Title
Autonomous probabilistic coprocessing with petaflips per second

Permalink
https://escholarship.org/uc/item/6mg4d18h

Authors
Sutton, B
Faria, R
Ghantasala, LA
et al.

Publication Date
2020

DOI
10.1109/ACCESS.2020.3018682

Peer reviewed
Autonomous Probabilistic Coprocessing with Petaflips per Second

Brian Sutton, Rafatul Faria, Lakshmi A. Ghantasala, Risi Jaiswal, Kerem Y. Camsari, and Supriyo Datta

Abstract—In this paper we present a concrete design for a probabilistic (p-) computer based on a network of p-bits, robust classical entities fluctuating between -1 and +1, with probabilities that are controlled through an input constructed from the outputs of other p-bits. The architecture of this probabilistic computer is similar to a stochastic neural network with the p-bit playing the role of a binary stochastic neuron, but with one key difference: there is no sequencer used to enforce an ordering of p-bit updates, as is typically required. Instead, we explore sequencerless designs where all p-bits are allowed to flip autonomously and demonstrate that such designs can allow ultrafast operation unconstrained by available clock speeds without compromising the solution’s fidelity. Based on experimental results from a hardware benchmark of the autonomous design and benchmarked device models, we project that a nanomagnetic implementation can scale to mark of the autonomous design and benchmarked device models, by available clock speeds without compromising the solution’s fidelity, but we will restrict our discussion to linear functions defined by a set of weights $W_{ij}$ such that

$$I_i(t + \tau_S) = \beta \sum_j W_{ij} m_j(t) \quad (1)$$

where $\beta$ is a constant and $\tau_S$ is the ‘synapse time’, that is the time it takes to recompute the inputs $\{I_i\}$ every time the outputs $\{m\}$ change. In software implementations, each BSN is updated repeatedly according to

$$m_i(t + \tau_N) = \text{sgn} [\tanh (I_i(t)) - r_{[-1,1]}] \quad (2)$$

where $r_{[a,b]}$ represents a random number in the range $[a, b]$, and $\tau_N$ is the ‘neuron’ time, that is the time it takes for a neuron to provide stochastic output $m_i$ with the correct statistics dictated by a new input $I_i$.

It is well-known [22] to highlight its role as the key element of an autonomous p-computer (ApC), similar to the role of a q-bit in a quantum computer. We note that such an autonomous architecture in the absence of any clocking...
circuitry that controls the updating p-bits has recently been demonstrated in small scale using 8 magnetic tunnel junction based p-bits [26]. With this experimental demonstration of an 8 p-bit design, it is important to understand if such a system can scale effectively.

Herein we use FPGA emulation to demonstrate the operation of a scaled version of such an autonomous computer up to 8100 (90 × 90) p-bits with all the necessary peripheral circuitry, including programmable synapses that are used to map different problems to the co-processor. The FPGA implementation presented in this paper is specifically designed to capture the autonomous operation of probabilistic bits that fluctuate in time allowing us to make performance projections of a scaled implementation of the demonstration presented in Ref. [26]. This paper demonstrates the feasibility of an ApC that performs the weight logic and p-bit functions defined by Eqs. (1) and (2) without the aid of sequencers as portrayed in Figs. 1(a) and 1(b). Our work is motivated by the compact, fast, energy efficient hardware that are currently being developed for the implementation of these functions [27] as shown in Fig. 1(c), which we emulate using existing CMOS devices on an easily reconfigurable, cloud accessible digital FPGA platform, Fig. 1(d).

In the following sections, we will present an emulation framework for the study of scaled autonomous probabilistic coprocessing and use the framework to provide performance predictions of a design based on nanomagnets, helping to motivate such an implementation. Section II will provide an overview of the ApC, our analysis methodology, and present an abstract autonomous p-bit model with corresponding device benchmarking. Section III provides an overview of the design and implementation of a p-computing coprocessing framework using a cloud-accessible FPGA platform. Section IV presents results for two distinct applications using the coprocessor, one involving combinatorial optimization and one involving emulated quantum annealing. Finally, sections V and VI discuss how these applications help show that accurate results can be obtained with a sequencerless probabilistic computer of the type envisioned by Feynman [28], implemented using modern devices to enable operation at ultrafast rates unconstrained by the available clock speed in a sequenced design.

II. AUTONOMOUS P-COMPUTING MODEL

The building block for our ApC has four components as shown in Fig. 1(a)
- **weight logic** to implement Eq. (1),
- **p-bit** to implement Eq. (5) described below,
- **write unit** to program the weights $W_{ij}$ and $\beta$
- **read unit** to access the individual p-bit outputs

Fig. 1(b) shows how multiple building blocks can be interconnected to form a p-computer. The tiling shown is based on nearest-neighbor connections, but the connections need not be limited to nearest-neighbor. We have also implemented all-to-all networks using the digital emulator shown in Fig. 1(d) as discussed in section III. In the following sub-sections, we will introduce what is meant by “autonomous” operation, present a digital model for such an autonomous p-bit, and finally benchmark the digital model against a physical device model.

![Fig. 1: Autonomous p-Computer (ApC): a. A weighted p-bit building block is used to construct an ApC that comprises four components: weight logic, p-bit logic, weight programming, and p-bit readout. b. The individual building blocks are interconnected to construct an ApC with a desired topology. A nearest-neighbor coupling layout is depicted. c. A projected MRAM based ApC using nanomagnetic devices for the p-bit with resistor-based weight logic can be used to form a compact, efficient building block. d. Using all-digital technology, an FPGA was used to construct and ApC that emulates the MRAM based design. An example composition of a p-bit with a linear, register based weight logic, a lookup table based activation function, a linear-feedback shift register based pseudo random number generator, and a pseudo-asynchronous attempt logic is shown.]
A. Autonomous p-bit Operation

Superficially Fig. 1 looks like other existing neural network architectures, like the one used for TrueNorth [29]. However, to our knowledge, earlier implementations have used time-multiplexing [29], [30] to share the same resource among different neurons and synapses, while our objective is to eliminate sequencing and time-multiplexing altogether so that we are not constrained by available clock speeds. TrueNorth for example uses 4096 neurosynaptic cores, each core having dedicated neuron and synaptic memory forming 256 logical neurons that are time multiplexed sequentially to implement $4096 \times 256 \approx 1M$ logical neurons [29]. For our sequencerless operation we would need 1M distinct building blocks for the same number of neurons. This would be impractical if we were relying on fully digital implementations, however the compact hardware implementations currently being developed makes such a design feasible. As an example, stochastic behavior of nanomagnets has recently attracted attention in the context of novel computing paradigms, and they show promise in probabilistic and neuromorphic applications [31]–[45]. For example, an MRAM-based p-bit requires only 3 transistors and a magnetic tunnel junction [46], while its digital emulation requires significantly more transistors. With such compact hardware, it is feasible to have one building block for every p-bit in order to support sequencerless operation that is not limited by clock speeds.

We call this sequencerless operation of ANNs “autonomous” to distinguish it from the asynchronous operation that is widely used in the context of Spiking Neural Networks [29], [30].

As a quantitative measure of an ApC’s speed of operation we use the number of flips per second ($f$), a flip being defined as a p-bit update attempt (i.e. it may choose not to actually flip). For purely sequential updating, the number of flips per second is $\sim 1/(\tau_S + \tau_N)$. However, as mentioned earlier, updating need not be purely sequential since unconnected BSNs can be simultaneously updated without loss of fidelity. If a number ($N_p$) out of the total number ($N$) of BSNs can be updated in parallel, then the number of flips per second will be much larger $\sim N_p/(\tau_S + \tau_N)$. Note, however, that in order to achieve this enhanced flip rate, the number of neurons that are simultaneously updated, $N_p$, have to be deliberately selected using a digital sequencer, so that the clock period, $\tau_{\text{clock}}$, limits the maximum number of flips per second:

$$f \leq \frac{N_p}{\tau_{\text{clock}}} \quad (\text{Sequenced mode}) \quad (3a)$$

This clock speed will be limited by the synapse and neuron times, $\tau_S$, $\tau_N$ respectively, and the overhead associated with clock distribution [47].

The objective of this paper is to present a framework for clockless operation [23] whose speed is limited only by the neuron and synapse speeds

$$f \leq \frac{N}{\tau_N} = \frac{sN}{\tau_S} \quad (\text{Autonomous mode}) \quad (3b)$$

where $s = \tau_S/\tau_N$. We will show that this autonomous mode provides high fidelity results without supervision keeping the fraction of detrimental simultaneous updates down to an acceptably low. Detrimental updates are managed simply by choosing a small $s$ so that $\tau_N$ is much longer than $\tau_S$, without using a digital sequencer to enforce a deliberate update order. As a result, $f$ is not limited by $\tau_{\text{clock}}$ and can continue to operate faster as the synapse time $\tau_S$ is lowered. For example if we have nearest neighbor connections with weights of $\{-1, 0, +1\}$, then the synapse can be implemented with short wires which respond in times less than 10 to 100 ps [48], much shorter than typical clock periods.

Eqs. (3a) and (3b) suggest that an autonomous design will allow faster operation (that is, more flips per second) if

$$\tau_S \leq \frac{sN}{N_p} \tau_{\text{clock}} \quad (4)$$

The factor $N_p/N$ represents the fraction of neurons that can be updated simultaneously, which depends on the fan-in, the nature of interconnections, and problem topology. For example, with nearest neighbor connections on a 2D square lattice as in Fig. 1(b) half the nodes can be updated simultaneously so that $N_p/N = 1/2$. The factor $s$ also depends on the interconnections, but it additionally depends on the nature of the problem and the degree of solution fidelity needed, as we will show in this paper.

Ideally, we would implement a scaled version of an ApC using nanomagnetic devices to explore the performance of such a design. However, given current technological limitations, we will model the operation of p-bit hardware using a digital FPGA platform. In the following section we discuss how we use a digital, synchronous device to emulate intrinsically asynchronous nanomagnets.

B. Autonomous p-bit Model

As digital platforms are inherently synchronous, we mimic autonomous operation by replacing Eq. 3 with a new hardware-inspired model, Eq. 5 below that we benchmarked against established device models (section II-C). These equations are based on SPICE simulations of Boltzmann networks where the update order of p-bits becomes irrelevant due to the symmetric coupling between connected p-bits. Such a clockless circuit corresponds to the asynchronous parallelism scheme used to realize Boltzmann Machines in hardware with no asymptotic guarantees for convergence [23] unless all p-bits operate with up-to-date information that is enabled by fast synapses [49]. This model is valid for such networks, however, for other networks such as those with directed connections, the update ordering of p-bits may be important and other hardware models more appropriate for these systems are likely required and are not discussed in this paper.

At each time step, all p-bits are free to flip and they do so with a probability $\sim s$ that is controlled by the input $I_i$ having a zero-input value $s(I_i = 0) = s_0 \ll 1$.

$$m_i(n+1) = m_i(n) \times \text{sgn}(e^{-r} - r_{[0,1]}) \quad (5a)$$

$$s = s_0 e^{-m_i(n) I_i(n)} \quad (5b)$$

As each p-bit flips with a probability $\sim s$ in each time step, the average time taken for a p-bit to respond is $1/s$. Since time
steps are measured in units of $\tau_S$, we have $\tau_N = (1/s) \times \tau_S$ as stated earlier. Unlike Eq. (2), Eq. (5) can be used to update all p-bits in parallel without explicitly worrying about simultaneous updates. With small values of $s_0$, the fraction of simultaneous updates is sufficiently small such that Eq. (5) in an unsequenced mode gives results equivalent to those obtained from careful sequencing using Eq. (2).

For a given network topology and embedded problem [50], [51], the value of $s$ that ensures convergence to thermal equilibrium must be identified in order to assess the amount of parallelism in the design. The desire is to find the smallest value of $s$ that converges the thermal equilibrium for the problem specific Boltzmann distribution. In section IV we explore two example problems and evaluate the degree of parallelism obtainable. For example, in the nearest-neighbor design of Fig. 7 a value of $s = 1/4$ ensures convergence for a network of 8100 neurons. This value of $s$ states that on average the number of neurons that update within each synapse delay is $N \times s = 2025$. However, as the problem complexity increases, i.e. the incorporation of on-site biases, the acceptable value of $s$ decreased to 1/12 as shown in Fig. 8. Ultimately, the term $s$ drives the physical design of the synapse and neuron implementation.

C. Model Benchmarking with Stochastic sLLG

The autonomous p-bit model of (5a) and (5b) was benchmarked against a coupled stochastic Landau-Lifshitz-Gilbert (sLLG) equation. Magnetization dynamics of the modeled circular stochastic nanomagnet are captured by solving the sLLG equation [52]:

$$(1 + \alpha^2) \frac{d\vec{m}}{dt} = -|\gamma| \vec{m} \times \vec{H} - \alpha |\gamma| (\vec{m} \times \vec{m} \times \vec{H}) + \frac{1}{qN_s} (\vec{m} \times \vec{I}_S \times \vec{m}) + \left( \frac{\alpha}{qN_s} (\vec{m} \times \vec{I}_S) \right)$$  \hspace{1cm} (6a)

where $\alpha$ is the damping coefficient, $\gamma$ is the electron gyromagnetic ratio, $N_s = N_s V_{ol}/\mu_B$ is the total number of Bohr magnetons in the magnet, $M_s$ is the saturation magnetization, $\vec{H} = \vec{H}_d + \vec{H}_a$ is the effective field including the out-of-plane ($\hat{x}$ directed) demagnetization field $\vec{H}_d = -4\pi M_s m_z \hat{x}$, as well as the thermally fluctuating magnetic field due to the three dimensional uncorrelated thermal noise $H_n$ with zero mean $\langle H_n \rangle = 0$ and standard deviation $\langle H_n^2 \rangle = 2kT/|\gamma| M_s V_{ol}$. Along each direction, $\vec{I}_S$ is the applied spin current to the nanomagnet. The HSPICE solver we employed is based on spherical coordinates that solve for $(\theta, \phi)$, but the noise is first included as three uncorrelated random magnetic fields in Cartesian coordinates before being turned into spherical coordinates [53]. The HSPICE solver uses the .trannoise function [53] and is benchmarked against our own MATLAB implementation that uses the Stratonovich convention [54], as well as the Fokker-Planck Equation [52], [53]. We have used a time step of 1 ps for the chosen parameters which is verified by comparing the equilibrium fluctuations of single nanomagnets that are obtained numerically, against the expected Boltzmann distribution.

![Benchmarking the behavioral model with sLLG using Euclidean distance](image)

**Fig. 2: Benchmarking the behavioral model with sLLG using Euclidean distance:** Using a random Sherrington-Kirkpatrick spin glass instance for different network sizes, $N$, the behavioral model is benchmarked against sLLG as a function of time. Each point on the graph represents the Euclidean distance from the ideal Boltzmann distribution and the ensemble solution obtained from the behavioral model and sLLG. The steady state error will depend on the number of ensembles as shown by the black dotted line.

Normally, a nanomagnet based p-bit thresholds its continuous output with an inverter [55], [56] that is typically included in SPICE simulations with additional transistors. In order to simplify numerical simulations in the present context, we artificially threshold the sLLG outputs (such that $m_z > 0 \rightarrow 1$ and $m_z < 0 \rightarrow 0$) at each time step. This allows a binarization of the sLLG outputs that allow each p-bit to have a binary output, according to Eq. (2). Note that in an actual device implementation, a single inverter is enough to threshold the output given that a moderate tunneling magnetoresistance value ($TMR=R_{AP}/R_P \approx 100\%$) leads to voltage fluctuations of $\approx 200$ mV's over a voltage division of $V_{DD} = 0.8$ V, which is more than enough for a single inverter to threshold these fluctuations to rail-to-rail voltages [55]. For a detailed analysis of the nanomagnet based p-bit design that includes device-to-device variations, see Ref. [26].

Individual p-bits are coupled according to:

$$I_{k}\langle t + \Delta t \rangle = \beta I_{0} \sum_j W_{ij} \text{sgn}(m_j(t))$$  \hspace{1cm} (7)

where, $I_{0}$ is the tanh fitting parameter of the sigmoidal response (sgn($m_j$) versus spin current $I_{k}$ along z-direction). Equation 7 and Equation 5 constitute the autonomous behavioral model that is used to benchmark the results of coupled sLLG equations, which we refer to as “behavioral model” hereon. In the benchmark, a circular disk magnet with a vanishing anisotropy ($H_K$) is used with the parameters: diameter $D = 150$ nm and thickness $t = 2$ nm, $\alpha = 0.01$, $M_s = 1100$ emu/cc, $H_K = 1$ Oe resulting in an autocorrelation time of $\tau_{corr} = 1.372$ ns and $I_{0} = 1$ mA. A fitting parameter of 1.4 is used in the behavioral model for $\tau_N$, i.e. $\tau_N = 1.4\tau_{corr}$.

We use a simulated Sherrington-Kirkpatrick [59] spin glass network with a random coupling matrix and random bias between -1 and +1. The benchmarking of the proposed be-
havioral model with the coupled sLLG network, analogous to the probabilistic circuit proposed in \([57]\), is accomplished by comparing two different quantities: (1) Euclidean distance and (2) Free energy.

Euclidean distance is defined by:

\[
ED = \sum_{i=1}^{N} (P_i - P_{i,Boltzmann})^2 \tag{8}
\]

where \(P_i\) is the probability of occurrence of the \(i\)-th configuration computed out of 4000 ensembles at each time step of the simulation. \(P_{i,Boltzmann}\) is computed from the joint probability distribution obtained from a Boltzmann law.

The second benchmark approach is based on a comparison of the free energy of the system with what is expected from the principles of statistical mechanics. Free energy is defined by \([58]\) the partition function \(Z\):

\[
FE = \frac{\ln(Z)}{-\beta} \tag{9}
\]

where, \(\beta\) is the pseudo-inverse temperature. Partition function \(Z\) is given by:

\[
Z = \sum_k \exp(-\beta E_k) \tag{10}
\]

where \(k\) represents different configurations of the network. Energy of a specific configuration is defined by:

\[
E_k = -0.5 \sum_{i,j, i \neq j} W_{ij} m_i m_j - h_i m_i - h_j m_j \tag{11}
\]

When numerically calculating free energy from the sLLG data, the following steps have been applied (similar to the importance sampling method described in \([59]\)):

1. The probability of different configurations, \(P_i\), are calculated out of 4000 ensembles for each time step.
2. For each \(P_i\) larger than a certain threshold value \(P_{th}\), the partition function \(Z_i = \exp(-I_0 E_i)/P_i\) is calculated, so that outliers are excluded.
3. For each \(Z_i\), the free energy \(FE_i = -\ln(Z_i)/I_0\) is calculated.
4. Finally the mean of all \(FE_i\) is computed.

The above method is suitable for small examples, but may not scale due to the difficulty in empirically calculating different probabilities \(P_i\) as the network size grows. The striking agreement between the sLLG model and the behavioral model given by Eq. \((5)\) shown in Fig. 2 and Fig. 3 helps establish the validity of Eq. \((5)\) as a suitable digital model of an autonomous, stochastic MTJ-based computer.

III. Emulation Framework

Having established a model for the autonomous p-bit operation, in this section we describe the design and implementation of an FPGA based framework to explore the performance, scalability, and other characteristics of an ApC.

A. Autonomous FPGA Coprocessor

An all-digital framework based on Eqs. \((5a)\) and \((5b)\) was developed, Fig. 4, to facilitate architectural exploration of an ApC, study various trade-offs in p-bit, weight logic, and topology design, and to accelerate the combinatorial optimization and sampling problems explored in section IV. The digital framework leverages reconfigurable computing devices to support rapid exploration of different designs. A Xilinx Virtex Ultrascale+ xcvu9p-flgb2104-2-i provided via Amazon Web Services F1 cloud-accessible EC2 compute instances was used for the ApC implementation. While a Xilinx FPGA was used in this work, the design is hardware agnostic and another device, e.g. an Intel Stratix FPGA, could readily be used.

As shown in Fig. 1(b) and Fig. 4(b), an ApC comprises multiple weighted p-bits arranged in various topologies, each supporting programmable problem instances. There are many options for the implementation of programmable control, weight logic, p-bits, and p-bit readout in a digital platform. The digital ApC of Fig. 4 comprises a modular weighted p-bit, Fig. 4(c), that can be organized into various topologies, Fig. 4(b), supporting programmed problem instances. An example weighted p-bit implementation is shown in Fig. 1(d) leveraging a memory-mapped weight-logic register bank supporting linear weight coupling. The output of the weight logic block is provided to a programmable, activation function look-up table that is used in conjunction with a Linear Feedback Shift Register (LFSR) based pseudo-random function to implement Eqs. \((5a)\) and \((5b)\). Additional options were developed and explored for these building block elements, see section III-B and Fig. 6, beyond what is shown in Fig. 1(d).

Interaction with the FPGA framework is provided through MATLAB MEX programs in a client-server command driven model, Fig. 4(a). Clients issue commands to select which of the pre-built topologies to program into the cloud FPGA.

Fig. 3: Benchmarking the behavioral model with sLLG using Free Energy: The free energy calculated for the random Sherrington-Kirkpatrick spin glass instance of Fig. 2 from the behavioral model is benchmarked against sLLG as a function of time for network sizes \(N = 16\) and \(N = 24\), showing convergence to the free energy obtained from Boltzmann law.
instance, the current pseudo-temperature for the network, problem specific weights, and options to pause or resume the network. Commands are also provided to support random sampling from the network for readout operation. Online annealing is directly supported through global update operations of the activation function look-up. All weights are dynamically programmable through memory-mapped operations. The server interfaces with a PCIe Express (PCIe) attached FPGA, interacting with the programmed design as commanded. Other clients such as Octave and Python are readily supported through a C++ abstraction layer leveraging networking and serialization libraries.

Fig. 5 depicts the logical organization of the FPGA ApC design used herein. All interaction with the FPGA is performed using a PCIe Gen3 x16 interface supporting direct memory access (DMA) transactions (requester). Programmable control of the design is accomplished using an AXI-Lite 32-bit completer interface and memory map decoder. The address space for the ApC is divided into a few regions: global control and information registers, p-bit readout array, and individual space for the ApC.

The global address space provides information on the ApC pertinent for client interaction with the system. This includes information such as the total number of p-bits in the design, the p-bit network topology, weight precision, and other useful run-time information such as the number of elapsed synapse delays between client sampling requests. Additionally, global control functions include the ability to pause and resume the network so that a client can access the global p-bit readout array. This readout array holds the current output of all p-bits sampled on each system clock cycle. Reads from this interface are performed when the network is paused to ensure atomic readout of all p-bits given the limited bandwidth of the readout interface.

Each p-bit in the system has a local memory space supporting programmable control of its function. Each p-bit has localized programmable weights enabled through registers or internal memory (RAM), a programmable activation function lookup table supporting direct look-up or interpolation, support for seeding the chosen pseudo random number generation (PRNG) function, and finally a set of control registers for the p-bit. The output of the p-bit programmable elements directly interface with the weight logic, activation function, PRNG, and comparator operation of the weighted p-bits. Online annealing is accomplished using a global bus broadcast when programming the activation function lookup tables, so that all p-bits are updated simultaneously. Alternatively, each p-bit’s activation lookup table can be independently controlled, allowing the exploration of non-uniform bias, local temperature effects, and other non-idealities, facilitating future opportunities for exploration.

B. Digital Building Blocks

A modular digital p-bit was designed to support different options for the p-bit building block elements. Each p-bit is logically partitioned into a unit for pseudo-randomness or “entropy”, a block for computing the activation function, and a portion that uses the results of a comparison between the activation function and PRNG to determine if a p-bit update attempt should occur. There are various ways to construct these elements using digital logic. In this design, a few select implementations were explored as shown in Fig. 6. A non-exhaustive exploration of design options described below was performed as part of initial trade-study. As quantitative results were not pursued, a qualitative assessment of the various options is provided for completeness.

Fig 6(a) and 6(b) show two methods for performing autonomous updates of each p-bit. Shown in Fig. 6(a) is the logic corresponding to Eq. 5a where the comparator provides

$$\text{sgn}(e^{-s} - r_{[0,1]})$$

This approach emulates autonomy while preserving fully synchronous operation of the digital design. This p-bit update logic was used for the problems in this work. The activation look-up is programmed with $e^{-s}$.

A single clock domain within the FPGA is used to synchronize the digital elements of each synapse and p-bit. For each global clock period, $t_{ck}$, the synapse logic requires $t_s = N_{syn} t_{ck}$ time to compute where $N_{syn}$ is the number of global clock events required. In the nearest-neighbor models, it is possible for $N_{syn}$ to need only a single cycle, however, as the weight precision and number of neighbors is increased, $N_{syn}$ also increases. Thus, when specifying an $s$ ratio, coupled with a design driven $N_{syn}$, the neuron time, encoded probabilistically in the look-up table, is

$$\tau_N = s^{-1} N_{syn} t_{ck}$$

Alternative approaches were explored and implemented for the p-bit updates including the use of free-running ring
Fig. 5: **Logical Organization and Memory Map:** The FPGA based ApC coprocessor is accessible to a host processor from a dedicated PCIe endpoint within the FPGA. This endpoint provides logical access to the internal memory map of the ApC. The ApC contains global information and control registers, a global p-bit array facilitating readout, and finally individual control of each weighted p-bit through a localized, dedicated address space.

oscillators, Fig. 6(b), to mimic the naturally stochastic update frequency of an MTJ based p-bit as described by Eq. (3). In this implementation, each p-bit has a dedicated free running ring oscillator that generates asynchronous “attempt” edges that are synchronized into a system clock domain. Each asynchronous edge determines when the p-bit should attempt to update based on the current output from a comparator according to Eq. (3), in which case the activation look-up is programmed with tanh. While the use of oscillators provides some degree of true randomness for when flip attempts occur, this benefit was marginal and did not outweigh the design complexity and overhead (e.g. area and power) introduced with their use. As a result, the design of Fig. 6(a) was selected for the attempt logic.

While the attempt logic is used to determine when an update attempt should be made, the logic relies on the output of a comparator to determine if a flip should occur. The comparator computes the sign of the difference between the output of a PRNG and the output of the programmed activation function. Shown in the second row of Fig. 6 are two PRNG implementations. A Linear Feedback Shift Register (LFSR) is a PRNG that provides pseudo randomness in a compact design at the expense of output quality, Fig. 6(c). While the LFSR may be sufficient for many problems, a higher-quality PRNG was implemented that requires minimal FPGA resources, but significantly improves the PRNG quality [60].

The second input to the comparator is from an activation function output, shown on the third row of Fig. 6(e). As implemented, a straightforward look-up table was sufficient for the designs in this paper. However, an improved interpolation based activation function [61] would improve the accuracy of the lookup results and may be necessary for certain problem classes.

An additional building block was created to leverage physical randomness and a built-in sigmoidal response from within a digital design as shown in Fig. 6(g). Leveraging a delay based building block [62] and flip-flop metastability, “true” entropy was used to construct a sigmoidal response as depicted in Fig. 6(f). However, over continued operation within the device, temperature and other variations caused the sigmoidal curves to drift, resulting in non-uniform bias and operation. As a result, this building block is not currently being used in the design; however, it does provide insight into non-idealities that may be encountered in a chip design.

Finally, we note that an ApC that emulates the physics of different hardware primitives including memristive [63] stochastic neurons could be implemented in an autonomous circuit, unlike the nanomagnet inspired equations Eq. 5a- Eq. 5b. Additionally, as discussed earlier, we do not explore directed networks in this work, however the overall FPGA architecture was designed to support the exploration of different hardware models and network topologies, hence they can be included here with minimal effort in the future.

**IV. APPLICATIONS AND VALIDATION**

In this section we explore two applications of the ApC using the FPGA framework: combinatorial optimization and quantum emulation.

**A. Combinatorial Optimization**

A common architecture used to solve combinatorial optimization problems is the nearest-neighbor Ising model as shown in Fig. 7(a). In the Ising model, each p-bit is connected
to its neighbor through a coupling matrix, $J_{ij}$, and is influenced by an on-site bias $h_i$. Note that this is trivially mapped into $W_{ij}$ as in Eq. (1). By mapping a problem of interest to this system, an Ising computer will intrinsically search for the lowest energy solution to the problem.

Typical implementations of these systems leverage some form of simulated annealing in hardware to guide the system into a low energy solution, using careful control and sequencing of spin-flip updates to avoid non-ideal spin updates [6], [64], [65]. In the context of a nearest-neighbor topology, the network can be split into two groups in a checkerboard-like pattern such that all spins in a given group can be updated in parallel [66]. This results in the ability to update half of all spins within a given clock period. According to Eq. (3a) this results in

$$f = \frac{N}{2\tau_{\text{clock}}}$$

An ApC can be used to solve the same class of problems, but to do so a value of $s$ must be found that produces a solution with the desired fidelity. By identifying this limit for $s$, an upper bound is placed on the synapse speed that must be obtained to achieve a competitive $f$ from Eq. (4).

Shown in Fig. [7] is a Max-Cut problem for which a black and white image was used to encode magnetic domains for the p-bits. The network is initialized to run at a high effective-temperature (low $\beta$) resulting in an effectively uncoupled network. The temperature is then gradually reduced according to an annealing scheduling until the network crystallizes at a low energy, in this case ideal, solution as shown in Fig. 7(b). Shown in Figs. 7(c), 7(d), 7(e) different values of $s$ are used to convey how simultaneous updating affects the ability of the network to converge to the solution. As $s$ is decreased from 1 to 1/2 and 1/4, the smaller values of $s$ result in a more effective convergence. While these results are heuristic in nature given their visual display, a quantitative energy based analysis conveys the same relationship as discussed in the following section with a demonstration of simulated quantum annealing.

For this problem, a value of $s = 1/4$ resulted in effective convergence to the ideal solution and well-behaved network operation. Given this result, the ApC has

$$f = \frac{N}{4\tau_s}$$

Comparing (12) and (13), as long as the synapse is twice as fast as the best $\tau_{\text{clock}}$ that could be obtained from a synchronous implementation, the network will perform the same number of flips per second without needing a sequencer.
Ref. [67], it has been recognized that thermodynamic features have long attracted interest. After its introduction by B. Quantum Emulation at higher temperatures, the network is more effective at finding low energy solutions, even though there is no convergence to the ideal solution. As the temperature is lowered, the features begin to emerge as lower temperatures are reached.

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

To each character, the problem is then solved leveraging an FPGA ApC co-processor (see Methods). The problem is then solved leveraging an FPGA ApC co-processor (see Methods). The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leveraging an FPGA ApC co-processor (see Methods).

The probability of an individual p-bit flipping is controlled through a reduction of the energy state solution to the problem of interest. The network converges to a low energy state, as shown in the images (inset). During this process, the network converges to a low energy state. The problem is then solved leverag...
average magnetization, $\langle m_z \rangle$. We include a symmetry breaking field in the $+z$ direction ($\Gamma_x = +1$) to obtain a net magnetization at vanishing transverse fields ($\Gamma_x = 0$). We obtain the exact density matrix by directly diagonalizing the quantum Hamiltonian (Eq. (15)): $\rho = \exp(-\beta H_Q)$, where we chose an inverse temperature of $\beta = 20$. Once $\rho$ is known, we compute the average magnetization by tracing it with the magnetization operator $S^z$, $\langle m \rangle = \text{tr} [S^z \rho] / \text{tr} [\rho]$. The exact solution is shown as a black solid line in Fig. 8(a).

The corresponding average magnetization is obtained by running the classical system that is described by Eq. (15) with $n = 250$ replicas ($250 \times 8 = 2000$ spins) using Eq. (13) in the FPGA emulator. Fig. 8(a) shows different $s$ values that are used to obtain the exact result. Clearly, choosing $s$ too high ($s = 1/12$) fails, but gradually decreasing $s$ allows the result to approach the exact result. We observe that $s = 1/12$ seems to be an optimal choice as decreasing $s$ further does not yield more improvement due to the chosen precision in the digital implementation. Specifically, as $s$ continues to reduce, the chosen precision of the arithmetic logic and look-up tables in the FPGA emulator design limits the accuracy of the solution.

Fig. 8: Emulating the Transverse Ising Hamiltonian: a. A 1D ferromagnetic linear chain ($J_{ij} = +2$) with $M = 8$ spins described by the transverse Ising Hamiltonian is emulated. The average $z$ magnetization is shown as a function of the transverse magnetic field $\Gamma_x$ at an inverse temperature of $\beta = 20$ using a network of $8 \times 250$ p-bits with periodic boundary conditions (shown as b). The 250 p-bits serve as replicas generated using a Suzuki-Trotter decomposition. The exact quantum Boltzmann solution is compared against the replicas generated using a Suzuki-Trotter decomposition. The average $z$ magnetization at vanishing transverse fields ($\Gamma_x \to 0$, $\langle m \rangle \to 1$), but it also followed the correct average magnetization at high magnetic fields. As such, this could be viewed as an example of sampling a probability distribution rather than finding the ground state of the system [21].

Typically, SQA algorithms initialize the system at a high magnetic field ($\Gamma_x$) and slowly remove it to keep the system in its ground state and guide it to the desired ground state of the Ising Hamiltonian. In Fig. 8, however, we have not changed the magnetic field as a function of time, but rather sampled from 200 ensembles, each separated by $\sim 30,000$ synapse delays, to obtain the system statistics. This means that not only was the system guided to the expected ground state ($\Gamma_x \to 0$, $\langle m \rangle \to 1$), but it also followed the correct average magnetization at high magnetic fields. As such, this could be viewed as an example of sampling a probability distribution rather than finding the ground state of the system [21].

The replica discretization for the present implementation of SQA incurs errors of order $O(\beta^3/r^2)$ where $\beta$ is the inverse pseudo-temperature and $r$ is the number of replicas [70]. The explicit form of the error provides a guide to reduce errors at a fixed $\beta$ at the expense of adding more p-bits. As an example, note the increasing error as $\Gamma_x/J$ in Fig. 8(c) where increasing $\Gamma_x$ compared to a fixed $J$ is like increasing $\beta$. In both examples (Fig. 8-9) we discuss, we have made comparisons to exact calculations that did not require a careful optimization of the number of replicas since we were guided by the errors with respect to exact calculations.

As a second example, we illustrate the trade-off between the number of replicas and the size of the quantum system when limited to a fixed number of p-bits, for example in an FPGA with finite resources. The emulation in Fig. 8 modeled a 1D Transverse Field Ising Model (TFIM) with $M = 8$ spins and with 250 replicas (with a total of 2000 p-bits) to obtain an accurate result at a low (dimensionless) temperature of $\beta = 20$. In Fig. 9, we show another example with a much larger system of $M = 250$ spins but using only 10 replicas per quantum spin for a total of 2500 p-bits.

In this example, instead of plotting average magnetization, we plot the average correlation (i.e. $\langle m_i^2 m_0^2 \rangle$) between lattice points at a fixed inverse temperature ($\beta = 0.744$) and...
that are due to simultaneous parallel updating in Boltzmann degree of parallel updates, reminiscent of similar oscillations when the parameter exceeds this value, we observe systematic oscillations without the need for a controlling sequencer. But in order to make use of ultrafast flip rates, \( f \), enabled by short \( \tau_s \) times, it is important that the building blocks be energy efficient to ensure that power levels are acceptable:

\[
P = f \varepsilon
\]

where \( P \) is the power budget and \( \varepsilon \) is the energy required per flip.

In Table 1, we compare representative Ising computers based on sequenced designs [6], [64] to the autonomous designs implemented in and projected by this work, focusing on nearest-neighbor implementations for this discussion. The last row of the table shows \( \varepsilon \) for both the sequenced and autonomous designs. As shown in the table, the FPGA based 8K-spin ApC achieves an energy per flip that is similar to the Janus II sequenced FPGA implementation. However, this comparison applies some artificial constraints on the Janus II design: namely that all spins must be simultaneously resident in the device at one time. This is a requirement for unclocked autonomous designs that for the sake of comparison we applied to the sequenced design. A clocked, sequenced design can pause the system and leverage external memory for storage of neuron state, providing a large pool of logical neurons, though limited by available memory bandwidth. Additionally, the ability to pause the network enables time-multiplexing and can harness the ability to re-use logic resources.

The FPGA results naturally consume more power and have reduced density [81]. The 8K spin result of Table 1 has \( \sim 18 \) W of static power dissipation due to the periphery included in the FPGA design, not all of which is used. Using the FPGA to ASIC power ratio of 14 [81], a naive migration of the design to an ASIC would result in an \( \varepsilon \) of \( \sim 4.0 \times 10^{-3} \). By translating these approaches to a tailored ASIC implementation, the energy efficiency per flip increases and the ability to increase the density of resident neurons within a device increases substantially. Extending this further, it should be feasible to obtain designs with \( \sim 10 \) petaflips per second with a power budget of \( \sim 10 \) W, but we need devices with \( \varepsilon \) \( \sim 1 \) fJ that also support a density of 1M devices.

The CMOS based SRAM design of Hitachi [6] has an estimated energy per flip of \( \varepsilon \sim 50 \) fJ, though the neuron density limits the ability of the approach to obtain an \( f \) of petaflips per second. Using a hybrid CMOS and MTJ design as would be encountered in modern commercial MRAMs [78], it should be feasible to obtain \( \sim 10 \) petaflips per second projected in the last column of Table 1[27]. Modern MRAMs can achieve Gb densities; however, we limit the projection to a 1 Mb density based on a target power consumption of \( \sim 20 \) W for the neurons and synapses in the design.

It should be noted that the 20 W power target was arbitrarily chosen. In principle the autonomous designs can leverage clocking to choose when global p-bit updates should occur, much like the FPGA emulator discussed in the methods section. While this approach can save power, it limits the utility of the MTJ based approach by constraining the flips per second \( f \) to the same limits of Eq. (3a) due to the clocking scheme. Even with this limit in place, as the connectivity between neurons increases beyond nearest-neighbor, the sequencing logic becomes more complex while the ApC only requires a balance

V. Discussion

These example applications help demonstrate the feasibility of an ApC governed by Eq. (3) to follow Boltzmann statistics without the need for a controlling sequencer. But in order to

![Graph](image-url)

**Fig. 9:** Transverse Ising system with \( M=250 \) q-bits: a. A 1D ferromagnetic linear chain \((J_z=+1)\) with \( M = 250 \) spins is modeled using a network of \( 10 \times 250 \) p-bits with periodic boundary conditions at an inverse temperature of \( \beta = 0.744 \) with \( \Gamma_x = 0.5 \). The measured quantity is the average correlation measured between the first lattice point and different lattice distances \((L)\), which decay to zero as \( L \) increases. Results are shown at different \( s \) values (inset). For each lattice distance and \( s, 10^5 \) samples from a free-running FPGA implementation were collected spaced \( \approx 640,000 \) synapse delays apart. Samples are further averaged over replicas to obtain the mean correlation for the quantum system.
of $s$ to ensure proper convergence, though both approaches still face challenges with routing congestion as $N$ grows. In the case of all-to-all connectivity, the sequencing logic reduces in complexity and technically can be implemented with a single time-multiplexed weighted p-bit. In this situation, the benefits of an ApC begin to degrade, except for the elimination of memory bottlenecks with the use of distributed weights and the avoidance of time-multiplexing. A 500 node all-to-all network was implemented using the FPGA emulator, and the resulting $s$ was directly proportional to the number of neurons, affirming the reduced benefit.

VI. CONCLUSION AND FUTURE WORK

In this work we presented a vision for an autonomous probabilistic computer for applications in optimization and machine learning. Using stochastic nanomagnets as intrinsic p-bits, an ApC based on these devices provides an opportunity to realize a scalable co-processor achieving high-performance operation. As the technology is not yet available to build a scaled device, we presented a benchmarked behavior model for the autonomous computer that facilitated direct study of the ApC. This behavioral model was implemented in an FPGA to establish a framework for the study of various applications and design trade-offs. The results presented in Table I demonstrate that with the removal of sequencers, the ability to run at speeds limited only by synapse delays, and the ability scale to millions of neurons, all within an accessible power budget, the proposed ApC is a compelling alternative to clocked, sequential designs for stochastic ANN coprocessing.

We have also emphasized a key metric, flips per second, as a figure-of-merit for emerging probabilistic annealers that quantify their performance in terms of a problem or substrate independent manner. Improving flips per second by application specific, massively parallel or autonomous architectures using nanodevices as we have suggested could lead to efficient domain-specific, probabilistic coprocessors that can outperform conventional implementations in the future.

TABLE I: Comparison of Sequential and Asynchronous Ising Computers: Ising computers proposed to-date have used sequential updating mechanisms based on CMOS technology. These designs achieve spin update times of $\sim 1 - 4$ ps/flip. The 20K SRAM chip from [6] achieves an energy of 50 fJ per flip. The autonomous CMOS implementations demonstrated in this work obtain a similar spin update time as sequenced implementations and comparative energy per flip as sequential FPGA implementations. Using CMOS and MTJ technologies as proposed herein, a highly efficient design with petaflips per second with 2 J per flip is projected.

| Sequence | Autonomous (This Work) |
|----------|-------------------------|
| Technology | CMOS (SRAM) | CMOS (FPGA) | CMOS (FPGA) | CMOS + MTJ |
| Total Power (W) | 0.05 | 25 | 55 | 32 | 19.25 |
| Number of Neurons (N) | 20K (80 x 256) | 2K | 5K (8 x 250) | 8.1K (90 x 90) | 1M |
| Simulated Delay ($\tau_s$) (ps) | $s = \tau_s/\tau_N$ | 1/12 | 1/12 | 1/10 |
| Neuron Delay ($\tau_N$) (ps) | 10,000 | 4,000 | 8,000 | 8,000 | 10 |
| Flips per Second (f) | $1 \times 10^{12}$ | $2.5 \times 10^{11}$ | $2.08 \times 10^{10}$ | $2.5 \times 10^{11}$ | $1 \times 10^{16}$ |
| Spin Update Time (1/f) (ps/flip) | 1 | 4 | 48 | 4 | 0.0001 |
| Energy per Flip (e) (nJ/flip) | $5 \times 10^{-5}$ | 0.1 | 2.64 | 0.13 | 1.33 x $10^{-6}$ |

a Calculations based on information extracted from [6]. Design used sequential implementation, therefore we assume ($\dagger$) $s = 1$ and an optimal updating scheme such that half of all neurons are updated every step such that $f = (0.5)N/\tau_s$. Synapse and neuron delays assumed to operate at interaction frequency of 100 MHz.

b Calculations based on information extracted from [64]. Janus II supports a much larger number of neurons using external memory, multiple FPGAs, and data shuffling between devices. Here we limit the comparison to only a single spin processor assuming all spins are co-located on one FPGA, as would be required for an autonomous design. As with (a), we assume ($\dagger$) $s = 1$. With all spins on one chip, an optimal updating scheme updates half of all neurons every step for $f = (0.5)N/\tau_s$. Based on this, the spin update time calculated here is 4 instead of 2 as in [64]. We note that this discrepancy is negligible when compared to the projected autonomous coprocessor that can be orders of magnitude faster. Design assumed to operate at operating frequency of 250 MHz.

c Power consumption measured from maximum power draw during computation. 2K QA topology based on 16-bit precision operations and 8K Ising is based on 2-bit precision operations. Selected $s$ values based on Figs. 7 and 8.

d Assuming chip density of 1M based on memory density available from commercial ST-MRAM [78]. With nearest-neighbor connections, a synapse delay of 10 ps assumed [48]. Assumed neuron delay of $\sim 100$ ps and a steady-state neuron power consumption 10 pW [27]. Overhead from nearest-neighbor memresistive cross-bar and external communication logic estimated as $\sim 0.25$ W using 5 memresitors and 1 op-amp per neuron [79]. Similar order of magnitude values can be obtained from [80].

REFERENCES

[1] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, “A Survey of Neuromorphic Computing and Neural Networks in Hardware,” arXiv:1705.06963 [cs], May 2017, arXiv: 1705.06963.

[2] G. Detorakis, S. Dutta, A. Khanna, M. Jerry, S. Datta, and E. Neftci, “Inherent weight normalization in stochastic neural networks,” in Advances in Neural Information Processing Systems, 2019, pp. 3286–3297.

[3] L. Buesing, J. Bill, B. Nessler, and W. Maass, “Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons,” PLoS computational biology, vol. 7, no. 11, p. e1002211, 2011.

[4] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1,” arXiv preprint arXiv:1602.02830, 2016.

[5] S. Boixo, T. F. Ronnow, S. V. Isakov, Z. Wang, D. Wecker, D. A. Lidar, J. M. Martinis, and M. Troyer, “Evidence for quantum annealing with more than one hundred qubits,” Nature Physics, vol. 10, no. 3, pp. 218–224, Mar. 2014.
