nature electronics

Supplementary information

https://doi.org/10.1038/s41928-024-01182-4

Training deep Boltzmann networks with sparse Ising machines

In the format provided by the authors and unedited

SUPPLEMENTARY INFORMATION

A. FPGA Implementation

We present the experimental results in the main paper using a hybrid classical-probabilistic computer. Here, we discuss the technical details of the FPGA-based p-computer implemented on a Xilinx Alveo U250 data center accelerator card. The basic architecture is presented in FIG. S1.

1. *p-bit and MAC Unit*

A single p-bit consists of a tanh lookup table (LUT) for the activation function, a pseudorandom number generator (PRNG), and a comparator to implement Eq. (4). Digital input with a fixed point precision (10 bits with 1 sign, 6 integers, and 3 fraction bits) provides tunability through the activation function. The effect of different fixed point precisions has been explored in Supplementary Section E. In this work, we used a high-quality 32-bit PRNG (xoshiro [1]). There is also a multiplier–accumulator (MAC) unit to compute Eq. (3) based on the neighbor p-bit states and provide input signal for the LUT as shown in FIG. S1a.

2. *Bipolar to binary conversion*

As described in the main article, Eq. (2)-(4) are represented with the bipolar state of the variables. However, for the FPGA-based implementation of p-computer, using a binary state of variables is more convenient since digital CMOS operates with gnd (0) and VDD (1) and Boolean logic can naturally represent the p-bit state. Therefore, bipolar to binary conversion is done on the weights and biases before being sent to the FPGA, according to the following mapping:

$$J_{\text{binary}} = 2J_{\text{bipolar}} \tag{S.1}$$

$$h_{\text{binary}} = h_{\text{bipolar}} - J_{\text{bipolar}}\,\mathbf{1} \tag{S.2}$$

where, $\mathbf{1}$ is a vector of ones of size $N \times 1$, and N is the number of p-bits. All the variables in the MAC unit are also calculated in binary notations. To convert the stored activation function values of LUT from bipolar to binary representation, the $\tanh(x)$ is mapped to $[1 + \tanh(x)]/2$. Finally, the output p-bit is represented with binary states $m_i \in \{0, 1\}$.

3. *Clocking unit*

In the main paper, we discussed graph coloring to color p-bit blocks achieving massive parallelism, which we define as the linear scaling of probabilistic flips/ns with respect to increasing graph size. To achieve this parallelism, we utilized multiple built-in clocks inside the FPGA board to drive the PRNGs within the p-bit blocks, as shown in FIG. S1a. The mixed-mode clock manager (MMCM) block, available in the Xilinx Alveo U250 data center accelerator card, generates equally phase-shifted and same-frequency parallel stable clocks. The input to the MMCM is a 300 MHz differential pair system clock created on a low-voltage differential signaling (LVDS) clock-capable pin (FIG. S1b). These clocks are highly precise with minimal noise or phase deviation. By triggering the colored p-bit blocks with these phase-shifted clocks, we achieve massive parallelism.

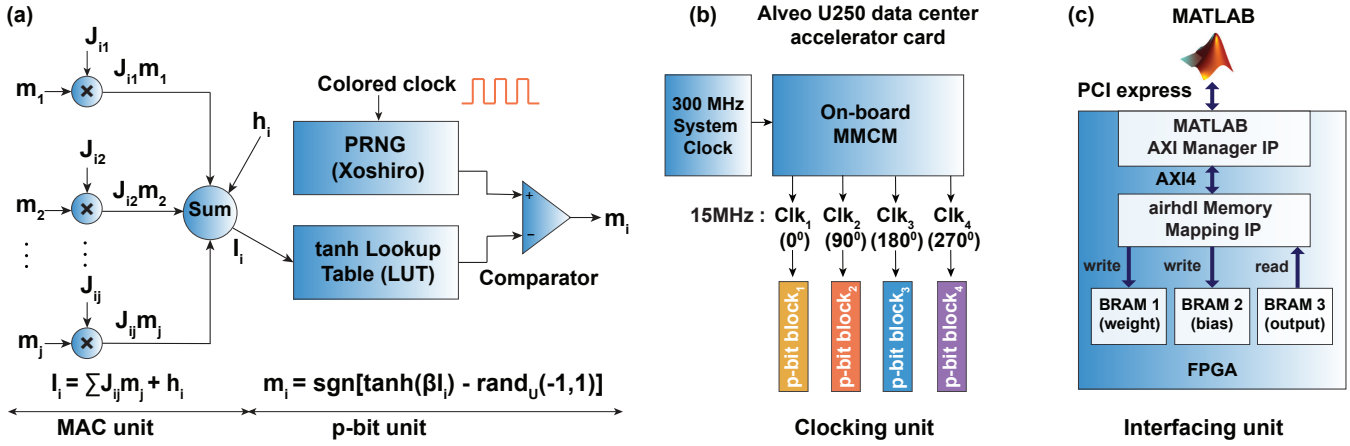

FIG. S1. (a) The MAC (multiplier–accumulator) unit implements Eq. (3). The p-bit unit consists of a xoshiro pseudorandom number generator (PRNG), a lookup table for the activation function (tanh), and a comparator to generate a binary output. (b) A built-in clocking unit generates equally phase-shifted and same-frequency parallel clocks to trigger the PRNGs inside the colored p-bit blocks. (c) A PCIe interfacing unit transfers data between MATLAB and the FPGA.

4. *Interfacing unit*

We use MATLAB as an Advanced eXtensible Interface (AXI) manager to communicate with the FPGA board through a PCI express interface (FIG. S1c). We have designed an AXI manager 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' [2] is used to manage the memory mapping of the registers into the block rams (BRAMs) inside the FPGA. We used two BRAMs to write the weights and biases from MATLAB and another BRAM to save the p-bit states that MATLAB reads out.

B. Readout architecture with mirror p-bit

FIG. S2 shows our readout architecture that can take a "snapshot" of the system at a given time. The nature of the learning algorithm is that only a handful of equilibrium samples are needed to estimate correlations and averages without requiring continuous samples. By way of mirror (or copy) p-bit registers, we decouple system p-bits (that are always on) from snapshot states that are saved to memory. In this way, while the system goes through a large number of samples, we are able to take a sufficient number of samples that are saved to local memory (at our own accord) which can then be read by a CPU.

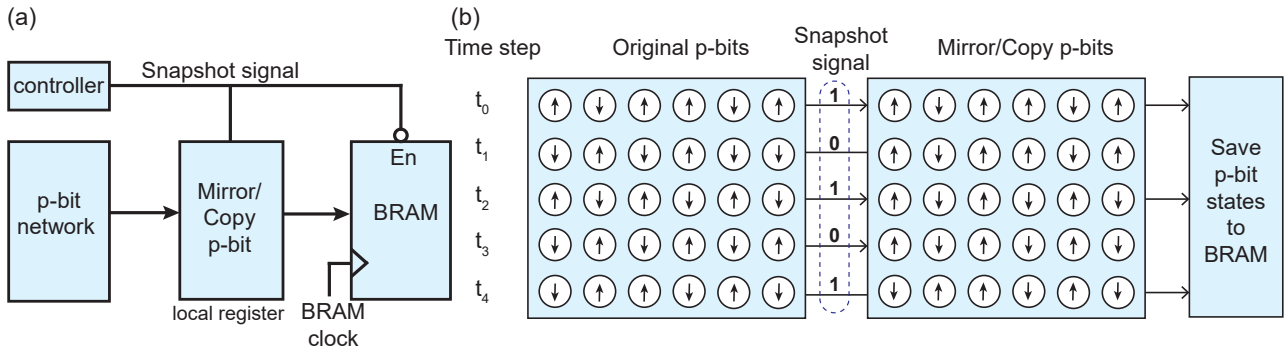

FIG. S2. (a) Block diagram showing the mirror (or copy) p-bit architecture with a snapshot signal. The controller block generates the snapshot signal at which time the original p-bit states are copied to local memory (registers) and at the inverted snapshot signal those states are saved into block memory (BRAM), only once. Subsequent zero signals from the snapshot signal do nothing to mirror/copy p-bits or to BRAM. (b) Conceptual diagram to visualize the operation of the snapshot signal.

C. Graph density of sparse DBMs

The networks we present in the manuscript (Pegasus 4,264 p-bits and Zephyr 3,360 p-bits) are highly sparse, which we quantitatively show in FIG. S3. We use the typical graph density metric, measured by

$$\text{graph density} = \frac{\text{number of edges in the network}}{\text{number of edges in a fully-connected network}} \times 100 \tag{S.3}$$

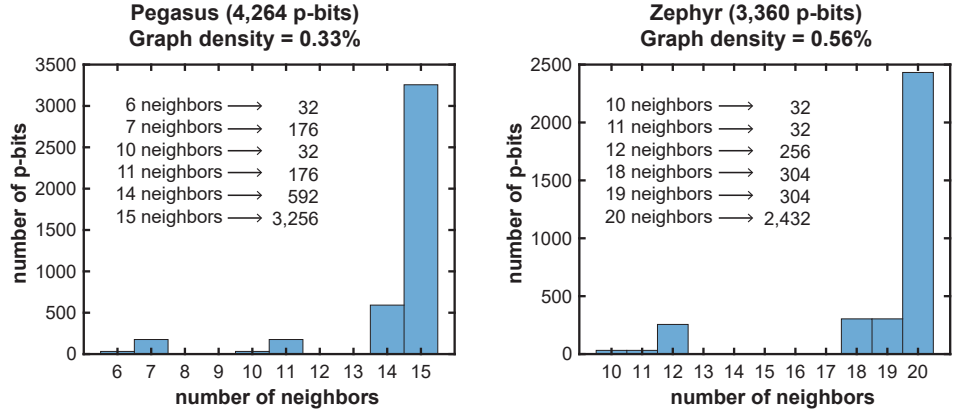

FIG. S3. The graph density and the neighbor distribution of sparse DBMs (Pegasus and Zephyr) where graph density, $\rho = 2|E|/(|V|^2 - |V|)$ where $|E|$ = the number of edges and $|V|$ = the number of vertices in the graph. The density of Pegasus 4,264 p-bits is 0.33% and 3,256 p-bits have the maximum number of neighbors 15. Zephyr (3,360 p-bits) has a density of 0.56% and 2,432 p-bits have the 20 maximum neighbors.

By this metric, the network we have shown in FIG. 1e (Pegasus, 4264) only has a graph density of 0.33%. Moreover, we also show a vertex degree distribution of the network providing a histogram of nodes and the number of their neighbors. On the right, the same metrics are shown for the Zepyhr graphs with similar results.

D. Training accuracy of MNIST/100

We have studied the effect of training the sparse DBMs (Pegasus 4,264 p-bits) with a small subset of data before training the full MNIST. In this setting, we chose 100 images from MNIST to train on the sparse network with our massively parallel architecture. To train these 100 images, we used 10 mini-batches, having 10 images in each batch and the same set of hyperparameters as in training full MNIST. The training accuracy reached 100% within 1,000 epochs as illustrated in FIG. S4. We also explored different values of the 'regularization' parameter ($\lambda = 0.005, 0.001$) which is generally used to keep the weights from growing too much. In our case of sparse DBM, we did not observe any significant difference between a small regularization and without any regularization. The poor test accuracy here is an indication of overfitting due to the small size of the training set (only 100 images). We observed similar accuracy on Zephyr graphs with 3,360 p-bits for 100 images.

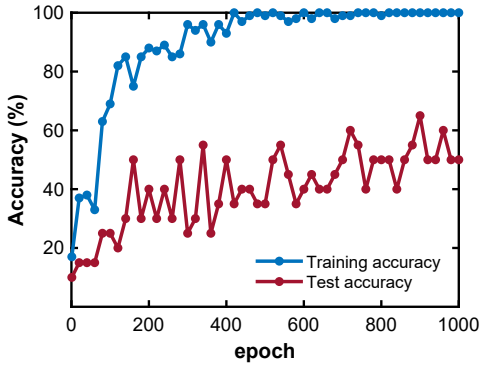

FIG. S4. Accuracy of training 100 images with sparse DBMs up to 1,000 epochs. Training is accomplished with 10 mini-batches and CD-10^5. All the trained 100 images and 20 unseen images have been tested here.

E. Effect of weight precision

The results we have in the main paper utilized a weight precision of 10 bits (1 sign, 6 integer, and 3 fraction bits i.e., s$\{6\}\{3\}$). Here, we explore different weight precisions by changing the fraction bit width and compare the results to identify the effect of weight precision on accuracy. We trained full MNIST with 1,200 mini-batches for 200 epochs, using five different weight precisions of s$\{6\}\{2\}$, s$\{6\}\{3\}$, s$\{6\}\{5\}$, s$\{4\}\{2\}$ and s$\{3\}\{2\}$. We chose a Pegasus 2,560 p-bit graph as a sparse DBM for this experiment that fits into the FPGA since increasing bit precision reduces the available resources.

The weight update is accomplished in MATLAB (double-precision floating-point with 64 bits), but before the new weights are loaded to the FPGA, they are converted to the corresponding fixed-point precision. The choice of hyperparameters remains the same for all cases. The test accuracy goes to $\approx 88\%$ in each case (with 17,984 parameters) and there is no remarkable difference among the accuracy of the different weight precisions between s$\{6\}\{5\}$ to s$\{6\}\{3\}$, accuracy starts degrading at or below s$\{4\}\{2\}$ (FIG. S5a). We also trained full MNIST on RBM (512 hidden units) using both float64 and s$\{6\}\{3\}$ weight precision for 200 epochs. The test accuracy remains the same for these two different precisions as shown in FIG. S5b.

To further study the impact of weight precision on the *generative* properties of the sparse DBM network, we have conducted image completion experiments. FIG. S5c shows inference experiments where we obscure half of an image and let a trained network evolve to the corresponding minima (by annealing the network from $\beta = 0$ to $\beta = 5$). We observe that while s$\{6\}\{3\}$ can complete this task, precisions below s$\{4\}\{2\}$ start failing.

F. Mixing times in Pegasus graph (3,080 p-bits)

We described the mixing times in the main paper showing the results (FIG. 4) from our largest size Pegasus (4,264 p-bits). Here, we show another graph, Pegasus with 3,080 p-bits to measure the mixing time of the network. Unlike the main model, for this experiment, we trained full MNIST for only 50 epochs (instead of 100) using the same hyperparameters as mentioned in the main article with different numbers of sweeps starting from CD-10^2 to CD-10^6. Test accuracy improves significantly when we take more than CD-10^4 per epoch.

G. Image generation with Zephyr

In the main paper, FIG. 1f displays the images generated with Pegasus (4,264 p-bits) graph and the procedure is described in Section VI. Here we explored image generation with a different type of sparse DBM, Zephyr (3,360 p-bits) that also reached

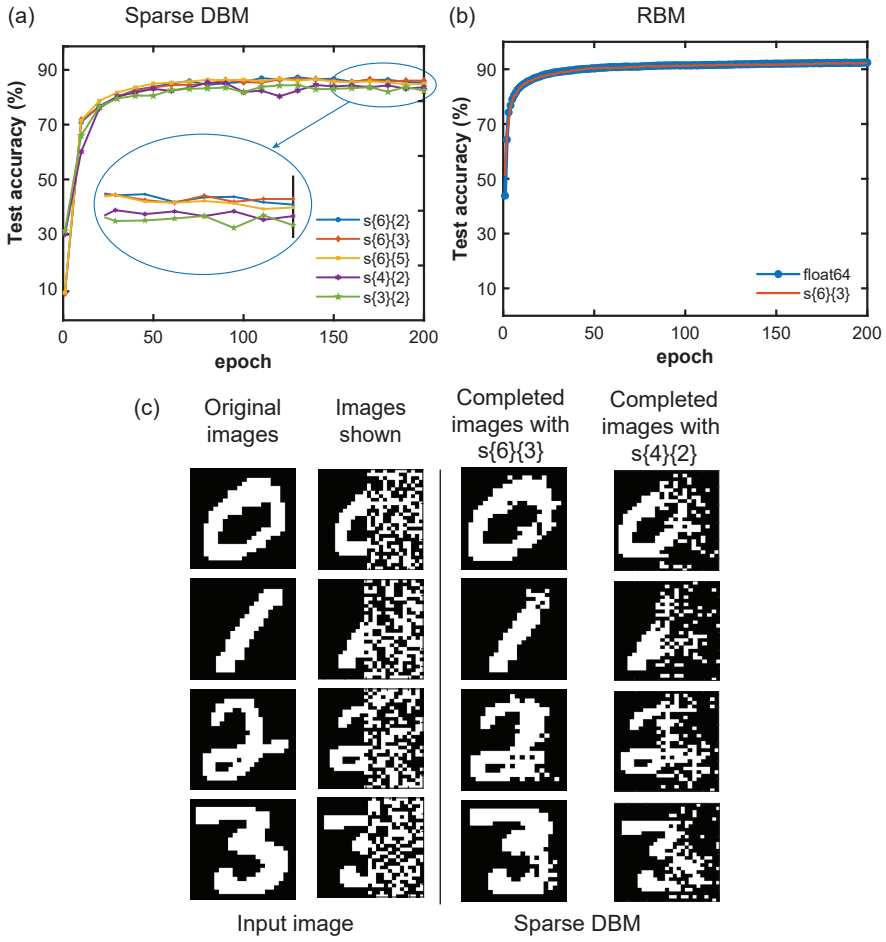

FIG. S5. (a) Test accuracy of full MNIST with sparse DBMs (Pegasus 2,560 p-bits) up to 200 epochs with five different fixed point precisions of weights (s{6}{2}, s{6}{3}, s{6}{5}, s{4}{2} and s{3}{2}). (b) Test accuracy of full MNIST for RBM (512 hidden units) with double-precision floating point 64 bits and s{6}{3}. (c) Image completion examples with sparse DBMs for fixed point precisions of weights s{6}{3} and s{4}{2} (annealing schedule varies from $\beta = 0$ to $\beta = 5$ with 0.125 steps). With s{6}{3} precision, the network can complete the images where the right half of the image starts from random noise. Below s{4}{2}, the network fails to complete the images.

$\approx 90\%$ accuracy with randomized indices as demonstrated in the main FIG. 5c (bottom). The generated images with Zephyr as shown in FIG. S7 are slightly different from the Pegasus ones.

H. Momentum to the learning rule

In our training, we used the momentum in our update rules, which are empirically added to the learning rule we discuss in the next section. By retaining a portion of the last update to the weights, momentum helps increase the effective learning rate [3]. The effective increase in the learning rate is equivalent to multiplying it by a factor of 1/(1-α) where α is denoted as momentum. Using this process, the algorithm can increase the effective learning rate without causing unstable oscillations, which ultimately speeds up the convergence of the training process [4]. We modify the learning rule equations in the main Eq. (5) and Eq. (6) by introducing the momentum term as follows:

$$\Delta J_{ij}(n) = \varepsilon\Big(\langle m_i m_j\rangle_{\text{data}} - \langle m_i m_j\rangle_{\text{model}}\Big) + \alpha\Delta J_{ij}(n-1) \tag{S.4}$$

$$\Delta h_i(n) = \varepsilon\Big(\langle m_i\rangle_{\text{data}} - \langle m_i\rangle_{\text{model}}\Big) + \alpha\Delta h_i(n-1) \tag{S.5}$$

where n represents the j^{th} index (ranging from 1 to the number of batches) in the Extended Data Fig.1.

I. Maximum Likelihood for Boltzmann Networks

The basic idea of Boltzmann networks is to start from a physics-inspired variational guess, that the data distribution will be approximated by a model whose probability p^M for a given input vector $m^{(i)}$ (i, being the input index) obeys the Boltzmann

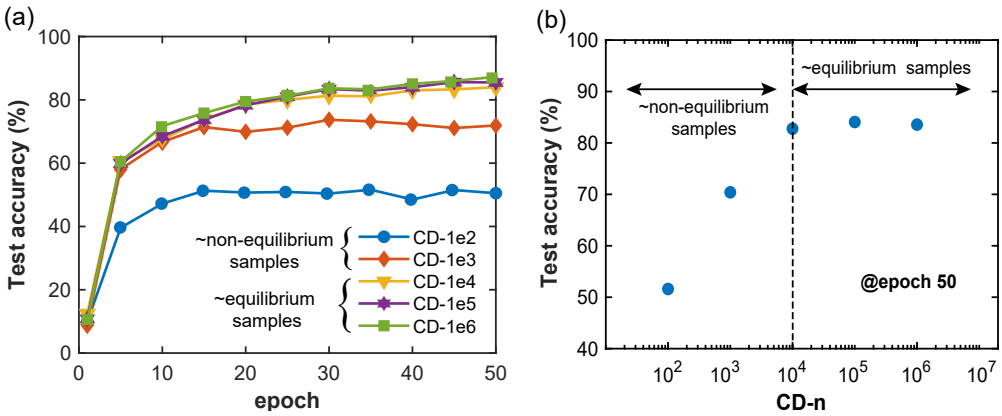

FIG. S6. (a) Test accuracy after training full MNIST up to 50 epochs with different numbers of sweeps using sparse DBMs (Pegasus 3,080 p-bits). For our sparse graph, to mix the Markov chain properly we need minimum CD-10^4. Reducing the number of sweeps significantly degrades the quality of mixing in the chain. (b) Test accuracy as a function of CD-n at epoch 50 showing the equilibrium and non-equilibrium samples.

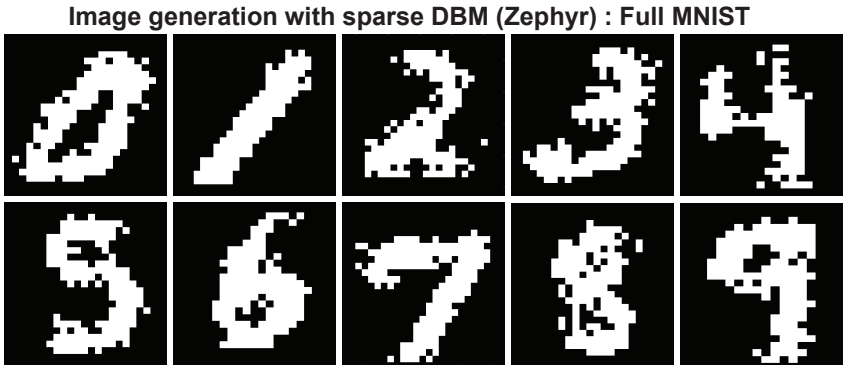

FIG. S7. Image generation examples with sparse DBM Zephyr (3,360 p-bits) after training the network with the full MNIST dataset.

law (ignoring biases in our derivation for simplicity):

$$p^M\left(m^{(i)}\right) = \frac{1}{Z}\exp\left[\sum_{er} J_{er} m_e^{(i)} m_r^{(i)}\right] \tag{S.6}$$

In our setting, we have a system of M fully visible p-bits connected in some arbitrary graph topology. The problem is learning a "truth table" with exactly N_T lines of inputs in it. The model is going to try to select these N_T states in the space of 2^M possible discrete probabilities. Like in any other ML model, fitting every line of the truth table exactly will overfit, but the Boltzmann formulation given by Eq. (S.6) smooths out the sum of "delta function"-like data vectors in the 2^M space, which can later be used for generating new samples.

We define a p^V as the probability distribution of the data, corresponding to the visible bits. Then, a maximum likelihood estimation minimizing the Kullback–Leibler divergence between the data and the model can be used to derive the learning rule, by taking the negative derivative of $KL(p^V \| \, p^M)$:

$$KL(p^V \| \, p^M) = \sum_{i=1}^{N_T} p_i^V \log\left[\frac{p_i^V}{p_i^M}\right] \tag{S.7}$$

where i is the index of truth table lines N_T. To simplify the analysis, we consider fully visible networks where p_i^V is independent of J_{mn} for any network topology since p_i^V represents the data distribution.

$$\frac{\partial KL\left(p^V \| \, p^M\right)}{\partial J_{mn}} = \frac{\partial p_i^V}{\partial J_{mn}} \log\left[\frac{p_i^V}{p_i^M}\right] = \sum_{i=1}^{NT} \frac{p_i^V}{p_i^M}\left(\frac{-\partial p_i^M}{\partial J_{mn}}\right) \tag{S.8}$$

$$\frac{\partial p_i^M}{\partial J_{mn}} = \frac{\partial}{\partial J_{mn}} \left[\frac{1}{Z} \exp \left[\sum_{er} J_{er} m_e^{(i)} m_r^{(i)} \right] \right] \tag{S.9}$$

$$Z = \sum_{j=1}^{2^N} \exp \left[\sum_{er} J_{er} m_e^{(j)} m_r^{(j)} \right] \tag{S.10}$$

where the index j represents all possible states from 1 to 2^M for the model.

$$\begin{aligned} \frac{\partial p_i^M}{\partial J_{mn}} &= \frac{-1}{Z^2} \cdot \frac{\partial Z}{\partial J_{mn}} \cdot \left[\exp \sum_{er} J_{er} m_e^{(i)} m_r^{(i)} \right] + \frac{1}{Z} \cdot \frac{\partial}{\partial J_{mn}} \left[\exp \sum_{er} J_{er} m_e^{(i)} m_r^{(i)} \right] \\ &= \frac{-\partial Z}{\partial J_{mn}} \cdot \frac{1}{Z} \cdot p_i^M + m_m^{(i)} \cdot m_n^{(i)} \cdot p_i^M \\ &= - \sum_{j=0}^{2^N-1} m_m^{(j)} m_n^{(j)} \cdot p_j^M \cdot p_i^M + m_m^{(i)} m_n^{(i)} \cdot p_i^M \end{aligned} \tag{S.11}$$

$$\begin{aligned} -\frac{\partial KL\left(p^V \| \, p^M\right)}{\partial J_{mn}} &= \sum_{i=1}^{N_T} \frac{p_i^V}{p_i^M} \cdot \frac{\partial p_i^M}{\partial J_{mn}} \\ &= \sum_{i=1}^{N_T} \frac{p_i^V}{{p_i}^M} \left[m_m^{(i)} \cdot m_n^{(i)} \cdot p_i^M - \sum_{j=0}^{2^M-1} m_m^{(j)} \cdot m_n^{(j)} \cdot p_j^M \cdot p_i^M \right] \\ &= \left[\sum_{i=1}^{N_T} p_i^V \cdot m_m^{(i)} m_n^{(i)} - \sum_{i=1}^{N_T} p_i^V \sum_{j=0}^{2^M-1} m_m^{(j)} \cdot m_n^{(j)} \cdot p_j^M \right] \\ &= \left[\langle m_m m_n \rangle_{\text{data}} - \langle m_m m_n \rangle_{\text{model}} \right] \end{aligned} \tag{S.12}$$

which gives the familiar learning rule. A similar learning rule in terms of the averages can be derived by accounting for the biases in the energy, which we ignored for simplicity.

J. Comparison with state-of-the-art GPUs and TPUs

In the main paper, we show how graph-colored FPGA achieves massive parallelism to provide a few orders of magnitude faster sampling throughput than traditional CPUs. Here in Supplementary Table S1, we also compare the sampling speed to some state-of-the-art (SOTA) Ising machines implemented on the latest GPUs and TPUs. The throughput reported in this work up to 64 billion flips per second outperforms the numbers reported by the SOTA Ising solvers in GPUs and TPUs. It is also important that this comparison is not completely accurate and favors the GPU/TPU implementations for two reasons: First, all the GPUs and TPUs discussed here are solving simple, nearest-neighbor chessboard lattices, unlike the irregular and relatively high-degree (with up to 20 neighbors) graphs used in this work. Second, GPU/TPU implementations generally use low precision {+1,-1} weights (compared to 10 bits of weight precision in our work) and thus can explore only a few discrete energy levels. Both of these features are heavily exploited in reporting a large degree of flips/ns in these solvers and their performance would presumably be much worse if they were implemented in the same graphs with the same precision we discuss in this work.

TABLE S1. Optimized GPU and TPU implementations of Markov Chain Monte Carlo sampling with regular chessboard lattices. It is important to note that these TPU and GPU implementations solve Ising problems in sparse graphs, however, their graph degrees are usually restricted to 4 or 6, unlike more irregular and higher degree graphs.

Sampling method	**topology**	**max. degree**	**flips/ns**
Nvidia Tesla C1060 GPU [5, 6]	Chessboard	4	7.98
Nvidia Tesla V100 GPU [7]	Chessboard	4	11.37
Google TPU [7]	Chessboard	4	12.88
Nvidia Fermi GPU [8]	Chessboard	4	29.85

K. Image completion with DBM and RBM

A typical task for energy-based generative models is that of image completion as opposed to image generation. Image completion is relatively easier since the network is clamped near local minima. In the main manuscript, we show how an iso-

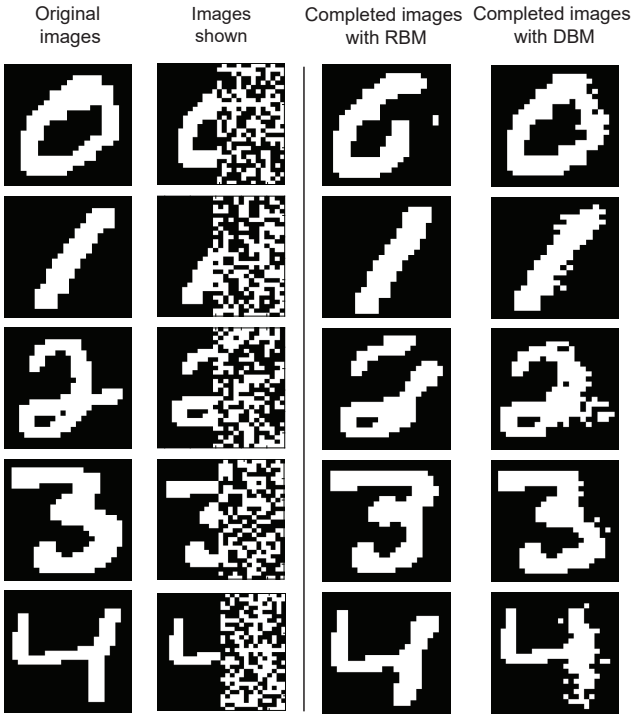

FIG. S8. Image completion examples with RBM (4,096 hidden units and CD-100) and sparse DBM (Pegasus 4,264 p-bits). Only the left half of the images is shown (clamped) to the networks while the other half is obscured. The label bits are also clamped and the annealing schedule varies from $\beta = 0$ to $\beta = 5$ with 0.125 steps.

accuracy RBM with around 3M parameters cannot perform image generation. Here, we show that an RBM can complete the easier task of image completion. We clamp half of the visible bits along with the labels and clamp the other half to noise. Then we let the trained network evolve to the corresponding minima by annealing the network from $\beta = 0$ to $\beta = 5$. Results are shown for a sparse DBM (4,264 p-bits) and RBM (4,096 hidden units, trained with CD-100). We observe that despite failing at image generation, the RBM performs similarly to our sparse DBM in image completion as shown in FIG. S8.

L. Quality of parallel samples

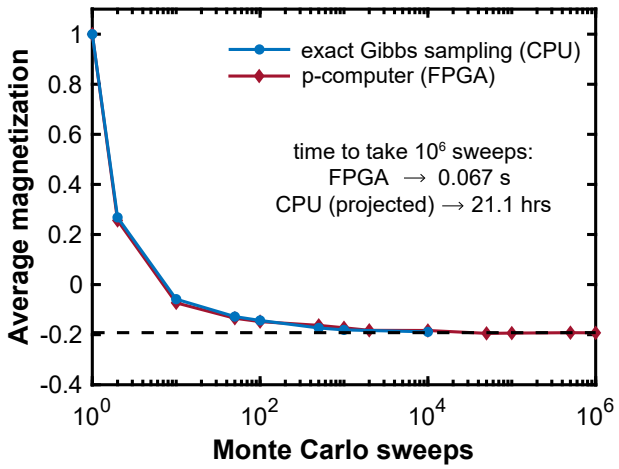

FIG. S9. Average magnetization as a function of Monte Carlo sweeps in a CPU (exact Gibbs sampling) vs. parallelized FPGA. See text for details.

To explicitly show the quality of our parallel samples in our graph-colored architecture (in the main Section 'p-Computer architecture'), we have performed the following "inference" experiment in the CPU performing exact Gibbs sampling vs. our parallelized FPGA using a sparse DBM (Pegasus 3,080 p-bits):

- We start with an MNIST-*trained* Pegasus network with (3080 p-bits) with known weights and biases.
- We initialize all p-bits to the +1 state at time step 1 and define a "network magnetization", $\langle m \rangle_k = \sum_i^N (m_i)/N$
- We perform Exact (sequential) Gibbs Sampling in a CPU and our parallelized Gibbs Sampling in the FPGA for M=100 times, measuring $\langle m \rangle_k$ for each run, $k \in 1, \ldots, M$. We obtain an *ensemble-averaged* $\langle m \rangle$.
- We then compare these averaged magnetizations, *as a function Monte Carlo sweeps* taken in the FPGA and CPU.

FIG. S9 shows the results of this experiment showing near identical relaxation between the FPGA and the CPU. FPGA takes about 0.067 seconds to take a million samples as opposed to a projected 21.1 hours from the CPU (we did not take more than 10,000 samples over 100 iterations in the CPU, since at that point both models converged). These numbers are in accordance with our expectations from their relative flips/second numbers and they establish that the samples taken by the FPGA follow the Gibbs sampling process.

M. Grayscale training of full Fashion MNIST

To test the differences between sparse DBMs and RBMs, we used our largest network (Pegasus 4264 p-bits) to train full Fashion MNIST [9], a more challenging dataset than MNIST [10]. Fashion MNIST consists of 28×28 grayscale images of 70,000 fashion products (60,000 in the training set and 10,000 in the test set) from 10 categories (e.g. t-shirt/top, trouser, sneaker, bag, pullover, and others), with 7,000 images per category. We have trained this dataset using our sparse DBM on the Pegasus graph. There are 4264 p-bits with 30,404 parameters in this network where the number of visible units is 784, 50 label units, and 3430 hidden units.

Our approach to grayscale images is based on time-averaging inspired by the stochastic computing approach of Ref. [11], where grayscale images between 0 and 1 are treated as the time-averaged probability of activation for p-bits. During the positive phase, we choose N (e.g., 20, 50, 100) binary samples from this probability and clamp the visible nodes as described in the Extended Data FIG. 1 and Section V. Here, we used 1200 mini-batches containing 50 grayscale images in each batch during training. To train Fashion MNIST, we used 20 binary (black and white) samples for each grayscale image resulting in a total of 60,000×20 training images. We found that the number of black and white samples can vary depending on the dataset as a hyperparameter.

To test classification accuracy, we also used 20 black and white samples for each grayscale image to perform the same softmax classification as described in the main article. After the network sees those 20 black and white samples to form a grayscale image, we check the labels to establish classification accuracy. Using this scheme of grayscale images, our sparse DBM with 30,404 parameters can reach around 80% in 120 epochs as shown in FIG. S10a. We trained RBMs with different numbers of parameters as listed in Table S2 using the same approach as sparse DBM. The iso-parameter RBM (43 hidden units and 34k parameters) can reach a maximum of 72% accuracy on full Fashion MNIST while the million-parameter RBMs can go to around 85% test accuracy. As in MNIST, we see that RBM requires the order of a million parameters to reach sparse DBM accuracy.

Using a similar approach to image generation as described in the main article, we can also generate images of fashion products with our sparse DBM as shown in FIG. S10b. Similar to other cases, for image generation, we only clamp label bits for a

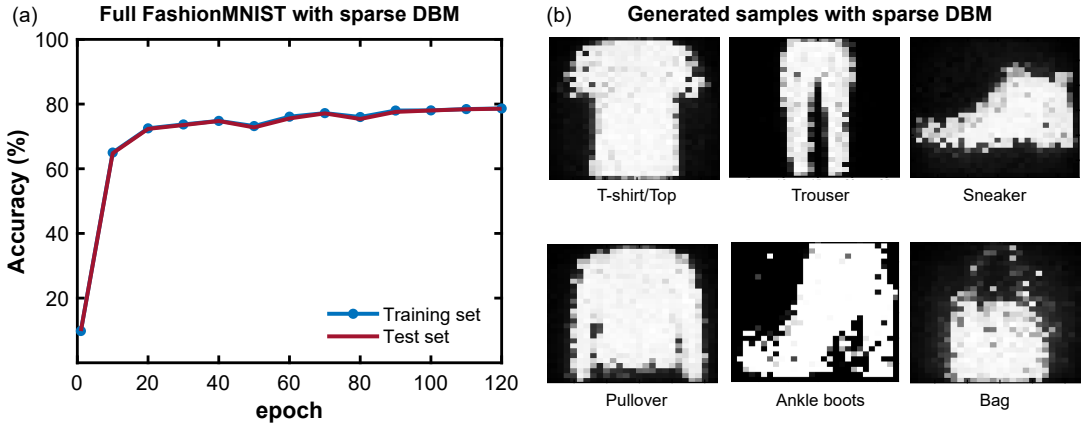

FIG. S10. (a) Fashion MNIST accuracy can reach around 80% in 120 epochs with sparse DBM. Full Fashion MNIST (60,000 images) with 20 black and white samples per grayscale image is trained on sparse DBM (Pegasus 4,264 p-bits and 30,404 parameters) using CD-10^5, batch size = 50, learning rate = 0.003, momentum = 0.6 and epoch = 120. Test accuracy shows the accuracy of the whole test set (10,000 images) and the training accuracy represents the accuracy of 10,000 images randomly chosen from the training set. (b) Image generation examples with sparse DBM after training the network with full Fashion MNIST dataset by annealing the network from $\beta = 0$ to $\beta = 1$ with 0.125 steps.

TABLE S2. Fashion MNIST accuracy with different sizes of RBMs.

Number of hidden units	number of parameters	maximum accuracy (%)
43	34×10^3	71.90
64	50×10^3	76.56
128	101×10^3	77.45
256	203×10^3	78.45
1264	1×10^6	85.56
2048	2×10^6	84.72
4096	3.25×10^6	82.64

particular image. We anneal the network slowly from $\beta = 0$ to $\beta = 5$ with a 0.125 increment using the final weights and biases. Then we check the 784 visible p-bits after time-averaging the collected samples to obtain grayscale images. Using a similar procedure, we observe that none of the RBMs can generate images despite having a maximum of 85% accuracy. We observed that different annealing schedules (e.g., with slower β changes) do not help RBM image generation.

N. Grayscale CIFAR-10/100

To see whether the same conclusions hold for our architectural and algorithmic ideas for sparse DBMs, we trained 100 images (10 images from each class) from the CIFAR-10 dataset [12]. Due to resource limitations in our FPGA, we could not increase the number of parameters, hence we used Pegasus 4,264 p-bits as sparse DBM to train the grayscale CIFAR-10/100. The CIFAR-10 dataset consists of 32×32 color images of 10 different classes (i.e., airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), with 6000 images per class. There are 50000 training images and 10000 test images.

We converted the color images into grayscale [13] using the 'rgb2gray' function in MATLAB. We used 1024 visible units, 5 sets of labels (50 p-bits), and 3190 hidden units arranged in 2 layers as shown in the inset of FIG. S11a. Utilizing the same approach of binary transformation from grayscale images as described in Section M, we have trained the CIFAR-10/100 dataset with 100 black and white samples per grayscale image with 10 mini-batches (having randomly selected 10 grayscale images in each batch). Training accuracy of this dataset with sparse DBM can reach around 90% in 2000 epochs as shown in FIG. S11a while the iso-parameter RBM (40 hidden units) accuracy is only 68% as listed in Table S3. RBMs reach the same levels of accuracy using between 264,000 and 1 million parameters, in line with our earlier results.

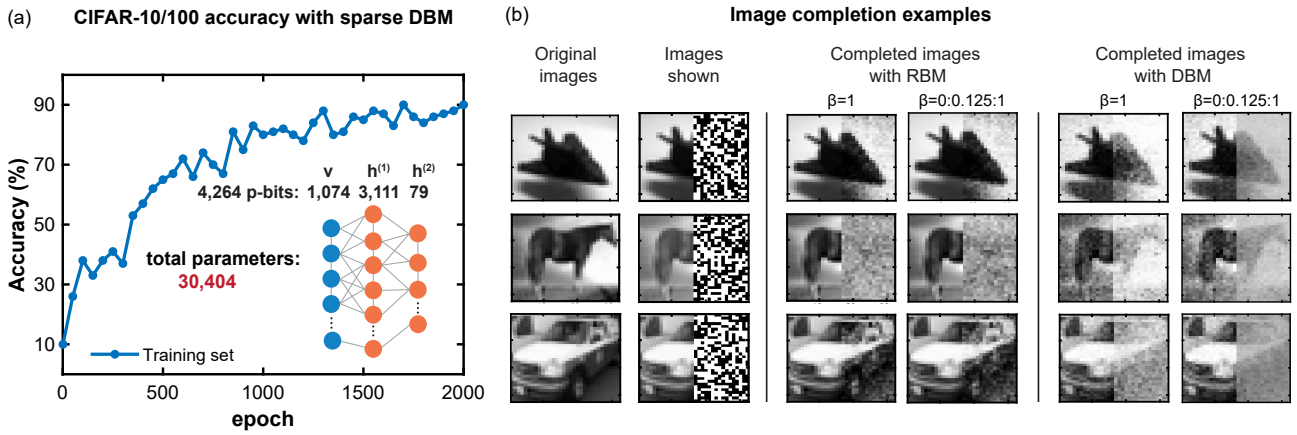

FIG. S11. (a) Training accuracy of 100 images from CIFAR-10 is around 90% in 2000 epochs with sparse DBM. Training is accomplished with 100 black and white samples per grayscale image using CD-10^5, batch size = 10, learning rate = 0.006, momentum = 0.6, and epoch = 2000. (b) Image completion examples with RBM (4096 hidden units) and sparse DBM. Only the left half of the grayscale images are shown (clamped) to the networks (using 100 black and white samples) while the other half is obscured. The label bits are also clamped and the annealing schedule varies from $\beta = 0$ to $\beta = 5$ with 0.125 steps, and for the other case β is kept to 1.

Image generation for CIFAR-10 in this reduced setting with only 100 images in the training set failed both for sparse DBM and RBM. For this reason, we also examined the image completion task (details in Section K) with RBM and sparse DBM as shown in FIG. S11b. We clamped only the left half of a grayscale image (using 100 black and white samples) along with the corresponding label bits and checked the right half of that image. In this case, both RBM (4096 hidden units) and sparse DBM performed similarly, in this much harder setting.

TABLE S3. CIFAR-10/100 accuracy with different sizes of RBMs.

Number of hidden units	number of parameters	maximum accuracy (%)
40	41×10^3	68
64	66×10^3	83
128	132×10^3	88
256	264×10^3	88
968	1×10^6	99
2048	2×10^6	100
4096	4×10^6	100

O. Full graph topologies: Pegasus 4,264

Below we show the full Pegasus network topology with 4,264 p-bits and its sparse deep BM representation.

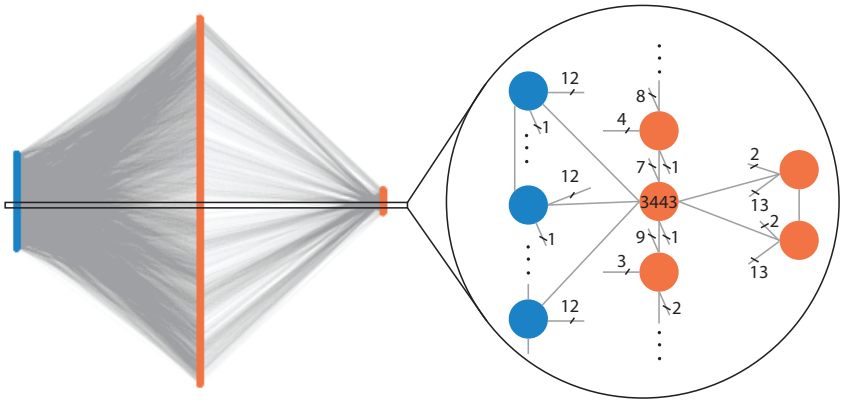

FIG. S12. Layered embedding of the 4,264 p-bit Pegasus graph of FIG. S13, illustrating the sparse DBM architecture: the first layer is visible p-bits with 834 nodes, second and third layers are the hidden p-bits with 3,226 and 204 nodes respectively. There are also some intralayer connections within each layer. An example is shown in the right circle which shows the neighboring connections around node 3,443. The number next to a line represents the number of wires grouped in that branch, the total number being the fan-out of a given p-bit (vertex).

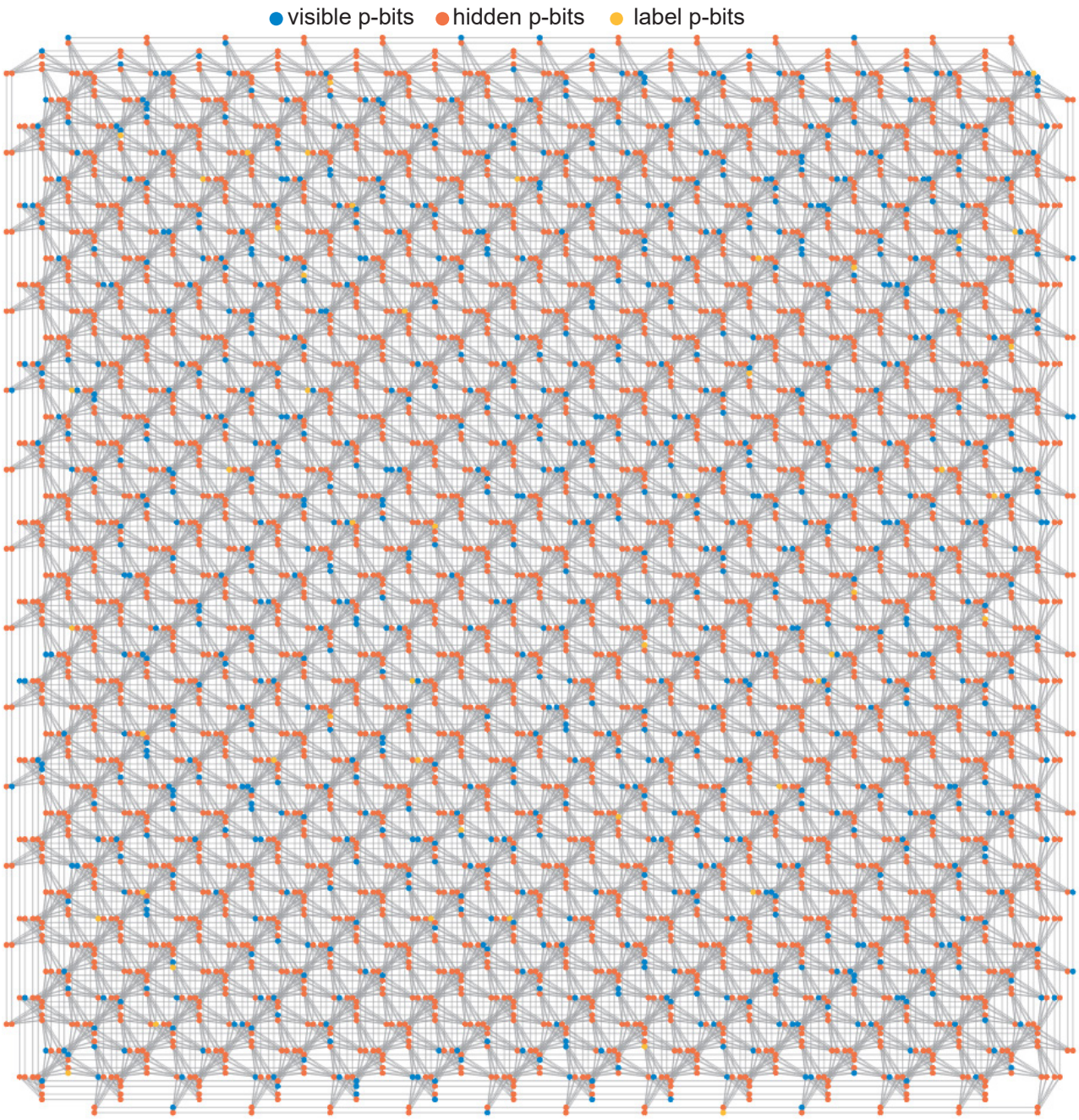

FIG. S13. The original sparse DBM network (Pegasus: 4,264 p-bits) used in this work with marked-up visible (blue), hidden (orange), and label (yellow) units.

REFERENCES

[1] D. Blackman and S. Vigna, Scrambled linear pseudorandom number generators, ACM Transactions on Mathematical Software (TOMS) **47**, 1 (2021).

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

[3] G. E. Hinton, A Practical Guide to Training Restricted Boltzmann Machines, in *Neural Networks: Tricks of the Trade: Second Edition*, edited by G. Montavon, G. B. Orr, and K.-R. Müller (Springer Berlin Heidelberg, Berlin, Heidelberg, 2012) pp. 599–619.

[4] T. Tieleman, Training restricted Boltzmann machines using approximations to the likelihood gradient, in *Proceedings of the 25th international conference on Machine learning* (2008) pp. 1064–1071.

[5] B. Block, P. Virnau, and T. Preis, Multi-GPU accelerated multi-spin Monte Carlo simulations of the 2D Ising model, Computer Physics Communications **181**, 1549 (2010).

[6] T. Preis, P. Virnau, W. Paul, and J. J. Schneider, GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model, Journal of Computational Physics **228**, 4468 (2009).

[7] K. Yang, Y.-F. Chen, G. Roumpos, C. Colby, and J. 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* (2019) pp. 1–15.

[8] Y. Fang, S. Feng, K.-M. Tam, Z. Yun, J. Moreno, J. Ramanujam, and M. Jarrell, Parallel tempering simulation of the three-dimensional Edwards–Anderson model with compact asynchronous multispin coding on GPU, Computer Physics Communications **185**, 2467 (2014).

[9] H. Xiao, K. Rasul, and R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, arXiv preprint arXiv:1708.07747 (2017).

[10] R. Liao, S. Kornblith, M. Ren, D. J. Fleet, and G. Hinton, Gaussian-Bernoulli RBMs Without Tears, arXiv preprint arXiv:2210.10318 (2022).

[11] T. Hirtzlin, B. Penkovsky, M. Bocquet, J.-O. Klein, J.-M. Portal, and D. Querlioz, Stochastic computing for hardware implementation of binarized neural networks, IEEE Access **7**, 76394 (2019).

[12] A. Krizhevsky and G. Hinton, CIFAR-10 and CIFAR-100 datasets, https://www.cs.toronto.edu/~kriz/cifar.html (2009).

[13] X. Qin, X. Luo, Z. Wu, and J. Shang, Optimizing the sediment classification of small side-scan sonar images based on deep learning, IEEE Access **9**, 29416 (2021).