Physics-inspired Ising Computing with Ring Oscillator Activated p-bits

‡ Navid Anjum Aadit*, ‡ Andrea Grimaldi†, Giovanni Finocchio†,○
and Kerem Y. Camsari*,§
†Department of Mathematical and Computer Sciences, Physical Sciences and Earth Sciences,
University of Messina, Messina, Italy
*Department of Electrical and Computer Engineering, University of California, Santa Barbara,
Santa Barbara, CA, 93106, USA
‡ ***Equally contributing authors***

***Abstract*—The nearing end of Moore's Law has been driving the development of domain-specific hardware tailored to solve a special set of problems. Along these lines, probabilistic computing with inherently stochastic building blocks (p-bits) have shown significant promise, particularly in the context of hard optimization and statistical sampling problems. p-bits have been proposed and demonstrated in different hardware substrates ranging from small-scale stochastic magnetic tunnel junctions (sMTJs) in asynchronous architectures to large-scale CMOS in synchronous architectures. Here, we design and implement a truly asynchronous and medium-scale p-computer (with $\approx$ 512 p-bits) that closely emulates the asynchronous dynamics of sMTJs in Field Programmable Gate Arrays (FPGAs). Using hard instances of the planted Ising glass problem on the Chimera lattice, we evaluate the performance of the asynchronous architecture against an ideal, synchronous design that performs parallelized (chromatic) exact Gibbs sampling. We find that despite the lack of any careful synchronization, the asynchronous design achieves parallelism with comparable algorithmic scaling in the ideal, carefully tuned and parallelized synchronous design. Our results highlight the promise of massively scaled p-computers with millions of free-running p-bits made out of nanoscale building blocks such as stochastic magnetic tunnel junctions.**

***Index Terms*—p-bits, combinatorial optimization, planted Ising, Chimera lattice, asynchronous computing, massive parallelism, magnetic tunnel junctions**

I. Introduction

With the nearing end of Moore's Law, domain-specific hardware and architectures are growing rapidly. The notion of performing *some tasks* more efficiently (area, speed and/or energy) rather than improving performance for *general purpose* computing has led to the proliferation of special-purpose accelerators. With their widespread use, hard optimization problems have been a primary target of this approach and a variety of different domain-specific hardware architectures have emerged (see, Ref. [1] for a general and recent review).

As an example of this growing trend, probabilistic bits or p-bits were introduced [2] as a building block which can accelerate a broad family of algorithms including Monte Carlo, Markov Chain Monte Carlo [3], Quantum Monte Carlo, statistical sampling for Bayesian inference and Boltzmann machine learning [4] methods. p-bits have been shown to be compatible with powerful optimization techniques such as parallel tempering [5] with competitive performance relative to all other Ising machines (classical and quantum) in select problems such as integer factorization and Boolean satisfiability [6]. Their combination with sophisticated algorithms [7] could yield further advantages.

A natural advantage of the p-bit model is its native mapping to the Ising Model and to the natural generalization of Ising Models. This ensures that coupled p-bits can systematically probe the exact Boltzmann distribution through Gibbs or Metropolis sampling without any approximations or reductions, often necessary in alternative, non-bistable abstractions of the Ising spin.

One particularly promising small-scale demonstration of p-bits in an *asynchronously* operating mode was performed in Ref. [8]. Combined with key breakthrough experiments demonstrating nanosecond fluctuations in suitably designed low barrier magnetic tunnel junctions (MTJ) [9], [10], these results suggest the intriguing possibility of designing $>$ million bit probabilistic computers [11] in light of the remarkable advances in the magnetic memory chip industry reaching gigabit densities [12], [13]. Even though large scale p-bit emulators have been designed and tested in FPGAs or ASICs, [3], [6], [11], [14], [15], virtually all of these implementations have been on *synchronous* hardware where a global clock controlled the information flow.

In this paper, we make a first attempt in designing and building a physics-inspired, truly *asynchronous* architecture, which more closely emulates the dynamics of interacting nanodevice-based p-bits, analogously to interacting bodies (FIG. 1, upper panel). We achieve this by an unconventional use of FPGAs where individual p-bits are activated by decoupled ring oscillators and can have overlapping and out-of-phase clocks with different frequencies. Considering how variations may influence individual p-bit behavior in magnetic tunnel junction based designs [16] the behavior of asynchronous p-computers with built-in variations is worth investigating.

To compare the performance of the truly asynchronous p-computer in the FPGA, we choose the planted Ising model where a hard optimization problem is generated with a planted solution [17], [18], allowing a reliable evaluation of the asynchronous design with respect to exact Gibbs sampling.

§Corresponding authors: ○gfinocchio@unime.it, §camsari@ece.ucsb.edu

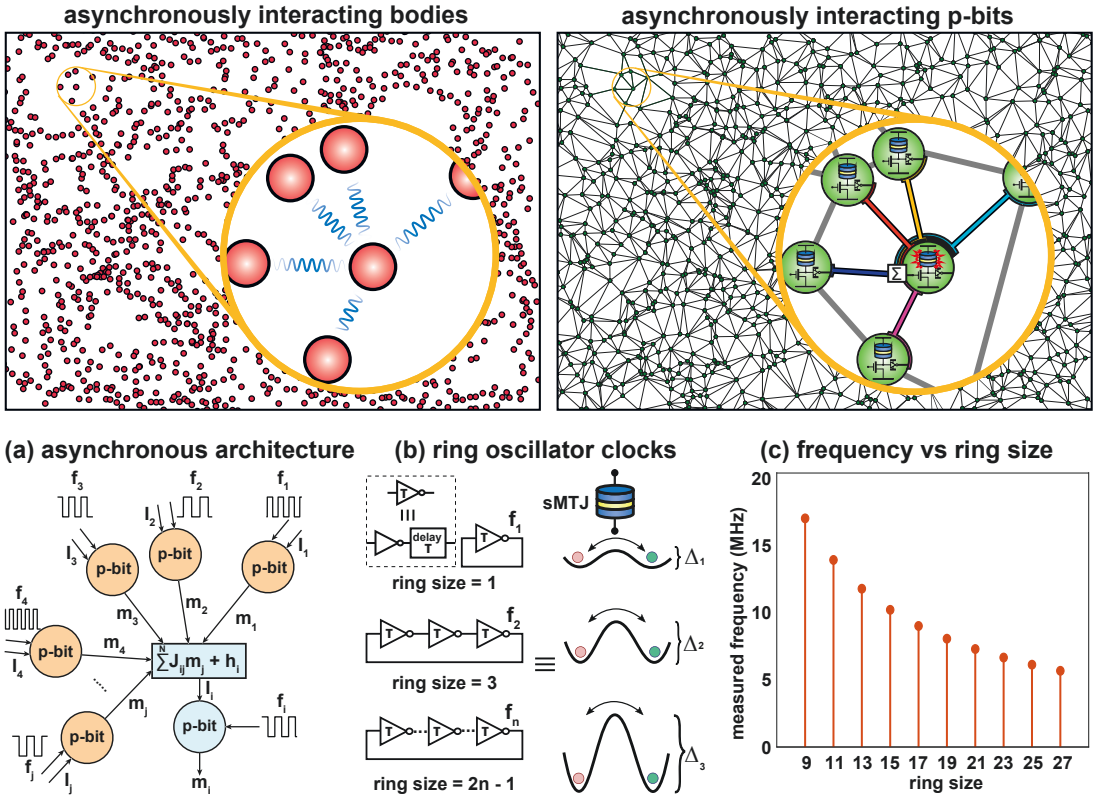

FIG. 1. Upper panel: Physics-inspired analogy between asynchronously interacting bodies and p-bits: both systems are asynchronous, local (sparse connectivity) and massively parallel. (a) Asynchronous computer architecture: The local field (Eq. 1) for each p-bit is computed combinationally. Each p-bit has a different clock with asynchronous activation (b) Asynchronous ring oscillator-based clocks (ROSC) where we draw an analogy between stochastic Magnetic Tunnel Junction (sMTJ)-based p-bits and ROSCs. Large rings correspond to large energy-barrier sMTJs with higher retention times. (c) Measured frequency range for ROSCs with different ring sizes in the FPGA implementation.

II. Physics-inspired Architecture

The main equations of the p-bit model (FIG. 1a) involves stochastic activation and a local field (synapse) calculation, given by:

$$m_i = \text{sgn}(\tanh(\beta I_i) - r_U) \quad I_i = \sum J_{ij} m_j + h_i \qquad (1)$$

where m_i represents the bipolar p-bit state (± 1), r_U is a uniform random number between $(-1, +1)$ and $[J], \{h\}$ are the weights and biases for a given problem and β is the inverse temperature.

Standard Gibbs sampling iterates Eq. 1 to reach the Boltzmann distribution defined by the weights and typically involves a *serialized* update procedure with nested `for` loops. One way to avoid this serial `for` loop is to perform block updates between unconnected p-bits. This approach when applied in software is called "chromatic sampling" [19] and a low-level hardware realization of it was recently reported in Ref. [6]. However, this design also involves careful (synchronous) equal phase shifting between the blocks so that multiple blocks do not update simultaneously.

In this work, inspired by truly asynchronous small-scale implementations of p-computers with nanodevices (based on stochastic MTJs [4], [8]), we implemented a physics-inspired, truly asynchronous Ising Computer where different p-bits receive clocks with different frequencies with random phases. In contrast to synchronous designs, no careful engineering between the clocks of asynchronous p-bits were made. Moreover, unavoidable variations of sMTJs in highly scaled p-computers with nanodevices would make such an engineering extremely difficult if not impossible. We found that despite the deliberate randomization of p-bit clocks and unavoidable collisions breaking exact Gibbs sampling, the physics-inspired design exhibited massive parallelism observed in carefully tuned synchronous designs, not observed in standard CPU-based Gibbs sampling (FIG. 3).

Ring Oscillator Generation: A ROSC clock consists of an odd number of looped NOT gates. In our FPGA (Xilinx, VCU118), we attach controllable delays to our inverters to make logical delays comparable to wire delays (FIG. 1b). We designed the delay unit as a flip flop with a very fast master clock (300 MHz) compared to the ROSC frequencies that essentially acts as a combinational delay unit. In this way, we were able to obtain highly regular ROSC clocks as a function of ring sizes whose frequencies were measured by specially designed counters (FIG. 1c). In our experiments, we used 10 ROSCs to drive 512 p-bits in a Chimera lattice. Each p-bit has a pseudorandom number generator, which is a 32-bit Linear Feedback Shift Register (LFSR). The ROSCs activate the LFSRs of the p-bits randomly based on the frequencies. In this work, we have distributed the clocks among the p-bits uniformly between 5 and 17 MHz. However, different distributions for the clocks, e.g., Gaussian, could be used.

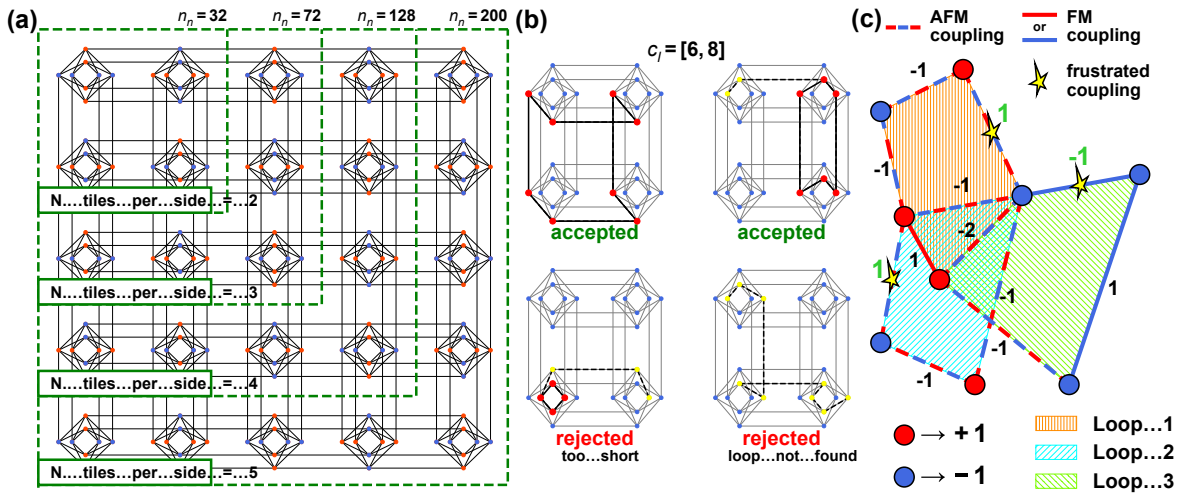

FIG. 2. Planted Ising problems. (a) Chimera lattice with different sizes, depending on number of tiles. Different tiles were chosen in the same hardware to change problem sizes, similar to Ref. [17] but all problems used a fixed 512-spin Chimera in our hardware. (b) Loop/close generation process shown for $c_l = [6, 8]$. (c) Example graph with three loops (the colored hatchings) and a planted solution (the colors of the nodes). For each loop, the weights of the couplings between neighbors are assigned to be either ferromagnetic (FM) interactions or antiferromagnetic (AFM), based on the plant. One randomly selected coupling in each loop is flipped (spark symbol).

Since the Chimera graph is bi-partite, we did not assign the same clock to two p-bits that are on the same partition to avoid systematic parallel updates between connected p-bits. Future work will consider *dynamic* clocking schemes where each p-bit can have a different "retention time" much like MTJ-based p-bits.

III. Planted Ising Model

An important class of hard optimization problems are those with "planted" ground states that allow effective evaluation of performance. We construct frustrated spin glasses with planted solutions, following [17], on a 512-spin Chimera graph where we used different number of tiles for different problem sizes (FIG. 2a). A Hamiltonian generated by this process is the sum of several local frustrated Hamiltonians which we will call "clauses". A planted solution will be used to define these clauses so that it will be the ground energy of the final Hamiltonian. Every instance is characterized by two parameters: the clause density α, defined as n_c/n_n, where n_c is the number of clauses and n_n is the number of nodes of the graph, and the length of possible loops that form the clauses, $c_l = [l_{min}, l_{max}]$, where $l_{min/max}$ is the min/max loop length, respectively. In this work, we chose $\alpha = 0.4$ and $c_l = [4, 8]$ for all our instances used in this paper, that run on the same 512-spin Chimera lattice in our hardware.

Clause generation: A total of $n_c = \alpha n_n$ clauses is generated. Each clause is an ordered sequence of nodes that creates a loop of acceptable length in the graph. To obtain one, following Ref. [17], we pick a random node and start a non-backtracking random walk of at most l_{max} steps. If the walker lands on an already visited node, it means that a loop was formed and the node can be considered its initial point. If the length of the loop is $> l_{min}$ the clause is accepted, if it is not or if the maximum number of steps is reached without closing the loop, the process is repeated. FIG. 2b shows a few examples of this process. A planted solution s is generated by creating a random array of -1s and $+1$s of length n_n. A clause can be defined as $c_m = \{n_1, n_2, ..., n_k, n_{k+1}\}$, with $k \in [l_{min}, l_{max}]$, where $n_{k+1} = n_1$, representing the closing of the loop. Now, $\forall i \in [1, k] : i \neq j$ we increase $J_{n_i,n_{i+1}}$ by $s_i s_{i+1}$, while for $j \in [1, k]$, picked at random, we increase $J_{n_j,n_{j+1}}$ by $-s_j s_{j+1}$. This last step serves to create a frustrated loop. Once this is done for all clauses, the final J is calculated by summing J and J^T and by normalizing so that all J lie between $[-1, +1]$.

IV. Performance Comparison

We follow the time-to-solution formulation [17], [18] to measure performance of the physics-inspired asynchronous architecture.

$$\mathrm{TTS}(\tau, p_R) = \tau N_r(\tau, p_R) \tag{2}$$

where N_r is the expected number of repetitions we need to perform an annealing schedule of time τ to the energy ground state at least once with probability p_R. N_r is defined as:

$$N_r(\tau, p_R) = \frac{\ln(1 - p_R)}{\ln(1 - p_S(\tau))} \tag{3}$$

where p_S is the probability of success in finding the ground state in one annealing process of length τ.

To evaluate the performance of our asynchronous architecture, we compared it to serialized Gibbs sampling on CPU (2.6 GHz) and to synchronous colored Gibbs sampling on FPGA. We investigated the scaling difficulty of planted Ising instances with fixed c_l and α across several Chimera graphs increasing in size by changing the number of tiles used in a Chimera, as illustrated in FIG. 2. For each set of tiles, we generated 25 planted Ising instances, performing 50 simulated annealing trials to estimate the success probability, $p_S(\tau)$, for each instance (FIG. 3). Then an average success probability $p_S(\tau)$ over all instances for a given tile was obtained. The reference probability p_R was set to 99%.

Given our limited number of trials and for simplicity, we discarded the p_S points when they were exactly zero, since

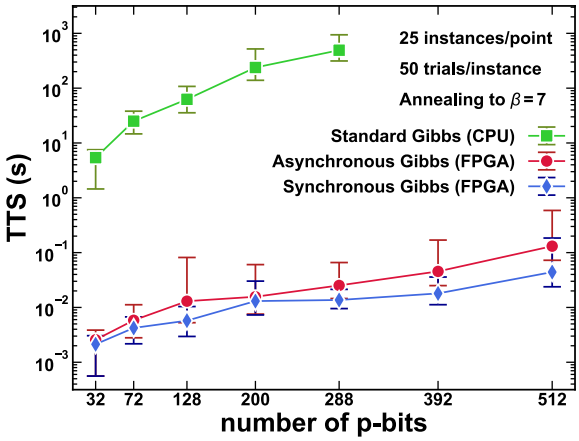

FIG. 3. Performance comparison of synchronous (graph colored-tuned), asynchronous (physics-inspired, not colored, not tuned) and standard Gibbs (CPU) samplers on the planted Ising problem.

these lead to an infinite TTS. For easier instances when averaged $p_S = 1$, we set $\mathrm{TTS} = \tau$ since a single trial reaches the ground state within τ seconds.

We present the TTS for solving 25 instances of the planted Ising problems of 7 different sizes in FIG. 3. The standard Gibbs (CPU) was implemented in Python using optimized libraries for matrix calculations. The final two points were not computed because of time limitations. We solved the exact same instances on the FPGA programmed with the asynchronous ROSC activated 512 p-bits with a fixed point representation using 10-bits. As a reference, we also solved the same instances using *synchronous* chromatic Gibbs sampling on the same FPGA where careful phase shifting ensures no simultaneous or incorrect updates (where I_i calculation is not complete) occur between neighboring p-bits (as in Ref. [6]). On the other hand, the asynchronous solver is expected to take samples with both of those errors when p-bit clocks are closely separated. Our experiment investigates the usefulness of such samples.

We solved each instance using a simple simulated annealing schedule performing 50 trials per instance. We defined a linear annealing schedule from $\beta = 0.5$ to 7 with a fixed annealing time, $\tau = 0.0014$ s for each trial. The asynchronous architecture received 10 clocks ranging from 5 to 17 MHz uniformly distributed among the p-bits, the synchronous architecture was set up with two stable and oppositely phase shifted clocks, having approximately the average frequency (9.375 MHz) of the 10 ROSC clocks. We believe this arrangement made two designs equivalent beyond the *asynchronous and inexact dynamics* of the ROSC since both designs approximately take the same amount of samples within the fixed annealing time τ. The key result we obtained is shown in FIG. 3. We observe a clear scaling difference between the CPU implementation of standard serialized Gibbs sampling and the massively parallel FPGA implementations which gain a scaling factor of $\approx N$ in their flips/second due to their massively parallel architecture. Both solvers provide a roughly 5-orders of magnitude prefactor improvement over the CPU. Intriguingly, the scaling of the synchronous and asynchronous FPGA remain very similar, despite the possibility of many collisions (parallel or incorrect updates) in the asynchronous design. Indeed, the carefully tuned synchronous design performs strictly better than the asynchronous one in all instances. Nevertheless, it is encouraging to observe that the asynchronous design without any carefully engineered clocks or tuning performs nearly as well, leading to the promising possibility of truly asynchronous, million bit p-computers with stochastic MTJs or other nanodevices.

ACKNOWLEDGMENT

K.Y.C. and N.A.A. acknowledge support through National Science Foundation (CCF 2106260) and K.Y.C. through the Samsung GRO program. A.G. and G.F. were supported under the project PRIN 2020LWPKH7 funded by the Italian Ministry of University and Research and by the PETASPIN Association (www.petaspin.com).

[1] N. Mohseni, P. L. McMahon, and T. Byrnes, "Ising machines as hardware solvers of combinatorial optimization problems," *arXiv preprint arXiv:2204.00276*, 2022.
[2] K. Y. Camsari et al., "Stochastic p-bits for invertible logic," *Physical Review X*, vol. 7, no. 3, p. 031014, 2017.
[3] J. Kaiser et al., "Benchmarking a probabilistic coprocessor," *arXiv preprint arXiv:2109.14801*, 2021.
[4] J. Kaiser et al., "Hardware-aware in situ learning based on stochastic magnetic tunnel junctions," *Physical Review Applied*, vol. 17, no. 1, p. 014016, 2022.
[5] A. Grimaldi et al., "Spintronics-compatible approach to solving maximum-satisfiability problems with probabilistic computing, invertible logic, and parallel tempering," *Physical Review Applied*, vol. 17, no. 2, p. 024052, 2022.
[6] N. A. Aadit et al., "Massively parallel probabilistic computing with sparse ising machines," *arXiv preprint arXiv:2110.02481*, 2021.
[7] M. Mohseni et al., "Nonequilibrium monte carlo for unfreezing variables in hard combinatorial optimization," *arXiv preprint arXiv:2111.13628*, 2021.
[8] W. A. Borders et al., "Integer factorization using stochastic magnetic tunnel junctions," *Nature*, 2019.
[9] K. Hayakawa et al., "Nanosecond random telegraph noise in in-plane magnetic tunnel junctions," *Physical Review Letters*, vol. 126, no. 11, p. 117202, 2021.
[10] C. Safranski et al., "Demonstration of nanosecond operation in stochastic magnetic tunnel junctions," *Nano Letters*, vol. 21, no. 5, pp. 2040–2045, 2021.
[11] B. Sutton et al., "Autonomous probabilistic coprocessing with petaflips per second," *IEEE Access*, vol. 8, pp. 157 238–157 252, 2020.
[12] G. Finocchio et al., "The promise of spintronics for unconventional computing," *Journal of Magnetism and Magnetic Materials*, vol. 521, p. 167506, 2021.
[13] S. Bhatti et al., "Spintronics based random access memory: a review," *Materials Today*, vol. 20, no. 9, pp. 530–548, 2017.
[14] A. Z. Pervaiz et al., "Weighted p-bits for fpga implementation of probabilistic circuits," *IEEE transactions on neural networks and learning systems*, 2018.
[15] S. C. Smithson et al., "Efficient cmos invertible logic using stochastic computing," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 6, pp. 2263–2274, 2019.
[16] R. Rahman and S. Bandyopadhyay, "Variability of binary stochastic neurons employing low energy barrier nanomagnets with in-plane anisotropy," *arXiv preprint arXiv:2108.04319*, 2021.
[17] I. Hen et al., "Probing for quantum speedup in spin-glass problems with planted solutions," *Physical Review A*, vol. 92, no. 4, p. 042325, 2015.
[18] T. Albash and D. A. Lidar, "Demonstration of a scaling advantage for a quantum annealer over simulated annealing," *Physical Review X*, vol. 8, no. 3, p. 031016, 2018.
[19] J. Gonzalez et al., "Parallel gibbs sampling: From colored fields to thin junction trees," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. JMLR Workshop and Conference Proceedings, 2011, pp. 324–332.