which we call a Stochastic Processing Unit (SPU). It is divided into four main stages with two internal decoupling FIFOs and an inverse transform method is used for the discrete sampler.

Given the data and neighbor labels, the first stage computes the total energy of a possible label $E(i)$ each cycle. The energy $E(i)$ is then dynamically scaled using subtraction to maximize the dynamic range (Eq. 1). Both $E(i)$ and $E_s(i)$ are 8-bit unsigned integers. 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

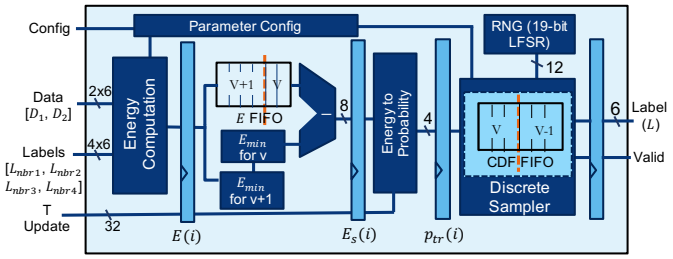


Fig. 2. The SPU pipeline derived from RSU-G [20]. The sampling stage is replaced with a CMOS discrete sampler.

between [0,1] using floating-point in software, where T is a fixed parameter per outer iteration. However, the probability is scaled using Eq. 2 and truncated using Eq. 3 to match the unsigned integer representation, where $P_{bits} = 4$ 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 (Eq. 3). 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. 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.

$$E_s(i) = E(i) - E_{min} \tag{1}$$

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

$$p_{tr}(i) = \lfloor 2^{\lfloor \log_2 p_s(i) \rfloor} \rfloor \tag{3}$$

The SPU supports two operating modes: 1) pure-sampling and 2) optimization (simulated annealing). Pure-sampling iteratively generates Gibbs samples using constant temperature T, where T is considered a parameter obtained during model training. When converged, the estimated distribution of a random variable (e.g., distribution of possible disparities in a pixel) can be obtained by collecting the latest N samples. An exact result can be obtained from the mode of the estimated distribution, the most frequent label. The optimization mode uses simulated annealing to converge to an exact result faster than pure-sampling by strategically decreasing the temperature T, but cannot provide an estimated distribution. The previous work [20] evaluates only the optimization mode.

We implement the SPU in Verilog and use QuestaSim simulation to evaluate the end-point result quality of the same three applications assessed in the previous work [20]: image segmentation, motion estimation, and stereo vision. Tab. I shows the result quality comparison between FP64 software and the SPU. Each result is collected by a single run per dataset in optimization mode. We validate that the SPU with a simple 19-bit LFSR as its random number generator (RNG) achieves the same result quality as the software. Image segmentation results indicate the same conclusion and are omitted for brevity. We also obtain similar high quality applicatoin

TABLE I
SPU RESULT QUALITY FROM ONE RUN PER DATASET.

Motion estimation[1]			**Stereo vision**[2]		
Dataset	Software	SPU	Dataset	Software	SPU
dimetrodon	0.600	0.611	*art*	26.8%	27.7%
rubberwhale	0.371	0.376	*poster*	12.3%	11.0%
venus	0.460	0.449	*teddy*	26.9%	27.8%

[1] Metric is end-point error [1]. Lower is better.
[2] Metric is bad-pixel percentage [16]. Lower is better.

results on an Intel Arria 10 FPGA prototype. Despite these results we are left with the question: *What do the results in Tab. I indicate about accelerator statistical robustness*? The short answer is *nothing*. The following sections present our initial efforts toward providing a better answer.

III. 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 [6], [9]. 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, that are based on foundations in statistics, to establish three pillars for assessing a probabilistic accelerator's statistical robustness: 1) sampling quality [17], 2) convergence diagnostic [5], and 3) goodness of fit. Each pillar has at least one quantitative measure. Collectively these pillars evaluate correctness, 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. Previous work addresses some statistical metrics for MCMC accelerators (e.g., KL-divergence and QQ plots [13], ESS/second [12], and goodness of fit tests [15]). These metrics all belong to one of three pillars proposed in this work and we argue all three pillars are needed to fully characterize the statistical robustness of an MCMC accelerator.

Note that the statistical robustness is jointly affected by the algorithm and hardware architecture. Therefore, we compare hardware results with FP64 software as the baseline to extract the impact of hardware optimizations.

A. 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 nontrival 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. Theoretically, $ESS = n$ provides the best sampling quality where all samples are independent. Since closed form expressions for ESS are difficult, we estimate ESS using known initial positive sequence (IPS) method [12], [17]. Note that ESS has no definition when all collected samples have the same value (zero variance), which is possible in practice as shown in Sec. IV. Many MCMC problems are high-dimensional (a 320×320 image has 102,400 dimensions). An ideal metric can report a scalar ESS value to account for all dimensions as a single random variable. While methods exist to report multivariate ESS [18], to our knowledge they aren't practical in our case and they do not allow zero variance for any variable. In this paper, we calculate ESS per dimension (random variable/pixel) and report the arithmetic mean ESS. Sec. IV describes how we handle zero variance cases.

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.

B. Pillar 2: Convergence Diagnostic

An important question for an MCMC method is when to stop iterating, determined by when the MCMC is converged. Multiple methods exist to measure the convergence [5]. We use Gelman-Rubin's $\hat{R}$ (potential scale reduction factor) [3], a popular quantitative method to measure whether a univariate random variable (e.g., a pixel in stereo vision) is converged at a certain iteration. Gelman-Rubin's $\hat{R}$ 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]. As a rule of thumb [3], [7], a univariate random variable is considered converged when $\hat{R} < 1.1$. Typically larger $\hat{R}$ indicates that more iterations are needed to converge.

A scalar convergence diagnostic is preferred for multi-dimensional applications. Similar to ESS, handling high dimensions and the random variables with estimated zero variance ($W = 0$) is challenging using existing methods [3]. The original Gelman-Rubin's $\hat{R}$ metric has no definition at $W = 0$. We 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 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.

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

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 software, the additional iterations may reduce its effective speedup.

C. 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 to a reference software result, and 2) using a distribution divergence measurement to evaluate all possible data inputs and provide worst-case divergence.

1) 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.

2) Data-independent Analysis: Recall from Sec. II-A that step-one 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 P_{hw} from software P_{sw} provides insights on why the results are good (or bad), how the hardware may perform on unobserved data, and the worst-case divergence in arbitrary data inputs. This can be measured using a popular divergence measure Kullback-Leibler (KL) divergence. However, KL-divergence 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 in our study. We use the symmetric method Jensen-Shannon Divergence (JSD) [11] instead as the divergence measurement. 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

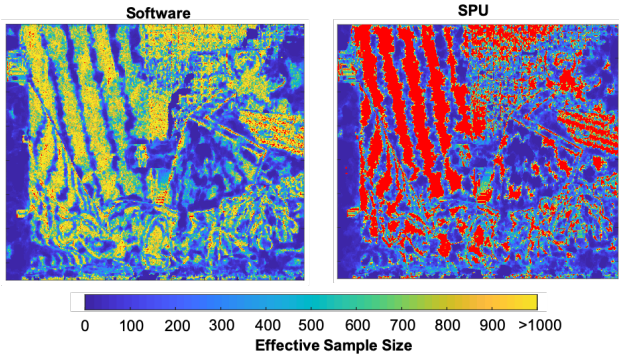
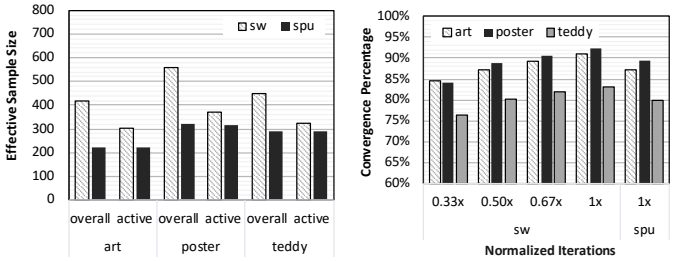


Fig. 3. ESS per random variable in stereo vision *teddy*. Red regions correspond to zero variance. Red regions in the SPU overlap high ESS regions with small variances in the software.



(a) Mean overall and active ESS (b) Convergence percentage

Fig. 4. Sampling quality (a) and convergence diagnostic (b) results in stereo vision. Higher is better.

start from evaluating 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.

IV. Analyzing Existing Hardware

We apply the three pillars of statistical robustness on an existing hardware, the Stochastic Processing Unit (SPU) described in Sec. II-C.

A. Methodology

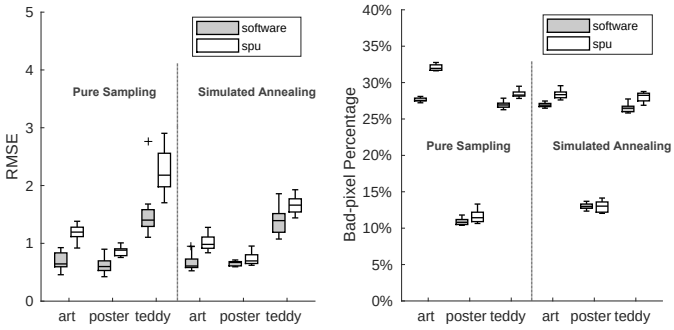
In this work we consider a single SPU, as it is sufficient to explore the statistical robustness questions. We primarily utilize MATLAB for both the FP64 software and for a functionally equivalent SPU simulator. Importantly, we also have SPU implementations in Verilog, Chisel, and HLS all with verified results. We choose stereo vision and motion estimation as our test applications. The concept of applying MRF Gibbs Sampling on motion estimation is similar to stereo vision as described in Sec. II-A, except the output label is a 2D motion vector of a pixel, indicating where the pixel moves to in the next frame. We pick three datasets from Middlebury [1], [16] for each application, as in the previous work [20]. We also considered, but omit, image segmentation since it converges too fast to produce meaningful statistical measurements. Recall the SPU supports two operating modes (sampling and optimization). 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. For brevity, we mainly present stereo vision results and summarize motion estimation results. More can be found elsewhere [21].

B. Results Analysis

1) 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. Fig. 3 shows an example ESS per random variable in stereo vision *teddy* dataset. Red regions indicate the random variables have zero variance, and thus considered inactive and ESS cannot be calculated. Due to truncating small probabilities to zero, the SPU has more inactive random variables (26.2%) than the software (1.4%). Zero variance means the probability of a possible label is large enough that all random samples pick the same label, which can indicate convergence. The variance of corresponding inactive random variables in software are consistently small, indicating the random variable are likely to consistently pick the same label as well—a concentrated distribution. Therefore, high inactive percentage does not necessarily imply bad result quality.

Fig. 4a shows the ESS arithmetic mean for a single MCMC run per dataset in stereo vision. We verify that different runs have small ESS differences (< 6). The mean "overall" ESS eliminates the random variables with zero variance in software and hardware, respectively. Fig. 3 reveals that the inactive regions in the SPU (red) correspond to the high ESS regions in the software due to small but non-zero variance (yellow), and thus overall ESS is biased toward software. Therefore, we also report the mean "active" ESS which only includes the regions with non-zero variances in both 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 variances needs to be evaluated and we are actively looking for methods to account for these regions. The software has 1.1-1.4× higher active ESS than the SPU in stereo vision and around 1.2× in motion estimation (not shown). This implies the SPU needs to run 1.1-1.4× iterations to reach the same active ESS as the software.

2) 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 et al. [7]. Fig. 4b shows the stereo vision results. The number of iterations is normalized with respect to SPU runs. 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 (not



(a) RMSE (b) End-point result quality

Fig. 5. Stereo vision goodness of fit results. Lower is better. Solid boxes show the range from 25th to 75th percentile. Horizontal lines inside boxes are the medians. Whiskers include the range of data not considered as outliers (within 1.5× interquartile range). Pluses are outliers.

shown). More than 99.5% of random variables with $W = 0$ in 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 [19], [20] shows that a pipeline with the same data interface provides the speedups of at least 2.8× and up to 84×.

3) Goodness of Fit: Fig. 5a 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. Whiskers of the software and the SPU overlap in all stereo vision benchmarks, suggesting close results. The software and the SPU produce closer results in simulated annealing optimization mode. Fig. 5b shows the end-point result quality using ground truth data and application metrics. Whiskers of the software and the SPU overlap except for *art* in stereo vision in sampling mode. Software and SPU whiskers overlap in optimization mode, indicating the difference in end-point result quality is very small. This is consistent with the single-run results in Tab. I. 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, motion estimation benchmark *dimetrodon* has consistently lower end-point result error in the SPU than in FP64 software, but has higher RMSE in the SPU (see elsewhere [21]). This result indicates two insights: 1) software with higher precision does not necessarily produce better end-point result quality, and 2) a higher RMSE compared to software does not always indicate worse end-point result quality. 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 three pillars beyond end-point result quality is necessary to evaluate correctness.

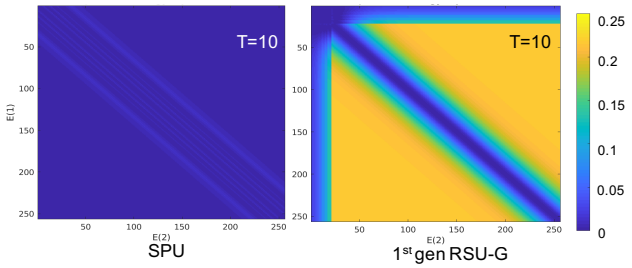


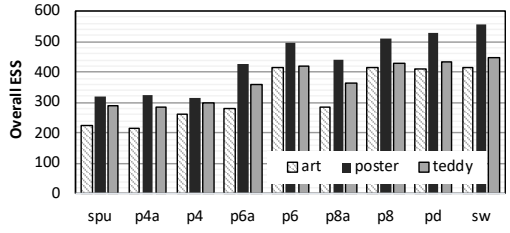
Fig. 6. Jensen-Shannon Divergence comparison between the SPU (left) and a previously proposed accelerator 1st-gen RSU-G (right).

We analyze the Jensen-Shannon Divergence of SPU relative to software with FP64 probability representation. Our goal is to provide insights about why hardware exhibits good or bad application end-point results and how it may perform 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 $E(i)$ from 0 to 255 in integer, corresponding to arbitrary data inputs, Fig. 6 plots JSD in $T = 10$ (initial T in optimization mode) with two different SPU microarchitectures: 1) the SPU described in Sec. II-C and 2) an early design 1st generation RSU-G [19] that was shown to lack sufficient precision and dynamic range [20]. These results clearly show the problems with the 1st-gen RSU-G. The more recent SPU JSDs are negligible in most energy inputs (blue regions), whereas the 1st-gen RSU-G has high JSD for many inputs (yellow) and becomes worse when T decreases (not shown). 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.

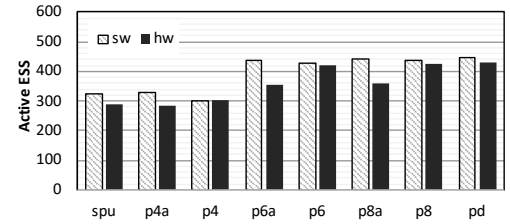
V. Design Space Exploration: A Case Study

The previous section shows that architectural optimizations might have negative influence on the statistical robustness, even though producing comparable end-point results to FP64 software. *Can we achieve desirable statistical robustness without the commensurate overhead of FP64?*

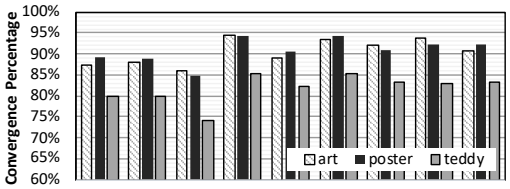
The SPU pipeline (Fig. 2) has several design parameters related to bit precision that potentially influence statistical robustness, including energy $E(i)$ and $E_s(i)$, scaled and truncated probability $p_{tr}(i)$, and RNG output bits. We fix energy $E(i)$ and $E_s(i)$ at 8 bits based on previous work [13], [20]. 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, 6, 8-bit $p_{tr}(i)$s. The influence of RNG output bits is small compared to $p_{tr}(i)$ and we find a 19-bit LFSR with 12-bit RNG outputs does not reduce the statistical robustness or result quality across all design points. The SPU truncates all $p_{tr}(i)$s to the nearest 2^n values (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 achieve a stall-free design. We evaluate the statistical robustness of each scaled probability design point with and without 2^n approximation. The above



(a) Overall ESS

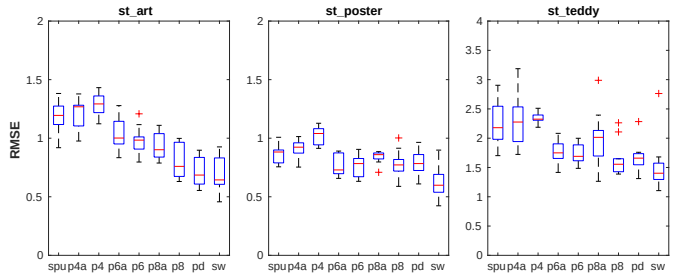


(b) Active ESS in *teddy*

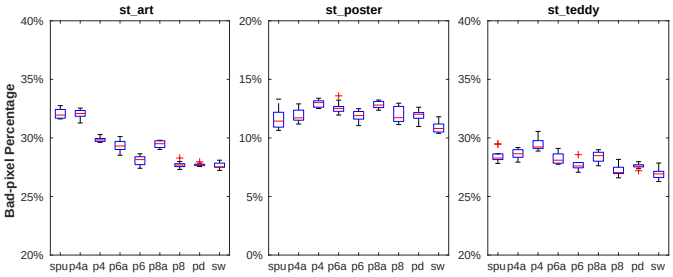


(c) Convergence percentage

Fig. 7. Stereo vision sampling quality (a), (b) and convergence diagnostic (c) in the design points.



(a) RMSE



(b) End-point result

Fig. 8. Stereo vision RMSE and end-point results in the design points (sampling mode).

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; higher precision requires more area and power affecting the number of SPU units in systems with limited area/power budget.

A. Evaluating Design Parameters

Figs. 7 and 8 show our design space results. For brevity, we show only stereo vision results 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 a FP64 back-end for probability conversion and discrete sampling ("pd").

1) Sampling Quality: Fig. 7a shows the overall ESSs (respectively omit random variables with zero variances for each design) increase when more bits are added, partly as a result of fewer random variables with zero variances. 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. Fig. 7b shows the active ESS for the *teddy* dataset. Recall active ESS masks out the random variables inactive in either software or the 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. Results for motion estimation (omitted) are similar.

2) Convergence Diagnostic: Fig. 7c shows the convergence percentage 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 (results not shown).

3) Goodness of Fit: Fig. 8a shows 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. Application end-point results in Fig. 8b exhibit the same trends. All designs produce comparable result quality to the software in optimization mode (not shown). Motion estimation results are summarized elsewhere [21]. 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 to the software.

B. Resource Usage

1) FPGA Resource Usage: We evaluate three different implementations of the SPU ("spu") on an Intel Arria 10 FPGA: 1) an optimized hand-written Verilog implementation with 4-bit scaled probability, 2) a High-Level Synthesis (HLS) implementation (HLS-int) that matches the hand-written Verilog (but using HLS basic integer data-types), and 3) an HLS

TABLE II
AREA AND POWER 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

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. HLS-int (frequency 369MHz) is close to the Verilog (374MHz) implementation in terms of performance both with 1 initiation interval cycle, but consumes more resources (3.7× ALMs, 7.3× memory, and 2.5× DSPs. See [21]). HLS-fp consumes 13.7× ALMs, 33.7× memory, 6.3× DSPs compared to Verilog and most importantly performs remarkably worse due to its lower frequency (320MHz) and throughput (3 initiation interval cycles) caused by the FP addition. Clearly, naively implementing the SPU in FP32 consumes too much resources and significantly reduces the performance benefits. A human-designed architecture is needed to improve efficiency.

2) ASIC: We estimate the ASIC area/power for various design points. Circuits elements written in Chisel are synthesized in a predictive 15nm library [14] using Synopsys Design Compiler. Memory elements (FIFOs and LUTs) are estimated using Cacti 7 in 22nm technology, the smallest supported technology. Tab. II summarizes the results. Total area/power numbers are the sum of 15nm circuitry and 22nm memory elements. Power is estimated at 1GHz. All designs can run up to 3.3GHz, bounded 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.

VI. CONCLUSION

Domain-specific accelerators require correctness evaluation. 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 work takes a first step toward defining metrics and a three-pillar framework for evaluating correctness of probabilistic accelerators beyond application end-point result quality. We apply the three pillars on an existing hardware accelerator and surface design issues that cannot be revealed by only using end-point result quality. The three pillars guide the design space exploration and achieve considerable improvements in the statistical robustness by slightly increasing the bit precision.

ACKNOWLEDGEMENTS

This project is supported in part by Intel, the Semiconductor Research Corporation and the National Science Foundation (CNS-1616947).

REFERENCES

[1] S. Baker, D. Scharstein, J. P. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, no. 1, pp. 1–31, Mar 2011.

[2] S. S. Banerjee, Z. T. Kalbarczyk, and R. 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*, ser. ASPLOS '19. New York, NY, USA: ACM, 2019, pp. 515–528.

[3] S. P. Brooks and A. Gelman, "General methods for monitoring convergence of iterative simulations," *Journal of Computational and Graphical Statistics*, vol. 7, no. 4, pp. 434–455, 1998.

[4] W. Cheng, L. Ma, T. Yang, J. Liang, and Y. Zhang, "Joint lung ct image segmentation: a hierarchical bayesian approach," *PloS one*, vol. 11, no. 9, 2016.

[5] M. K. Cowles and B. P. Carlin, "Markov chain monte carlo convergence diagnostics: a comparative review," *Journal of the American Statistical Association*, vol. 91, no. 434, pp. 883–904, 1996.

[6] R. Ge, H. Lee, and A. Risteski, "Simulated tempering langevin monte carlo ii: An improved proof using soft markov chain decomposition," *arXiv preprint arXiv:1812.00793*, 2018.

[7] A. Gelman and D. B. Rubin, "Inference from iterative simulation using multiple sequences," *Statistical science*, vol. 7, no. 4, pp. 457–472, 1992.

[8] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, no. 7553, pp. 452–459, 2015.

[9] J. E. Johndrow, J. C. Mattingly, S. Mukherjee, and D. Dunson, "Optimal approximating markov chains for bayesian inference," *arXiv preprint arXiv:1508.03387*, 2015.

[10] G. G. Ko, Y. Chai, R. A. Rutenbar, D. Brooks, and G.-Y. Wei, "Flexgibbs: Reconfigurable parallel gibbs sampling accelerator for structured graphs," in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2019, pp. 334–334.

[11] J. Lin, "Divergence measures based on the shannon entropy," *IEEE Transactions on Information theory*, vol. 37, no. 1, pp. 145–151, 1991.

[12] S. Liu, G. Mingas, and C. Bouganis, "An exact mcmc accelerator under custom precision regimes," in *2015 International Conference on Field Programmable Technology (FPT)*, Dec 2015, pp. 120–127.

[13] V. Mansinghka and E. Jonas, "Building fast bayesian computing machines out of intentionally stochastic, digital parts," *arXiv preprint arXiv:1402.4914*, 2014.

[14] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech, and J. Michelsen, "Open cell library in 15nm freepdk technology," in *Proceedings of the 2015 Symposium on International Symposium on Physical Design*, ser. ISPD '15. New York, NY, USA: ACM, 2015, pp. 171–178.

[15] F. A. Saad, C. E. Freer, N. L. Ackerman, and V. 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*, 2019, pp. 1640–1649.

[16] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002.

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

[18] D. Vats, J. M. Flegal, and G. L. Jones, "Multivariate output analysis for markov chain monte carlo," 2015.

[19] S. Wang, X. Zhang, Y. Li, R. Bashizade, S. Yang, C. Dwyer, and A. R. Lebeck, "Accelerating markov random field inference using molecular optical gibbs sampling units," in *Proceedings of the 43rd International Symposium on Computer Architecture*, ser. ISCA '16. Piscataway, NJ, USA: IEEE Press, 2016, pp. 558–569.

[20] X. Zhang, R. Bashizade, C. LaBoda, C. Dwyer, and A. R. Lebeck, "Architecting a stochastic computing unit with molecular optical devices," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, June 2018, pp. 301–314.

[21] X. Zhang, R. Bashizade, Y. Wang, C. Lyu, S. Mukherjee, and A. R. Lebeck, "Beyond application end-point results: Quantifying statistical robustness of mcmc accelerators," *arXiv preprint arXiv:2003.04223*, 2020.