Accelerating Bayesian Inference on Structured Graphs Using Parallel Gibbs Sampling

Glenn G. Ko*, Yuji Chai*, Rob A. Rutenbar†, David Brooks*, Gu-Yeon Wei*
*Harvard University, Cambridge, USA
†University of Pittsburgh, Pittsburgh, USA
gko@seas.harvard.edu, yuc927@g.harvard.edu, rutenbar@pitt.edu, {dbrooks, guyeon}@eecs.harvard.edu

***Abstract*—Bayesian models and inference is a class of machine learning that is useful for solving problems where the amount of data is scarce and prior knowledge about the application allows you to draw better conclusions. However, Bayesian models often requires computing high-dimensional integrals and finding the posterior distribution can be intractable. One of the most commonly used approximate methods for Bayesian inference is Gibbs sampling, which is a Markov chain Monte Carlo (MCMC) technique to estimate target stationary distribution. The idea in Gibbs sampling is to generate posterior samples by iterating through each of the variables to sample from its conditional given all the other variables fixed. While Gibbs sampling is a popular method for probabilistic graphical models such as Markov Random Field (MRF), the plain algorithm is slow as it goes through each of the variables sequentially. In this work, we describe a binary label MRF Gibbs sampling inference architecture and extend it to 64-label version capable of running multiple perceptual applications, such as sound source separation and stereo matching. The described accelerator employs a chromatic scheduling of variables to parallelize all the conditionally independent variables to 257 samplers, implemented on the FPGA portion of a CPU-FPGA SoC. For real-time streaming sound source separation task, we show the hybrid CPU-FPGA implementation is 230x faster than a commercial mobile processor, while maintaining a recommended latency under 50 ms. The 64-label version showed 137x and 679x speedups for binary label MRF Gibbs sampling inference and 64 labels, respectively.**

***Index Terms*—bayesian inference, markov chain monte carlo, gibbs sampling, hardware accelerator, markov random field**

I. Introduction

The recent advancement in machine learning, especially deep learning, has been tightly coupled with the advancement of computing power, which enabled us to create and evaluate more sophisticated models that can span over billions of parameters [1]. Coupled with GPUs that are excellent at doing linear algebra operations such as matrix multiplication, there have been numerous state-of-the-art results achieved in many distinct domains, including computer vision, natural language processing and more. However, the power of deep learning has been mostly limited to discriminative models and supervised learning. Although newer networks for generative models such as, Generative Adversarial Networks [2] and Variational Autoencoders [3], [4], have been introduced, dealing with applications that has large amount of unlabeled data or with substantial uncertainty and noise is normally reserved for Bayesian models and inference.

Bayesian models pose challenges that are different from deep learning. They tend to have more complex structure, that may be hierarchical with dependencies between different distributions Without an easily-defined common computational kernel and hardware accelerators like GPU and TPU [5], and software frameworks such as Tensorflow [6] and PyTorch [7], it is hard to build and run models fast and efficiently. This work explores hardware accelerators for Bayesian models and inference which has growing interest and demand.

Probabilistic graphical models are Bayesian models described in a graph in which the nodes and the edges between them encode the dependency between the random variables [8]. They provide an intuitive way to describe Bayesian models, which could have very complicated dependencies and distributions among the variables. The process of questioning the model and obtaining the desired answer from it is referred to as inference and it is known to be computationally expensive, often intractable. In order to solve these large problems, approximate inference methods are employed. We accelerate one of the most commonly used Markov Chain Monte Carlo (MCMC) method called Gibbs sampling by offloading the compute to a parallel and parameterized architecture [9]. By employing chromatic scheduling of conditionally independent variables to parallel Gibbs samplers, a sequential Gibbs sampling process is computed in parallel.

We demonstrate a hybrid CPU-FPGA implementation for an unsupervised learning application, sound source separation, which is often called Blind Source Separation (BSS) when minimal or no information is provided about the sources or the mixing process [10]. Source separation is modeled as a 2-D Markov Random Field (MRF) and the FPGA accelerator is configured on a hybrid CPU-FPGA SoC to speed up the Gibbs sampling inference. An array of 257 parallel Gibb samplers is implemented on Xilinx Zynq UltraScale+ ZCU102-ES2 board. By partitioning the compute between CPU and FPGA for the sound source separation task and offloading the Gibbs sampling inference computation to the FPGA, we are able to achieve 230x speedup over running the application on the ARM Cortex-A53 while keeping the latency under 50ms. We further describe a 64-label implementation that is capable of running computer vision tasks such as stereo matching while achieving 137x over A53 on the binary labels MRF Gibbs sampling inference used

This work was supported by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA.

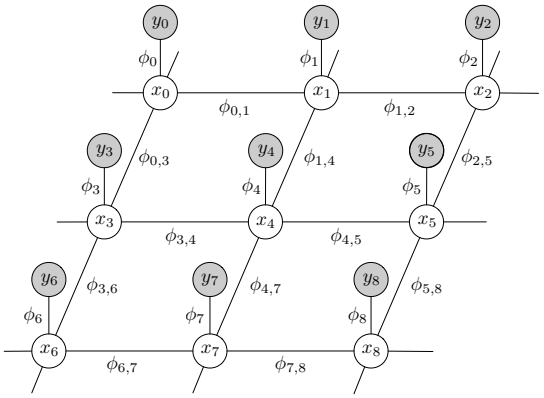

Fig. 1. The figure is showing nine nodes in a pairwise MRF where the white nodes and gray nodes each representing hidden and observed variables, x and y. The MRF represents a factorized distributions where ϕ_i and ϕ_i, j corresponds to factors that represent the relationship between the variables.

for source separation and 679x for the 64 labels.

In Section II, we quickly overview the model, inference, and the application discussed. Section III describes the parameterized architecture that accelerates Gibbs sampling on MRFs with chromatic scheduling of conditionally independent variables. Section IV discusses the hybrid CPU-FPGA SoC implementation of real-time streaming sound source separation. Section V extends the binary label MRF Gibbs sampling inference to 64 labels which extends the support for other perceptual applications such as stereo matching. Section VI lists other previous research efforts made to parallelize and accelerate Gibbs sampling inference. We conclude with final remarks and present future research directions in Section VII.

II. Background

A. Bayesian inference on Markov random field

A pairwise MRF with four-connected neighbors is most often used for labeling problems in computer vision [11]. Figure 1 shows the factorized distribution on a four-connected pairwise MRF. Given a node $i \in V$, the set of all nodes, the relationship between the label on the node x_i and the observed data y_i is represented as potential ϕ_i. The edge between node i and j, $(i, j) \in \eta$ represents the affinity between the neighboring nodes using potential $\phi_{i,j}$. The posterior distribution of MRF is the product of all node and edge factors:

$$P(x, y) = \frac{1}{Z} \prod_{i \in V} \phi_i(x_i, y_i) \prod_{i,j \in E} \phi_{i,j}(x_i, x_j) \qquad (1)$$

where Z is called a partition function or a normalizing constant; it is the sum of all possible values of ϕ. Following Gibbs distribution formulation, probability distribution in (1) is proportional to the sum of energy functions which is equivalent to taking the negative log of (1):

$$E(x, y) = \sum_{i \in \nu} \theta_i(x_i, y_i) + \sum_{i,j \in \varepsilon} \theta_{i,j}(x_i, x_j), \qquad (2)$$

where $\theta_i(x_i, y_i)$ and $\theta_{i,j}(x_i, x_j)$ are energy functions often referred to as data cost and smoothness cost respectively [12], [13]. To find the maximizing assignment of x in (1), we can instead perform Maximum A Posteriori (MAP) inference to find labels x that minimizes the energy function as follows:

$$x^* = \arg\max P(x|y) = \arg\min_x E(x|\theta) \qquad (3)$$

Algorithm 1 Gibbs sampling on MRF

Initialize $x^{(0)}$
for $t = 0$ to T **do**
  for $i = 0$ to N **do**
    $x_i^{(t+1)} \sim P(x_i | x_{north}^{(t)}, x_{south}^{(t)}, x_{west}^{(t)}, x_{east}^{(t)})$
  end for
end for
return x

B. Gibbs sampling inference on MRF

Gibbs sampling is a popular Markov chain Monte Carlo (MCMC) inference method to find the set of labels x that would minimize the energy function. The Gibbs sampler is constructed by iteratively sampling each variable in the MRF given all the neighboring variables as shown in Algorithm 1. The algorithm shows the 4-connected neighbors observed for the sampled node as all the other variables are conditionally independent given the neighbors.

C. Gibbs sampling for Source Separation

While there are many different perceptual applications with varying number of possible labels for each of the nodes, we initially focused on binary label MRFs and Gibbs sampling inference on them. Applications such as image segmentation [14] and sound source separation [15] can be solved using binary MRFs. Sound source separation application modeled as a binary MRF can be solved using Gibbs sampling inference.

Sound source separation consists of multiple functions to enable unsupervised learning of Gaussian parameters through expectation-maximization (EM). The means and variance of each of the sources are updated after Gibbs sampling inference and repeated until convergence. A pair of sound mixture spectrograms are used to compute an auditory cue commonly used for sound localization called interaural level difference (ILD), A, which is the log ratio of the energy values seen at each of the time-frequency point of the spectrograms. The data cost is defined as a mixture of two Gaussian distributions and the smoothness cost is a simple $L1$ norm to penalize the having different labels between the neighbors. The computed costs are used to perform Gibbs sampling inference. The source separation using Gibbs sampling has two nested loops, where the inner loop performs Gibbs sampling for N nodes in the MRF for T iterations and the outer EM loop that updates the MRF distribution using improved Gaussian parameters through unsupervised learning.

Source separation using Gibbs sampling steps:

1) Get ILD values for an audio frame.
2) Generate data and smoothness costs using Gaussians.

3) Perform T iterations of Gibbs sampling on N nodes.
4) Generate binary masks and update Gaussians.
5) Repeat 2) to 4) for EM iterations.
6) Go back to 1) for a new audio frame.

III. Flexible Gibbs Sampling Architecture for MRF

The 32-bit fixed point datapath for the binary MRF Gibbs sampler, shown in Figure 2, consists of three stages: total cost computation, generation of probability distributions, and sampling a new label. The inputs are data costs for the sampled node and the labels of the neighboring nodes. After each Gibbs sampling iteration, the sampler produces a new label that is stored back to the labels memory.

A. Three-stage datapath

1) Total cost computation: For 2-D grid MRFs with smoothness costs that do not depend on the input, the actual values of the smoothness costs computed will not vary from node to node and can be found statically. We can offload the total cost computation to the accelerator by storing pre-computed smoothness cost values for all permutations of label pairs into local smoothness cost tables in each of the Gibbs samplers. This optional table trades logic area with lower latency, which can be desirable for perceptual applications where latency is one of the most important design specifications. Gibbs samplers compute energy for each node and evaluate probability distribution as shown in Figure 2. It shows how the data costs for all possible labels summed with smoothness cost table values indexed by the neighboring labels to generate a total cost.

2) Generation of conditional distributions: Using the total cost found in the previous stage, the conditional distributions are generated using the Boltzmann distribution formulation, which takes the total energy, scales it by a constant β, and finds the negative exponential value. The negative exponential function unit uses Taylor polynomials and LUTs for error values with 16 bits of precision. The result is the conditional distribution given the label value and it is accumulated to find cumulative sums of these distributions. The cumulative sums for all possible labels are stored in a FIFO.

3) Sampling new label: Once the cumulative sums for iterating through all possible labels have been computed and stored in a FIFO, these values are compared against $SamplingProbability$, a threshold for sampling a new label for the node, obtained by scaling an uniform random number by the final sum of conditional distributions. A 32-bit uniform random number generator with 43-bit LFSR and 37-bit CASR [16] was implemented to generate the uniform random number needed. $SamplingProbability$ is compared against each of the cumulative sums until a sum larger than the sampling probability is found. The label corresponding to the sum is selected as a new label for the variable.

B. Parallel Gibbs sampling using 2-color scheduling

It is worth noting that in an MRF sampling a new label for a node in a graph involves looking at the labels for the neighboring nodes, the nodes that are connected to the observed node by edges, but not the others. By applying graph coloring, we can create a checkered pattern 2-coloring of the nodes in the MRF, where nodes of one color is conditionally independent of all the other nodes of the same color. This allows us to sample all the same colored nodes in parallel across two phases for each color while preserving ergodicity, therefore, resulting in targeted stationary distribution [17]. This reduces the runtime from $O(n)$ where n is the number of labels to $O(k)$ where k is the number of colors, as long as we have enough samplers to process all the nodes of same color concurrently. The programmable hardware scheduler assigns conditionally independent variables to parallel Gibbs sampler array. The scheduler also supports fine-grain scheduling for each samplers to avoid waiting for all the other samplers to finish but to sample ahead and start executing the next sampling iteration minimizing the latency of the sampling.

C. Computation by row

Ideally, we would want to have as many processing elements as the number of nodes in a graph. However, as such idealism is unlikely for most case given resource constraints, the architecture instead assumes enough number of processing elements per row of an MRF and chromatically schedules conditional independent variables of same color in parallel to the array of processing elements row by row.

To perform chromatic scheduling of Gibbs sampling, the MRF is divided into two dark and light colors. All the nodes of same color in the row will be sampled concurrently and the other color together in the next phase. This allows us to use one address space to address the row and read all the costs and labels required for that row and also two other address spaces, one above and one below, to find labels for the neighboring values as shown in Figure 3. Any other shapes, such as square tiles, will have more complicated indexing or redundant reads from the memory compared to processing row by row.

For a complete iteration of Gibbs sampling, Gibbs samplers will first process dark nodes in each row simultaneously and finish all dark nodes in a row-by-row fashion. After all dark nodes are updated with new labels, then all light nodes are updated with new labels in a similar row-by-row fashion. To process each Gibbs node, four of its neighbors labels are required to find its smoothness cost. Neighbors diagram for light and dark nodes are shown in the first row of Figure 3. In the second row, the neighbors locations for a row of same color nodes are shown. To sample dark nodes of Row j, light nodes labels from Row $j-1$, Row j and Row $j+1$ are needed. This particular way of label accessing pattern facilitated us to store labels by their colors and their rows in the buffer. This minimizes the number of addresses to read concurrently in a single cycle, avoiding issues that could rise with limited number of ports in memory elements.

IV. Source separation on a hybrid CPU-FPGA SoC

Using the architecture for binary Gibbs sampling inference on MRF, we built a hybrid CPU-FPGA implementation of source separation on Xilinx Zynq UltraScale+ ZCU102 board, which is a SoC with a quad-core ARM Cortex-A53 and FPGA

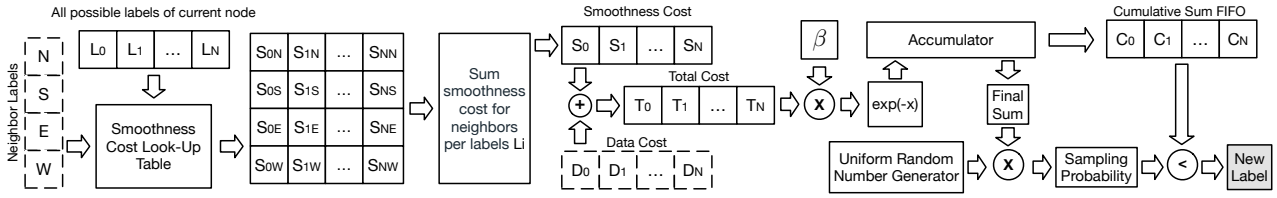

Fig. 2. The datapath for Gibbs sampler is shown with inputs in dotted lines. In the first stage, the sampler takes in a node's neighbors' labels and its data cost as inputs, to generate probabilities for each possible label values. Then in second stage, the sampler computes Boltzmann distribution using the total cost and stores the cumulative sums of the distribution in a FIFO. The $SamplingProbability$ is found using the final cumulative sum scaled by the uniform random number generated from a pseudo-random number generator. The cumulative sums in the FIFO are compared against the $SamplingProbability$ until the cumulative sum for the label is larger than the sampling probability or the last label has been reached, and the corresponding label is returned as the new label.

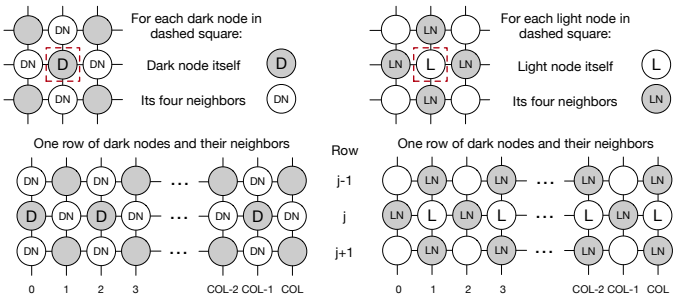

Fig. 3. This diagram shows the neighbors for nodes graph. All nodes are divided into dark and light nodes for 2-color chromatic Gibbs sampling. Neighbors of a single dark or light node is shown in the first row and neighbors of a row of dark or light nodes is shown in the second row.

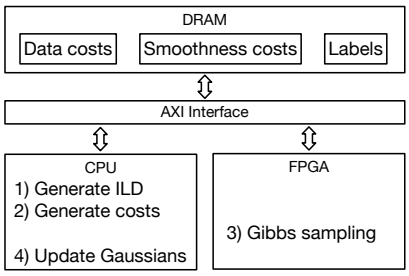

Fig. 4. Compute partition between CPU and FPGA for streaming sound source separation task and the date stored in DRAM. From the spectrograms obtained from sound mixtures, ILD is generated and converted to data cost. The smoothness cost is also pre-computed and labels are randomly generated.

logic. The FPGA design was coded in Verilog and synthesized using Xilinx Vivado. From the list of functions for running source separation mentioned in Section II-C, Figure 4 shows the compute fabric in which each of the functions are executed.

From the input audio frame, ILD values are computed on the CPU. Using ILD values and Gaussian distribution parameters, the data costs for the audio frame are computed and stored in DRAM. We use $L1$ norm for the smoothness cost, which is pre-computed for each application and stored in DRAM. As mentioned above, Gibbs samplers can have local smoothness cost tables that can be used to look up costs based on neighbors' labels. Labels are randomly initialized and stored in DRAM. Prior to running Gibbs sampling on the MRF, we utilize large BRAM space on the FPGA to buffer the inputs and the intermediate states, or labels, for the MRF, coming from DRAM. Then for a given number of iterations, Gibbs sampling is executed on the FPGA and the resulting new labels for the MRF are copied back to the DRAM. As shown in Figure 4, the Gibbs sampling on the MRF is done on the accelerator, while other operations are handled by the CPU.

The binary MRF Gibbs sampling accelerator is configured on the FPGA with an array of 257 parallel Gibbs samplers. 257 nodes enables 2-color parallel Gibbs sampling on a row of the MRF, for row size 513 or less. In order to fit 257 nodes on the Zynq board, two approximation of multipliers as shifters in the datapath was made to make sure the DSPs on the board didn't run out. The MRF dimension of 24x513 is supported for real-time source separations. The size is optimized for application of sound source separation for a given BRAM capacity on the FPGA. Since the audio input is transformed into 513 channels of discrete frequency, using 257 samplers allows sampling one row simultaneously when using chromatic scheduling.

A. Experimental Setup

To evaluate the binary MRF Gibbs sampling accelerator, we compared the hybrid CPU-FPGA implementation with running the software purely on the ARM Cortex-A53 on the Zynq SoC. The experimental setup involves compiling C++ code for source separation for ARM and measuring the runtime of running it purely on the CPU. This is compared against offloading the Gibbs sampling function on the FPGA accelerator. Both implementations allocate the input data for the Gibbs sampling on the DRAM, resulting in identical setup. We assume a streaming version of sound source separation where we take 64 ms audio frames where half is overlapped with the previous frame and perform source separation on that frame.

Sound source separation consists of functions listed in Section II-C. We have empirically found the optimal number for the outer EM loop to be 4 iterations, as reported in [18] as well. The number of Gibbs sampling iteration will be increased to find the best signal-to-distortion ratio (SDR), a metric used to evaluate the quality of source separation.

B. Mobile CPU vs. FPGA

Hybrid CPU-FPGA offer an opportunity to create an accelerator for essential computational kernels in machine learning workloads, such as matrix multiplication and in our case, Gibbs sampling, to reduce latency, power and runtime. We partitioned the workload according to Figure 4 and compared running the Gibbs sampling function on the ARM Cortex-A53. While

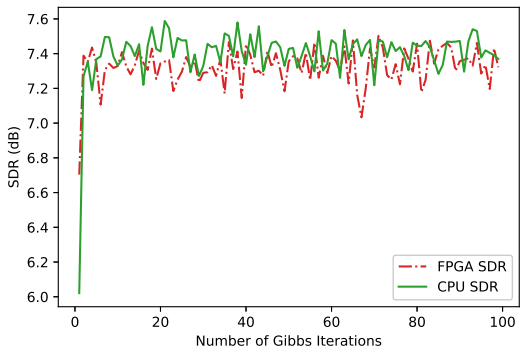

Fig. 5. SDR of sound source separation for increasing number of Gibbs sampling iterations. SDR improves rapidly up to 10 iterations and saturates.

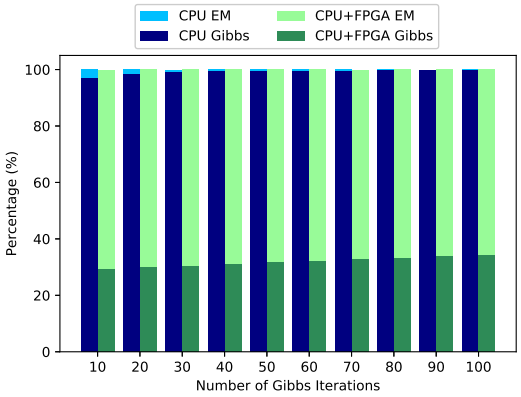

Fig. 6. The percentage of compute time for Gibbs sampling function and the rest of EM loop is shown for CPU and the hybrid CPU-FPGA. The hybrid CPU-FPGA implementation allows Gibbs sampling inference to scale well with increasing number of iterations.

other functions of the source separation program could reap the benefits of having multiple cores, the Gibbs sampling, a sequential algorithm is inherently not suitable for SIMD operations. It is only by exploring the algorithmic scheduling to figure out the sampling order that would not affect the ergodicity (i.e., pertaining to the true distribution) of the sampling process. The Gibbs sampling with the chromatic scheduling allows us to sample conditionally independent variables in parallel.

Preliminary experiments using multi-threading on a server-grade Intel Xeon and a commodity laptop GPU showed gains of 10x and 20-30x, respectively, over a single-threaded software implementation. The speedups are suboptimal due to the complexity of the sampling computation, dependencies between variables and the large amount of random numbers needed per iteration. They were measured running the entire sampling process until convergence to show a realistic speedup with the overhead of creating multiple threads, propagating updates to different parts of the graph and moving the data between caches and memories. We also observed linear speedup with low number of threads but as the threads were increased, the speedups plateaued, especially with a larger number of discrete states. The experiments indicate a scaling problem on CPU and GPU, which prohibits real-time execution and highlight how a hardware scheduler could help us overcome overheads to achieve real-time performance on an FPGA.

While CPU and GPU have large enough memories to fit MRFs used for our applications, the FPGA has limited local memory. Instead of performing chromatic scheduling on the entire MRF, we use a tile-based approach of dividing the MRF into multiple tiles with optional overlapping to control the amount of mixing between the tiles. Our design includes a hardware scheduler which can be automatically generated for different number of labels and MRF sizes to perform chromatic scheduling of a selected region in the MRF. Our previous binary-label version achieved 1048x speedup over CPU [9].

Signal-to-Distortion-Ratio (SDR) of sound source separation running on the hybrid CPU-FPGA SoC with the FPGA Gibbs sampling accelerator running at 100 MHz is plotted in Figure 5. The implementation achieves SDR of 7.33 dB, slightly less than 7.41 dB of the software version, but is better than the prior MRF-based source separation hardware implementations [18], [19]. It runs more than fast enough to satisfy the International Telecommunication Union (ITC) recommended mouth-to-ear delay requirement of 200 ms, taking 116 ms for 10 EM iterations and 50 Gibbs sampling iterations per EM.

Running source separation on CPU doesn't scale well with increased Gibbs iterations as the bulk of the compute time is spent on Gibbs sampling as shown in Figure 6. On the other hand, on the hybrid CPU-FPGA implementation, the time spent for Gibbs sampling is minor compared to the rest of EM loop.

V. Expanding to Multiple Labels

For computer vision applications such as stereo matching and image restoration that uses more than binary labels, we can configure the architecture to handle larger number of labels. Stereo matching typically deal with less than 64 labels [12]. To examine an argument for a more general Gibbs sampling inference accelerator with the support for multiple application, we extended our binary label MRF Gibbs sampling inference accelerator to a 64 labels on the same Xilinx Zynq platform.

Table I compares the resource utilization for binary and 64-label versions. Table II shows the runtime, power, and energy numbers for the 64-label version compared to the binary version. The expanded BRAM usage used for input buffers shows how large the input data could grow to be for increasing the number of labels. As shown in Table I, the compute logic (DSPs) is no longer of the bottle neck as we can only instantiate 24 Gibbs samplers using 64 labels due to increased input buffer.

We compared the relative speedups of original binary MRF Gibbs sampling, 64-label version, and 64-label version running binary MRF Gibbs sampling (not 64 labels), against running the inference on the A53. The 64-label version running binary MRF Gibbs sampling achieved a speedup of 137x for 50 Gibbs sampling iterations, which is roughly 5.7x per sampler, which is roughly similar to 4x per sampler of the binary version. For running 64-label Gibbs sampling inference, it achieved a speedup of 679x for 50 iterations, which is roughly 28x per sampler, much greater than the binary version. Having substantial acceleration across different number of labels

TABLE I
UTILIZATION FOR GIBBS SAMPLERS WITH BINARY AND 64 LABELS

SAMPLERS	LABELS	LUT	(%)	FF	(%)	BRAM	(%)	DSP	(%)
257	2	254270	92.77	257299	46.94	290	31.80	2056	81.59
24	64	72920	26.61	150025	27.37	896	98.25	384	15.24

TABLE II
COMPARISON OF RUNTIME POWER AND ENERGY

HW	RUNTIME (S)	POWER (W)	ENERGY (J)	SAMPLES/S	E /SAMPLE (NJ/S)	E RED. (%)
CPU	0.952951	3.512	3.347	0.646 M	5436.59	0
257	0.000909	5.357	0.005	677.228 M	7.910	99.85
24	0.006949	4.527	0.031	88.588 M	51.102	99.06

suggests that a general MRF Gibbs sampling accelerator for multiple application is plausible.

The dimensions for the MRF has not been changed from 24x513 and this indicates a challenge when solving problems that has larger number of labels on a limited hardware resources. The storage of the labels for the entire MRF doesn't take up as much space and could be cached locally. However, the storage for the input costs could grow significantly. One of the solutions would be to stream in the input costs without buffering them locally. This will save a great amount of memory space but would have many redundant data transfers when iterating Gibbs sampling over the same input values multiple times.

Another solution is to partition the graphs into multiple tiles and processing them independently. Our software experiments have shown that it is actually possible to get decent qualitative results on stereo matching and image restoration using tiles where each of the tiles are iterated multiple times without synchronization of updated labels between them. This effectively performs asynchronous Gibbs updates, or often called "Hogwild Gibbs sampling," [20] where synchronization between the labels are not enforced every Gibbs iteration. The next version will expand on the current work to support asynchronous Gibbs sampling with a focus on energy efficient memory subsystem for caching intermediate data.

VI. RELATED WORK

The simplest way to parallelize Gibbs sampling is by running multiple chains independently in parallel. This gives you more samples but does not produce more accurate samples faster. It also requires more memory for the duplicate chains.

Another class of algorithms are approximate sampling algorithms. Even the original Gibbs sampler proposed by Geman and Geman [21] is approximate as it samples all variables simultaneously on separate processors. However, such samplers are not ergodic and does not converge to the correct stationary distribution. Blocked or blocking Gibbs sampling updates blocks of parameters one at a time rather than update parameters one at a time. If the parameters in the block are highly-correlated, it can improve convergence and mixing. There are other methods that incorporate extensions to recover ergodicity such as methods proposed for Latent Dirichlet Allocation (LDA) [22], [23]. These work provided extensive empirical work on Approximate Distributed LDA (AD-LDA) [22], which samples in parallel without globally synchronizing the sampler state after each update and demonstrated its effectiveness in sampling LDA models with same predictive performance as those generated by sequential Gibbs sampling.

Chromatic Gibbs sampling falls under single chain Gibbs sampling method. It uses graph coloring technique to enable parallel scheduling of sequential scan Gibbs sampling [17]. This sampler is ergodic and can be massively parallel depending on how the graph is colored. It has also proven to be a great match for MRFs as the graph-coloring is facile due to its structure. It also has the benefit of not having to worry about ergodicity when implementing an accelerator.

One strategy of achieving parallelism is to run Gibbs sampling updates in parallel without global synchronization of the sampler state after each update, which is referred to as Hogwild or asynchronous Gibbs sampling [20]. More recently, there have been several work that studied asynchronous Gibbs sampling [24]–[26]. While AD-LDA empirically shown that asynchronous updates work, these recently work has shown better understand of why it works and how to keep the bias low and mixing fast. [27] randomly selects a node's neighbors and samples but the pairwise MRF, we have targeted does not have a lot of neighbors and it is unlikely to benefit from this method.

There exist several work on Gibbs sampling hardware. One approach incorporates novel devices such as spiking VLSI substrate with stochastic leak and threshold properties [28] and molecular optical device called resonance energy transfer circuits [29]. Another work focuses on building combinational stochastic circuit and uses conventional substrate (FPGA) as a demonstration vehicle [30]. All of these methods are not using standard digital CMOS-based fabrics and logic circuits. There have been a couple of digital CMOS-based work using ASIC flow [18], [19]. These only have a naive single-chain sequential Gibbs sampler with no parallelism.

VII. CONCLUSION

We demonstrated a hybrid CPU-FPGA implementation of sound source separation to show how our parameterized architecture for binary Gibbs sampling inference could to used to accelerate real-time perceptual applications to reduce the runtime, latency, and power compared to running it on a quad-core ARM Cortex-A53. We employed 257 Gibbs sampler array and used chromatic scheduling to parallelize Gibbs sampling inference to achieve up to 230x speedup over A53 while meeting the ITU's recommended latency with 50 ms.

We also show how the architecture could be configured to support 64 labels, which is enough to run a computer vision application, stereo matching. We showed an array of 24 samplers for 64 labels on same MRF size of 24x513 for a comparison. The 64-label version achieved 137x speedup and 99.06% energy reduction over A53, on 50 iterations binary MRF Gibbs sampling used in sound source separation. For 64-label MRF Gibbs sampling, it achieved up to 679x speedup with much greater 28x speedup per Gibbs sampler, compared to binary MRF Gibbs sampling.

REFERENCES

[1] J. Dean, G. S. Corrado, R. Monga, K. Chen, M. Devin, Q. V. Le, M. Z. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, and A. Y. Ng, "Large scale distributed deep networks," in *NIPS*, 2012.

[2] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[3] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *International Conference on Learning Representations*, 2013.

[4] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 1278–1286. [Online]. Available: http://proceedings.mlr.press/v32/rezende14.html

[5] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*. IEEE, 2017, pp. 1–12.

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[7] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[8] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

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

[10] J. . Cardoso, "Blind signal separation: statistical principles," *Proceedings of the IEEE*, vol. 86, no. 10, pp. 2009–2025, Oct 1998.

[11] S. Z. Li, *Markov Random Field Modeling in Computer Vision*. Berlin, Heidelberg: Springer-Verlag, 1995.

[12] 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, pp. 7–42, Apr 2002. [Online]. Available: https://doi.org/10.1023/A:1014573219977

[13] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 6, pp. 1068–1080, June 2008.

[14] R. W. and, "A class of discrete multiresolution random fields and its application to image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 1, pp. 42–56, Jan 2003.

[15] 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*, Sept 2012, pp. 1–6.

[16] T. E. Tkacik, "A hardware random number generator," in *International Workshop on Cryptographic hardware and embedded systems*. Springer, 2002, pp. 450–453.

[17] 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*, 2011, pp. 324–332.

[18] G. G. Ko and R. A. Rutenbar, "Real-time and low-power streaming source separation using markov random field," *J. Emerg. Technol. Comput. Syst.*, vol. 14, no. 2, pp. 17:1–17:22, May 2018. [Online]. Available: http://doi.acm.org/10.1145/3183351

[19] G. G. Ko and R. 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)*, March 2017, pp. 2477–2481.

[20] M. Johnson, J. Saunderson, and A. Willsky, "Analyzing hogwild parallel gaussian gibbs sampling," in *Advances in Neural Information Processing Systems*, 2013, pp. 2715–2723.

[21] S. Geman and D. Geman, "Stochastic relaxation, gibbs distributions, and the bayesian restoration of images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 6, no. 6, pp. 721–741, Nov. 1984. [Online]. Available: https://doi.org/10.1109/TPAMI.1984.4767596

[22] D. Newman, A. Asuncion, P. Smyth, and M. Welling, "Distributed algorithms for topic models," *J. Mach. Learn. Res.*, vol. 10, pp. 1801–1828, Dec. 2009. [Online]. Available: http://dl.acm.org/citation.cfm?id=1577069.1755845

[23] A. Smola and S. Narayanamurthy, "An architecture for parallel topic models," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, 2010.

[24] C. De Sa, K. Olukotun, and C. Ré, "Ensuring rapid mixing and low bias for asynchronous gibbs sampling," in *JMLR workshop and conference proceedings*, vol. 48. NIH Public Access, 2016, p. 1567.

[25] A. Terenin, D. Simpson, and D. Draper, "Asynchronous distributed gibbs sampling," 09 2015.

[26] A. Terenin, S. Dong, and D. Draper, "Gpu-accelerated gibbs sampling: a case study of the horseshoe probit model," *Statistics and Computing*, pp. 1–10.

[27] C. De Sa, V. Chen, and W. Wong, "Minibatch gibbs sampling on large graphical models," *arXiv preprint arXiv:1806.06086*, 2018.

[28] S. Das, B. U. Pedroni, P. Merolla, J. Arthur, A. S. Cassidy, B. L. Jackson, D. Modha, G. Cauwenberghs, and K. Kreutz-Delgado, "Gibbs sampling with low-power spiking digital neurons," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 2704–2707.

[29] 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.

[30] V. K. Mansinghka, E. M. Jonas, and J. B. Tenenbaum, "Stochastic digital circuits for probabilistic inference," *Massachussets Institute of Technology, Technical Report MITCSAIL-TR*, vol. 2069, 2008.