Massively Parallel Probabilistic Computing with Sparse Ising Machines

Navid Anjum Aadit,[1, *] Andrea Grimaldi,[2] Mario Carpentieri,[3] Luke Theogarajan,[1]
John M. Martinis,[4, 5, 6] Giovanni Finocchio,[2, †] and Kerem Y. Camsari[1, ‡]

[1]*Department of Electrical and Computer Engineering, University of California, Santa Barbara, Santa Barbara, CA, 93106, USA*
[2]*Department of Mathematical and Computer Sciences, Physical Sciences and Earth Sciences, University of Messina, Messina, Italy*
[3]*Department of Electrical and Information Engineering, Politecnico di Bari, Bari, Italy*
[4]*Department of Physics, University of California, Santa Barbara, Santa Barbara, CA, 93106, USA*
[5]*Quantala, Santa Barbara, CA, 93105, CA, USA*
[6]*Zyphra Technologies Inc., San Francisco, CA, USA*

(Dated: February 22, 2022)

Inspired by the developments in quantum computing, building domain-specific classical hardware to solve computationally hard problems has received increasing attention. Here, by introducing systematic sparsification techniques, we demonstrate a massively parallel architecture: the sparse Ising Machine (sIM). Exploiting sparsity, sIM achieves ideal parallelism: its key figure of merit − flips per second − scales linearly with the number of probabilistic bits (p-bit) in the system. This makes sIM up to 6 orders of magnitude faster than a CPU implementing standard Gibbs sampling. Compared to optimized implementations in TPUs and GPUs, sIM delivers 5-18x speedup in sampling. In benchmark problems such as integer factorization, sIM can reliably factor semiprimes up to 32-bits, far larger than previous attempts from D-Wave and other probabilistic solvers. Strikingly, sIM beats competition-winning SAT solvers (by 4-700x in runtime to reach 95% accuracy) in solving 3SAT problems. Even when sampling is made inexact using faster clocks, sIM can find the correct ground state with further speedup. The problem encoding and sparsification techniques we introduce can be applied to other Ising Machines (classical and quantum) and the architecture we present can be used for scaling the demonstrated 5,000−10,000 p-bits to 1,000,000 or more through analog CMOS or nanodevices.

I. INTRODUCTION

Markov Chain Monte Carlo (MCMC) algorithms have made a significant impact in the history of computing [1]. MCMC methods are among the most powerful randomized algorithms with a wide range of applications in Artificial Intelligence (AI) [2]. Powerful MCMC methods such as Metropolis and Gibbs sampling have been widely applied to training generative neural networks [3], probabilistic inference in belief networks [4], calculating physical observables in classical and quantum systems [5, 6] and solving computationally hard combinatorial optimization problems [7].

Designing domain-specific hardware to accelerate such computationally hard problems of AI has been receiving increasing attention with the slowing pace of Moore's Law. There have been a number of approaches to build special-purpose hardware to solve computationally hard problems. A class of such solvers (also known as Ising Machines) specifically solve quadratic energy models or the Ising model, typically mapped to problems in NP [8–25],

$$E = -\left(\sum_{i<j} J_{ij} m_i m_j + \sum h_i m_i\right) \tag{1}$$

($m_i \in \pm 1$, J_{ij}=J_{ji}, $h_i \in \mathbb{R}$), where quadratic terms ($m_i m_j$) in the energy translate to a linear "synapse" or an interconnection matrix which can be expressed as a graph ($I_i = -\partial E/\partial m_i$):

$$I_i = \sum J_{ij} m_j + h_i \tag{2}$$

Choosing the activation function of individual probabilistic bits as

$$m_i = \mathrm{sgn}\left[\tanh(\beta I_i) - \mathrm{rand}_U(-1,1)\right] \tag{3}$$

ensures the system states, $\{m\}$, are visited according to their corresponding Boltzmann probability:

$$p_i \propto \exp\left[-\beta E(\{m\})\right] \tag{4}$$

where β is introduced as an "inverse temperature" and can be used to enhance or suppress probabilities corresponding to energy minima. The dynamical evolution of Eq. (2)-(3) enables probabilistic sampling and inference, learning weights of a stochastic neural network or performing search or optimization in the exponential state space of the model. Such a machine evolving to the Boltzmann distribution defined by Eq. (4) is called a Boltzmann Machine, after the pioneering work of Hinton and colleagues [26, 27].

So far, nearly all dedicated Ising Machines have specifically focused on optimization problems, with the exception of D-Wave's quantum annealers which have been applied to problems beyond combinatorial optimization [28, 29].

Typically in Gibbs sampling (a type of MCMC method), Eq. (2)-(3) are updated iteratively to dynamically evolve the Markov chain such that the network eventually reaches the Boltzmann distribution defined by Eq. (4). A practical

* maadit@ece.ucsb.edu
† giovanni.finocchio@unime.it
‡ camsari@ece.ucsb.edu

difficulty lies in the serial nature of this evolution: connected nodes need to be updated one after the other since parallel updating leads to repeated oscillations in the network state, preventing the network from converging to the Boltzmann distribution. The need for sequential updating inherently serializes the network evolution, signified by the nested for-loops in standard descriptions of Gibbs sampling [30].

II. SUMMARY OF MAIN RESULTS

This work is about overcoming the fundamental difficulty of sequential updating by combining algorithmic and architectural ideas to build a sparse Ising Machine (sIM) for solving combinatorial optimization and probabilistic sampling problems. The present implementation is on a Field Programmable Gate Array (FPGA) however the architecture is general and can have many different implementations ranging from digital CMOS to energy-efficient nanodevices such as Magnetic Tunnel Junctions [31, 32] (see Supplementary Information Section A, B).

The sIM achieves near-ideal parallelism for MCMC sampling as long as the interconnection matrix J is sparse. A key feature of our framework is in its generality: we first show that *any* optimization function can be efficiently represented as a sparse (but irregular) graph through principles of invertible logic [22, 33]. We then outline techniques for further sparsification using additional nodes without any approximation. Next, we develop a massively parallel architecture to implement Eq. (2)-(3), exploiting the sparsity of the graph, where the graph is defined by J. This is achieved by using multiple phase shifted clocks controlling the activation of probabilistic bits (p-bits) (Eq. (3)). The p-bits are interconnected through a multiply-accumulate (MAC) unit (Eq. (2)).

This architecture can be considered to be a low level hardware-level implementation of chromatic Gibbs sampling [34] where large blocks of conditionally independent nodes are updated in parallel. For this sampling to be exact, the MAC must finish its computation before the next color block is updated. An unexpected finding however is even when color blocks are updated before the MAC operation is completed, the network is often able to find exact ground states in model optimization problems. This inexact Gibbs sampling approach is reminiscent of the Hogwild!-Gibbs algorithm [35] and we show how this overclocking strategy can lead to further advantages. We provide error models and an analysis of inexact Gibbs sampling with analytical estimates of limiting behaviors.

The idea of block updating is commonly used for regular graphs. For example, as first noted in Ref. [36], when the graph is bipartite (as in Restricted Boltzmann Machines or chessboard lattices), trivial colorings (with two colors, black and white or four colors in King's graphs) are possible and this is often exploited in updating each color block in parallel [4, 19, 37–41]. We also note that parallelization techniques in multiple FPGAs have recently been explored for other types of Ising Machines [24], however these are based on completely different algorithms unrelated to the computational model we use based on Eq. (2)-(4).

Compared to prior works on block updating [4, 19, 37–41], our contributions are twofold: First, we extend the block updating scheme such that it applies to regular *and* irregular graphs with the only requirement that the graphs are sparse enough to be colored by a few colors (typically $\leq$ 4-8). This generalization is significant, considering most practical instances of combinatorial optimization problems have irregular graph representations.

Second, we provide exact sparsification methods which can be applied to sparsify dense graphs. We believe that both of these methods can be useful for other Ising Machines and different problems.

We tested the resultant sIM on model problems and achieved three key results:

- In solving Boolean satisfiability problems, the sIM is able to beat competition-winning SAT solvers (2020, 2017) in run-time by up to 4-700x to reach 95% accuracy.

- In probabilistic sampling, the sIM delivers a measured 5-18x speedup over the optimized TPU and GPU implementations (Table I). Against a standard CPU implementation, we measure up to 6-orders of magnitude speedup.

- In integer factorization, the sIM can reliably find the absolute ground state for semiprimes up to 32-bits, far larger than what has been reported for D-Wave's quantum annealers or similar probabilistic solvers [15, 20, 22, 42–44] (Table II). Robust factorization up to 32-bit numbers seem to be the largest by far among these alternatives.

In this paper, we use integer factorization as a computationally hard optimization problem to compare the performance of sIM with respect to D-Wave and other Ising Machines. Expressing the integer factorization problem as a satisfiability (or spin-glass) instance is not expected to be practically relevant, as studied in detail in Ref. [45]. However, critical subroutines of the best algorithms for factoring may potentially be accelerated using improved satisfiability through dedicated hardware or algorithms [46].

A striking result is to show speedups over recent competition-winning SAT solvers in approximate optimization, since SAT solvers have been optimized and fine-tuned after decades of research and development. In contrast, the sIM is using a standard simulated annealing algorithm without any detailed fine-tuning. Further improvements using more sophisticated algorithms such as Parallel Tempering should increase the performance of the sIM. Moreover, experimental developments in emerging nanodevice technologies [47, 48] such as Magnetic Tunnel Junction based asynchronous probabilistic computers [15] can use the same architectural and algorithmic ideas we develop in this paper to provide additional speedup [25].

The organization of this paper is as follows: Section III shows how any combinatorial optimization problem can be converted to an invertible probabilistic circuit (p-circuit) using the basic building blocks (AND, OR, NOT, Full Adder) with compact interconnection matrices and discrete weights, followed by a discussion on reconfigurability. We illustrate in Section IV how a given sparse p-circuit can be colored with a few colors using an approximate graph coloring algorithm, DSATUR [49]. We elaborate on the details of the hardware architecture implementing the coupled equations and propose further sparsification techniques to overcome the limitations of high fan-out circuits. Next, we report our experimental results on integer factorization and Boolean satisfiability problems in Sections V-VII as well as detailed comparisons of the sIM with respect to CPU, GPU and TPU implementations of MCMC. Finally, in Section VIII we discuss the effects of overclocking p-bits to perform inexact and asynchronous Gibbs sampling for further improvement.

III. COMPOSABILITY AND RECONFIGURABILITY

A. Composing invertible circuits from elementary gates

Invertible logic (discussed in detail in Refs. [33, 50]) allows composing probabilistic generalizations of a universal set of basic gates such as COPY/NOT (2 p-bits), AND (3 p-bits), OR (3 p-bits) and Full Adders (FA, 5 p-bits). The basics of such invertible gates and their operation have been discussed extensively [22, 51, 52]. Here, we show some of the J matrices for a set of elementary probabilistic gates in the Supplementary Information Section C and D, where we also elaborate on how to compose p-circuits corresponding to any given Boolean function. Similar to conventional digital circuits, the use of basic p-logic gates to a hierarchical design of larger p-circuits results in highly sparse representations amenable to the massive parallelism discussed in this paper.

The ability of invertible logic circuits to operate in reverse gives rise to a convenient method of solving inverse problems, as noted in the related paradigm of memcomputing and by D-Wave [42, 53]. Therefore, hard combinatorial optimization problems such as integer factorization and Boolean satisfiability (SAT) can be solved in hardware using invertible multipliers or by invertible Boolean circuits corresponding to a given satisfiability instance. Fig. 1a shows a classic m-bit multiplier circuit composed of multiple AND gates and FAs. In the reverse direction, this circuit works as an n-bit factorizer circuit ($n = 2m$) where we clamp the output bits to the n-bit product to be factored. Similarly, we clamp the output of the SAT solver circuit in Fig. 1b to 1 and find the input variables satisfying all the clauses in the reverse direction. Each clause is represented by multiple OR gates; if any input variable is negated, we modify the corresponding weight (J) matrix of the OR gate accordingly. The output bits of the OR gates can be clamped to 1 either directly or by using an AND gate at the end. In this work, we also focus on the 3SAT, a special form of the satisfiability problem where each clause has exactly three variables in the conjunctive normal form (CNF) [54]. We collected 3SAT instances as .CNF files from the UBC SATLIB library [55] and map them to interconnection matrices (J), composed of elementary probabilistic gates. Further details about instances used in this work are provided in the Methods Section X A.

B. Reconfigurability

The invertible circuits composed out of elementary probabilistic gates are reconfigurable, able to accommodate different instances of a given problem. For example, a 32-bit factorizer can factor any two numbers of up to 32-bits by a suitable clamping of the bias values (h_i) of the output bits. Similarly, invertible Boolean circuits can be designed to accommodate many different instances of the SAT problem to function as general SAT solvers. For example, a 250-variable SAT solver can solve a 50-variable SAT instance by an appropriate clamping and multiplexing of input bits.

The reconfigurability of invertible logic circuits provides an alternative to the usual method of embedding a native graph to a target graph referred to as the minor graph embedding (MGE) problem [56, 57]. As we show in the Supplementary Information Section E 2, typical MGE algorithms often fail to find a mapping for the problems we considered, and even when a mapping is found the number of auxiliary spins is too large [58]. Reconfigurability using invertible logic with sparsification [50, 59] is a much more practical alternative to MGE for the problems we considered. For example, reconfigurable 32-bit factorization graph requires a Chimera target with $\approx$ 10,000 spins, while the sparsification technique introduced in this work requires only $\approx$ 2,000 spins. (see Supplementary Information Section E 2 for details).

We also note that the universal nature of the Boolean satisfiability problem enables another layer of reconfigurability since many combinatorial optimization problems are mapped to satisfiability instances with minimal overhead [60]. For example, the Maximum-Cut problem, a common benchmark for many Ising Machines, can be efficiently mapped to a Boolean satisfiability instance [61].

IV. ARCHITECTURE DESIGN FOR MASSIVE PARALLELIZATION

A. Graph coloring

When a quadratic energy model described by Eq. (1) is chosen, an invertible logic p-circuit can be represented as a graph where each node represents a p-bit and each edge represents the connection between the p-bits. Fig. 1c illustrates the graph of an 8-bit factorizer p-circuit encoded with 52 p-bits.

As discussed in Section II, the coloring is used to exploit the trick that allows the parallel update of unconnected (conditionally independent) p-bits. We first color the graph using a heuristic graph coloring algorithm DSATUR [49]. Coloring a graph with exactly the minimum possible colors is NP-hard [62], however, this is unimportant for our purposes since we use DSATUR as a greedy algorithm which may or may not find the optimum coloring.

We find that when the overall graph density is low (e.g., $\lessapprox$ 1%), irregular graphs containing nodes with hundreds of neighbors can be colored by a few colors in line with theoretical results on coloring sparse graphs [63]. For example, for the 8-bit factorizer graph only five colors are used and as a result, we need five parallel and equally phase-

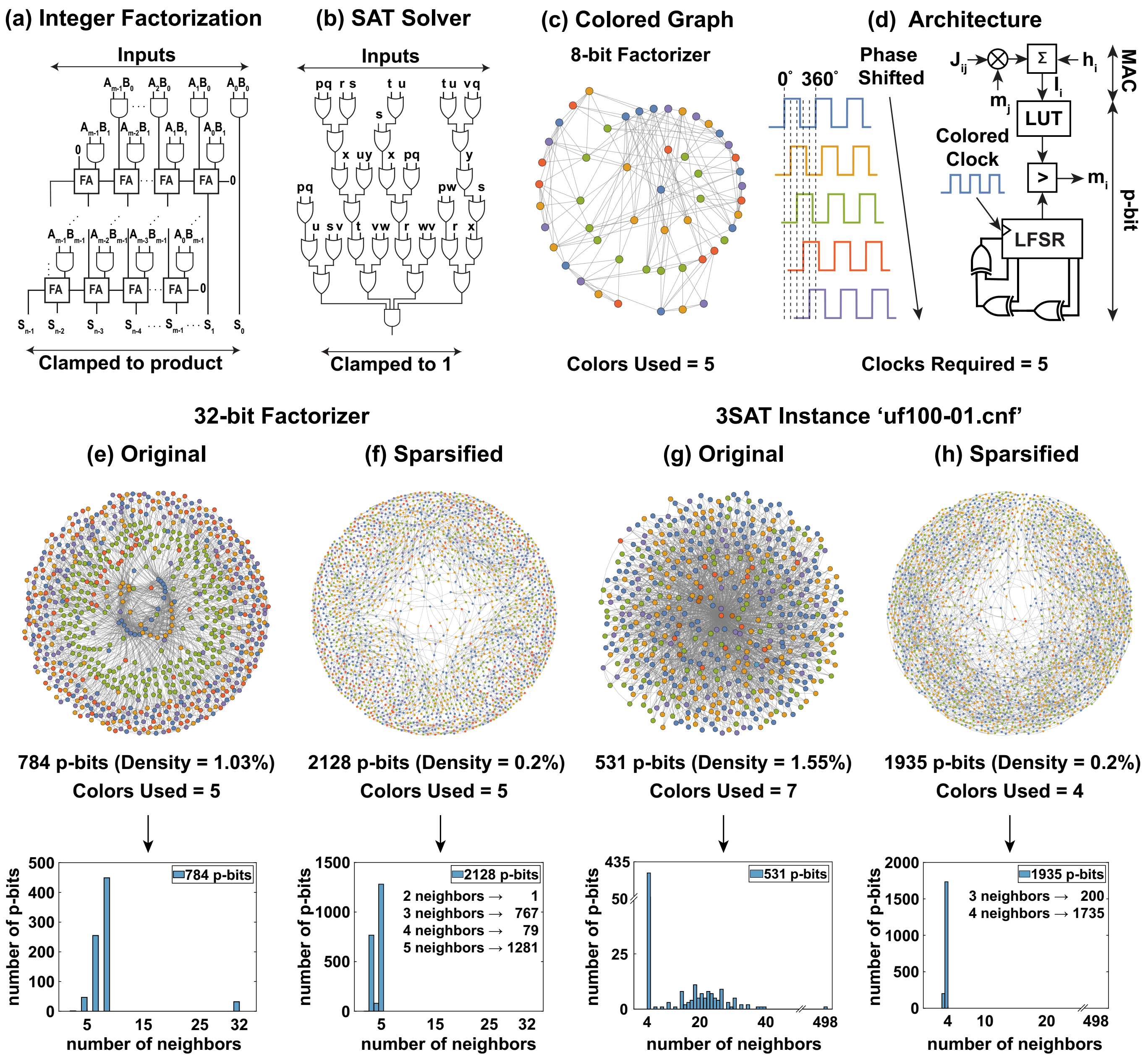

Fig. 1. Combinatorial optimization with invertible logic. (a) A digital multiplier circuit composed of AND gates and full adders (FAs) that can be run invertibly to solve n-bit integer factorization problem. (b) A Boolean circuit composed of OR and AND gates that can be run invertibly to solve satisfiability problems. (c) An example 8-bit factorizer graph colored with 5 colors. (d) An example architecture with 5 parallel and equally phase-shifted clocks, implementing the MAC (Eq. (2)) and the p-bit (Eq. (3)) equations. (e) The original graph of a 32-bit factorizer developed using invertible logic requires 5 colors but it contains nodes with up to 32 neighbors, limiting hardware implementations. (f) Exact sparsification techniques are used to reduce nodes with a large number of neighbors at the expense of additional bits. (g) The original version of the 3SAT instance 'uf100-01.cnf' developed using invertible logic. (h) The sparsified version with nodes having a maximum of 4 neighbors only. Graph density, ρ is defined by Eq. (5).

shifted clocks in the sIM (Fig. 1d). The equal phase shift is required to avoid any concurrent edges of the clocks. In our architecture (Fig. 1d), different color blocks receive different clocks to their RNGs, ensuring no neighboring p-bits flip at the same time for exact Gibbs sampling. This is a constraint we relax in Section VIII.

In the case of exact Gibbs sampling, our architecture ensures the entire network is updated in parallel in one clock period (of any color) while ensuring an effectively sequential operation.

As a quantitative measure of sparsity in the resultant invertible p-circuits we use the notion of graph density ρ. For an undirected graph ρ is defined as

$$\rho = \frac{2|E|}{|V|^2 - |V|} \tag{5}$$

where $|V|$ is the number of nodes (vertices), corresponding to the number of p-bits in the sIM and $|E|$ is the number of edges, corresponding to the interconnections in the $[J]$ matrix. With this definition an all-to-all (or a complete) graph has a ρ of 100%. We find that when p-circuits are composed of universal gates, both the factorization and the satisfiability graphs are sparse (see Supplementary Fig. S5). For example, consider the graph of a 32-bit factorizer p-circuit presented in Fig. 1e having 784 p-bits with a ρ of 1.03%, requiring only 5 distinct colors. In this example, five parallel and equally phase-shifted clocks in the sIM are adequate to implement this p-circuit for massively parallel computation.

B. Sparsification of problem graphs

There is a limitation on the clock speed depending on the maximum number of neighbors for each p-bit. The neighbor distribution of the 32-bit factorizer graph reveals that it has 32 p-bits with 32 neighbors (Fig. 1e). In order to ensure every single color block is updated with the latest state of each neighbor, the MAC unit implementing Eq. (2) needs to finish its computation before the next color block is updated. With binary models the multiplication consists of simple multiplexing (the weights are either selected or they are ignored). This means that the addition for the weights needs to be completed within the n^{th} of a clock period, where n is the number of colors. In the present example, this requires large adders to add 32 s-bit numbers within this short period, where s is the bit precision of the weights.

Therefore even when the sparse graphs as in Fig. 1e,g require a few colors the p-bits with 32 or 498 neighbors introduce large synapse (addition) delays creating a severe bottleneck for how fast the clocks can be operated. To overcome this limitation, we have developed exact sparsification techniques to remove the nodes with a large number of neighbors without changing the structure of the optimization problem. The main idea in this approach is to split a given p-bit representation between two p-bits coupled by a ferromagnetic (J >0) interaction which we call a COPY gate [19, 56, 57]. The ferromagnetic interaction ensures that at the end of an annealing schedule (high β), the ground states of the split and fused models are identical. We show a formal proof in the Supplementary Information Section D.

Fig. 1f shows the sparsified graph of the 32-bit factorizer p-circuit with 2128 p-bits and a graph density of 0.2%. It is colored with 5 colors as before, however, the neighbor distribution reveals the maximum number of neighbors, k is limited to 5 which minimizes the adder delay and allows fast clocks. In general, such sparsification techniques always introduce extra p-bits, however, in scaled implementations individual p-bits are almost always cheaper than complicated synapse interactions.

Similarly, the original graph of the 3SAT instance 'uf-100-01.cnf' can be colored using 7 colors (Fig. 1g). It is also a sparse graph with 531 p-bits and a graph density of 1.55%. However, the neighbor distribution shows that some p-bits have more than 5 connections and one p-bit has 498 connections. This p-bit corresponds to the node where the outputs of all clauses meet. A sparsified version of this graph is illustrated in Fig. 1h with 1935 p-bits and a graph density of 0.2%. As before, the sparsification ensures the graph has a maximum of $k = 4$ neighbors for each p-bit and hence avoids large adder delays, requiring 4 colors (clocks).

Even though we show specific cases of sparsification (with $k = 4$ neighbors for satisfiability and $k = 5$ for factorization), we have analyzed the effect of sparsification (as a function of k) on system size and performance in the Supplementary Information Section E. We note that limiting the number of neighbors per p-bit to a *fixed* value ($k = 4$, $k = 8$ etc.) ensures that the adder delays from the MAC unit do not grow with system size. In other words, no matter how large the global system becomes, only the local neighborhood of a p-bit (with k-neighbors) needs to communicate faster than the p-bit clocks, ensuring the scalability of the approach.

In the following sections, we present our results for integer factorization and SAT solving implemented in the massively parallel sIM architecture. We implemented the sIM architecture on a Xilinx Virtex UltraScale+ FPGA VCU118 Evaluation board for our experiments. The details of the FPGA architecture and its design choices are included in the Supplementary Information Section B. We used a simple simulated annealing algorithm [64] with a linear schedule for all experiments reported in this work.

V. COMPARING SPARSE ISING MACHINE WITH EXISTING HARDWARE (CPU, GPU AND TPU)

A. CPU comparison with approximate factoring

In order to test the parallelism achieved by our approach, first, we compare the sIM implementation of *parallelized* Gibbs sampling with a CPU implementation of standard (serialized) Gibbs sampling [30]. Our purpose in this comparison is to stress the asymptotic scaling differences between the standard approach and our method, rather than a pure performance comparison which we perform later in Section V B, against highly optimized and fine-tuned GPU/TPU implementations.

We define approximate factorization as reaching 99% of the absolute ground energy from 14-bit to 50-bit semiprimes. We do not attempt exact factorization since the absolute ground state is very difficult to be reached using simulated annealing [64] and the CPU practically never reaches there. While finding approximate factors is not useful for the problem of integer factorization, many other optimization problems benefit from high-quality and approximate solutions. As such, we treat approximate factorization as a computationally hard problem benchmark.

We use the same annealing schedule and the same sparsified graphs for the comparison between an unoptimized serial MATLAB implementation on a CPU and the parallel FPGA implementation of the sIM. For each problem, we have attempted to factor 10 different numbers 10 times to make

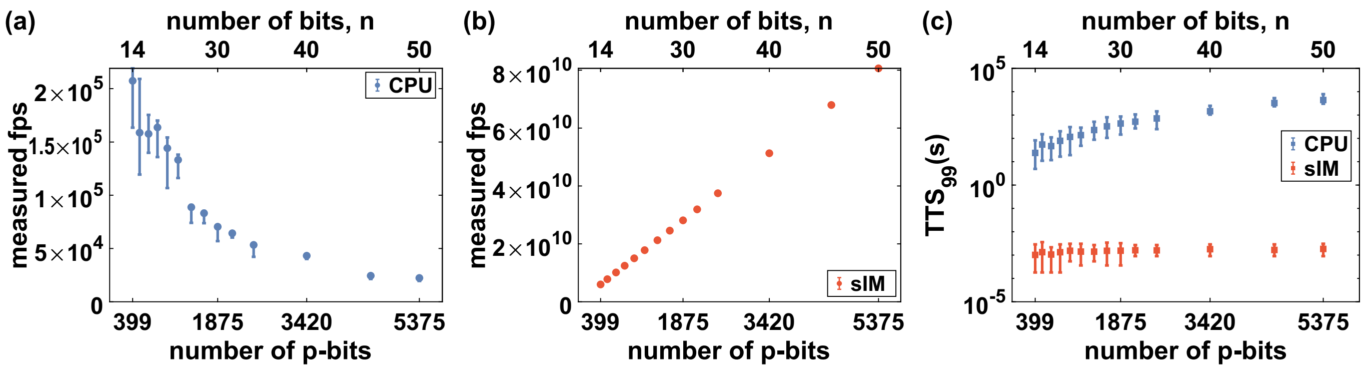

Fig. 2. Performance scaling between parallel (sIM) and serial (CPU) implementations. (a) The CPU flips per second (fps) as a function of graph size is shown, larger graph sizes drastically limiting the fps. (b) The measured sIM fps as a function graph size is shown, showing ideal parallelism scaling linearly with system size in contrast to the serial CPU. (c) Time to solution comparison for approximate factorization between the CPU and the sIM as a function of graph size. TTS_{99} or time to solution, is defined as the time to reach 99% of the ground state. TTS_{99} requires up to 4408.93 s for the 50-bit factorization in the CPU. For the sIM, it is showing almost a flatline behavior with up to $\approx 10^6$x performance improvement.

sure we collect enough statistics and test the robustness of the system.

As a key metric, we first focus on the flips per second (fps) as a figure of merit. The fps corresponds to the *correlated* flips per second that can be taken by the system where every flip is made based on the latest state of the network, between physically connected nodes while avoiding simultaneous updates. Indeed, various hardware implementations for MCMC solvers have reported this metric [38, 39, 65, 66]. The fps can be thought of as the effective processing speed for MCMC. The key point of the parallel architecture we design for the sIM is that its fps *increases linearly* with the number of p-bits corresponding to a problem.

Fig. 2a,b present a comparison of fps between the inherently serial CPU and the massively parallel sIM for the factorization problem. The CPU calculation is done in MATLAB by an iterative solution of Eq. (2)-(3) using standard Gibbs sampling [30] with simulated annealing. Even though optimization techniques including graph coloring by multiple threads could improve our MATLAB implementation, our purpose is to stress the massive parallelism achieved by the sparse architecture we develop over a standard implementation of Gibbs sampling.

MATLAB runs on the Knot Cluster at the Center for Scientific Computing (CSC) server, UCSB featuring an Intel Xeon Processor E5630 running at up to 2.8 GHz. On the other hand, we have used five equally phase-shifted 15 MHz clocks in the sIM and assigned those clocks to the p-bits based on a previously calculated graph coloring. In the sIM, each p-bit updates in parallel, achieving an effective clock speed of $N\times 15$ MHz where N is the number of p-bits. 15 MHz is chosen to satisfy the timing requirements for the additions to be completed and better FPGAs or hardware implementations can be envisioned to reach even higher frequencies.

It is important to note that we directly measured the flips per second of the sIM by means of specially designed reference counters in the FPGA (See the Methods Section X D for the measurement details). The maximum flips per second (fps) achieved by the CPU is limited to around 10^5 (Fig. 2a). Moreover, fps quickly starts to decline for the CPU with the increasing number of p-bits. The main reason for this is due to the sequential nested for-loops in standard Gibbs sampling [30]. On the contrary, the sIM collects up to 80 - 100 billion fps for the highest problem sizes. Crucially, the fps increases linearly with the increasing number of p-bits as shown in Fig. 2b due to its massively parallel architecture. Starting at an fps of 5.99×10^9 for the 14-bit factorization, it achieves a maximum fps of 8.06×10^{10} for the 50-bit factorization.

While we do measure an increasing fps for the sIM as a function of graph size, an important arising question is whether all these samples are useful or not. In order to test this question, we define a performance metric, the time to solution, TTS_{99} as the time to reach 99% of the absolute ground energy. Note that in the case of factorization, the exact solution is planted, in other words, we know the factors of a given product and therefore we have access to the exact ground state energy. Fig. 2c shows a steep rise of TTS_{99} from 14-bit to 50-bit factorization for the CPU. The fastest case requires TTS_{99} of 23.84 s for the 14-bit factorization while the slowest one requires 4408.93 s for the 50-bit factorization. In contrast, the sIM shows a roughly constant mean value for all the problems, requiring 1.02 ms for the 14-bit factorization and 1.84 ms for the 50-bit factorization (See the Methods Section X E for TTS measurement details). The 50-bit factorization is over a 2.4×10^6x improvement over the CPU. The difficulty of the approximate factorization is clearly increasing with increasing problem size. The reason for the constant time to solution for the sIM despite the increasing difficulty of the problem for larger problem sizes can be attributed to the massive parallelism of the sIM where increasing problem size also increases the fps. Therefore, we can conclude that the measured fps shown in Fig. 2b is a real improvement. This makes the massive parallelism of the sIM very different from trivially parallel p-bits sampling independently.

Platform	Graph	flips/ns
Nvidia Tesla C1060 GPU [65, 66]	Chessboard	7.98
Nvidia Tesla V100 GPU [39]	Chessboard	11.37
Google TPU [39]	Chessboard	12.88
Nvidia Fermi GPU [38]	Chessboard	29.85
FPGA sIM [This work]	Irregular	143.80
Nanodevice sIM [Projected] [15, 25, 47, 48]	Irregular	1,000,000

TABLE I. Comparison of the FPGA-based and nanodevice-based (projected) sIM with optimized GPU and TPU implementations of Markov Chain Monte Carlo sampling. Unlike the GPUs and TPUs, the sIM can support regular chessboard lattices as well as irregular graphs shown in this paper. For the sIM, fps is a size dependent metric as shown in Fig. 2,4. In this table, the best fps achieved in the largest system is quoted.

Platform	Integers Factored (up to)
D-Wave 2000Q [42]	143 (8-bit)
CMOS Inv. Logic [22]	598 (10-bit)
Stochastic MTJ [15]	945 (10-bit)
FPGA RBM [20]	43621 (16-bit)
D-Wave 2000Q [43]	223357 (18-bit)
D-Wave 2000Q [44]	249919 (18-bit)
FPGA sIM [This work]	4277546633 (32-bit)

TABLE II. Comparison between the state-of-the-art hardware factorizers and the FPGA-based sIM. Note that all of these solvers treat integer factorization as a frustrated spin-glass problem and perform classical or quantum annealing (or ordinary sampling). We show best reported numbers and single instances, in the case of sIM, random semiprimes up to 32-bits can be reliably factored, see Section VI.

B. GPU and TPU comparison

Table I summarizes the performance benchmarking of the current work (sIM) with the state-of-the-art GPUs and TPUs. An important distinction between these comparisons is that virtually all optimized implementations of GPUs and TPUs make use of a regular 2D chessboard lattice where partitioning the graph into two color blocks becomes the key piece that enables parallelism. By contrast, in our examples, we show that for realistic instances of combinatorial optimization problems using invertible Boolean circuits, the resulting graphs are sparse but not necessarily regular or bipartite (2-color). Even though theoretical results suggest simple nearest-neighbor models could be sufficient to model any other problem [67], accelerating problem graphs in between nearest-neighbor and all-to-all will be critically important in practice.

The performance of the sIM flips per second (fps) grows linearly with the number of p-bits in a network, hence the fps becomes a size dependent metric. We observe the sIM reaches up to 143.8 flips/ns at one of the largest graph nodes (4793 p-bits). This is an 18.03x performance gain over the single Nvidia Tesla C1060 GPU with multi-spin coding [65, 66]. It also outperforms the Google Cloud TPU implementation of the 2D Ising model by a factor of 11.17 and its reference Nvidia Tesla V100 GPU by a factor of 12.65 [39]. Furthermore, the sIM provides 4.82x more flips/ns than the Nvidia Fermi GPU coded for simulating the 3D Edwards–Anderson model with parallel tempering [38].

Beyond the FPGA-based sIM implementation we consider in this paper, we make performance projections for massively parallel asynchronous sIMs using nanodevices (see Supplementary Section A). Magnetic Tunnel Junctions (MTJ) have recently attracted attention as building blocks for probabilistic computation because of their extreme scalability. The magnetic memory industry have integrated up to billions of single MTJs to replace various parts of the memory hierarchy [68]. Through minimal modifications such MTJs can be made stochastic [15, 69], providing the expensive random number generation with negligible hardware cost. Stochastic MTJs have been demonstrated to provide fast fluctuations (τ =1 ns / flip) [47, 48] and at least a million MTJs ($N = 10^6$) can be integrated in massively parallel architectures [15, 25] similar to what we consider in this paper. Following the linear scaling law we demonstrated in Fig. 2b, such sIMs can provide N/τ flips per second reaching 1 million flips per nanosecond (Table I), provided that the connectivity of the hardware is sparse enough to enable the ideal parallelism demonstrated in this paper.

VI. EXACT FACTORIZATION OF SEMIPRIMES UP TO 32-BITS

Beyond approximate factorization, we have also performed exact factorization with the sIM. As mention in Section II, we consider the integer factorization problem as a benchmark to compare the performance of our sIM implementation against other probabilistic solvers or D-Wave's quantum annealers. We found that the sIM can factor random semiprimes up to 32-bits reliably (Supplementary Information, Fig. S7). To the best of our knowledge, this result is by far the best among all other approaches of solving factorization as an optimization problem, for example by D-Wave and others [15, 20, 22, 42–44]. Table II presents a comparison of integer factorization across the state-of-the-art hardware platforms, and the sIM reports the largest factorization up to 32-bit. What is common to all these solvers is they express factorization as a frustrated spin-glass problem in which the ground state is searched using classical or quantum annealing.

From an algorithm perspective, factorization of 32-bit semiprimes is not difficult since even with trial division this is a relatively easy computation. From a statistical physics perspective, however, finding the doubly degenerate ground state of a frustrated spin glass in a 2^N dimensional space ($N >$ 2000 p-bits) is striking. Contrasting the time to solution shown for approximate factorization (Fig. 2c) to that of exact factorization (Fig. S7) as function of problem size, we observe a drastic difference in algorithmic scaling, indicative of a "golf course" like energy landscape where the ground state is well-hidden from "ordinary" approximate states that are easy to reach.

It is worth stressing that we used a simple, standard simulated annealing algorithm without any fine tuning or optimization. Our preliminary findings indicate parallel

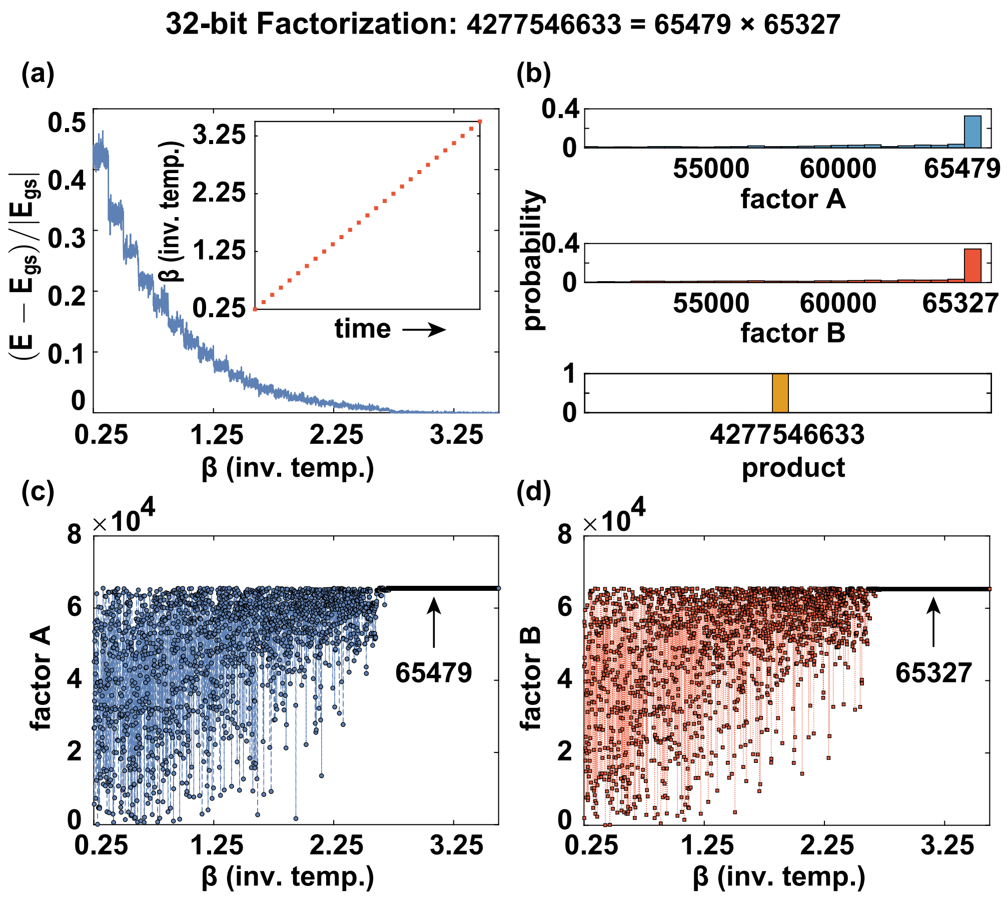

Fig. 3. Exact factorization of a 32-bit number, P= 4277546633 in the sIM. (a) The normalized energy as a function of inverse temperature, (β from Eq. (4). Inset shows the linear annealing schedule. (b) Histograms of the product P and the factors A, B over the entire annealing schedule. (c-d) Factors A and B at different β values, showing they converge to the right ground state at the highest β.

tempering or other algorithmic methods could improve the success probability of these results. We believe the success of our approach over similar alternatives is due to the sparsification methods that have enabled the massively parallel sampling architecture of the sIM.

Fig. 3 presents the exact factorization of a 32-bit number, P= 4277546633. The linear annealing schedule and the normalized energy of the system are presented in Fig. 3a. The absolute ground state is reached at the coolest temperature (the highest β). The histograms of the product P and the factors A, B over the entire annealing schedule are presented in Fig. 3b where the exact factors A = 65479 and B = 65327 are visited reliably. Fig. 3c and Fig. 3d reconfirm that the factors are consistently found at the highest β without any fluctuations.

While we do not show statistics in Fig. 3, in Supplementary Fig. S7, we report the time to find the exact factors (TTS_{100}) from 14-bit to 32-bit semiprimes. For any of these problems, the CPU fails to find the exact factors even over a very long time and therefore is excluded from the TTS_{100} report and we report the sIM times. As before, we attempt to factor 10 different numbers 10 times for each problem.

Unlike approximate factorization where we defined TTS_{99} as the average time the sIM takes before reaching 99% of the absolute ground state, we find that for exact factoring there is an (empirical) exponential dependence of the time with respect to problem size, in line with the belief that integer factorization is in NP where a known polynomial algorithm does not exist [70].

To reiterate, extrapolating our observed data with an exponential fit to estimate exact factoring, we find that factorization with this method is not a practical approach in the context of cryptography, in agreement with the observation from Ref. [45]. However, improving SAT solving with massively parallel hardware could still be useful in accelerating critical subroutines of the best factoring algorithms [46].

VII. BOOLEAN SATISFIABILITY WITH INVERTIBLE LOGIC

A. sIM vs. competition-winning SAT Solvers

As shown earlier in Fig. 1, it is possible to design invertible Boolean circuits in hardware corresponding to satisfiability instances using the principles of invertible logic. Here, we focus on solving 3SAT problems to demonstrate the hardware acceleration of our massively parallel sIM architecture. In

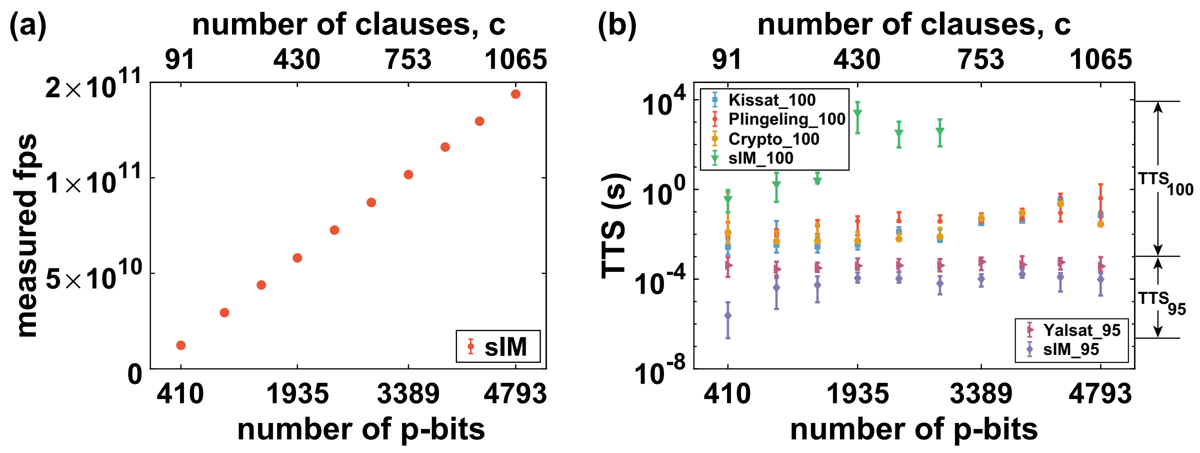

Fig. 4. Performance of the sIM vs. competition-winning SAT solvers. (a) The flips per second (fps) of the sIM increases linearly with the graph size, showing ideal parallelism. The sIM achieves a record fps of 1.44×10^{11} for the largest problem with 4793 p-bits and 1065 clauses. (b) Runtimes to solve the UBC SATLIB [55] 3SAT instances. Winning solvers from the SAT 2020 competition, namely Kissat, Plingeling, and, Cryptominisat [71, 72] find 100% solution (all clauses satisfied). SAT competition 2017 random track winner Yalsat [73] is set to find the 95% solution (95% of the clauses satisfied). In exact SAT solving, the sIM is slower than all professional SAT solvers. However, in approximate SAT solving to satisfy 95% of the clauses, the sIM outperforms all of these solvers (including local search-based SAT solvers, such as Yalsat) delivering the fastest solution.

particular, our purpose is to compare the sIM with the best possible software algorithms and we focus on competition winning SAT solvers as a benchmark. As previously, we first report the flips per second (fps) achieved by the sIM for different 3SAT instances defined by the number of their clauses (Fig. 4a). The sIM runs with four parallel and equally phase-shifted clocks operating at 30 MHz since in this case the sparsified graphs require only 4 colors. For the smallest instance 'uf20-01.cnf' with 20 variables and 91 clauses, the sIM achieves an fps of 1.23×10^{10} with 410 p-bits. For the largest instance 'uf250-01.cnf' with 250 variables and 1065 clauses, the sIM achieves a record fps of 1.44×10^{11} with 4793 p-bit, a 5 to 18x speed up over optimized TPUs and GPUs discussed in Section V (Table I).

Fig. 4b shows the run times to solve the UBC SATLIB [55] 3SAT instances using different professional SAT solvers and the sIM. We solve each instance 100 times to obtain enough statistics. We compare our results with award-winning solvers from the SAT 2020 competition, namely Kissat, Plingeling, and Cryptominisat [71, 72]. These conflict-driven clause learning (CDCL) solvers attempt to find the exact solution that satisfies all of the clauses. All these solvers are executed on the same Linux machine having a flagship Intel Core i9-10900 Processor running at up to 5.20 GHz. Time to solve all the clauses (100% solution) is reported as Kissat_{100}, Plingeling_{100}, and, Crypto_{100} respectively. We have used linear simulated annealing in the sIM to report the time to solve all the clauses, labeled as sIM_{100} in Fig. 4b.

As typical of simulated annealing [64], reaching the absolute ground state is difficult for the sIM. We do report the sIM_{100}, namely the time it takes for sIM to satisfy all clauses in a given 3SAT instance and find that despite the enormous number of flips per second taken by the massively parallel processor, we did not find the ground state beyond 2903 p-bits (Fig. 4b, sIM_{100}).

In many practical instances however, the user may not be interested in finding the absolute ground state of an optimization problem, and reaching approximate but practically useful solutions as quickly as possible is far more important. In such a paradigm, we find that the sIM beats all of the SAT solvers mentioned (Fig. 4b, sIM_{95}). Because CDCL-based solvers such as Kissat, Plingeling and Crpytominisat are programmed to find the exact solution, we also test the sIM against another solver Yalsat (2017 SAT competition random track winner) which keeps a current best solution around. We program the solver to stop when it reaches 95% of the solution (denoted as Yalsat_{95}). We run Yalsat on the same Linux Machine. sIM is also set to solve 95% of all the clauses and the time to solution is noted as sIM_{95}. We find that in this approximate SAT solving mode, the sIM provides the fastest approximate solution outperforming all of the professional SAT solvers by a factor of 4 to 700. We find that there is no failure even for the largest instance ('uf250-01.cnf') we can fit to our FPGA encoded with 4793 p-bits. It takes only 2.36 μs for the instance 'uf20-01.cnf' and 98.26 μs for the instance 'uf250-01.cnf' to solve 95% of the clauses. We expect larger improvements in more scaled implementations of our sIM architecture using more powerful FPGAs or application specific integrated circuits.

VIII. OVERCLOCKED GIBBS SAMPLING WITH SPARSE ISING MACHINES

The parallelized sparse Ising Machine architecture we introduced in this work implements Eq. (2)-(3) exactly. This is ensured by making every color block update with the most up-to-date neighbor information. Since this architecture is

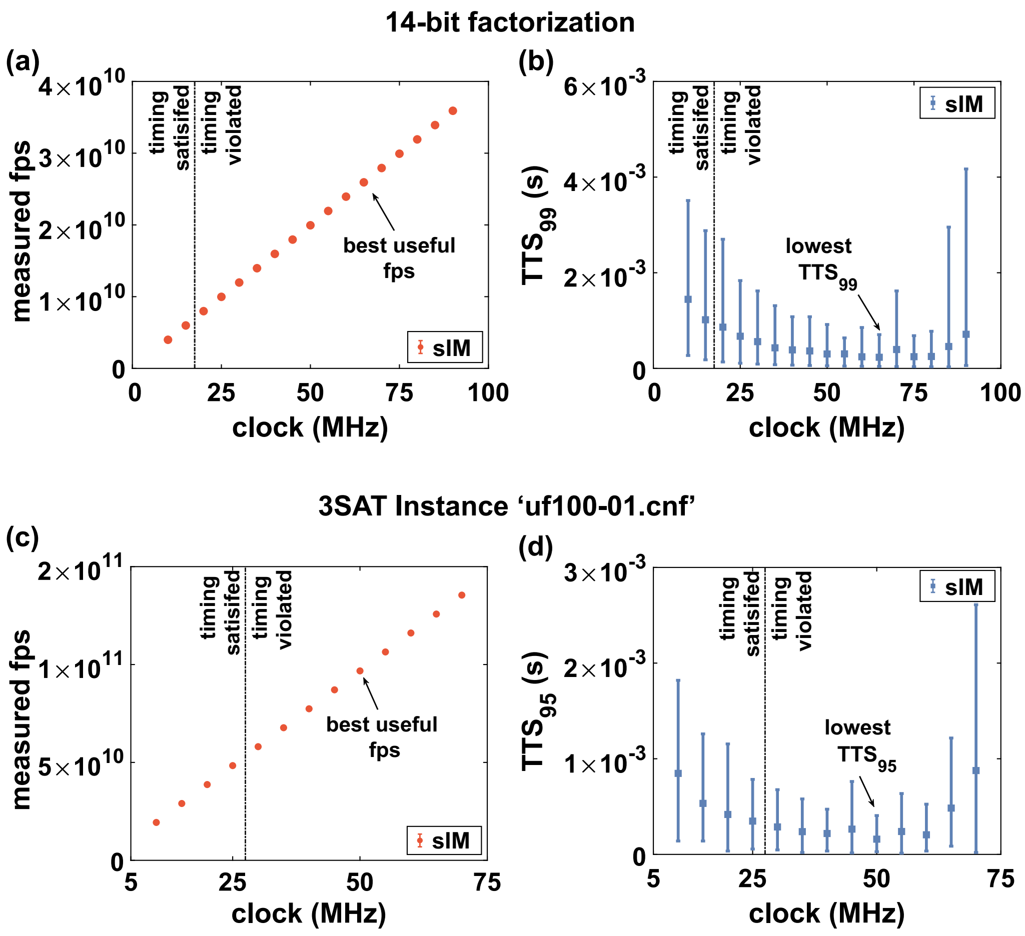

Fig. 5. Overclocked Gibbs Sampling with sIM. (a) 14-bit factorization: the flips per second (fps) as a function of p-bit clocks is shown. Timing is violated after 15 MHz. Based on (b), we observe that overclocking effectively improves fps up to 65 MHz. (b) Time to solution to reach 99% of the ground state, TTS_{99}, measured as a function of clock frequency. (c) 3SAT instance 'uf100-01.cnf': measured fps as a function of p-bit clocks shown. Timing is violated above 25 MHz. Based on the results of (d), the highest effective fps is achieved at 50 MHz. (d) Time to solution to satisfy 95% of the clauses, TTS_{95} as a function of clock frequencies. Both results (b)-(d) show qualitatively similar behavior of improving performance followed by a sharp decline, indicative of universal behavior.

inspired by asynchronous and physical p-bit implementations, for example using stochastic magnetic tunnel junctions, a natural question to consider is whether inexact Gibbs sampling where the p-bits do not update with the most up-to-date neighbor information is worth considering. This approach is reminiscent of asynchronous Gibbs sampling (or Hogwild! Gibbs) approaches that have been analyzed theoretically [35, 74, 75].

Here, we (systematically) investigate how increasing individual clock frequencies of color blocks and eventually performing *inexact* Gibbs sampling with old statistics at the p-bit level affects system performance. Remarkably, for two completely different problems (integer factorization and Boolean satisfiability), we observe qualitatively similar results as a function of increasing clock frequency, therefore increasing the number of messages dropped between neighbors, reminiscent of approximate message passing algorithms [76]. In both cases (Fig. 5b,d), we observe an initial decrease in time to solution with increasingly incorrect updates followed by a sharp increase. Increasing the clock frequencies naturally increase the fps of the main network (Fig. 5a,c), however, when messages are dropped beyond a certain threshold, the improvement in the fps does not help the network converge to the right answers.

To test the generality of overclocking, we analyzed inexact Gibbs sampling with systematically introduced errors in a 5 p-bit full adder circuit (Supplementary Section G). By introducing two different error models, we observe qualitatively similar behavior, where introducing a small amount of error (related to the number of messages dropped between neighbors) does not lead to significant deviations from the exact Boltzmann distribution. This explains why a moderate amount of overclocking is effective: increasing fps without introducing significant errors decreases time to solution, as observed in two different problems in Fig. 5. In Supplementary Section G, we also show how further overclocking reduces the network to a fully synchronous (parallel) updating state, providing analytical estimates of limiting behavior.

One additional reason why overclocking improves performance significantly is due to the slowing down of the network dynamics at lower temperatures (higher β). At the end of an annealing schedule, despite introducing timing failures in many critical paths (and potentially dropping a significant number of messages between neighbors), these critical paths are not activated because p-bits do not change their states frequently. The degree of resilience of the system to errors through inexact sampling is an important feature of such probabilistic methods [77] which can be exploited in truly asynchronous nanodevice-based implementations of sIM [15, 25, 47, 48].

IX. CONCLUSIONS

In this paper, we proposed and implemented a massively parallel architecture, the sparse Ising Machine, to parallelize a broad range of Markov Chain Monte Carlo (MCMC) algorithms useful for computationally hard problems. In particular, overcoming the fundamentally serial nature of MCMC algorithms such as Gibbs sampling, we have shown an architecture that can achieve ideal parallelism where the main metric of the sIM, the flips per second, scales linearly with the number of probabilistic bits in the system. This parallel architecture used several algorithmic ideas of combining invertible logic to produce sparse graph representations of combinatorial optimization problems such as Boolean satisfiability and integer factorization. Further sparsification was needed to ensure matrix multiplication and addition can be performed before independent p-bits are updated. The architecture used approximate graph coloring to parallelize sampling.

We have shown an FPGA-based implementation of this concept where we have achieved three major results. First, comparisons to an ordinary CPU implementation of Gibbs sampling showed that the sIM is able to achieve up to 6-orders improvement in flips per second, which directly translated to advantages in time to solution in the integer factorization problem. Comparisons to highly optimized GPU and TPU implementations, the sIM showed up to 5-18x measured speed up in flips per second, without the use of regular or simple graphs as commonly used for benchmarking purposes in GPUs and CPUs. Second, the sIM was able to factor semiprimes up to 32-bit integers, far larger than the best available results on factoring where an optimization approach is taken. And third, the sIM was able to beat competition winning SAT solvers in approximate satisfiability, delivering superior performance compared to the best possible classical approach in solving satisfiability problems. We have also shown how overclocking in the spirit of asynchronous Gibbs sampling [35] could lead to performance improvements.

These results were obtained in a FPGA platform where our problem sizes were limited to the number of probabilistic bits we could fit in a single device. Use of more powerful FPGAs would immediately extend the size of problems programmable to the sIM. The ideal parallelism we achieved in the architecture, coupled with algorithmic sparsification techniques we developed can further be exploited in highly scaled implementations. In particular, nanodevice (or analog CMOS-based) p-bits can produce significant improvements over our present results, as the search for domain-specific hardware in the beyond Moore era of electronics intensifies.

X. METHODS

A. Problem description

For the factorization problem, we generated random semiprime numbers from 14-bit to 50-bit using MATLAB. For each instance, 10 different numbers were generated. The graphs obtained using invertible logic and sparsification are very sparse (See Supplementary Fig. S5a).

For the SAT problem, we solved 3SAT instances (each clause has exactly 3 variables). The instances were collected as .CNF files from the UBC SATLIB library [55]. Similar to factorization, the 3SAT graphs are very sparse (See Supplementary Fig. S5b).

B. Simulated annealing

In simulated annealing, β is gradually increased over time. According to Eq. (2), multiplication of β and the input weights (J, h) are performed in MATLAB. The updated values of J and h are sent to the FPGA for every β over time to do simulated annealing.

C. Data READ/WRITE

MATLAB is used to READ/WRITE data from the FPGA through a USB-JTAG interface (see Supplementary Fig. S2a). A programmable timer is implemented in the FPGA. Using the timer, a global DISABLE signal is sent to the p-bits before a READ instruction. The timer is preset from the program (MATLAB) and all the p-bits are automatically frozen at the same time when the time is up. Once the p-bits are frozen, the data are READ using the USB-JTAG interface and sent to MATLAB for post-processing. When the READ instruction is DONE, the timer is RESET from MATLAB to resume the p-bits if necessary. Similarly, for the WRITE instruction, a global DISABLE signal is sent using the programmable timer to freeze the p-bits before sending the weights. Likewise the READ instruction, the timer is RESET from MATLAB to resume the p-bits after the WRITE instruction is DONE.

D. Measurement of fps

Each p-bit is designed with a programmable stopwatch counter in the FPGA. A global counter running parallelly is set to count up to a preset value at the positive edge of a known clock. When the global counter is DONE counting, a global DISABLE signal is broadcast to all other counters. Comparing the p-bit counter outputs (number of flips) with the global counter preset value, the time for the total flips is obtained. With this data, the fps of the sIM is measured experimentally for each p-bit. To measure the fps in the case of the CPU, built-in functions from MATLAB is used to measure the elapsed time and programmatically count the total flips in that time. With this data, the fps is measured in real-time. The error bars in all the figures are obtained by taking 100 measurements of fps.

E. Measurement of TTS

In the FPGA, a minimum time is set using the programmable timer to find the solution to the problem of interest. After that time, a global DISABLE signal is sent to READ the latest TTS. In iterations, the minimum time is incremented, and the p-bits are RESET. This process is repeated until the desired solution is reached. The latest TTS is reported as the TTS of the sIM for that problem. In measuring the TTS, we do not include the READ/WRITE times through the USB-JTAG interface. While we use a slow USB-JTAG interface (up to 33 MHz) for the convenience of using MATLAB, much faster R/W protocols such as PCI Express (up to 8 Gb/s) would remove this time entirely. To measure the TTS in the case of the CPU, a predefined minimum number of samples is set to find the solution. The number of samples is increased in iterations until the optimum solution is found by the CPU. The time to solution is recorded using the built-in function and the latest one is reported as TTS of the CPU for that problem. The error bars in all the figures are obtained by taking 100 measurements of TTS.

F. Setting up the SAT solvers

The online source codes of the SAT solvers are used to build the solvers on a Linux machine. For the CDCL SAT solvers, the time to find the 100% solution is measured using a simple Python script. For the local SAT solver Yalsat, the program is set to report the TTS for the current best solution.

ACKNOWLEDGMENTS

The authors are grateful to Brian M. Sutton, Daniel Eppens, Alan Ho and Masoud Mohseni for useful discussions. It is a pleasure to acknowledge Xilinx for hardware support. K.Y.C. and L.T. acknowledge support from the Institute of Energy Efficiency, UC Santa Barbara. K.Y.C. and N.A.A. acknowledge National Science Foundation support through CCF 2106260. The research of A.G., M.C., and G.F. has been supported by the Project No. PRIN 2020LWPKH7 funded by the Italian Ministry of University and Research and by Petaspin association (www.petaspin.com). Use was made of computational facilities purchased with funds from the National Science Foundation (CNS-1725797) and administered by the Center for Scientific Computing (CSC). The CSC is supported by the California NanoSystems Institute and the Materials Research Science and Engineering Center (MRSEC; NSF DMR 1720256) at UC Santa Barbara.

COMPETING INTERESTS

J.M.M. is affiliated with Zyphra Technologies Inc., San Francisco, CA, USA. All other authors have no competing interests.

DATA AVAILABILITY

The data that support the plots within this paper and other findings of this study are available from the corresponding author upon reasonable request.

CODE AVAILABILITY

The computer code used in this study is available from the corresponding author upon reasonable request.

REFERENCES

[1] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[2] Aydin Buluc, Tamara G Kolda, Stefan M Wild, Mihai Anitescu, Anthony DeGennaro, John Jakeman, Chandrika Kamath, Miles E Lopes, Per-Gunnar Martinsson, Kary Myers, et al. Randomized algorithms for scientific computing (rasc). *arXiv preprint arXiv:2104.11079*, 2021.

[3] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[4] Vikash K Mansinghka, Eric M Jonas, and Joshua B Tenenbaum. Stochastic digital circuits for probabilistic inference. *Massachussets Institute of Technology, Technical Report MITCSAIL-TR*, 2069, 2008.

[5] Alexandre Bouchard-Côté, Sebastian J Vollmer, and Arnaud Doucet. The bouncy particle sampler: A nonreversible rejection-free markov chain monte carlo method. *Journal of the American Statistical Association*, 113(522):855–867, 2018.

[6] Werner Krauth. Quantum monte carlo calculations for a large number of bosons in a harmonic trap. *Physical review letters*, 77(18):3695, 1996.

[7] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. Optimization by simulated annealing. In *Readings in Computer Vision*, pages 606–615. Elsevier, 1987.

[8] Peter L McMahon et al. A fully programmable 100-spin coherent ising machine with all-to-all connections. *Science*, 354 (6312):614–617, 2016.

[9] Masanao Yamaoka, Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, and Hiroyuki Mizuno. 24.3 20k-spin ising chip for combinational optimization problem with cmos annealing. In *2015 IEEE International Solid-State Circuits Conference-(ISSCC) Digest of Technical Papers*, pages 1–3. IEEE, 2015.

[10] Hayato Goto, Kosuke Tatsumura, and Alexander R Dixon. Combinatorial optimization by simulating adiabatic bifurcations in nonlinear hamiltonian systems. *Science advances*, 5(4):eaav2372, 2019.

[11] Tianshi Wang and Jaijeet Roychowdhury. Oim: Oscillator-based ising machines for solving combinatorial optimisation problems. In *International Conference on Unconventional Computation and Natural Computation*, pages 232–256. Springer, 2019.

[12] Ibrahim Ahmed, Po-Wei Chiu, and Chris H Kim. A probabilistic self-annealing compute fabric based on 560 hexagonally coupled ring oscillators for solving combinatorial optimization problems. In *2020 IEEE Symposium on VLSI Circuits*, pages 1–2. IEEE, 2020.

[13] Jeffrey Chou, Suraj Bramhavar, Siddhartha Ghosh, and William Herzog. Analog coupled oscillator based weighted ising machine. *Scientific reports*, 9(1):1–10, 2019.

[14] S Dutta, A Khanna, AS Assoa, H Paik, DG Schlom, Z Toroczkai, A Raychowdhury, and S Datta. An ising hamiltonian solver based on coupled stochastic phase-transition nano-oscillators. *Nature Electronics*, 4(7):502–512, 2021.

[15] William A Borders et al. Integer factorization using stochastic magnetic tunnel junctions. *Nature*, 2019.

[16] Marco Baity-Jesi, Rachel A Baños, Andres Cruz, Luis Antonio Fernandez, José Miguel Gil-Narvión, Antonio Gordillo-Guerrero, David Iñiguez, Andrea Maiorano, Filippo Mantovani, Enzo Marinari, et al. Janus ii: A new generation application-driven computer for spin-system simulations. *Computer Physics Communications*, 185(2):550–559, 2014.

[17] Maliheh Aramon, Gili Rosenberg, Elisabetta Valiante, Toshiyuki Miyazawa, Hirotaka Tamura, and Helmut G Katzgraber. Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics*, 7:48, 2019.

[18] Kasho Yamamoto, Kota Ando, Normann Mertig, Takashi Takemoto, Masanao Yamaoka, Hiroshi Teramoto, Akira Sakai, Shinya Takamaeda-Yamazaki, and Masato Motomura. 7.3 statica: A 512-spin 0.25 m-weight full-digital annealing processor with a near-memory all-spin-updates-at-once architecture for combinatorial optimization with complete spin-spin interactions. In *2020 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 138–140. IEEE, 2020.

[19] Saavan Patel, Lili Chen, Philip Canoza, and Sayeef Salahuddin. Ising model optimization problems on a fpga accelerated restricted boltzmann machine. *arXiv preprint arXiv:2008.04436*, 2020.

[20] S Patel et al. Logically synthesized, hardware-accelerated, restricted boltzmann machines for combinatorial optimization and integer factorization. *arXiv preprint arXiv:2007.13489*, 2020.

[21] Yuqi Su, Junjie Mu, Hyunjoon Kim, and Bongjin Kim. A 252 spins scalable cmos ising chip featuring sparse and reconfigurable spin interconnects for combinatorial optimization problems. In *2021 IEEE Custom Integrated Circuits Conference (CICC)*, pages 1–2. IEEE, 2021.

[22] S Smithson et al. Efficient cmos invertible logic using stochastic computing. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 66(6):2263–2274, 2019.

[23] Fuxi Cai, Suhas Kumar, Thomas Van Vaerenbergh, Xia Sheng, Rui Liu, Can Li, Zhan Liu, Martin Foltin, Shimeng Yu, Qiangfei Xia, et al. Power-efficient combinatorial optimization using intrinsic noise in memristor hopfield neural networks. *Nature Electronics*, 3(7):409–418, 2020.

[24] Kosuke Tatsumura, Masaya Yamasaki, and Hayato Goto. Scaling out ising machines using a multi-chip architecture for simulated bifurcation. *Nature Electronics*, 4(3):208–217, 2021.

[25] Brian Sutton et al. Autonomous probabilistic coprocessing with petaflips per second. *IEEE Access*, 8:157238–157252, 2020.

[26] David H Ackley, Geoffrey E Hinton, and Terrence J Sejnowski. A learning algorithm for boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

[27] Geoffrey E Hinton. Boltzmann machine. *Scholarpedia*, 2(5): 1668, 2007.

[28] Vivek Dixit, Raja Selvarajan, Muhammad A Alam, Travis S Humble, and Sabre Kais. Training restricted boltzmann machines with a d-wave quantum annealer. *Front. Phys. 9: 589626. doi: 10.3389/fphy*, 2021.

[29] Yaroslav Koshka and Mark A Novotny. Toward sampling from undirected probabilistic graphical models using a d-wave quantum annealer. *Quantum Information Processing*, 19(10): 1–23, 2020.

[30] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[31] Giovanni Finocchio, Massimiliano Di Ventra, Kerem Y Camsari, Karin Everschor-Sitte, Pedram Khalili Amiri, and Zhongming Zeng. The promise of spintronics for unconventional computing. *Journal of Magnetism and Magnetic Materials*, 521:167506, 2021.

[32] Julie Grollier, Damien Querlioz, KY Camsari, Karin Everschor-Sitte, Shunsuke Fukami, and Mark D Stiles. Neuromorphic spintronics. *Nature Electronics*, pages 1–11, 2020.

[33] K. Y. Camsari et al. Stochastic p-bits for invertible logic. *Physical Review X*, 7(3):031014, 2017.

[34] Joseph Gonzalez, Yucheng Low, Arthur Gretton, and Carlos Guestrin. Parallel gibbs sampling: From colored fields to thin junction trees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 324–332. JMLR Workshop and Conference Proceedings, 2011.

[35] Matthew J Johnson, James Saunderson, and Alan Willsky. Analyzing hogwild parallel gaussian gibbs sampling. *Advances in neural information processing systems*, 26:2715–2723, 2013.

[36] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence*, (6): 721–741, 1984.

[37] Glenn G Ko, Yuji Chai, Rob A Rutenbar, David Brooks, and Gu-Yeon Wei. Flexgibbs: Reconfigurable parallel gibbs sampling accelerator for structured graphs. In *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, pages 334–334. IEEE, 2019.

[38] Ye Fang, Sheng Feng, Ka-Ming Tam, Zhifeng Yun, Juana Moreno, Jagannathan Ramanujam, and Mark Jarrell. Parallel tempering simulation of the three-dimensional edwards–anderson model with compact asynchronous multispin coding on gpu. *Computer Physics Communications*, 185(10):2467–2478, 2014.

[39] Kun Yang, Yi-Fan Chen, Georgios Roumpos, Chris Colby, and John Anderson. High performance monte carlo simulation of ising model on tpu clusters. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–15, 2019.

[40] Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, and Masanao Yamaoka. Fpga-based annealing processor for ising model. In *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, pages 436–442. IEEE, 2016.

[41] Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, and Masanao Yamaoka. Implementation and evaluation of fpga-based annealing processor for ising model by use of resource sharing. *International Journal of Networking and Computing*, 7(2):154–172, 2017.

[42] E Andriyash et al. Boosting integer factoring performance via quantum annealing offsets. *D-Wave Technical Report Series*, 14, 2016.

[43] Raouf Dridi and Hedayat Alghassi. Prime factorization using quantum annealing and computational algebraic geometry. *Scientific reports*, 7(1):1–10, 2017.

[44] Shuxian Jiang, Keith A Britt, Alexander J McCaskey, Travis S Humble, and Sabre Kais. Quantum annealing for prime factorization. *Scientific reports*, 8(1):1–9, 2018.

[45] M Mosca et al. Factoring semi-primes with (quantum) sat-solvers. *arXiv preprint arXiv:1902.01448*, 2019.
[46] Michele Mosca, Joao Marcos Vensi Basso, and Sebastian R Verschoor. On speeding up factoring with quantum sat solvers. *Scientific Reports*, 10(1):1–8, 2020.
[47] Christopher Safranski, Jan Kaiser, Philip Trouilloud, Pouya Hashemi, Guohan Hu, and Jonathan Z Sun. Demonstration of nanosecond operation in stochastic magnetic tunnel junctions. *Nano Letters*, 21(5):2040–2045, 2021.
[48] Keisuke Hayakawa, Shun Kanai, Takuya Funatsu, Junta Igarashi, Butsurin Jinnai, WA Borders, H Ohno, and S Fukami. Nanosecond random telegraph noise in in-plane magnetic tunnel junctions. *Physical Review Letters*, 126(11):117202, 2021.
[49] Daniel Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22(4):251–256, 1979.
[50] Naoya Onizawa, Kaito Nishino, S Smithson, B Meyer, W Gross, Hitoshi Yamagata, Hiroyuki Fujita, and Takahiro Hanyu. A design framework for invertible logic. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pages 312–316. IEEE, 2019.
[51] JD Biamonte. Nonperturbative k-body to two-body commuting conversion hamiltonians and embedding problem instances into ising spins. *Physical Review A*, 77(5):052331, 2008.
[52] Ahmed Zeeshan Pervaiz, Lakshmi Anirudh Ghantasala, Kerem Yunus Camsari, and Supriyo Datta. Hardware emulation of stochastic p-bits for invertible logic. *Scientific reports*, 7(1): 10994, 2017.
[53] Fabio L Traversa and Massimiliano Di Ventra. Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 27(2):023107, 2017.
[54] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
[55] Holger H Hoos and Thomas Stützle. Satlib: An online resource for research on sat. *Sat*, 2000:283–292, 2000.
[56] William M Kaminsky and Seth Lloyd. Scalable architecture for adiabatic quantum computing of np-hard problems. *Quantum computing and quantum bits in mesoscopic systems*, pages 229–236, 2004.
[57] Vicky Choi. Minor-embedding in adiabatic quantum computation: I. the parameter setting problem. *Quantum Information Processing*, 7(5):193–209, 2008.
[58] Yuya Sugie, Yuki Yoshida, Normann Mertig, Takashi Takemoto, Hiroshi Teramoto, Atsuyoshi Nakamura, Ichigaku Takigawa, Shin-ichi Minato, Masanao Yamaoka, and Tamiki Komatsuzaki. Minor-embedding heuristics for large-scale annealing processors with sparse hardware graphs of up to 102,400 nodes. *Soft Computing*, 25(3):1731–1749, 2021.
[59] Makoto Kato, Naoya Onizawa, and Takahiro Hanyu. Design automation of invertible logic circuit from a standard hdl description. *IfCoLoG Journal of Logics and their Applications*, 8(5):1311–1333, 2021.
[60] Armin Biere, Marijn Heule, and Hans van Maaren. *Handbook of satisfiability*, volume 185. IOS press, 2009.
[61] Andrea Grimaldi, Luis Sánchez-Tejerina, Navid Anjum Aadit, Stefano Chiappini, Mario Carpentieri, Kerem Camsari, and Giovanni Finocchio. Spintronics-compatible approach to solving maximum-satisfiability problems with probabilistic computing, invertible logic, and parallel tempering. *Phys. Rev. Applied*, 17:024052, Feb 2022.
[62] Andrew Lucas. Ising formulations of many np problems. *Frontiers in Physics*, 2:5, 2014. ISSN 2296-424X.
[63] Mohsen Ghaffari and Christiana Lymouri. Simple and near-optimal distributed coloring for sparse graphs. *arXiv preprint arXiv:1708.06275*, 2017.
[64] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., 1989.
[65] Benjamin Block, Peter Virnau, and Tobias Preis. Multi-gpu accelerated multi-spin monte carlo simulations of the 2d ising model. *Computer Physics Communications*, 181(9):1549–1556, 2010.
[66] Tobias Preis, Peter Virnau, Wolfgang Paul, and Johannes J Schneider. Gpu accelerated monte carlo simulation of the 2d and 3d ising model. *Journal of Computational Physics*, 228 (12):4468–4477, 2009.
[67] Gemma De las Cuevas and Toby S Cubitt. Simple universal models capture all classical spin physics. *Science*, 351(6278): 1180–1183, 2016.
[68] Sabpreet Bhatti, Rachid Sbiaa, Atsufumi Hirohata, Hideo Ohno, Shunsuke Fukami, and SN Piramanayagam. Spintronics based random access memory: a review. *Materials Today*, 20 (9):530–548, 2017.
[69] Keito Kobayashi, William A Borders, Shun Kanai, Keisuke Hayakawa, Hideo Ohno, and Shunsuke Fukami. Sigmoidal curves of stochastic magnetic tunnel junctions with perpendicular easy axis. *Applied Physics Letters*, 119 (13):132406, 2021.
[70] Arjen K Lenstra, Hendrik W Lenstra Jr, Mark S Manasse, and John M Pollard. The number field sieve. In *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 564–572, 1990.
[71] Armin Biere Katalin Fazekas Mathias Fleury and Maximilian Heisinger. Cadical, kissat, paracooba, plingeling and treengeling entering the sat competition 2020. *SAT COMPETITION 2020*, page 50, 2020.
[72] Mate Soos, Jo Devriendt, Stephan Gocht, Arijit Shaw, and Kuldeep S Meel. Cryptominisat with ccanr at the sat competition 2020. *SAT COMPETITION 2020*, page 27, 2020.
[73] Armin Biere. Cadical, lingeling, plingeling, treengeling and yalsat entering the sat competition 2017. *Proceedings of SAT Competition*, pages 14–15, 2017.
[74] Constantinos Daskalakis, Nishanth Dikkala, and Siddhartha Jayanti. Hogwild!-gibbs can be panaccurate. *arXiv preprint arXiv:1811.10581*, 2018.
[75] Christopher De Sa, Chris Re, and Kunle Olukotun. Ensuring rapid mixing and low bias for asynchronous gibbs sampling. In *International Conference on Machine Learning*, pages 1567–1576. PMLR, 2016.
[76] Marc Mezard and Andrea Montanari. *Information, physics, and computation*. Oxford University Press, 2009.
[77] Xiangyu Zhang, Ramin Bashizade, Yicheng Wang, Sayan Mukherjee, and Alvin R Lebeck. Statistical robustness of markov chain monte carlo accelerators. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 959–974, 2021.
[78] Rafatul Faria, Kerem Y Camsari, and Supriyo Datta. Implementing bayesian networks with embedded stochastic mram. *AIP Advances*, 8(4):045101, 2018.
[79] Orchi Hassan, Supriyo Datta, and Kerem Y. Camsari. Quantitative evaluation of hardware binary stochastic neurons. *Phys. Rev. Applied*, 15:064046, Jun 2021.
[80] Wei Zhao and Yu Cao. Predictive technology model for nano-cmos design exploration. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 3(1):1–es, 2007.

[81] airhdl.com. airhdl VHDL/SystemVerilog Register Generator. https://airhdl.com.

[82] David Blackman and Sebastiano Vigna. Scrambled linear pseudorandom number generators. *arXiv preprint arXiv:1805.01407*, 2018.

[83] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[84] Masuo Suzuki. Relationship between d-dimensional quantal spin systems and (d+ 1)-dimensional ising systems: Equivalence, critical exponents and systematic approximants of the partition function and spin correlations. *Progress of theoretical physics*, 56(5):1454–1469, 1976.

[85] Jun Cai, William G Macready, and Aidan Roy. A practical heuristic for finding graph minors. *arXiv preprint arXiv:1406.2741*, 2014.

SUPPLEMENTARY INFORMATION

A. Characteristics of CMOS and nanodevice based p-bits

In Section I of the main paper, we have discussed Eq. (3) of a p-bit that can be implemented in several hardware platforms. A CMOS implementation of Eq. (3) is presented in Fig. S1a. A clock-triggered random number generator (RNG) provides the $\mathrm{rand_U}(-1,1)$ function and a lookup table (LUT) maps the tanh activation function. Finally, a comparator is used to trigger the flip of the p-bit. The input-output characteristics of a CMOS implemented p-bit is shown in Fig. S1b. Eq. (3) can also be implemented using nanodevices such as a 14-nm FinFET and a stochastic MTJ (see Fig. S1c) where a mapping between the dimensionless Eq. (3) and the device characteristics can be made [78]. The input-output characteristics of the nanodevice p-bit is presented in Fig. S1d. The parameters used for the simulation are included in Table S1. The simulation time steps to solve the stochastic Landau-Lifshitz-Gilbert equation intrinsically calculates an attempt time. For low-barrier nanomagnets, due to the lack of an energy-barrier there is no clear switching voltage at zero temperature, however, there is the notion of a 'pinning current' [79] which is a function of device parameters.

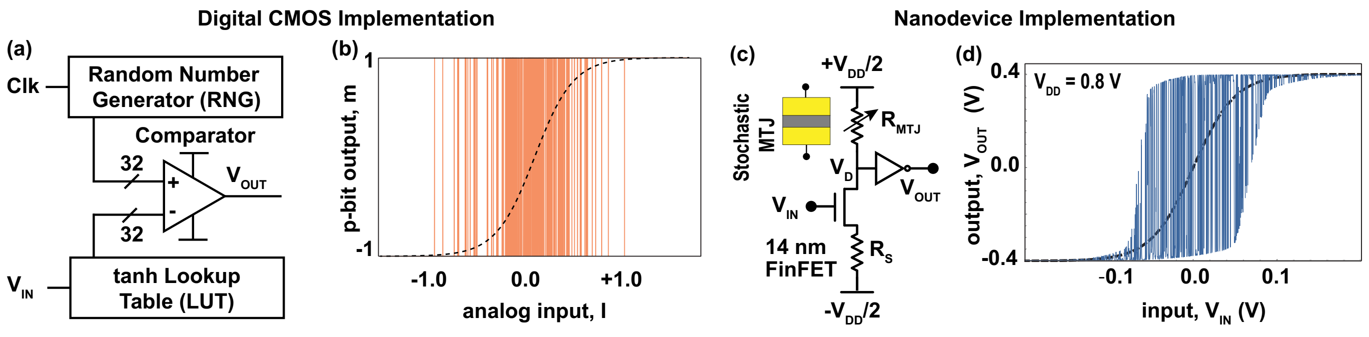

Fig. S1. (a) Digital CMOS implementation of Eq. (3) of a p-bit using an RNG, a LUT, and a comparator. (b) Input-output characteristics of the digital CMOS p-bit. (c) Nanodevice implementation of Eq. (3) using a 14 nm FinFET and a stochastic MTJ. (d) Input-output characteristics of the nanodevice p-bit.

Parameter	Value
Free layer energy barrier	≈ 0 kT
Free layer diameter	20 nm
Free layer thickness	2 nm
Free layer damping coeff. (α)	0.01
H_K (uniaxial anisotropy)	10^{-2} Oe
H_D (demagnetization field)	$4\pi M_s$
Saturation magnetization (M_s)	1100 emu/cc
Interface Polarization	$P = 0.7$
Tunneling Magnetoresistance (TMR)	$2P^2/1-P^2 = 192\%$
Average conductance for MTJ (G_0)	43 μS
NMOS technology model	14-nm HP FinFET, PTM [80]
Time step for integration	$\Delta t = 1$ ps

TABLE S1. Parameters used for nanodevice based p-bit simulation in Fig. S1d.

B. FPGA implementation of the sIM

We have presented the experimental results of the sIM in the main paper without technical details about the implementation of the architecture. Here, we discuss an FPGA based implementation of the sIM in a Xilinx Virtex UltraScale+ VCU118 Evaluation board. The basic architecture of the FPGA design is presented in Fig. S2.

1. Interfacing unit

We use MATLAB as an Advanced eXtensible Interface (AXI) master to communicate with the slave FPGA board through a USB-JTAG interface (Fig. S2a). We have designed an AXI master integrated IP on the board that transfers data with a 32-bit memory-mapped slave register IP via the fourth generation AXI (AXI4) protocol. An external website 'airhdl' [81] is used to manage the memory mapping of the registers.

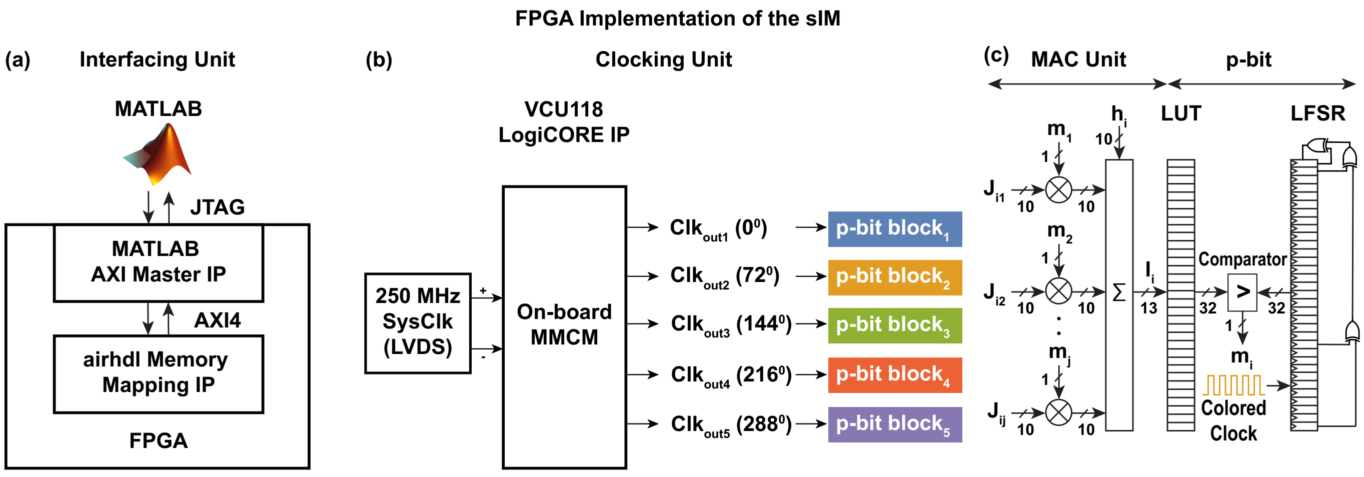

Fig. S2. FPGA based implementation of the sIM. (a) An interfacing unit to communicate between MATLAB and the FPGA. (b) A built-in clocking unit to generate equally phase-shifted parallel clocks to trigger the colored p-bit blocks. (c) The MAC unit to implement Eq. (2) where the colored clock from the general architecture is fed into the LFSR of the p-bit.

2. *Clocking unit*

Section IV in the main paper explains how approximate graph coloring can be used to color p-bit blocks for massive parallelism. For this, we have used built-in clocks on the FPGA board to drive the LFSRs inside the p-bit blocks as shown in Fig. S2b. A 250 MHz Low-voltage Differential Signaling (LVDS) system clock generates equally phase-shifted and parallel stable clocks using the on-board Mixed-Mode Clock Manager (MMCM) Module. This module is available in the VCU118 LogiCORE IP provided by Xilinx. The generated clocks are very accurate, have minimum jitter and minor phase error. The colored p-bit blocks get triggered with these phase-shifted clocks.

3. *MAC unit*

In Section II of the main paper, we described the MAC unit interconnecting the p-bits and computing Eq. (2). Fig. S2c illustrates the MAC unit implemented in the FPGA. In this work, we have used 32-bit linear-feedback shift registers (LFSRs) with taps [32, 22, 2, 1] as RNGs after extensive experiments with different types of RNGs (LFSR, Xoshiro128+ [82], and the Mersenne Twister [83]) with different bit-widths. The LUT bit-width is configured accordingly and a comparator compares the outputs of the LUT and the LFSR. The input weights (J, h) are programmable and we multiply them by β from the MATLAB level to implement simulated annealing.

4. *Correspondence between binary and bipolar variables*

Eq. (1)-(3) presented in Section I of the main paper use bipolar variables. It is more convenient to use binary variables for the FPGA-based implementation of the sIM. In the MAC unit, all the variables are calculated using binary notations where the final output for a p-bit is $m_i \in \{0, 1\}$. The bipolar to binary conversion is done using the following equations:

$$\mathrm{J}_{\text{binary}} = 2\mathrm{J}_{\text{bipolar}} \tag{S.1}$$

$$\mathrm{h}_{\text{binary}} = \mathrm{h}_{\text{bipolar}} - \mathrm{J}_{\text{bipolar}}\, \mathbb{A} \tag{S.2}$$

where, $\mathbb{A}$ is an $[N \times 1]$ vector of ones, and N is the number of p-bits. The activation function values stored in the LUT is converted from a bipolar representation to a binary representation by mapping $\tanh$ to $(1 + \tanh)/2$.

C. Basic logic gates for probabilistic computing

Section III A of the main paper illustrates how any invertible logic probabilistic circuit can be composed using basic logic gates and full adders to solve combinatorial optimization problems. Fig. S3a-l presents the basic logic gates (COPY/NOT/AND/OR) used to build such p-circuits. The J and h matrices have few unique weights that are highlighted using unique colors. The energy plot corresponding to the Boltzmann probability are also included in the figure. For example, for the AND gate, the ground states (states with the lowest energy) $[m_1\ m_2\ m_3]$ = $\{000, 010, 100, 111\}_{\text{bin}}$ correspond to the truth table of the AND gate.

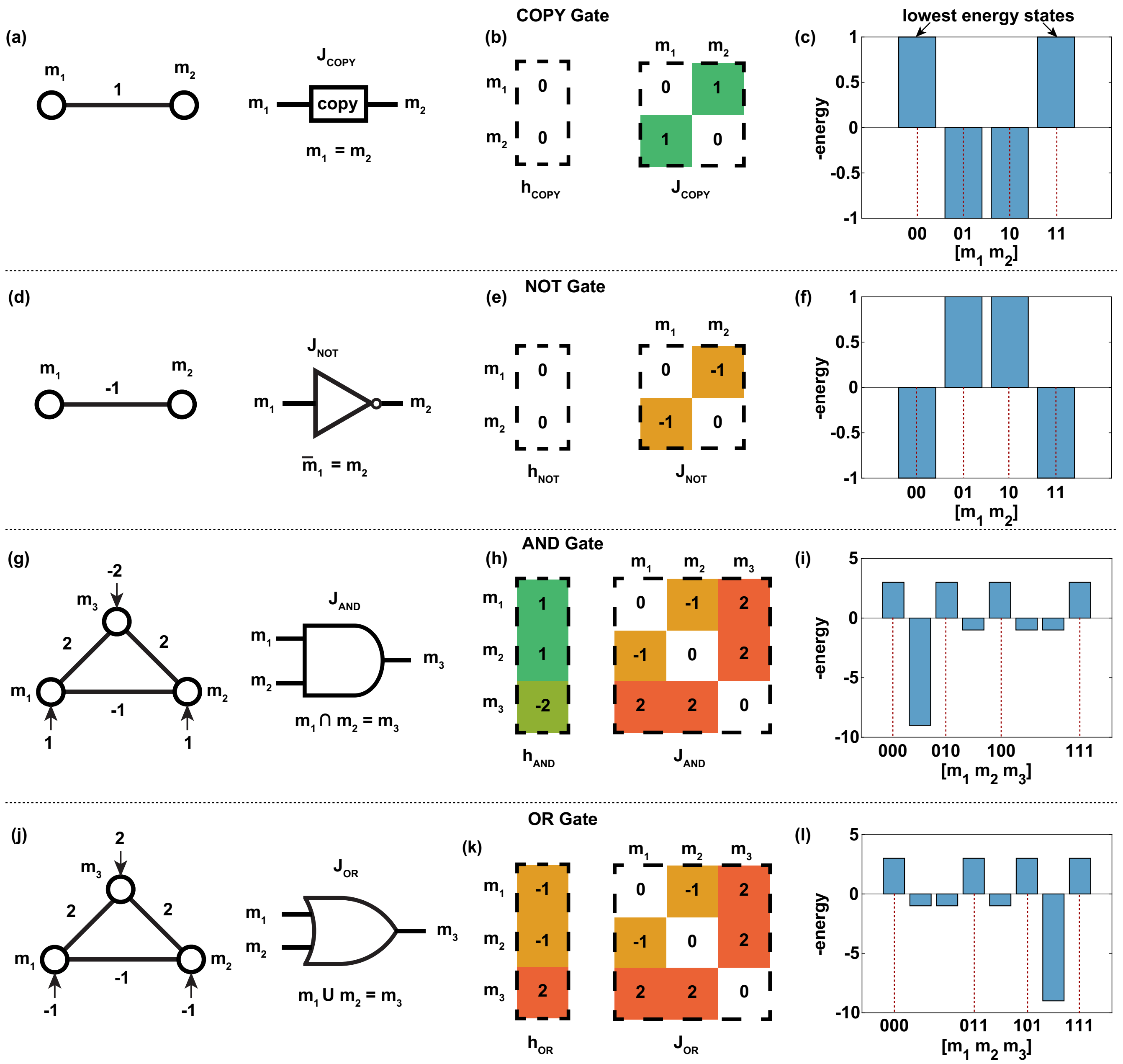

Fig. S3. Basic logic gates for probabilistic computing. (a)-(l) COPY/NOT/AND/OR gates with J and h matrices having few unique weights highlighted using unique colors (not to be confused with graph coloring). The ground states match the basic truth tables where the vectors $[m_1\ m_2]$ or $[m_1\ m_2\ m_3]$ are represented by the corresponding binary numbers.

Fig. S4a-d further illustrates how to compose an invertible logic p-circuit using these basic logic gates. The composite circuit combines the $[3 \times 3]$ J matrices of the AND and OR gates which become a $[5 \times 5]$ matrix after fusion of the common node. Similarly, it combines the $[3 \times 1]$ h matrices of the AND and the OR gates which become a $[5 \times 1]$ h matrix after fusion. The energy plot reveals that the ground states $[m_1\ m_2\ m_3\ m_4] = \{0001, 0101, 1001, 1110, 1111\}_{\mathrm{bin}}$ agree with the truth table of the composite circuit when $m_5 = 1$. The circuit has a maximum number of neighbours, $k = 4$. A sparser version of the circuit with $k = 3$ can be obtained by splitting m_3 into two nodes and inserting a copy gate between m_3 and m_3' (Fig. S4e-h). The ground states are $[m_1\ m_2\ m_3\ m_3'\ m_4] = \{00001, 01001, 10001, 11110, 11111\}_{\mathrm{bin}}$ when $m_5 = 1$. We present a mathematical justification in Supplementary Section D showing the equivalence between these two circuits.

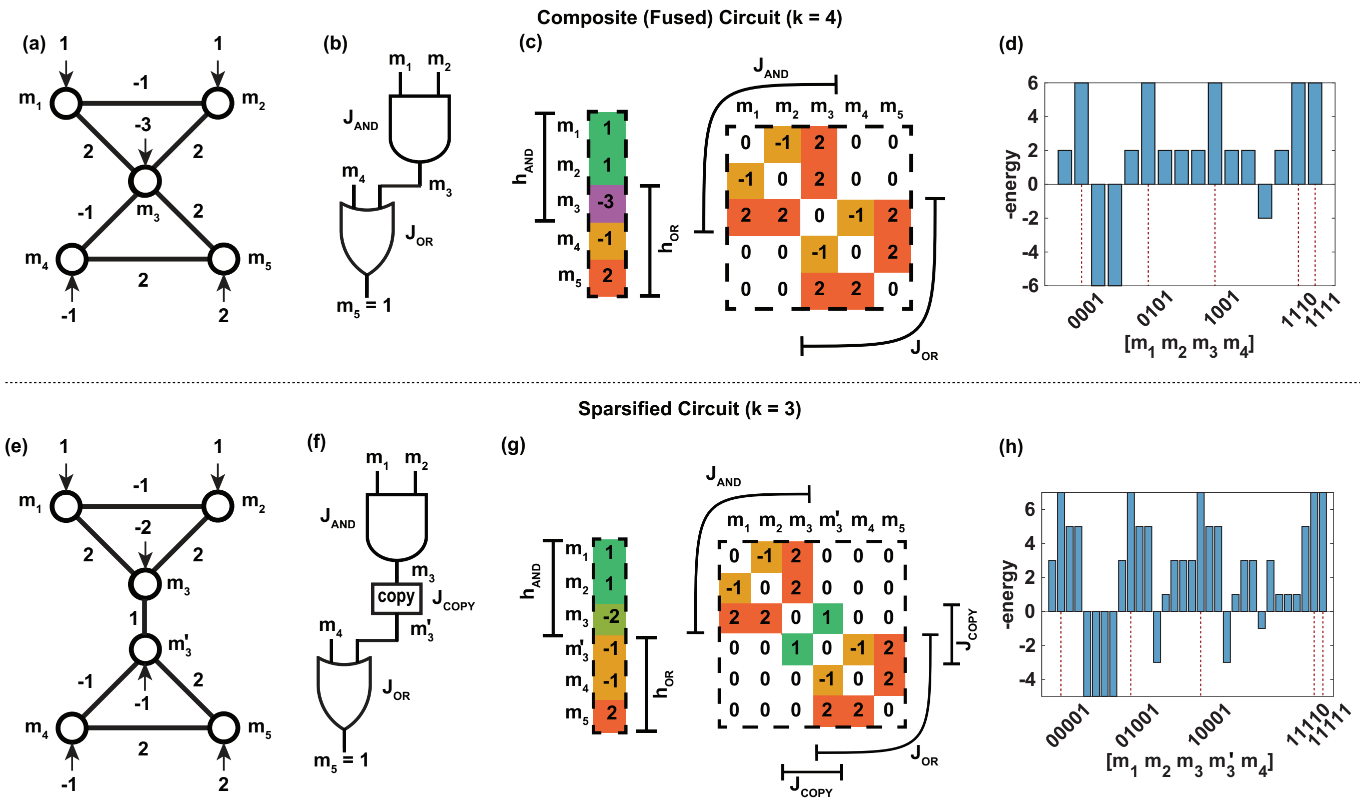

Fig. S4. Fusion and sparsification techniques for probabilistic gates. (a)-(d) A composite p-circuit built using an AND gate and an OR gate with corresponding J and h matrices and energies of states. Unlike the other examples, m_5 is clamped to 1 in this example. (e)-(h) A sparse p-circuit built by splitting m_3 into two nodes and inserting a copy gate between m_3 and m_3'. The energy plot shows the same ground states, showing that these two circuits are equivalent.

D. Fusion and Sparsification

In this section, our objective is to define two graph modification techniques we call fusion and sparsification. Fusion refers to combining multiple nodes (p-bits) to a single node. Sparsification refers to splitting a single node into multiple nodes to decrease the vertex degree (number of neighbors) of a given node. We define k as the maximum number of neighbors per node in a graph. $k = \infty$ represents the original problem (fused) without sparsification. In this section, we give a mathematical proof of the equivalence between a fused and a sparisified graph.

1. Mathematical equivalence of fused and sparsified graphs

Here, we would like to establish that as the annealing parameter β of Eq. (3) (main paper) is increased ($\beta \to \infty$ at the end of an annealing schedule), the sparsified and the fused circuits have the same ground state for a given optimization problem. The connection between the sparsification and the fusion also allows a natural way of composing p-circuits which we elaborate next. Suppose there are two subcircuits we are trying to connect in which m_A corresponds to the node from subcircuit A and m_B corresponds to the node from subcircuit B and that m_A and m_B need to be tied together. Both these systems are described by their respective energies (ignoring biases without loss of generality):

$$E_A = -\left(\sum_{\substack{i \neq A \\ i<j}} J_{ij} m_i m_j + \sum_j J_{Aj} m_j m_A \right) \tag{S.3}$$

$$E_B = -\left(\sum_{\substack{i \neq B \\ i<j}} J'_{ij} m_i m_j + \sum_j J'_{Bj} m_j m_B \right) \tag{S.4}$$

where we separated the energy terms corresponding to m_A and m_B from the rest of the subcircuits. If m_A and m_B are to be connected as a common node between these subcircuits, as in ordinary digital circuits, a positive interaction parameter (ferromagnetic, $J_T > 0$) can be used to connect them such that the total energy of the composed system is given as:

$$E = E_A + E_B - J_T m_A m_B \tag{S.5}$$

This situation corresponds to the sparsified network where the interaction parameter J_T corresponds to the COPY gate which ties the same logical value to m_A and m_B.

We continue the analysis with a physical observation. As the temperature is lowered ($\beta \to \infty$), m_A and m_B cannot differ in their states due to the large energy penalty incurred by βJ_T, or mathematically, $P(m_A \neq m_B) = \exp(-\beta J_T) \to 0$, independent of the specific value of J_T. This allows, in the bipolar notation where $m_A, m_B \in \pm 1$, the following trick: $m_A = m_B$ and $m_A^2 = 1$. This means that the last term in Eq. (S.5) becomes a constant and drops out of the final Boltzmann probabilities since any constant term in the energy cancels out:

$$P(m_1, \ldots, m_N) = \frac{1}{Z} \exp(\beta J_T) \exp[-\beta(E_A + E_B)] \tag{S.6}$$

where

$$Z = \sum \exp(-\beta E) = \exp(\beta J_T) \sum \exp -\beta(E_A + E_B) \tag{S.7}$$

This analysis indicates that for an annealed system ($\beta \to \infty$), irrespective of the strength of the coupling parameter J_T, there is no difference in the final probabilities between subcircuits A and B. For example, subcircuit A can be the fused circuit shown in Fig. S4a-d and subcircuit B can be the sparsified circuit shown in Fig. S4e-h. This analysis is similar to the behavior of the many replicas collapsing to a single qubit in the Suzuki-Trotter transformation, enabling a mapping between the thermodynamics of a many-body quantum system and a probabilistic system [84].

Going back to Eq. (S.3)-(S.4), and substituting $m_A = m_B$ and calculating the input to the node m_A by $I_A = -\partial E / \partial m_A$:

$$I_A = \sum J_{Aj} mj + \sum J'_{Bj} m_j \tag{S.8}$$

Eq. (S.8) shows the mathematical justification of adding the columns of fused nodes together, as shown in the composite circuit of Fig. S4a-d. Note that the rows of fused nodes also need to be added together to ensure the symmetry of the J matrix.

2. *Fused circuit*

For the n-bit factorizer circuit in Fig. 1a, we have $2m$ p-bits from the input bits of the AND gates. The output p-bits of the AND gates get fused with the corresponding input p-bits of the FAs, except the direct output of the first AND gate that represents S_0. It will be represented by a single p-bit. For the FAs, the first row has $4m+1$ p-bits instead of $5m$, since all the neighbor FAs have the c_{out} and c_{in} fused together. For the other rows of the FAs, we have $3m+1$ p-bits per row. This is because one of the input bits comes from the previous row. The number of p-bits in a fused n-bit factorizer p-circuit can be generalized as

$$\begin{aligned} N_{\text{fact}}^{(\text{fused})} &= 2m + (4m+1) + (3m+1)(m-2) + 1 \\ &= 3m^2 + m \end{aligned} \tag{S.9}$$

where $m = \frac{n}{2}$.

Likewise, an invertible logic 3SAT solver circuit (a special case of Fig. 1b with exactly 3 variables per clause and needs only 2 rows of OR gates) can be composed using fusion. Same input variable routed to multiple places is represented by a single p-bit instead of multiple p-bits. This way, we have exactly one p-bit for each input variable. The output p-bits of the OR gates in the first row get fused with the corresponding input p-bits of the OR gates in the second row. To encode this, we need exactly one p-bit for each clause. Finally, all the output p-bits of the OR gates in the second row are fused together and represented by a single p-bit clamped to 1. The number of p-bits in a fused 3SAT p-circuit can be generalized as

$$N_{\text{3SAT}}^{(\text{fused})} = c + v + 1 \tag{S.10}$$

where, c = number of clauses, and v = number of input variables.

The fused p-circuit is software-friendly since it keeps the state space smaller, however, it introduces a fan-out issue in the sIM due to having too many neighbors for some p-bits. It also slows down the clock speed as discussed in Section IV B in the main manuscript.

3. *Sparsified circuit*

In a sparsified p-circuit, we do not fuse the p-bits when it exceeds a predefined maximum number of neighbors, k for any p-bit. Here, we demonstrate two different ways of sparsifying a graph, one with an example of the integer factorization and the other with an example of the 3SAT solver.

In the factorizer p-circuit, we add a series of p-bits to the same-signal input bits using copy gates. Having set a maximum number of neighbors, k for each p-bit, we add a p-bit for every $(k-1)$ input bits to be connected. If the total number of p-bits added is more than 1, the process is started again until only one p-bit is added, which represents the actual input p-bit.

If no p-bits are fused, the n-bit factorizer circuit in Fig. 1a has $3m^2$ p-bits for the m^2 AND gates and $5m(m-1)$ p-bits for the $m(m-1)$ FAs. The number of p-bits for a sparsified n-bit factorizer p-circuit with a maximum number of neighbors, k per p-bit includes additional p-bits for the copy gates and can be generalized as

$$\begin{aligned} N_{\text{fact}}^{(\text{sparse})} &= 3m^2 + 5m(m-1) + 2mf(m,k) \\ &= 8m^2 - 5m + 2mf(m,k) \end{aligned} \tag{S.11}$$

where

$$\begin{aligned} m &= \frac{n}{2} \\ f(m,k) &= \sum_{i=1}^{\lceil \log_{k-1} m \rceil} a_i \\ a_0 &= m \\ a_i &= \left\lceil \frac{a_{i-1}}{k-1} \right\rceil \end{aligned}$$

In this work, we set $k=5$ for integer factorization and the expression can be approximated as

$$\begin{aligned} N_{\text{fact}}^{(\text{sparse})} &\approx 8m^2 - 5m + 2m\left\lceil \frac{m-1}{3} \right\rceil \\ &\approx 8m^2 - 5m + 2m\frac{m-1}{3} \\ &= \frac{26}{3}m^2 - \frac{17}{3}m \end{aligned} \tag{S.12}$$

For the 3SAT problem, instead of fusing, we connect the same-signal input bits by inserting a copy gate between every two input p-bits. This way, the input p-bits get correlated and also avoid additional neighbors. The output p-bits of the OR gates in the first row get fused with the corresponding input p-bits of the OR gates in the second row as before since it does not cost any fan-out issue. Finally, the output p-bits of the OR gates in the second row are fused in pairs and clamped to 1. Each clause has 2 OR gates and thus 5 p-bits since the middle p-bits get fused. However, since the output p-bits also get fused in pairs, we have 1 p-bit less for every 2 clauses. The final sparsified 3SAT solver p-circuit has a maximum number of neighbors $k=4$ only and the number of p-bits can be generalized as (c = number of clauses)

$$N_{\text{3SAT}}^{(\text{sparse})} = \lceil 5c - \frac{1}{2}c \rceil = \lceil \frac{9}{2}c \rceil \tag{S.13}$$

The sparsified p-circuit is hardware-friendly as it limits fan-out and allows fast clocks with small adder delay in the sIM.

4. *Graph density*

In Section IV B of the main manuscript, we have discussed how the sparse (less dense) graphs reduce adder delays by limiting the maximum number of neighbors, k in a graph. Here, we add an analytical expression for the maximum graph density and show how sparsity helps to scale the proposed architecture.

For a graph with k regular neighbors and $|V|$ nodes (vertices), the total number of edges ($|E_{\text{k-regular}}|$) is

$$|E_{\text{k-regular}}| = \frac{k|V|}{2} \tag{S.14}$$

For a graph with all-to-all connections and $|V|$ nodes, the total number of edges ($|E_{\text{all-to-all}}|$) is

$$|E_{\text{all-to-all}}| = \frac{|V|^2 - |V|}{2} \tag{S.15}$$

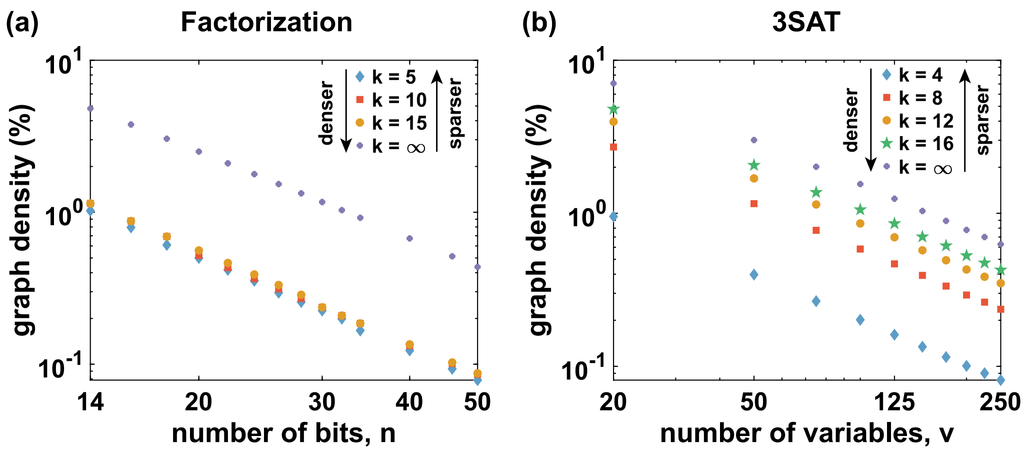

Fig. S5. Graph density as a function of problem size at different k values for (a) factorization and (b) 3SAT problem. k is defined as the maximum number of neighbors after sparsification. $k = \infty$ represents the original problem without sparsification.

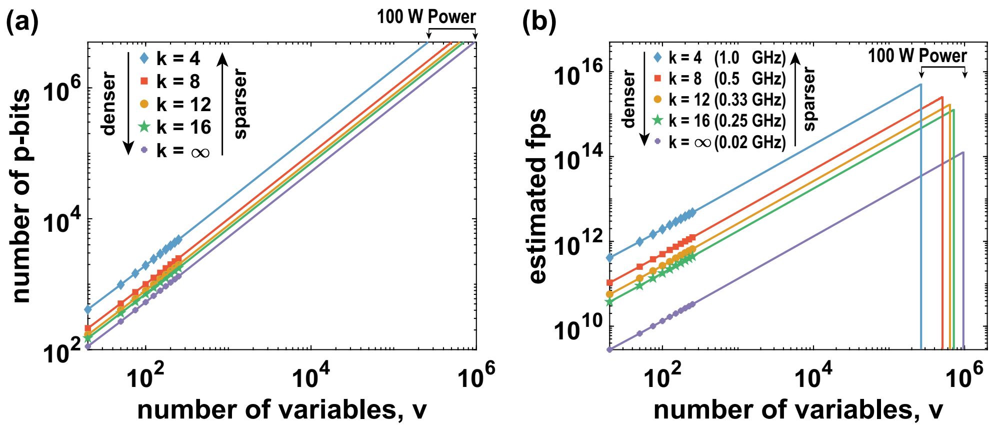

Fig. S6. Performance Projections. We consider projections up to a million p-bits for the 3SAT problem, assuming 20 μW per p-bit inspired by detailed device simulations [79]. (a) Number of p-bits as a function of problem size at different k values. (b) estimated fps as a function of problem size at different k values operating at different clock frequencies (inversely depending on k). For a given power budget of 100 W, the sparsest representation allows the fastest fps, however, for further scaling, denser graphs can be used with lower fps. k is defined as the maximum number of neighbors after sparsification. $k = \infty$ represents the original problem without sparsification.

Hence, a graph with a maximum of k neighbors will have maximum graph density (ρ_{max}):

$$\rho_{\text{max}} = \frac{|E_{\text{k-regular}}|}{|E_{\text{all-to-all}}|} = \frac{k}{|V|-1} \approx \frac{k}{|V|} \tag{S.16}$$

As Fig. S5 shows, for both the integer factorization and 3SAT instances, the graph density (Eq. (5)) as a function of problem size progressively *decreases* even without sparsification ($k = \infty$), hence they can be efficiently represented in a sparse, scalable hardware.

E. Scalability analysis

The proposed architecture has two main parts: p-bits and interconnections. Both p-bits and interconnections scale linearly in terms of resources. Each p-bit uses a fixed amount of resources (e.g., LUT, LFSR) which grows linearly with increasing number of p-bits [O(N)]. Similarly, the interconnections also grow linearly [O(N)], since we limit the maximum neighbors for a p-bit to a fixed number, k = 4, k = 8, etc.

Bits	Original Graph	Sparsification [This Work]	MGE (Chimera Graph) Fixed Spins $\approx$ 50000	MGE (King's Graph) Fixed Spins $\approx$ 50000	MGE (Grid Graph) Fixed Spins $\approx$ 50000
14	154 spins	2128 spins	1162 out of 50000 spins	1690 out of 50000 spins	Fails
16	200 spins	2128 spins	1465 out of 50000 spins	2734 out of 50000 spins	Fails
18	252 spins	2128 spins	1946 out of 50000 spins	3552 out of 50000 spins	Fails
20	310 spins	2128 spins	2888 out of 50000 spins	Fails	Fails
22	374 spins	2128 spins	3108 out of 50000 spins	Fails	Fails
24	444 spins	2128 spins	4766 out of 50000 spins	Fails	Fails
26	520 spins	2128 spins	5786 out of 50000 spins	Fails	Fails
28	602 spins	2128 spins	6017 out of 50000 spins	Fails	Fails
30	690 spins	2128 spins	8896 out of 50000 spins	Fails	Fails
32	784 spins	2128 spins	10320 out of 50000 spins	Fails	Fails

TABLE S2. Minor graph embedding (MGE) vs. invertible Boolean logic embedding for the integer factorization problem. We assume a fixed $\approx$ 50000-spin Chimera, King's and square grid topologies as target graphs for MGE. For the sparse Ising Machine, a fixed 2128-spin hardware can factor all integers up to 32-bits.

1. *Trade-off between resources and performance*

The optimum sparsity of a problem depends on a trade-off between resources (e.g, number of p-bits, interconnects) and performance (e.g., flips per second). The number of p-bits and the number of interconnects are limited by a given power and area budget. For example, Fig. S6 shows nanodevice (Magnetic Tunnel Junction) based projections for the 3SAT problem. Using MRAM technology we project that up to a million p-bits can be integrated within a power budget of 100 W since each p-bit dissipates around 20 μW based on detailed device simulations [79]. For further scaling, using denser graphs that can accommodate larger problems is possible, however, this approach shows diminishing returns beyond a point, where increasing the graph density does not help if the original problem (e.g., Boolean SAT) is already sparse (Fig. S6a).

While the sparsest representations allow the fastest flips per second (Fig. S6b), they lead to an increase in the number of p-bits. The operating clock frequency decreases linearly with the maximum number of neighbors, k since we assume the adder delay increases linearly as a function of k.

2. *Invertible Boolean logic vs. minor graph embedding*

Here, we report an illustrative comparison between invertible Boolean logic vs. minor graph embedding (MGE) applied to the integer factorization problem. For MGE, we use D-wave's minor-miner program [85] and assume that a fixed hardware with $\approx$ 50000 spins in Chimera, King's and square grid graph topologies needs to embed an original graph to factor different sizes of semiprimes, up to 32-bits.

For MGE, Table S2 shows that the Chimera graph requires $\approx$ 10000 spins to encode the 32-bit factorizer. The King's graph fails to encode beyond 18-bits and the grid graph always seems to fail. For the sparse Ising Machine, on the other hand, a 32-bit invertible multiplier can factor any number up to 32-bits. Only a sparsified graph with $k = 4$ having 2128 spins that can factor 32-bits is necessary and sufficient at all sizes.

F. Time to solution for exact factorization

We have reported exact factorization up to 32-bit semiprime numbers in Section VI. Here, we report the time to find the exact factors. An exponential fit with respect to the number of p-bits up to 5375 p-bits shows the difficulty in factoring numbers larger than 32-bit with the current annealing schedule (Fig. S7a). We describe this plot with the following equation:

$$t = t_0 \exp\left(\frac{N}{\tau}\right) \tag{S.17}$$

where N = number of p-bits, t = TTS, t_0 = pre-exponential factor, τ = time constant. From the fitted plot, we obtain $t_0 = 10^{-3.39}s$ and $\tau = 122.13$.

Another exponential fit with respect to the number of bits up to 50-bits is shown in Fig. S7b. We describe this plot with the following equation:

$$t = t_0 \exp\left(\frac{n}{\tau}\right) \tag{S.18}$$

where n = number of bits. From the fitted plot, we obtain $t_0 = 10^{-7.17}s$ and $\tau = 1.26$.

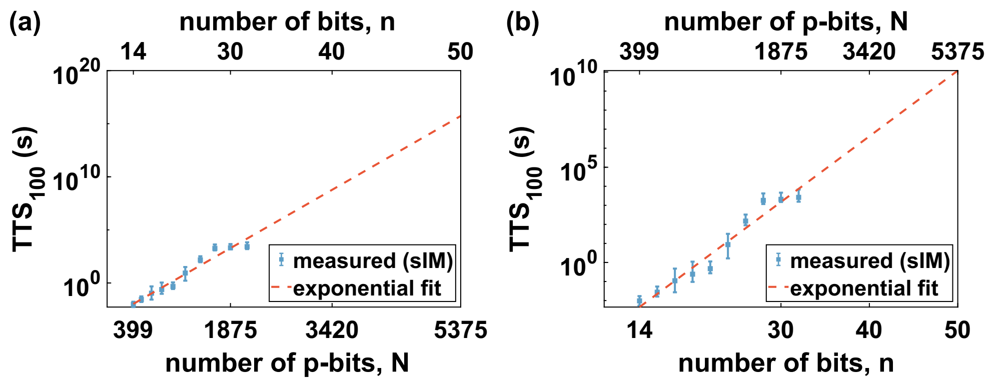

Fig. S7. Time to solution for exact factorization (TTS_{100}) of 14-bit to 32-bit semiprime numbers in the sIM and an exponential fit with respect to the (a) number of p-bits up to 5375 p-bits and (b) number of bits up to 50-bits.

G. Error models for inexact Gibbs sampling

In Section VIII of the main manuscript, we show how moderate overclocking leads to a decrease in the time to solution in two different problems. In order to analyze this phenomenon, we introduce two error models for inexact Gibbs sampling and apply them to systematically study a 5 p-bit full adder (FA) circuit. However, the conclusions and limits we obtain are generally applicable. The J and h matrices we used for the full adder are the following:

$$J_{FA} = \begin{pmatrix} 0 & -1 & -1 & +1 & +2 \\ -1 & 0 & -1 & +1 & +2 \\ -1 & -1 & 0 & +1 & +2 \\ +1 & +1 & +1 & 0 & -2 \\ +2 & +2 & +2 & -2 & 0 \end{pmatrix} \qquad h_{FA} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{S.19}$$

The two models are both based on the same principle: the J matrix is multiplied by an error mask matrix whose elements establish whether a neighbor connection is failing or not. If no nodes are failing, both mask models are a matrix of 1s.

In the first model we introduce, which we will refer to as the single mask model from here on, uses a mask of $+1$s and -1s to represent functioning and failing connections, respectively. For example, if a connection from node j to node i is failing due to overclocking, M^S_{ij} will be -1. Thus, Eq. (2) in the main text will be replaced by

$$I_i = \sum_j M^S_{ij} J_{ij} m_j + h_i \tag{S.20}$$

where M^S is the error mask for the single mask model. In the case considered, since J_{FA} is a 5×5 matrix and since the main diagonal elements cannot fail as a connection, the maximum number of errors is 20.

The second model, which we will refer to as the double mask model from here on, uses two complementary masks of 1s and 0s. In this case, Eq. (2) will be replaced by

$$I_i = \sum_j M^D_{ij} J_{ij} m_j^{\mathrm{new}} + \sum_j (1 - M^D_{ij}) J_{ij} m_j^{\mathrm{old}} + h_i \tag{S.21}$$

where m_j^{new} and m_j^{old} represent the updated and the non-updated values, respectively.

To test these models, we performed a systematic study where we considered every possible fraction of errors (defined as E_r) in the mask matrices (from $0/20$ to $20/20$). We simulated 400 random masks for each E_r taking 2×10^4 samples per mask. In Fig. S8, the average distributions resulting from all masks for each value of E_r are compared with the exact Boltzmann distribution. Both models exhibit qualitatively similar behavior. For a moderate number of errors the Kullback–Leibler (KL) divergence from the exact distribution does not increase significantly, but beyond a certain point errors diverge (Fig. S9). Eventually both models approach a distribution described by a parallelly updating network [64] whose steady-state is defined by the following equation:

$$p_k = p(m_1^{(k)}, m_2^{(k)}, \cdots, m_N^{(k)}) = \frac{1}{Z} \prod_{i=1}^{N} \cosh\left(\beta \left[\sum_j J_{ij} m_j^{(k)} + h_i\right]\right) \exp\left(\beta\, m_i^{(k)} h_i\right) \tag{S.22}$$

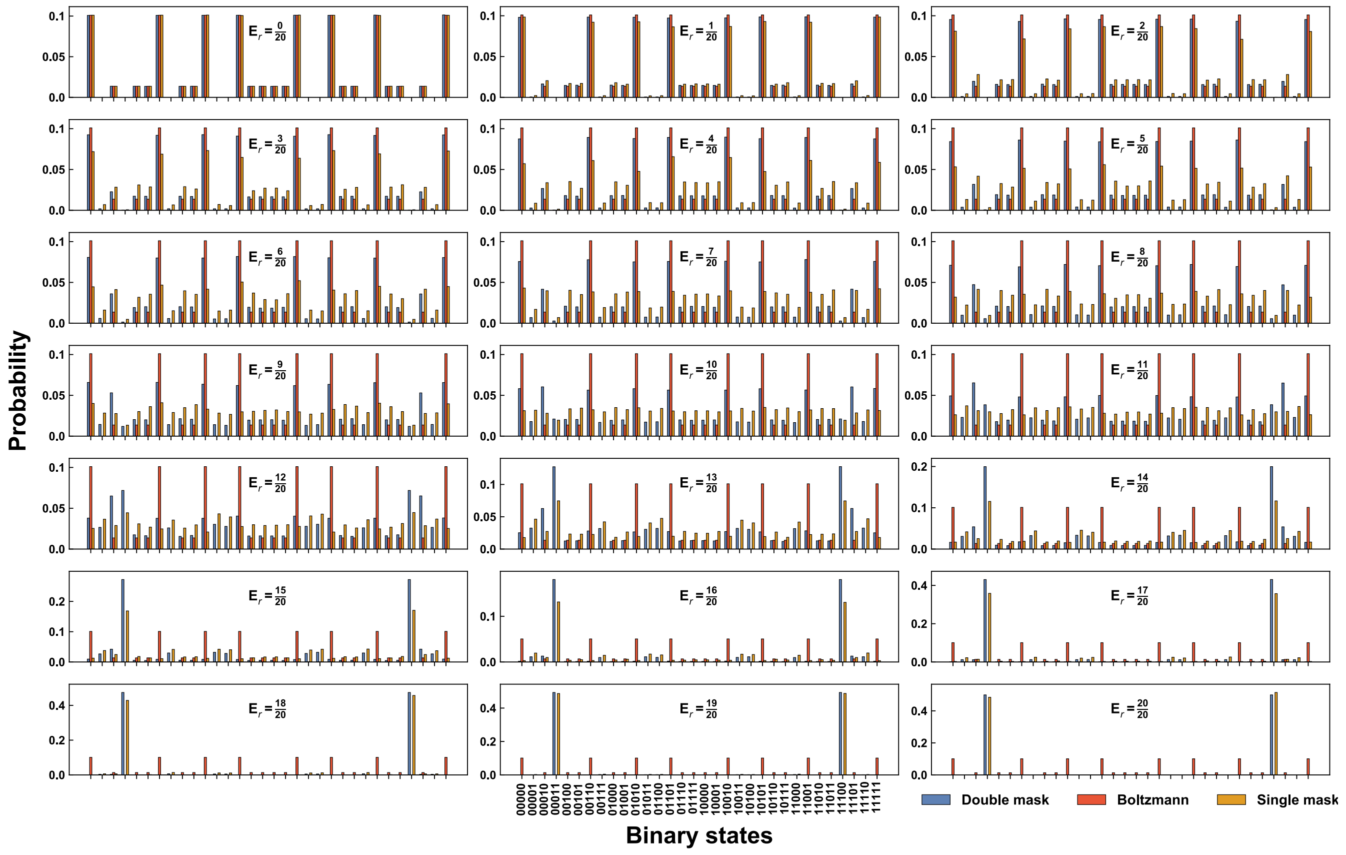

Fig. S8. Probability distributions of the single and double mask models vs. the exact Boltzmann distribution as a function of the fraction of errors E_r for a 5 p-bit full adder. Each plot is obtained by averaging over 400 random masks, each sampled 2×10^4 times with $\beta = 1$. Both models show relatively low deviations from the exact distribution at low values of E_r before dramatically worsening and converging to a parallel update distribution.

where k represents all possible states of $\mathbf{m}$ from $1, 2, \ldots 2^N$ and Z is a normalization constant ensuring probabilities add to 1.

The distribution defined by Eq. (S.22) quantitatively describes the steady-state distribution observed in Fig. S8 as E_r approaches 1, at which point the network updates are parallel.

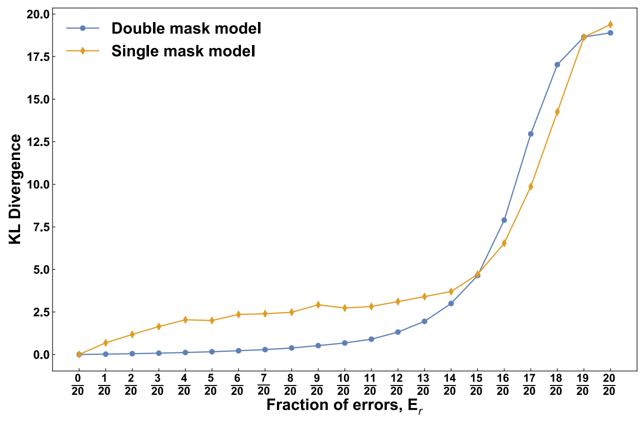

Fig. S9. Kullback–Leibler (KL) divergence of the distributions of the single and double mask models from the exact Boltzmann distribution as a function of the fraction of errors E_r. Both models exhibit (albeit with different degrees of accuracy to the real distribution) a flatline behavior with moderate values of E_r and a sharp increase in divergence when E_r reaches a certain threshold.