2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture

Accelerating Markov Random Field Inference Using Molecular Optical Gibbs Sampling Units

Siyang Wang, Xiangyu Zhang, Yuxuan Li, Ramin Bashizade, Song Yang, Chris Dwyer, Alvin R. Lebeck
Duke University
Durham, NC 27708
{*siyang.wang, xiangyu.zhang, yuxuan.li, ramin.bashizade, song.yang, c.dwyer, alvy*}@duke.edu

Abstract

The increasing use of probabilistic algorithms from statistics and machine learning for data analytics presents new challenges and opportunities for the design of computing systems. One important class of probabilistic machine learning algorithms is Markov Chain Monte Carlo (MCMC) sampling, which can be used on a wide variety of applications in Bayesian Inference. However, this probabilistic iterative algorithm can be inefficient in practice on today's processors, especially for problems with high dimensionality and complex structure. The source of inefficiency is generating samples from parameterized probability distributions.

This paper seeks to address this sampling inefficiency and presents a new approach to support probabilistic computing that leverages the native randomness of Resonance Energy Transfer (RET) networks to construct RET-based sampling units (RSU). Although RSUs can be designed for a variety of applications, we focus on the specific class of probabilistic problems described as Markov Random Field Inference. Our proposed RSU uses a RET network to implement a molecular-scale optical Gibbs sampling unit (RSU-G) that can be integrated into a processor / GPU as specialized functional units or organized as a discrete accelerator. We experimentally demonstrate the fundamental operation of an RSU using a macro-scale hardware prototype. Emulation-based evaluation of two computer vision applications for HD images reveal that an RSU augmented GPU provides speedups over a GPU of 3 and 16. Analytic evaluation shows a discrete accelerator that is limited by 336 GB/s DRAM produces speedups of 21 and 54 versus the GPU implementations.

1. Introduction

Statistical methods, machine learning in particular, are increasingly used to address important problems including, but not limited to: computer vision, robot/drone control, data mining, global health, computational biology, and environmental science. Many approaches in statistics and machine learning utilize probabilistic algorithms that generate samples from parameterized probability distributions (e.g., exponential distribution with a decay rate). Probabilistic algorithms offer the potential to create generalized frameworks with close ties to reality and in some cases are the only viable approach for solving certain classes of problems (e.g., high-dimensional inference).

The challenge we propose is to develop new hardware that directly supports a wide variety of probabilistic algorithms [13]. This paper takes the first steps toward meeting this challenge by exploiting the physical properties of molecular-scale optical devices. We build on recent work that provides a theoretical foundation for creating novel probabilistic functional units based on Resonance Energy Transfer (RET) that can approximate virtually arbitrary probabilistic behavior and generate samples from general distributions [42].

To meet the above challenge, we introduce the concept of a RET-based Sampling Unit (RSU), a hybrid CMOS/RET functional unit that generates samples from parameterized distributions. An RSU specializes the calculation of distribution parameters in CMOS and uses RET to generate samples from a parameterized distribution in only a few nanoseconds. There are a variety of distributions that could be implemented by an RSU; however, in this work we focus on an RSU that implements a distribution for use in a particular class of Bayesian Inference problems.

Bayesian Inference is an important, generalized framework that estimates a hypothesis (i.e., values for random variables) using a combination of new evidence (observations) and prior knowledge. Markov Chain Monte Carlo (MCMC) sampling is a theoretically important and powerful technique for solving the inference problem that iteratively and strategically samples the random variables and ultimately converges to an exact result. However, MCMC becomes inefficient for many inference problems in practice, especially those with high dimensionality (many random variables) and complex structure. MCMC can require many iterations to converge to a solution and the inner loop incurs the overhead of sample generation from prescribed distributions. Although deterministic inference algorithms can be faster than MCMC, they sacrifice accuracy (e.g., by approximation) and require more complex mathematical derivation. Similarly, problem specific non-Bayesian algorithms forego the benefit of a generalized framework and require reformulation for each problem.

1063-6897/16 $31.00 © 2016 IEEE
DOI 10.1109/ISCA.2016.55

Markov Random Field (MRF) Bayesian Inference can be used for a broad class of applications, including image processing (e.g., image segmentation, motion estimation, stereo vision, texture modeling), associative memory, etc. The overall goal is often to determine the most likely value for each random variable given the observed data (i.e., marginal MAP estimates). Given a specified MRF model, this is achieved in MCMC by iteratively sampling the random variables according to the conditional dependencies and then identifying the mode of the generated samples.

To accelerate MRF inference using MCMC, we introduce RSU-G, a Gibbs Sampling unit based on the 'first-to-fire' exponential unit proposed elsewhere [42]. Our specific RSU-G unit supports first-order MRFs with a smoothness-based prior, which includes many image processing applications (e.g., image segmentation, motion estimation, stereo vision). A survey of possible applications is provided elsewhere [36].

We experimentally demonstrate the functional operation of a rudimentary RSU-G using a macro-scale prototype comprised of discrete components, including an FPGA, laser sources, RET devices, and a PC. We use the prototype to segment a small image into two regions. To our knowledge this is the first demonstration of a RET-based molecular-scale optical probabilistic functional unit, and represents a significant step forward for this technology. The discrete nature of the prototype renders performance evaluation meaningless. Therefore, we use a combination of emulation and analytic evaluation to obtain performance estimates of architectures that incorporate RSU-Gs. We obtain area and power estimates for our proposed RSU from a combination of synthesis of the Verilog circuits using the Synopsys tools, Cacti, and first principles for the RET components. The synthesized circuits are verified in Modelsim.

To emulate the performance of RSU-G augmented processors, we replace appropriate code sequences with a sequence of instructions to emulate the latency of RSU-G instructions. This approach does not provide a functionally accurate evaluation but is sufficient to obtain performance estimates. We analytically investigate the performance for a discrete accelerator where the upper bound is dictated by memory bandwidth limitations. Our analysis shows that a GPU augmented with RSU-G units can achieve speedups of up to 3 and 16 for image segmentation and motion estimation, respectively. A discrete accelerator with 336 RSU-G units achieves speedups of 21 and 54 assuming a 336GB/s memory BW limitation. The novel optical components of RSU-G units consume very little power (0.16 mW) and area (0.0016 mm^2). Synthesizing the CMOS portions of RSU-G in 15nm reveals power of 3.75 mW and area of 0.0013 mm^2 for a total RSU-G power of 3.91 mW and area of 0.0029 mm^2.

Below we summarize our primary contributions:

- Introduce functional units (RSU) that provide samples from parameterized probability distributions.
- Design and implementation of RSU-G, a functional unit for Gibbs sampling in MCMC solvers for Bayesian Inference.
- Experimental demonstration of an RSU-G in a macroscale prototype, to our knowledge the first such demonstration.
- Performance evaluation of RSU-G augmented CPU and GPUs.

The remainder of this paper is organized as follows. Section 2 provides background, motivation and related work. Section 3 provides an overview of a generic RSU and how it can be incorporated into processors or a stand alone accelerator. We present the RSU-G design in Section 4 and implementation in Section 5. Section 6 provides more details on RSU-based architectures. Experimental results are described in Section 7 and we evaluate integrated designs in Section 8. Section 9 discusses limitations and future work, while Section 10 concludes.

2. Background, Motivation, and Related Work

The increasing use of machine learning in data analytics presents new challenges and opportunities for the design of computing systems. This section provides a brief overview of probabilistic algorithms, their associated challenges, reviews recent proposals to use nanoscale optical devices to overcome these challenges, and presents related work.

2.1 Probabilistic Algorithms

Recent theoretical advances in statistics and probabilistic machine learning demonstrate that many application domains can benefit from probabilistic algorithms in terms of simplicity and performance [28, 29]. Example problems include, but are not limited to, statistical inference, rare event simulation, stochastic neural networks (e.g., Boltzmann machines), probabilistic cellular automata and hyper-encryption.

Most probabilistic computations rely on sampling from application-specific distributions. For example, Bayesian Inference solvers often iteratively sample from common distributions such as gamma distribution and normal distribution, while rare event simulations may require many samples from a heavy-tailed distribution to obtain statistically significant results. The number of samples generated in these applications can be large; many thousands of samples per random variable with thousands of random variables.

The importance of sampling from various distributions led to the C++11 standard library including implementations for 20 different distributions. Although this greatly simplifies program development, it does not address the inherent mismatch between conventional digital computers and probabilistic algorithms. In particular, sampling requires control over a parameterizable source of entropy used for random selection. Therefore, generating a sample includes two critical steps: 1) parameterizing a distribution and 2) sampling from the distribution.

Consider Bayesian Inference, an important inference framework that 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. $p(X)$ is the prior distribution of X, and $p(D|X)$ is the probability of observing D given a certain value of X. In Bayesian Inference, the goal is to retrieve the posterior distribution $p(X|D)$ of the random variable X when D is observed. As the dimension of $X = [X_1, \ldots, X_n]$ increases, it often becomes difficult or intractable to numerically derive the exact posterior distribution $p(X|D)$. One approach to solve these inference problems uses probabilistic Markov Chain Monte Carlo (MCMC) methods that converge to an exact solution by iteratively generating samples for random variables. Obtaining each sample incurs at least the overhead of computing the distribution parameters and sampling from the distribution.

2.2 Sampling Overhead

Parameterizing a distribution is application dependent and requires computing specific values for a given distribution. For example, computing the decay rate for an exponential, the mean and variance for a normal, etc. For a class of Bayesian Inference problems that we study this may include computing a sum of distance values and can take at least 100 cycles to compute on an Intel E5-2640 processor (compiled with gcc –O3), and could be much higher. Other probabilistic algorithms may have different computations with varying complexity. Nonetheless, the time required to parameterize a distribution is an important source of overhead in probabilistic algorithms.

The second component for sampling is to generate the sample from the parameterized distribution. Devroye [9] provides a comprehensive overview of techniques for computationally generating samples from various distributions. Samples from general continuous or discrete distributions can be generated using algorithms such as inverse transform sampling and rejection sampling. Unfortunately, it can take hundreds of cycles to generate a sample with these approaches, and more complex multivariate sampling can take over 10,000 cycles. Table 1 shows how many cycles it takes to generate a sample for a few distributions using the C++11 library [1]. We obtain cycle counts on an Intel E5-2640 using the Intel Performance Counter Monitor and present the average of 10,000 samples (-O3 optimization).

Table 1: Cycles to Sample from Different Distributions.

Distribution Type	Cycles (average)
Exponential	588
Normal	633
Gamma	800

The overheads of calculating distribution parameters and sampling from the distribution are critically important to many probabilistic algorithms since they incur both overheads in their inner loop. Furthermore, multiple applications often use the same distribution and share the computation to parameterize the distribution. Therefore, accelerating sampling can have a significant impact on overall execution time.

2.3 Resonance Energy Transfer Sampling

Recent work provides a theoretical foundation for constructing physical samplers based on molecular-scale Resonance Energy Transfer (RET) networks [42]. RET is the probabilistic transfer of energy between two optically active molecules, called chromophores, through non-radiative dipole-dipole coupling [41]. When placed a few nanometers apart and their emission and excitation spectra overlap, energy transfer can occur between a donor and acceptor chromophore pair. A RET network is constructed by placing multiple chromophores in a physical geometry where chromophores interact through RET.

RET networks can approximate virtually arbitrary probabilistic behavior since they can implement sampling from phase-type distributions [42]. The probabilistic behavior of a RET network is determined by its physical geometry. Typically, a given RET network corresponds to a specific distribution. Sampling from the distribution occurs by illuminating the RET network and observing output fluorescence as a function of time (within a few nanoseconds). RET-based samplers may also reduce power consumption since the samples are generated in the form of single fluorescent photons.

Previous work outlined several possible samplers (e.g., Bernoulli and exponential) that can be composed to make more general samplers [42]. In this paper we focus on samplers for Markov Random Field (MRF) Bayesian Inference using Markov Chain Monte Carlo (MCMC) methods that utilize exponential samplers. We create probabilistic functional units that use CMOS specialization to accelerate distribution parameterization and RET networks to accelerate obtaining a sample from the parameterized distribution.

RET networks are integrated with an on-chip light source, e.g., quantum-dot LEDs (QD-LEDs), waveguide, and single photon avalanche detector (SPAD) to create a RET circuit. Each RET circuit can contain an ensemble of RET networks. A fully specified RET network can be conveniently and economically fabricated with sub-nanometer precision using hierarchical DNA self-assembly [19, 33]. RET circuits can also be integrated with hybrid electro-optical CMOS using back end of line processing [21, 22]. Section 3 provides an overview of RET-based Sampling Units (RSU) and how these units can be used in processor architectures. The remainder of this section briefly discusses alternative approaches.

2.4 Alternative Approaches and Related Work

Reducing or avoiding the overhead of sampling can be achieved by using deterministic algorithms or by introducing hardware specialization. One approach to avoid the overhead of sampling is to use alternative discrete

algorithms. For example, an alternative to MCMC is deterministic approximate inference methods such as Expectation Propagation (EP) and Variational Bayesian (VB). Although often more efficient in practice, these methods require more complex mathematical derivation and arbitrary assumptions that create divergence from the exact solution [27, 29, 43]. Domain scientists generally prefer the less complex mathematically, but more accurate pure solution if possible.

Another approach to accelerate sampling, and the one we advocate in this paper, is through specialization that incorporates novel devices. A Stochastic Transition Circuit and an FPGA implementation was proposed to efficiently update random variables given the graphical model of an inference problem [24, 25]. Abstractly, our units are an instance of a Stochastic Transition Circuit; however, our approach differs in that we exploit the physical properties of RET and can implement complex distributions that would be difficult in CMOS.

Similar to an RSU, other techniques propose using a physical process as a natural source of entropy to create samplers. The common physical processes used for this fall into four categories: noise, free running oscillator, chaos and quantum phenomena [35]. With a quantum-mechanical origin [16], RET provides true randomness and arbitrary sampling distributions. While previous works on RET between quantum dots support certain probabilistic computing applications [5, 30], RSUs are more general and can be used across a broad set of applications. Although often used in CMOS to generate random bits, thermal noise cannot provide provable randomness and requires complex post-processing to make the output appear more random. Further, its implementation is either difficult to parameterize or does not support arbitrary distributions, thus limits reuse across applications.

The Intel Digital Random Number Generator (DRNG) uses a thermal noise-based entropy source and includes two stages of post-processing: an AES-based entropy conditioner and a Pseudo Random Number Generator (PRNG) [2]. Based on our synthesis of the 256-bit AES conditioner at 45nm technology [3], this stage alone is comparable to the RSU-G_1 unit proposed later in terms of area, power consumption, and throughput. The full DRNG requires more area and power.

Probabilistic CMOS (PCMOS) can implement discrete samplers by using thermal noise in electronic circuits to make probabilistic switches with a set of tunable parameters [8, 31]. Unfortunately, this requires amplifying the noise to a specific magnitude, and can be energy and area inefficient. The exact noise level of each probabilistic switch relies on the probabilities of all values and requires normalization. Furthermore, PCMOS switches essentially implement Bernoulli random variables, and cannot be flexibly organized to generate samples from general distributions.

Other recent work explores augmenting processors with specialized Neural Processing Units (NPU) [4, 10] to achieve speedup and power savings using analog circuits, but focuses specifically on using neural networks to approximate deterministic functions.

RSUs can implement a broad class of distributions, can be easily parameterized dynamically by changing RET circuit inputs (e.g., QD-LED intensity values), and eliminate normalization for some cases by using relative ratios of distribution parameters (e.g., exponential decay rates). RSUs provide an efficient hardware platform for probabilistic programming languages [7, 24] by providing native probabilistic support. Exploring language support for RSUs is an interesting future direction, but first we must develop specific units and provide an architecture for probabilistic computing.

3. Architecture Overview

There are many possible ways to expose an RSU to software, ranging from adding functional units to an existing processor to a discrete accelerator. Our goal in this section is to present a general approach for constructing functional units with RET-based samplers and high-level architectures that utilize these units. We present two potential architectures: 1) augmenting a GPU with sampling units and 2) a discrete accelerator designed to maximize memory bandwidth utilization.

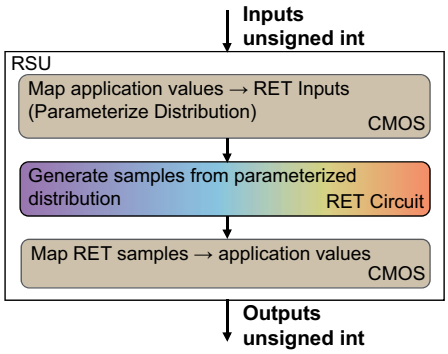


Figure 1: Generic RSU Block Diagram.

A generic RSU is a hybrid of CMOS and RET technology, and its inputs and outputs are unsigned integers that correspond to values of interest to the application. A block diagram of an RSU is shown in Figure 1. An RSU performs a series of three operations: 1) map application values to RET inputs, 2) generate samples, 3) map RET output to application value. Steps 1 and 3 are implemented using conventional CMOS specialization, whereas step 2 is a RET circuit that exploits the probabilistic behavior of RET networks. Step 1 is where distribution parameterization occurs, and involves converting application values into RET circuit inputs (e.g., QD-LED intensity values). Step 2 samples from the parameterized distribution using one or more RET circuits, and step 3 converts the RET circuit output value back to an application data type for the sample. Section 4 presents details of a specific RSU to support Bayesian Inference using MCMC.

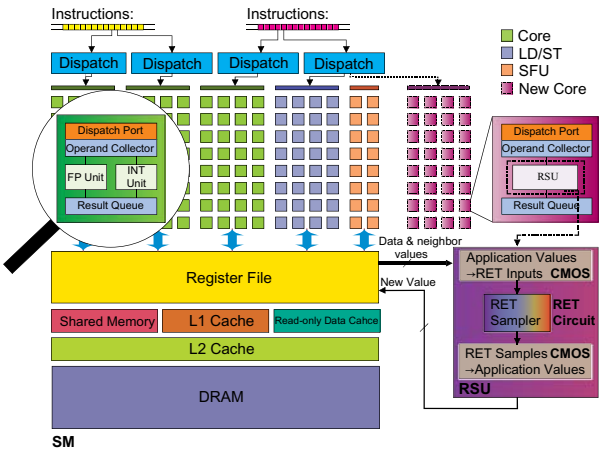


Figure 2: GPU Augmented with RSUs.

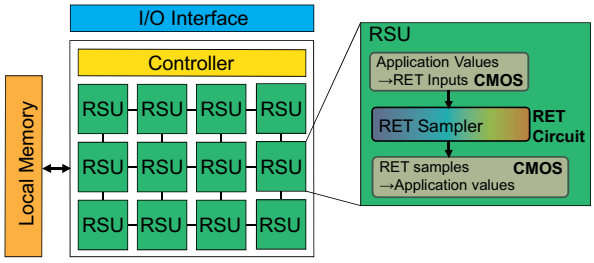


Figure 3: A Discrete Accelerator using RSUs.

Figure 2 shows a GPU architecture (single streaming processor) modified to include RSUs and Figure 3 shows a custom discrete accelerator design using RSUs. Augmenting a conventional CPU would be similar to the GPU design. For processor integration, additional instructions are required to access the RSU, and we discuss these in more detail in the context of our specific RSU in Section 6. From a program's perspective, an RSU is a multi-cycle, pipelined functional unit that takes several random variable values as input and produces a single random variable value as output. The values are represented as unsigned integers.

A discrete accelerator would operate similar to existing encryption, motion, or other specialized unit. Generally, these systems first place relevant data at a specified memory address, the accelerator then iterates over the data using custom control logic, and the CPU is notified of completion. Accelerators can often achieve much higher performance than programmable systems since they can be tailored to specific problems, they also serve as a method to analyze performance bounds.

RSUs can be designed for a variety of probabilistic algorithms, and exploring the large design space requires many man hours. In this paper we explore a small region of the overall design space by focusing on an RSU designed specifically to accelerate certain types of inference problems.

4. A RET-based Gibbs Sampling Unit

This section presents an RSU designed specifically for Markov Random Field inference using MCMC. We first summarize Markov Random Fields and MCMC approaches to perform inference, and then present our RSU designed to accelerate this broad class of applications.

4.1 Markov Random Fields

A Markov Random Field (MRF) is a type of graphical model used in Bayesian Inference. An MRF is a set of random variables that satisfies the Markov properties described by an undirected graph [20]. MRFs are suitable for many interesting and important applications such as low-level computer vision and associative memory [12, 20, 36]. In this paper, we focus on first-order MRFs with smoothness-based priors, homogeneity and isotropy (i.e., position and orientation independence), with discrete random variables. Extending our work to other types of MRFs is future work.

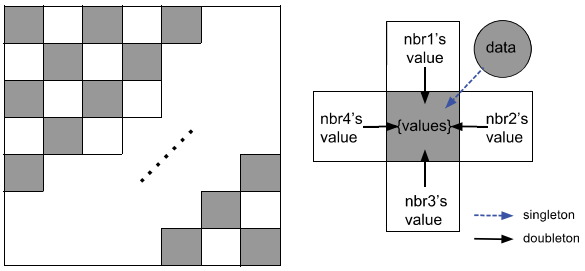


Figure 4: A First-order MRF.

Figure 4 shows an example first-order MRF where each random variable has four neighbors and is conditionally independent of all non-adjacent random variables when conditioned on the four neighbors. More specifically, the full conditional probability of each random variable $X_{i,j}$ is the exponential of the sum of five clique potential energies (with normalization):

$$p(X_{i,j}|X_{-(i,j)}, D) \propto \exp\left\{-\frac{1}{T}\left[Ec_S(X_{i,j}, D) + Ec_D(X_{i,j}, X_{i-1,j}) + Ec_D(X_{i,j}, X_{i,j-1}) + Ec_D(X_{i,j}, X_{i+1,j}) + Ec_D(X_{i,j}, X_{i,j+1})\right]\right\}. \quad (1)$$

$X_{i,j}$ is a random variable that can take on M possible values, or more commonly called *labels* in this context. $Ec_S(X_{i,j}, D)$ is the singleton clique potential energy that relates $X_{i,j}$ to the observed data D, and $Ec_D(X_{i,j}, X_{i',j'})$ is the doubleton clique potential energy that relates $X_{i,j}$ to a neighboring random variable. T is a fixed constant.

4.2 MCMC and Gibbs Sampling

For many problems, directly solving the equations above can be computationally expensive. Therefore, MCMC methods are often employed. Among the algorithms for generating MCMC samples, Gibbs sampling and Metropolis

sampling are the most commonly used [29], and for our applications we use Gibbs sampling.

Gibbs sampling generates a new sample of a random variable ($X_{i,j}$) directly from its full conditional distribution when conditioned on the current labels of the other random variables. In a first-order MRF, this is achieved by calculating the probability for each of the M possible labels a random variable can take on using Equation (1) and randomly selecting a label according to the discrete distribution.

Each MCMC iteration updates all random variables once to obtain one MCMC sample. The number of operations per iteration linearly depends on the number of possible labels for each random variable and the number of random variables. However, different labels can be evaluated simultaneously and random variables that are conditionally independent can be updated concurrently, exposing significant parallelism for some problems. Specifically, the first-order MRF in Figure 4 allows all the gray random variables to be updated simultaneously. Similarly, all the white random variables can be updated simultaneously.

4.3 RSU-G Design

A discrete sampler with M outcomes can be constructed using M exponential samplers parameterized by the probabilities of taking each outcome [42]. Generating one new sample for $X_{i,j}$ requires M different samples, each from uniquely parameterized exponential distributions. We exploit this observation to construct a RET-based Gibbs Sampling Unit (RSU-G_1) using a single RET circuit (G_1).

In this paper we consider only MRFs with smoothness-based priors where the energy (i.e., logarithm of probability) of taking on a label is the sum of four doubleton clique pontential energies and one singleton clique potential energy [36]. Each doubleton clique potential energy is a measure of distance between the label being evaluated and the current label of a neighbor. Typically, the distance measure is defined for the label space and is problem specific, here we consider grayscale valued images and use a simple squared difference norm as the metric (Equation 2). The singleton clique potential energy is application specific; RSU-G implements it as the squared difference between two data values to directly support applications such as image segmentation, motion estimation and stereo vision and is extendable to other applications by precomputing their singleton energy externally and sending into one data input.

$$\begin{aligned} &\mathrm{Ec_D}(X_{i,j} = x, X_{i+1,j} = x') \\ &\quad = \mathrm{Ec_D}(X_{i,j} = [x1, \ldots, xn], X_{i+1,j} = [x1', \ldots, xn']) \\ &\quad = d^2(x, x') = w_D \sum_k (xk - xk')^2 \quad (2) \end{aligned}$$

RSU-G utilizes the 'first-to-fire' design based on the property of competing exponential random variables [42]. In this design, a RET circuit is used as an exponential sampler that generates exponentially distributed samples. The M exponential samplers are parameterized by the energies of taking on each label. The key aspect in this design is parameterizing the exponential samplers, based on the neighboring random variables' current labels and the singleton energy, and recording the time to fluorescence (TTF) from each exponential sample. The label that produces the shortest TTF is chosen as the label for $X_{i,j}$.

4.4 Limited Precision

Previous work found that 8-bit precision for energy calculations is sufficient for many applications [25]. Given the specific distance measure we use here, only 3 bits are needed for scalar values and 6 bits for vector values in the doubleton calculation, as described later. Although extra bits can increase the number of possible labels, the energies of different labels start to overlap resulting in equal selection probability. These close or redundant labels do not necessarily improve the solution quality; however the time required to update one random variable increases since there are more labels to be sampled. In these cases, we recommend collapsing the equally likely labels into a single label before execution.

5. RSU-G_1 Implementation

An RSU-G implementation can take many forms. On one end of the spectrum it could be constructed to iterate over the M possible labels using a single RET circuit (RSU-G_1). On the other extreme end of the spectrum it could use M distinct RET circuits to simultaneously evaluate all M possible labels in a single step (RSU-G_M). In the middle are designs with K RET circuits that take M/K steps to obtain the result (RSU-G_k).

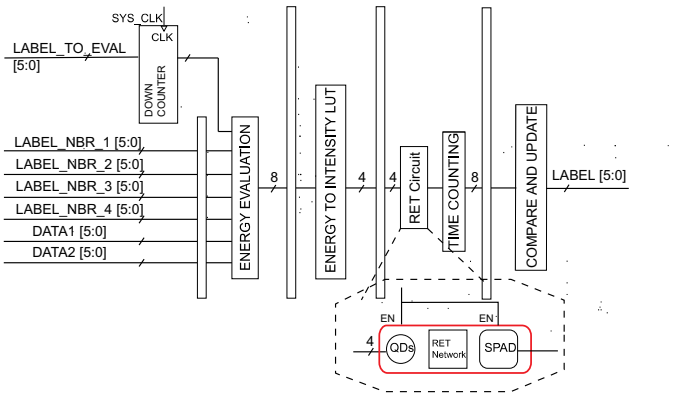


Figure 5: An RSU-G_1 Implementation Diagram.

5.1 Overview

Our preliminary RSU-G_1 implementation, shown in Figure 5, evaluates one possible label for a random variable (K = 1) per step, and iterates to evaluate all M labels. Given the limited label precision, we use 6-bit unsigned integers to represent random variable labels (M <= 64). RSU-G_1 is a multicycle pipelined functional unit that takes 7+(M-1) cycles to obtain a random variable sample in steady state.

The design can be easily extended to evaluate up to 64 labels (RSU-G_{64}) in 12 cycles at the expense of additional area. Exploring a configurabe RSU design is part of our future work.

There are five main components in any RSU-G implementation: 1) label decrement/input, 2) energy computation, 3) energy to intensity mapping, 4) RET circuits (for sampling), and 5) selection. Label decrement is used to iterate over all M possible labels. The energy computation performs the distance calculation to obtain the exponential decay rate which maps through a lookup table to a QD-LED intensity. The RET circuit samples the exponential distribution and the resulting TTF for the sample is used in the selection block to choose the lowest from all M possible labels.

For many applications M is fixed and an initial down counter value can be set at the start of the program. To obtain a sample for a random variable $X_{i,j}$, RSU-G requires five 6-bit inputs, one for each neighbor and its data value. Some applications need an additional data value that changes for each possible label. The output is a single 6-bit value that represents the new label for the random variable.

5.2 Pipeline Stages

Here we describe each of the stages in our initial RSU-G_1 pipeline implementation. This does not necessarily represent the most optimized design.

Label. The first stage sets the inputs necessary for evaluating a given possible value for the random variable. This includes the given value to evaluate, the current values of the four neighboring random variables, and the input data values. The down counter is initialized with the maximum possible value for the random variable (M-1), and the other inputs are stored in registers. We assume this initialization is overlapped with evaluations of previous random variables. On subsequent evaluation cycles the down counter is decremented to iterate over the values, and the other values (except for the second data value) remain unchanged until the next random variable evaluation.

Energy Calculation. The second stage computes the clique potential energies, a first step in distribution parameterization. Each cycle a new energy is computed since the down counter changes. The 6-bit value can represent either a 2D vector or a scalar. For the 2D vector $[x1, x2]$, the 6-bit value is split into 3 bits for x1 and 3 bits for x2. The distance measure is calculated separately for the two entries between two neighboring random variables and then summed to obtain the doubleton energy. When the value of the random variable is a scalar, only the first entry (3 bits) is used and the second entry is set to zero. We found this limited precision to be sufficient for many applications. Similarly, the singleton energy is a distance measure between two data inputs, a calculation that is application dependent, e.g., in motion estimation it can be a weighted squared difference between two grayscale values. We assume that any scalar weights in the singleton calculation are pre-factored from the input data. The 8-bit energy for a possible label is calculated by summing the five clique potential energies and passed onto the next stage.

Intensity Mapping. The third stage implements the second component of distribution paramaterization by mapping the 8-bit energy value to a corresponding QD-LED intensity. We use a lookup table to find the corresponding 4-bit signal that provides the input to a RET circuit to control the binary on/off state of its four QD-LEDs. The QD-LEDs are sized to provide a suitably large dynamic range of intensities to match the precision in relative probabilities we demonstrate with the RSU-G_2 hardware prototype described later.

RET Sampling. This stage activates a RET circuit to obtain a sample from the RET network. We simultaneously enable the QD-LEDs and the SPAD of the given RET circuit. The time to the first photon detection (TTF) is recorded using an 8-bit shift register that is clocked 8x faster than the system clock. It may take multiple system clock cycles before a RET circuit generates an output and can be reused for another evaluation. We elaborate on this later and use replicated RET circuits to sustain single cycle operation of the RSU-G pipeline.

Selection. In the final stage, the selection block records the shortest TTF for each possible label. Each cycle the previously shortest TTF is compared against the new TTF and the shorter is recorded as the current best TTF (with its label). After evaluating all labels (i.e., the down counter reaches zero), the best label is returned as the new sample of the random variable $X_{i,j}$.

5.3 Replicated RET Circuits

The TTF of a RET circuit is probabilistic, and for RSU-G follows an exponential distribution. Samples from the tail of this distribution can become arbitrarily long and the delay depends on the fluorescent lifetime of the chromophores in the RET networks. The RSU-G_1 design presented here requires four 1ns cycles for the RET circuits to reach a quiescent state, ensuring it is safe to proceed with a new sampling operation.

However, the four cycle delay creates a structural hazard in the pipeline. We use four replicated RET circuits in RSU-G_1 to overcome the hazard. This allows us to share the parameterization, timing and selection logic among all four replicates. We use a simple two-bit counter for round-robin scheduling of sampling operations across the four RET circuits and sustain a throughput of one label sample per cycle (requiring M cycles for a single random variable).

The above design represents the smallest RSU-G_1 design that produces one possible label evaluation per cycle. Utilizing more RET circuits can further reduce latency by evaluating multiple possible labels per cycle. The extreme is RSU-G_{64} that evaluates up to the maximum of 64 possible labels simultaneously by using 256 RET circuits. This design can sustain a throughput of one random variable

sample per cycle. Exploring the space of RSU design is ongoing work.

6. RSU Architectures

There are many possible ways to expose sampling units to software, ranging from adding functional units to an existing processor to a discrete accelerator. In this section we present two potential designs: 1) augmenting a GPU with RSU-G units and 2) a discrete accelerator designed to maximize memory bandwidth utilization.

The operation of the RSU-G unit can be viewed in terms of operations performed once: 1) per application, 2) per MCMC iteration, 3) per random variable evaluated (a pixel in our applications), and 4) per potential random variable label. Each RSU-G unit requires initializing the intensity map table and down counter (max label value) at the start of each application. For each random variable (pixel in our applications), RSU-G requires the four neighbor labels (for doubleton calculations) and its associated data (e.g., grayscale value for singleton calculations). Finally, for each potential label, the singleton calculation may also need information from a target location (pixel grayscale).

6.1 Augmenting Processors with RSU-G Units

To augment a processor or GPU with RSU-G units requires instructions to perform each of the four steps described above. To achieve this, we introduce a single instruction *RSU op, regsrc, regdest* that includes an operation, a source register and a destination register. The *op* field specifies a target RSU-G control register and if the instruction should read a result from the RSU-G unit. The instruction stalls if a result is not yet ready, and resets the unit to begin the next evaluation when it returns the result. Depending on the specific ISA, this instruction can be encoded in a variety of ways. Our current RSU-G design requires 3 bits to specify one of 6 control registers (map table hi, map table low, down counter, neighbor 0-3, singleton A, singleton D) and an additional bit for reading the result. We assume an *RSU* instruction takes one cycle to execute and that a thread can issue only one per cycle (i.e., 32 for a 32-wide warp/wavefront).

Initialization. The intensity map table is 128 total bytes (256 entries x 4 bits) and can be initialized using the *RSU* instruction twice by packing values into a 64-bit register. The down counter is initialized with a single 6-bit value using the *RSU* instruction. The total initialization time is only 3 cycles.

Execution. The remaining values transferred to the RSU-G unit are 6-bit quantities and we assume are packed into 32 or 64-bit registers. This allows for a peak sustained initiation rate of one random variable sample per cycle, but is application dependent.

We assume the per pixel instructions can be overlapped with the final iterations of the down counter, when necessary, and staged to begin executing the next pixel as soon as possible for example by using software pipelining. When the *RSU* instruction that reads a result completes (i.e., returns a new label), a new pixel execution can begin. This requires a small amount of control circuitry to reset the down counter and write values into neighbor and singleton control registers.

Context Switches. An RSU-G may maintain state over many cycles (i.e., iterating over many labels). On a general-purpose core, the operating system could save and restore the intermediate state of each RSU-G unit when an exception/interrupt occurs. This allows switching between multiple applications that are all using the RSU-G units. This requires saving the map tables, counter, neighbor labels, singleton data, and any potential solutions stored in the selection unit. To reduce this state we can identify each instance of random variable evaluation (step 3 above) as the boundary for an idempotent sequence and restart execution at step three [14, 18]. This reduces the state to only per application (map tables and initial counter value) and would add only a few cycles per RSU-G unit. GPUs currently do not support context switches, so it is not an issue.

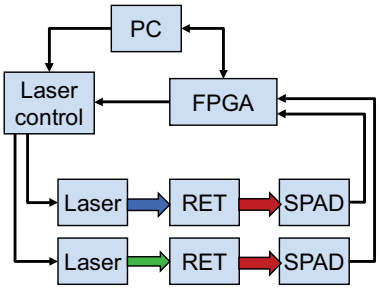
6.2 Discrete Accelerator

An alternative to augmenting an existing processor or GPU with RSU-G units is to design a custom discrete accelerator. This removes the constraints placed on general-purpose cores to support a wide variety of applications and instead allows us to focus on achieving the maximum performance. This design assumes that all control and data movement is implemented using custom logic where datapaths and register sizes can be specialized to match RSU-G's. We expect this to be the highest performing approach and analytically investigate this upper limit on performance.

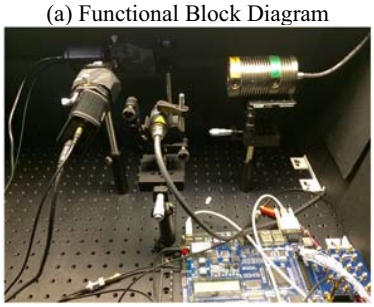
7. A Macro-Scale RSU-G_2 Prototype

To experimentally demonstrate the operation of a RET-based sampling unit we developed a macro-scale prototype of an un-pipelined RSU-G_2. Figure 6 shows a block diagram and a picture of our setup. We use this prototype to demonstrate the ability to parameterize a distribution for sampling and for use in a proof-of-concept image application. These experiments are an important step forward in the development of RET-based sampling units.

The prototype RSU follows the basic design of a generic RSU, with parameterization and a RET circuit for sampling the distribution. Macro-scale lasers are used to illuminate two RET networks and two SPADs to detect the output fluorescence from each network, or channel, to implement a RSU-G_2 (two-wide). The FPGA implements the time-to-fluorescence circuitry with sufficient timing precision to resolve 250ps differences in photon arrival times. Parameterization is performed in software on the PC by varying the laser source intensity to achieve different relative probabilities of photon detection between channels.



(a) Functional Block Diagram



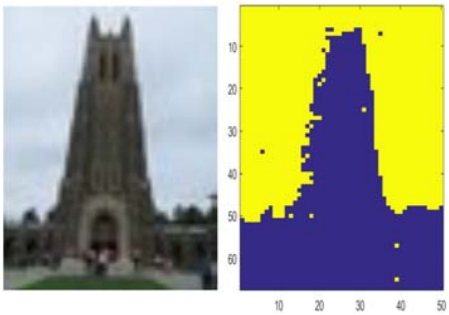
(b) Photo of Prototype

Figure 6: Macroscale RSU-G Prototype.

Our first experiment demonstrates the ability to parameterize the distribution. We vary the ratio of relative probabilities for the two RET circuits from 1 to 255 and observe that the prototype can achieve desired pairwise relative probabilities within 10% when the ratio is below 30, and 24% for higher ratios. A finer characterization and control of the prototype components could further improve the accuracy.

We also use the prototype on a simple image segmentation problem with only two possible labels (e.g., foreground and background). In this demonstration a PC executes the outer loop of an MCMC solver for the image segmentation MRF. Energy (singleton and doubleton) calculation and intensity mapping is also performed in software and the RSU-G_2 is used to sample from the output label distribution.

Figure 7 shows an input image (50x67) and representative output after 10 iterations of the MCMC. The prototype RSU-G_2 sampling takes no longer than ~2μs per pixel (since these are discrete components the hardware incurs significant electrical delays) but is dwarfed by the delays incurred to interface the proprietary laser controller (60 sec/image-iteration). These delays can be reduced significantly by electro-optical integration with CMOS.



(a) (b)

Figure 7: Image Segmentation on the Prototype: (a) Input Image and (b) the Sample at the 10th Iteration.

We conclude from the prototype demonstration that the fundamental operational principle of the RSU-G is sound and that it can be used as the underpinning for MRF based Bayesian Inference problems. Next, we evaluate the potential performance of the RSU-G.

8. Evaluation

This section provides preliminary analysis of RSU based systems in terms of performance, power and area.

8.1 Methodology

We use several image-processing applications (image segmentation, stereo vision matching, and dense motion estimation) for evaluation. Our image segmentation application assigns one of five possible values (labels) to each pixel by grouping similar pixels based on intensity [11, 37]. The stereo vision application similarly assigns one of 5 labels to align two images [39]. The dense motion estimation application searches over a 7x7 block to find the most likely position of a pixel in a subsequent frame (49 possible values) [17]. Although application specific implementations for these problems exist, our goal is to demonstrate the potential of RSU-G for the general MRF-MCMC Bayesian Inference framework. We use a single core of an Intel E5-2640 for image segmentation and stereo vision, but focus primarily on using an NVIDIA GTX Titan X GPU, for image segmentation and motion estimation. We functionally verify against MATLAB versions of the algorithms.

For performance evaluation, we first create best-effort implementations for the two applications via standard MCMC in C/C++ and CUDA as baselines. Then we replace select code sequences in the baseline implementations whose functionality is encapsulated in RSU-G units with appropriate instructions to match the theoretical timing of *RSU* instructions. We examine the assembly to ensure that all loads and stores are still present in the modified programs. We run image segmentation for 5,000 MCMC iterations and motion estimation for 400 iterations and measure the time to complete all iterations.

The standard MCMC process could be optimized by pre-calculating singleton values for each pixel and storing them in memory. Singletons are constant over all iterations, and thus can be pre-calculated and loaded from memory when needed rather than computed again. However, this optimization doesn't scale well; the memory required grows linearly with the number of pixels and the number of labels. Moreover, GPUs have limited global memory and this optimization will not work well for large images and large label sets. Nonetheless, we implement this optimization as an additional comparison for two of our applications.

We use the OpenMP *omp_get_wtime* routine on the CPU and the CUDA *nvprof* profiling tool to time all the iterations for the baseline implementations and the RSU-G emulated versions. This approach ignores the area overhead required to support the RSU-G units; however, it provides an estimate of how future chip area could be used to improve performance. Developing a more detailed, cycle accurate evaluation framework is ongoing work.

Finally, we analytically bound the speedup of RSU-G units using memory bandwidth as the limiter, which can be regarded as organizing RSU-G units in a specialized discrete accelerator. Based on memory bandwidth limits and the number of bytes needed for each pixel each iteration, we calculate an upper bound on performance for our current RSU-G design.

8.2 Performance

Figure 8 illustrates the speedups for our two applications with different image sizes on a GPU augmented with RSUs. For image segmentation, RSU-G_1 systems provides speedup of 3.2 over the baseline GPU for images of size 320x320, and 3.0 for HD images of size 1080x1920. Speedups over the optimized GPU implementation are 2.5 and 2.4 for small and HD images, respectively. For dense motion estimation, RSU-G_1 systems provides speedup of 6.4 and 12.8 for 320x320 images and achieve 7.5 and 16.06 for HD images. Dense motion estimation benefits from a wider RSU-G_4 design since it has more labels to evaluate (M=49) and achieves speedups of 23 for small images and 34 for HD images over the baseline GPU implementation.

Table 2: Application Execution Time (seconds).

Size	GPU	Opt GPU	RSU-G_1	RSU-G_4
Image Segmentation				
Small	0.3	0.23	0.09	0.09
HD	3.2	2.6	1.1	1.1
Dense Motion Estimation				
Small	0.55	0.27	0.04	0.02
HD	7.17	3.35	0.45	0.21

Our applications spend approximately equal time between energy calculations (parameterization) and sampling; therefore, the performance improvement is equally divided between the CMOS specialization of parameterization and RET-based sampling. Secondary effects from using RSU-G also influence speedup. Fewer instructions take less time to execute, but also reduces register pressure and increases processor occupancy. These two factors both contribute to our observed speedups.

We also ran sequential versions of image segmentation and stereo vision on an Intel E5-2640. The achieved speedup of an RSU-G_1 augmented processor was over 100; however, the GPU is a better comparison given the embarrassingly parallel nature of the applications.

Discrete Accelerator. GPUs have many architecture features that keep RSU-G units from providing their maximum performance, such as memory architecture, warp

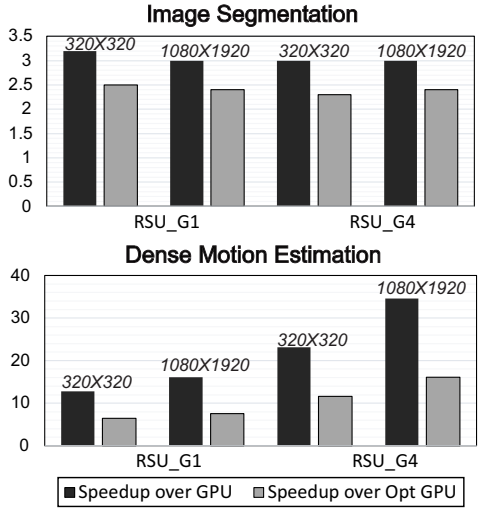


Figure 8: RSU Speedup over GPU.

granularity, and scheduling policy. A discrete accelerator could achieve even better performance by trading off flexibility. We use DRAM bandwidth to bound the best-case performance that a discrete accelerator can achieve by considering how much data is required for each application.

For image segmentation, each pixel needs 5 bytes (1 initial intensity, 4 neighboring pixels' labels) per MCMC iteration. Motion estimation needs 54 bytes (49 destination intensities and 1 initial intensity for singleton calculation, 4 neighboring pixels' labels) per pixel per MCMC iteration. Using image sizes and DRAM bandwidth limits, we can compute execution time.

Assuming DRAM bandwidth is 336GB/s, the GTX Titan X bandwidth, and the accelerator consumes data at DRAM bandwidth. For image segmentation, the accelerator achieves an additional 12.1x and 7x speedup over the RSU-G_1 augmented GPU for 320x320 and 1080x1920 (HD) images, respectively. Dense motion estimation achieves additional speedup of 6.5x and 3.4x for 320x320 and 1080x1920 images, respectively. The lower speedup for HD images is because HD images saturate the GPU while 320x320 images don't. Thus, for a discrete accelerator, the upper bound of speedups over standard MCMC on the GPU is 39 (image segmentation) and 84 (dense motion estimation) for 320x320 images and 21 and 54 for HD images. The discrete accelerator achieves speedup of only 1.55x over the RSU-G_4 augmented GPU for motion estimation of HD images, since RSU-G_4 nearly saturates memory BW. The number of RSU-G units required to achieve these speedups is #units = BW/frequency/bytes_per_cycle. Assuming the Titan X 1GHz frequency and that each unit consumes only 1 byte of data per cycle. Achieving these speedups requires approximately 336 RSU-G_1 units in the accelerator. Further speedups are possible by using on-chip storage to increase memory bandwidth and staging image frames. The number

of RSU-G units needed scales linearly with available memory bandwidth.

8.3 Power & Area

We obtain power and area estimates of RSU-G_1 from the synthesized Verilog in 45nm, Cacti, and a predictive 15nm process [26] for the CMOS portions and first principles for the RET circuit [6, 15, 23, 32, 34]. We do not scale the RET circuit.

Power. The power for a single RSU-G_1 in 15nm is 3.91mW and is dominated by the electrical power 3.75mW, the RET circuits consume only 0.16mW. A GPU augmented with RSU-G units (3072 in total) consumes 12W of additional power when they are all active. The accelerator with 336 units bounded by 336GB/s DRAM consumes only 1.3W for the RSU-G units. Additional power would be consumed for the memory controller and the control logic. Table 3 lists the power consumption breakdown by RSU-G_1 component.

Table 3: Power Consumption for a Single RSU-G_1.

Power(mW)	45nm (590MHz)	15nm (1GHz)
Logic	7.20	2.33
RET Circuit	0.16	0.16
LUT	3.92	1.42*
Total	11.28	3.91

*theoretically scaling LUT from 32nm to 15nm [40].

Area. We estimate area of a RSU-G unit by first observing that the SPAD (~1μm^2 [6, 23, 32]) and QD-LEDs (~16*25μm^2 [15, 34]) dominate the RET circuit area requirements. The volume of RET network ensemble (~N*20*20*2nm^3) is very small and can reside in a layer above the SPAD. Therefore, we estimate a single RET circuit requires 400μm^2 and all the RET circuits in an RSU-G_1 unit require 0.0016mm^2. Table 4 lists the area breakdown by RSU-G_1 component.

Table 4: Area for a Single RSU-G_1.

Area(μm^2)	45nm	15nm
Logic	2275	642
RET Circuit	1600	1600
LUT	1798	656*
Total	5673	2898

*theoretically scaling LUT from 32nm to 15nm [38, 40].

9. Limitations and Future Work

The current RSU-G implementation is for very specific MRF problems. Extending the design to support other MRF problems is a short-term goal. We are actively investigating the width and depth of RSU pipelines, including configurable width for single cycle random variable sampling. Our performance evaluation is only approximate since we use emulation and we are actively developing a more accurate performance evaluation platform.

Although different aspects of integration with CMOS have been demonstrated, a complete integration has not been demonstrated. Another issue is chromophore (RET Network) longevity; the presence of oxygen limits the number of excitation cycles through the equivalent of a wear-out process. We can address this issue in two ways: 1) using a larger number of RET networks per RET circuit and 2) encapsulating the chromophores to protect against oxygen.

Exploring a generalized RSU that allows programmability across parameterization and distribution is an interesting, but challenging area of future work. The long-term goal is to explore using the RSU for general problems in statistical inference and stochastic neural networks as well as domain specific applications such as rare event simulation.

10. Conclusion

Markov Random Field Bayesian Inference is a common problem in probabilistic machine learning. MCMC is a probabilistic algorithm for solving these problems that relies on sampling from parameterized distributions. We present RSU-G, a molecular-scale optical Gibbs sampling unit based on Resonance Energy Transfer between chromophores to accelerate MRF Inference. By implementing specified clique potentials, an RSU-G unit can be designed for any first-order MRF problem. We present a specific RSU-G for first-order MRFs with smoothness-based priors, which encompasses a large number of applications in low-level computer vision.

We experimentally demonstrate the basic operation of RSU-G to provide samples from parameterized exponential distributions. We explore the architectural integration of RSU-G units in two designs: 1) augmenting processors (CPU/GPU) with RSU-G units and 2) a custom discrete accelerator. Evaluation of these architectures shows significant potential speedup for low-level vision applications with low power consumption and reasonable area.

There are many possible ways to implement samplers using RET in different applications, and we explore a small portion of the overall space. Nonetheless, the work presented here represents a significant step forward for exploiting emerging technologies to provide architectural support for probabilistic algorithms.

Acknowledgments

This project is supported in part by the Defense Advanced Research Projects Agency (DARPA) (grant W911NF-13-1-0096) and the National Security Science and Engineering Faculty Fellowship (NSSEFF) ONR (grant N00014-15-1-0032).

References

[1] *C++ Pseudo-Random Number Generation Library*. Available: http://en.cppreference.com/w/cpp/numeric/random

[2] *Intel® Digital Random Number Generator (DRNG) Software Implementation Guide*. Available: https://software.intel.com/sites/default/files/managed/4d/91/DRNG_Software_Implementation_Guide_2.0.pdf

[3] *Verilog Implementation of AES as Specified in NIST FIPS 197*. Available: https://github.com/secworks/aes

[4] R. S. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, "General-Purpose Code Acceleration with Limited-Precision Analog Computation," *Proceedings of the 41st Annual International Symposium on Computer Architecture (ISCA),* 2014 2014.

[5] M. Aono, M. Naruse, S.-J. Kim, M. Wakabayashi, H. Hori, M. Ohtsu, and M. Hara, "Amoeba-Inspired Nanoarchitectonic Computing: Solving Intractable Computational Problems Using Nanoscale Photoexcitation Transfer Dynamics," *Langmuir,* vol. 29, pp. 7557-7564, 2014/07/28 2013.

[6] S. Assefa, F. Xia, and Y. A. Vlasov, "Reinventing Germanium Avalanche Photodetector for Nanophotonic on-Chip Optical Interconnects," *Nature,* vol. 464, pp. 80-84, 03/04/print 2010.

[7] J. Bornholt, T. Mytkowicz, and K. S. McKinley, "Uncertain<T>: A First-Order Type for Uncertain Data," in *Asplos '14: Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, 2014, pp. 51-66.

[8] L. N. Chakrapani, B. E. S. Akgul, S. Cheemalavagu, P. Korkmaz, K. V. Palem, and B. Seshasayee, "Ultra-Efficient (Embedded) Soc Architectures Based on Probabilistic CMOS (PCMOS) Technology," in *Design, Automation and Test in Europe, 2006. DATE '06. Proceedings*, 2006, pp. 1-6.

[9] L. Devroye, *Non-Uniform Random Variate Generation*: Springer-Verlag, 1986.

[10] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural Acceleration for General-Purpose Approximate Programs," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, 2012, pp. 449-460.

[11] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* pp. 721-741, 1984.

[12] S. Geman and C. Graffigne, "Markov Random Field Image Models and Their Applications to Computer Vision," in *Proceedings of the International Congress of Mathematicians*, 1986, p. 2.

[13] G. D. Hager, M. D. Hill, and K. Yelick, *Opportunities and Challenges for Next Generation Computing (White Paper)*. Computing Community Consortium, 2015.

[14] M. Hampton and K. Asanović, "Implementing Virtual Memory in a Vector Processor with Software Restart Markers," in *Proceedings of the 20th annual international conference on Supercomputing*, Cairns, Queensland, Australia, 2006, pp. 135-144.

[15] M. T. Hill and M. C. Gather, "Advances in Small Lasers," *Nat Photon,* vol. 8, pp. 908-918, 12//print 2014.

[16] G. Juzeliūnas and D. L. Andrews, "Quantum Electrodynamics of Resonance Energy Transfer," in *Advances in Chemical Physics*, ed: John Wiley & Sons, Inc., 2007, pp. 357-410.

[17] J. Konrad and E. Dubois, "Bayesian Estimation of Motion Vector Fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 14, pp. 910-927, 1992.

[18] M. d. Kruijf and K. Sankaralingam, "Idempotent Processor Architecture," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, Porto Alegre, Brazil, 2011, pp. 140-151.

[19] C. LaBoda, H. Duschl, and C. L. Dwyer, "DNA-Enabled Integrated Molecular Systems for Computation and Sensing," *Accounts of chemical research,* vol. 47, pp. 1816-1824, 2014.

[20] S. Z. Li and S. Singh, *Markov Random Field Modeling in Image Analysis* vol. 26: Springer, 2009.

[21] L. Luan, R. D. Evans, N. M. Jokerst, and R. B. Fair, "Integrated Optical Sensor in a Digital Microfluidic Platform," *Sensors Journal, IEEE,* vol. 8, pp. 628-635, 2008.

[22] L. Luan, M. W. Royal, R. Evans, R. B. Fair, and N. M. Jokerst, "Chip Scale Optical Microresonator Sensors Integrated with Embedded Thin Film Photodetectors on Electrowetting Digital Microfluidics Platforms," *Sensors Journal, IEEE,* vol. 12, pp. 1794-1800, 2012.

[23] S. Mandai, M. W. Fishburn, Y. Maruyama, and E. Charbon, "A Wide Spectral Range Single-Photon Avalanche Diode Fabricated in an Advanced 180 nm CMOS Technology," *Optics express,* vol. 20, pp. 5849-5857, 2012.

[24] V. K. Mansinghka, "Natively Probabilistic Computation," PhD, Brain and Cognitive Sciences, MIT, 2009.

[25] V. K. Mansinghka and E. Jonas, "Building Fast Bayesian Computing Machines out of Intentionally Stochastic, Digital Parts," *Computing Research Repository (ArXiv),* vol. abs/1402.4914, 2014.

[26] 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*, Monterey, California, USA, 2015, pp. 171-178.

[27] T. P. Minka, "Expectation Propagation for Approximate Bayesian Inference," in *Proceedings of the Seventeenth conference on Uncertainty in artificial intelligence*, 2001, pp. 362-369.

[28] M. Mitzenmacher and E. Upfal, *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*: Cambridge University Press, 2005.

[29] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*: MIT press, 2012.

[30] M. Naruse, M. Aono, and S.-J. Kim, "Nanoscale Photonic Network for Solution Searching and Decision Making Problems," *IEICE Transactions on Communications,* vol. E96.B, pp. 2724-2732, 2013.

[31] K. V. Palem, "Energy Aware Computing through Probabilistic Switching: A Study of Limits," *Computers, IEEE Transactions on,* vol. 54, pp. 1123-1137, 2005.

[32] D. Palubiak, M. M. El-Desouki, O. Marinov, M. Deen, and Q. Fang, "High-Speed, Single-Photon Avalanche-Photodiode Imager for Biomedical Applications," *Sensors Journal, IEEE,* vol. 11, pp. 2401-2412, 2011.

[33] C. Pistol and C. Dwyer, "Scalable, Low-Cost, Hierarchical Assembly of Programmable DNA Nanostructures," *Nanotechnology,* vol. 18, pp. 125305-9, 2007.

[34] G. Shambat, B. Ellis, J. Petykiewicz, M. A. Mayer, A. Majumdar, T. Sarmiento, J. S. Harris, E. E. Haller, and J. Vuckovic, "Electrically Driven Photonic Crystal Nanocavity Devices," *IEEE Journal of Selected Topics in Quantum Electronics,* vol. 18, pp. 1700-1710, 2012.

[35] M. Stipčević and Ç. K. Koç, "True Random Number Generators," in *Open Problems in Mathematics and Computational Science*, ed: Springer, 2014, pp. 275-315.

[36] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors," *Pattern Analysis and Machine Intelligence, IEEE Transactions on,* vol. 30, pp. 1068-1080, 2008.

[37] T. Szirányi, J. Zerubia, L. Czúni, D. Geldreich, and Z. Kato, "Image Segmentation Using Markov Random Field Model in Fully Parallel Cellular Network Architectures," *Real-Time Imaging,* vol. 6, pp. 195-211, 2000.

[38] S. Taejoong, R. Woojin, J. Jonghoon, Y. Giyong, P. Jaeho, P. Sunghyun, B. Kang-Hyun, B. Sanghoon, O. Sang-Kyu, J. Jinsuk, K. Sungbong, K. Gyuhong, K. Jintae, L. YoungKeun, K. Kee Sup, S. Sang-Pil, Y. Jong Shik, and C. Kyu-Myung, "13.2 a 14nm FinFET 128Mb 6T SRAM with VMIN-Enhancement Techniques for Low-Power Applications," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, 2014, pp. 232-233.

[39] M. F. Tappen and W. T. Freeman, "Comparison of Graph Cuts with Belief Propagation for Stereo, Using Identical MRF Parameters," in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, 2003, pp. 900-906.

[40] S. Thoziyoor, N. Muralimanohar, J. Ahn, and N. Jouppi, "Cacti 5.3," *HP Laboratories, Palo Alto, CA,* 2008.

[41] B. Valeur and M. N. Berberan-Santos, *Molecular Fluorescence: Principles and Applications*: John Wiley & Sons, 2012.

[42] S. Wang, A. R. Lebeck, and C. Dwyer, "Nanoscale Resonance Energy Transfer-Based Devices for Probabilistic Computing," *Micro, IEEE,* vol. 35, pp. 72-84, 2015.

[43] J. M. Winn and C. M. Bishop, "Variational Message Passing," in *Journal of Machine Learning Research*, 2005, pp. 661-694.