Optimizing Markov Random Field Inference via Event-driven Gibbs Sampling

Ramin Bashizade, Xiangyu Zhang, Sayan Mukherjee, Alvin R. Lebeck
Duke University
Durham, NC, USA
ramin@cs.duke.edu, xiangyu.zhang@duke.edu, sayan@stat.duke.edu, alvy@cs.duke.edu

***Abstract*—Markov Random Field (MRF) is a powerful graphical model for representing a wide range of applications in statistical machine learning. MRF encodes the conditional dependence among random variables (RVs). One approach to solving problems represented by MRF is using probabilistic algorithms such as Gibbs sampling. These methods go through all RVs in MRF and update them iteratively, until converged to the final result. In this work, we build on three observations to skip updating RVs that cannot change their value during the current iteration, hence avoiding unnecessary work: i) after the warm-up period, most RVs tend to not change values very often, ii) an RV can only change its value if either it has a non-concentrated probability distribution function (PDF), or at least one of the RVs on which it is conditionally dependent has changed its value, and iii) approximation techniques used for hardware specialization make it increasingly likely that RVs have concentrated PDFs. Therefore, we introduce event-driven Gibbs sampling which only updates RVs when necessary. Our analysis shows significant speedup can be gained for two image analysis applications.**

I. Introduction

Markov Random Field (MRF) is a powerful graphical model for representing a wide range of applications in statistical machine learning [6], [8], [12]. MRF encodes the conditional dependence among random variables (RVs). One approach to solving problems represented by MRF is using probabilistic algorithms such as Gibbs sampling [3]. These methods go through all RVs in the MRF model and update them iteratively, until converged to the final result. The update process relies on sampling from probability distributions, which is often computationally intensive. This is due to the significant overhead of sampling in conventional processors [14]. Furthermore, Gibbs sampling at first appears to be a sequential algorithm because updating each RV depends on the latest value of all other RVs in the model.

Despite these challenges, the statistical properties of these algorithms, especially their interpretability, make them an attractive alternative approach to deep learning. Therefore, developing solutions to accelerate these algorithms, such as hardware specialization and extracting parallelism from the model, are of significant importance. To this end, deploying pseudo-random number generation can help reduce the sampling inefficiency [15]. Additionally, to avoid the overhead of serial execution, one can take advantage of the conditional independence of RVs, i.e., develop a schedule which allows multiple independent RVs be updated in parallel [3]–[5], [13].

Orthogonal to the aforementioned techniques to optimize the hardware implementation of Gibbs sampling, we can adopt algorithmic optimizations to avoid performing unnecessary work and thus, further speedup the execution. In this work, we build on three observations to eliminate updating RVs that are guaranteed to not change their values during the current iteration: i) after the warm-up period, most RVs tend to not change values very often, ii) an RV can only change its value if either it has a non-concentrated probability distribution function (PDF), i.e., it has non-zero probabilities of taking on multiple values, or at least one of the RVs on which it is conditionally dependent has changed value, i.e., its PDF has changed, and iii) approximation techniques used for hardware specialization make it increasingly likely that RVs have concentrated PDFs. Therefore, we introduce event-driven Gibbs sampling (EDGS). In this scheme, optimized queues are used to keep track of RVs that must be updated. To be more specific, a RV is added to the queue if i) another RV on which it is conditionally dependent changes its value, or ii) it does not have a concentrated PDF. Our evaluations show $22.2\% - 57.7\%$ speedup can be gained for two image analysis applications.

The rest of the paper is organized as follows. Section II provides a brief background about probabilistic algorithms and the motivation for our work. EDGS is explained in Section III. Section IV presents RV range queue, an optimized queue structure that complements EDGS. Evaluation results are presented in Section V. Finally, Section VI discusses the future directions and concludes the paper.

II. Background and Motivation

A. Probabilistic Algorithms

Bayesian inference combines new evidence and prior beliefs to update the probability estimate for a hypothesis. Consider D as the observed data and X as the latent random variable. The prior distribution of X is $p(X)$ and $p(D \mid X)$ is the probability of observing D given a certain value of X. In Bayesian inference, the goal is to retrieve the posterior distribution $p(X \mid D)$ of the random variable X when D is observed. As the dimensions of D and X increase, it often becomes difficult or intractable to numerically derive the exact posterior distribution $p(X \mid D)$.

One approach to solving these inference problems is to use probabilistic Markov chain Monte-Carlo (MCMC) methods,

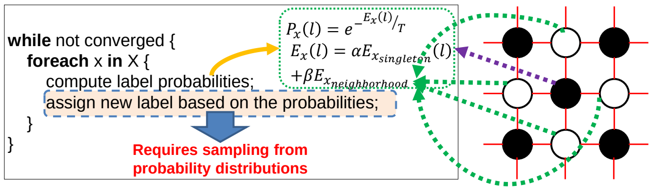

while not converged {
foreach x in X {
compute label probabilities;
assign new label based on the probabilities;
}
}
Requires sampling from probability distributions
$P_x(l) = e^{-E_x(l)/T}$
$E_x(l) = \alpha E_{x_{singleton}}(l)$
$+\beta E_{x_{neighborhood}}$

Fig. 1. Markov Chain Monte Carlo algorithm (left) for Markov Random Field (right) inference. Note that sampling is performed in the inner loop.

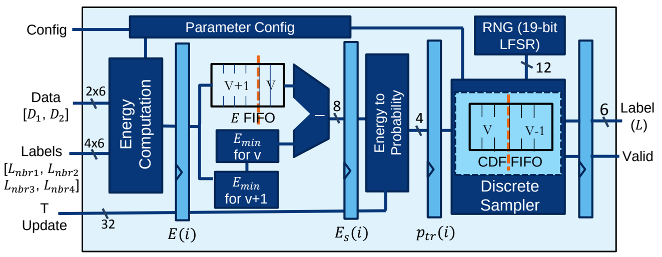


Fig. 2. Microarchitecture of the SPU reproduced from [16].

e.g., Gibbs sampling [3], that converge to an exact solution by iteratively generating samples for RVs. Figure 1 shows this process. The detailed explanation is provided elsewhere [3].

In practice MCMC becomes inefficient for many problems that have high dimensionality (i.e., many RVs) and complex structure. It can require many iterations before convergence, and the inner loop in Figure 1 includes generating samples from probability distributions which is computationally expensive for conventional processors [14] and thus, a specialized accelerator is needed to address these shortcomings.

B. First-order Markov Random Field

Markov Random Field (MRF) is a graphical model for representing a wide range of applications in statistical machine learning. MRF encodes the conditional dependence among RVs. Figure 1 (right) illustrates an example model and its connection with the Gibbs sampling algorithm. In the model, each RV depends on its four immediate neighbors. Due to this structure, the RVs can be divided into two regions so that all of RVs in each region are conditionally independent. This enables us to generate a chromatic schedule to update all RVs in each region in parallel [3]–[5], [13].

C. Stochastic Processing Unit

Zhang et al. propose a Gibbs sampling function unit, called Stochastic Processing Unit (SPU), that utilizes specialization and pseudo-random number generation to accelerate MCMC computations [16]. Figure 2 demonstrates the microarchitecture of this function unit. It is composed of four main stages, namely energy computation (equation in Figure 1), dynamic energy scaling (Equation 1), energy to probability conversion (Equations 2 and 3), and sampling.

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

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

$$P_{tr}(l) = \lfloor 2^{\lfloor \log_2 P_s(l) \rfloor} \rfloor \tag{3}$$

Energy computation takes the singleton data and neighbor labels, all 6-bit values, and computes the energy of a possible label, $E(l)$ in Figure 1, where α and β are application parameters. Next, $E(l)$ is dynamically scaled by subtracting the minimum energy of all labels from it to maximize the dynamic range. Energy values (raw and scaled) are 8-bit unsigned integers. The scaled energy $E_s(l)$ is then converted to a scaled probability represented by a 4-bit unsigned integer. The original probability (real number in $[0, 1]$) is calculated using $exp(-E_s(l)/T)$, in which T is a per iteration parameter. To avoid using floating-point function units, though, the probability is scaled using Equation 2, and then truncated using Equation 3. $P_{bits} = 4$ ensures the scaled probability is in $[0, 16)$, which allows for representing the number using 4 bits. This approximation technique, called probability cut-off, causes the scaled probabilities less than one to be rounded down to zero, and is one of the main factors for many of RVs having concentrated PDFs. Afterward, Equation 3 approximates the scaled probabilities to the nearest power of two, i.e., $P_{tr} \in \{0, 1, 2, 4, 8\}$. The possible values of $P_{tr}(l)$ can be pre-computed and stored in a look-up table (LUT). These values must be updated if T changes.

The last stage generates a sample per RV based on $\{P_{tr}(0), P_{tr}(1), ..., P_{tr}(L-1)\}$, where L is the number of labels, using the least significant twelve bits of a 19-bit Linear Feedback Shift Register (LFSR) to implement the inverse transform sampling. The SPU's throughput is one RV update per L cycles, if it receives the appropriate input (i.e., neighborhood labels and singleton data) at every cycle.

The SPU can be used in one of two modes: i) pure sampling, or ii) optimization. The main difference between these two modes is that in pure sampling, the parameter T is the same for all Gibbs sampling iterations, whereas in optimization (simulated annealing), T gradually decreases to help faster convergence to a final solution [3]. Therefore, in optimization mode, as the iterations proceed, picking the labels with higher energies becomes less and less likely, thus making the PDF more concentrated. Our goal in Section III is to detect and avoid updating the RVs that have concentrated PDFs and unchanged neighbors, consequently speeding up the execution of the algorithm.

D. Stable Random Variables

As the execution of Gibbs sampling algorithm makes progress, RVs gradually converge to their final labels. This process is further accelerated in the optimization mode. Therefore, it becomes unnecessary to update all RVs at every iteration because some of them simply cannot change their label. Figure 3 shows the number of times RVs change labels normalized to the total number of times they are updated in three input sets for two applications of stereo vision and motion estimation each. Based on the figure, only $22\% - 46\%$ of RV updates

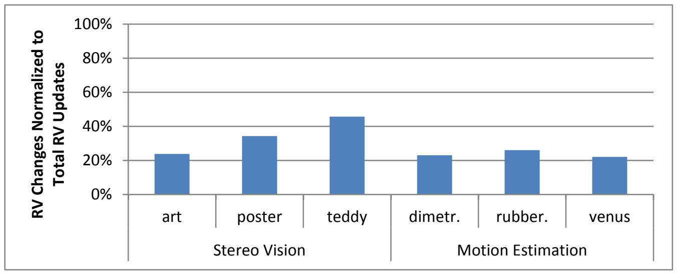
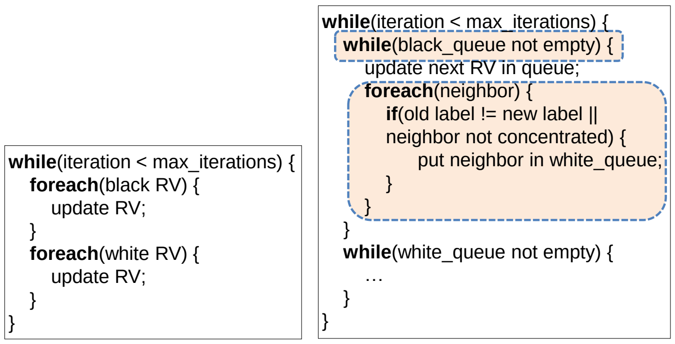


Fig. 3. Number of times that RVs change labels normalized to the total number of times they are updated, for stereo vision and motion estimation.



```
while(iteration < max_iterations) {
    foreach(black RV) {
        update RV;
    }
    foreach(white RV) {
        update RV;
    }
}
```

(a)

```
while(iteration < max_iterations) {
    while(black_queue not empty) {
        update next RV in queue;
        foreach(neighbor) {
            if(old label != new label ||
            neighbor not concentrated) {
                put neighbor in white_queue;
            }
        }
    }
    while(white_queue not empty) {
        …
    }
}
```

(b)

Fig. 4. Comparison of update procedure when (a) all RVs are updated at each iteration, and (b) when EDGS is adopted. The shaded regions highlight the additional work that must be done in EDGS compared to the baseline. The process for white RVs in EDGS is omitted for the sake of brevity.

result in changing labels. Consequently, there is opportunity for up to $54\% - 78\%$ speedup. However, not all of this speedup can be gained. If a RV simply *does not* change its label does not mean that it *cannot* do so. Some other conditions must be met for us to be able to skip updating RVs, which is explained in Section III.

III. Event-driven Gibbs Sampling

A RV can only change its label if i) it has more than one label with non-zero probability (i.e., non-concentrated PDF), or ii) at least one of the RVs on which it is conditionally dependent changes its label and thus, changing this RV's PDF. In other words, if a previous computation resulted in a PDF concentrated on one label, which is likely due to the probability cut-off technique in the SPU, the PDF is going to remain that way until something in its neighborhood changes. Therefore, we update variables in two cases, i) if at least one of its neighbors change, or ii) it did not have a concentrated PDF to begin with. We call this optimization event-driven Gibbs sampling (EDGS). This technique is similar to vertex programming in graph algorithms [9], [10], but we customize it for the context of MRF inference with Gibbs sampling.

We utilize specialized queues to keep track of RVs that must be updated. The queue's structure is explained in detail

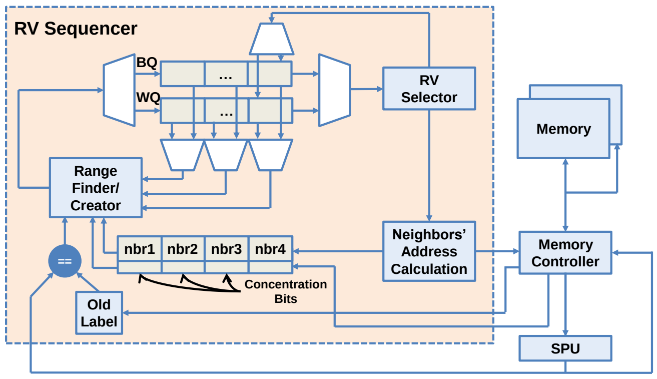


Fig. 5. Schematic of an accelerator with RQs implementing EDGS.

in Section IV. We use two queues to hold RVs in alternate rounds of the chromatic schedule described in Section II-B, i.e., one queue for holding black RVs and another for white RVs.

Figure 4 compares EDGS and the plain vanilla update scheme that updates all RVs at every iteration. The shaded regions show the extra work that must be done in EDGS to read RVs from the queues and write new RVs to them. In order to simplify writing new RVs to the queue, we check the necessary conditions when an RV's neighbors are being updated. This works because the conditional independence in first-order MRF is mutual. Therefore, while updating a RV, we compare its new label with its old label, and if the two labels do not match, we put all the neighbors in their corresponding queue. Additionally, we have to check if the neighbors have concentrated PDFs. To do so, we add an extra bit to each RV which is set only if the RV has a concentrated PDF.

IV. Random Variable Range Queue

Keeping RVs that need to be updated in queues simplifies selecting the next RV to update compared to a design in which such RVs are marked using an extra bit. However, the area overhead of additional queues capable of potentially storing all RVs is significant. To mitigate this problem, we propose using range queues (RQs) instead. Entries in a RQ are ranges of RVs instead of individual RVs. For instance, N entries $1, 2, ..., N$ in an ordinary queue would be compressed to a single entry $[1 : N]$ in a RQ. The intuition is that since RVs that are close to each other are expected to exhibit similar behavior, they will form large enough ranges that result in significantly smaller queues. Even in the worst case when every other RV must form a separate range (because two consecutive RVs would fall in the same range), the queue capacity would not be higher than an ordinary queue.

Nevertheless, the queue size in a hardware implementation must be fixed at the design time. Fortunately, RQ's flexibility means performance gain can be traded off with queue capacity, which makes designing the RQ for correctness easier. Consider a scenario when a new RV must be added to the RQ, but the RQ has run out of space. In this case, we can always extend

TABLE I
Application parameters used in evaluations.

Motion Estimation						
	α	β	initial T	Labels	Size	Iterations
dimetrodon	6	6	400	49	584×388	3000
rubberwhale			500		584×388	
venus			400		210×190	
Stereo Vision						
	α	β	initial T	Labels	Size	Iterations
art	6	7	10	28	348×278	1000
poster				30	435×383	500
teddy				56	450×375	1000

the bounds of the range closest to the new RV such that it encompasses the RV. This is always valid because it is fine to update extra stable RVs. Other policies may be used as well, such as contracting the RQ by merging the two closest ranges together, but this will add more complexity to the control logic and could adversely affect the clock rate.

Figure 5 shows the schematic of the accelerator augmented with RQs to implement EDGS. First, inside RV sequencer, which is responsible for sequencing through RVs that must be updated, RV selector reads a range from the head of the appropriate RQ and after selecting one RV from it, updates its bounds. Then the addresses of the neighbors are calculated. These addresses are stored in registers, to be added to the alternate RQ if deemed necessary at the end of the update process, and also sent to the memory controller for accessing memory. The old label of the RV is also written to a register to be later compared with the new label and decide whether the neighbors must be updated. After the memory controller receives the neighbors' addresses, it reads the data from appropriate memory units and sends them to the SPU. It also sends concentration bits to the RV sequencer to be stored along neighbor addresses. Finally, once the update process is finished, the SPU sends the new label to the memory controller to be written back to memory, and to the RV sequencer. After comparing the new label to the old one, if any neighbors must be updated, the range finder/creator searches the appropriate RQ to find the proper index to put the neighbor at.

V. Evaluation

We use two image analysis applications to evaluate our design, namely stereo vision [2], [11] and motion estimation [7]. Stereo vision reconstructs the depth information of objects in a field captured from two cameras by matching the pixels between the two images. In motion estimation, the goal is to determine the motion vectors of pixels between two consecutive frames of a video.

We implemented the applications in C++ using double-precision floating point, and mimicked the SPU behavior by driving the probabilities of labels with energies less than $1/8$ of the maximum energy to zero (recall dynamic scaling and probability cut-off in Section II-C). We assume the RQs can accommodate as many ranges as necessary. Our preliminary

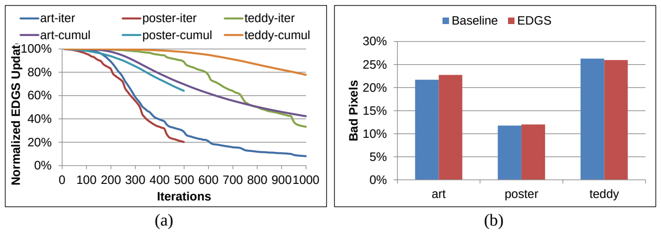


Fig. 6. Per iteration and cumulative number of RV updates when utilizing EDGS, normalized to the total number of updates in the baseline (a), and quality results of EDGS compared to the baseline (b), for stereo vision.

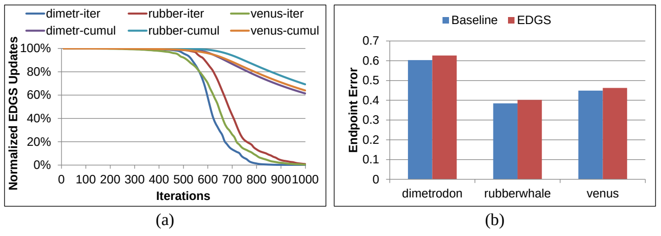


Fig. 7. Per iteration and cumulative number of RV updates when utilizing EDGS, normalized to the total number of updates in the baseline (a), and quality results of EDGS compared to the baseline (b), for motion estimation.

results show that limited-size RQs still provide notable performance gains, but more rigorous evaluations are future work.

We evaluate each application using three input image sets from Middlebury [1], [11]. Table I summarizes the parameters used for each input. Parameter names correspond to those in Figure 1. In order to compare the quality of the results, we compare against a MATLAB implementation which uses double-precision floating-point to perform the computations. We use bad-pixel (BP) percentage as the metric for stereo vision [11], and end-point error (EPE) as the metric for evaluating motion estimation results [7].

Figures 6 and 7 demonstrate performance and quality results for the two applications. Considering the random nature of the algorithm, as expected EDGS has almost no impact on the final result in terms of quality of the output. However, EDGS successfully reduces the cumulative amount of RV updates by $22.2\% - 57.7\%$ and $30.8\% - 38.6\%$ for stereo vision and motion estimation, respectively. Furthermore, the sharper decline in the number of per-iteration RV updates can be used as a metric to determine faster convergence.

VI. Conclusion and Future Work

Markov Random Field (MRF) is a powerful graphical model for representing numerous applications. It encodes the conditional dependence among random variables (RVs). Probabilistic algorithms such as Gibbs sampling [3] can be used to solve problems represented by MRF. Gibbs sampling is an iterative method which goes through all RVs in MRF and updates them until converged to the final result. The update process needs sampling from probability distributions, which is computationally intensive. Therefore, it is desirable to avoid unnecessary RV updates if possible. To this end, in this paper we propose event-driven Gibbs sampling (EDGS). In this scheme, we use optimized range queues (RQs) to keep track of RVs that must be updated. Our evaluations using two image analysis applications show that speedups of $22.2\% - 57.7\%$ can be gained for various input data sets.

As future work, we plan to refine the design of RQs to make them more amenable to hardware implementation. We also plan to design and implement the high-level architecture of an accelerator that takes advantage of EDGS and RQs.

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. [Online]. Available: https://doi.org/10.1007/s11263-010-0390-2

[2] S. T. Barnard, "Stochastic stereo matching over scale," *International Journal of Computer Vision*, vol. 3, no. 1, pp. 17–32, 1989.

[3] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 721–741, 1984.

[4] J. Gonzalez, Y. Low, A. Gretton, and C. Guestrin, "Parallel gibbs sampling: From colored fields to thin junction trees," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. Dudík, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 324–332. [Online]. Available: http://proceedings.mlr.press/v15/gonzalez11a.html

[5] E. M. Jonas, "Stochastic architectures for probabilistic computation," Ph.D. dissertation, Massachusetts Institute of Technology, 2014.

[6] M. Kim, P. Smaragdis, G. G. Ko, and R. A. Rutenbar, "Stereophonic spectrogram segmentation using markov random fields," in *2012 IEEE International Workshop on Machine Learning for Signal Processing*. IEEE, 2012, pp. 1–6.

[7] J. Konrad and E. Dubois, "Bayesian estimation of motion vector fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 9, pp. 910–927, Sept 1992.

[8] S. Z. Li, *Markov random field modeling in computer vision*. Springer Science & Business Media, 2012.

[9] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "Graphlab: A new framework for parallel machine learning," *arXiv preprint arXiv:1408.2041*, 2014.

[10] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 135–146. [Online]. Available: https://doi.org/10.1145/1807167.1807184

[11] D. Scharstein, R. Szeliski, and R. Zabih, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," in *Proceedings IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV 2001)*, Dec 2001, pp. 131–140.

[12] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE transactions on pattern analysis and machine intelligence*, vol. 30, no. 6, pp. 1068–1080, 2008.

[13] A. Terenin, S. Dong, and D. Draper, "Gpu-accelerated gibbs sampling: a case study of the horseshoe probit model," *Statistics and Computing*, vol. 29, no. 2, pp. 301–310, 2019.

[14] 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 *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, June 2016, pp. 558–569.

[15] 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.

[16] 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.