Increasing Ising Machine Capacity with Multi-Chip Architectures

Anshujit Sharma, Richard Afoakwa, Zeljko Ignjatovic and Michael Huang
{anshujitsharma,richard.afoakwa,zeljko.ignjatovic,michael.huang}@rochester.edu
Department of Electrical and Computer Engineering, University of Rochester
Rochester, New York, USA

ABSTRACT

Nature has inspired a lot of problem solving techniques over the decades. More recently, researchers have increasingly turned to harnessing nature to solve problems directly. Ising machines are a good example and there are numerous research prototypes as well as many design concepts. They can map a family of NP-complete problems and derive competitive solutions at speeds much greater than conventional algorithms and in some cases, at a fraction of the energy cost of a von Neumann computer.

However, physical Ising machines are often fixed in its problem solving capacity. Without any support, a bigger problem cannot be solved at all. With a simple divide-and-conquer strategy, it turns out, the advantage of using an Ising machine quickly diminishes. It is therefore desirable for Ising machines to have a scalable architecture where multiple instances can collaborate to solve a bigger problem. We then discuss scalable architecture design issues which lead to a multiprocessor Ising machine architecture. Experimental analyses show that our proposed architectures allow an Ising machine to scale in capacity and maintain its significant performance advantage (about 2200x speedup over a state-of-the-art computational substrate). In the case of communication bandwidth-limited systems, our proposed optimizations in supporting batch mode operation can cut down communication demand by about 4-5x without a significant impact on solution quality.

CCS CONCEPTS

• **Computer systems organization → Analog computers**.

KEYWORDS

Ising machine, scaling, multi-chip, nature-based computing

ACM Reference Format:
Anshujit Sharma, Richard Afoakwa, Zeljko Ignjatovic and Michael Huang. 2022. Increasing Ising Machine Capacity with Multi-Chip Architectures. In *The 49th Annual International Symposium on Computer Architecture (ISCA '22), June 18–22, 2022, New York, NY, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3470496.3527414

ISCA '22, June 18–22, 2022, New York, NY, USA

ACM ISBN 978-1-4503-8610-4/22/06…$15.00
https://doi.org/10.1145/3470496.3527414

1 INTRODUCTION

Nature apparently does a lot of computation all the time, solving differential equations, performing random sampling, and so on. We have harnessed some of it of course. The transistors, for example, can be turned on/off and are the foundation for most of our computers today. But this is different from harnessing nature's computational capability at some higher level, for example, to solve an entire problem. Indeed, some very powerful algorithms are inspired by nature [2, 33, 59]. It is not hard to imagine that if a computing substrate is nature-based, we could solve a certain set of problems much more quickly and efficiently than through mapping it to the von Neumann interface. One particular branch of this effort that has seen some recent rapid advance is Ising machines.

In a nutshell, Ising machines leverage nature to seek low-energy states for a system of coupled spins. A number of problems (in fact, all original NP-complete problems [36]) can be expressed as an equivalent optimization problem of the Ising formula (more on that in Sec. 2). Though existing Ising machines are largely in the form of prototypes and concepts, they are already showing promise of (much) better performance and energy efficiency for specific problems.

However, when the problem size is beyond the capacity of the machine, the problem can no longer be mapped to the hardware. Intuitively, with some form of divide and conquer, we can create smaller sub-problems that can map to the hardware and thus still benefit from the acceleration of an Ising machine. As it turns out, with the state-of-the-art algorithm employed by D-Wave [13], the effective speedup of a system employing such a divide-and-conquer strategy quickly diminishes as the size of the problem increases. As an example – and more details in Sec. 3.1 – a 500-node machine can reach a speedup of 600,000 over a von Neumann solver (simulated annealing); using the same machine to help solve a 520-node problem will only achieve a speedup of 250.

In this paper, we first analyze the problem of simple divide-and-conquer strategy. We show why such a strategy is fundamentally limited in its performance by "glue" computation. Thus, we need machines that are designed from ground up to obviate such glue. Next, we investigate hardware designs to achieve that goal. Later on, we perform experimental analyses that show the design can indeed scale to larger problems while maintaining high performance; achieving more than 6 orders of magnitude speedup over a sequential solver and over 2200x speedup over a state-of-the-art computational accelerator.

2 BACKGROUND AND RELATED WORK

2.1 Principles of Ising machines

The Ising model is used to describe the Hamiltonian of a system of coupled spins. Each spin has one degree of freedom and takes one of two values ($\sigma_i \in \{-1, 1\}$). The energy of the system is a function of pair-wise coupling of the spins ($J_{ij} = J_{ji}$) and the interaction of some external field (μ) with each spin (h_i). The resulting Hamiltonian is as follows:

$$H = -\sum_{i,j} J_{ij}\sigma_i\sigma_j - \mu\sum_i h_i\sigma_i \quad (1)$$

Given such a formulation, a minimization problem can be stated: what state of the system ($[\sigma_1, \sigma_2, ...]$) has the lowest energy. A physical system with such a Hamiltonian naturally tends towards low-energy states. It is as if nature always tries to solve the minimization problem, which is not a trivial task. Indeed, the cardinality of the state space grows exponentially with the number of spins, and the optimization problem is NP-complete: it is easily convertible to and from a generalized max-cut problem, which is part of the original list of NP-complete problems [30].

Thus, if a physical system of spins somehow offers programmable coupling parameters (J_{ij}, μ and h_i in Eq. 1), they can be used as a special purpose computer to solve optimization problems that can be expressed in Ising formula (Eq. 1). In fact, all problems in the Karp NP-complete set have their Ising formula derived [36]. Also, if a problem already has a QUBO (quadratic unconstrained binary optimization) formulation, mapping to Ising formula is as easy as substituting bits ($b_i \in \{0, 1\}$) for spins: $\sigma_i = 2b_i - 1$.

Because of the broad class of problems that can map to the Ising formula, building nature-based computing systems that solve these problems has attracted significant attention [6, 9, 11, 14, 26, 31, 32, 41, 56]. Loosely speaking, an Ising machine's design goes through four steps:

(1) Identify the physical variable to represent a spin (be it a qubit [27], the phase of an optical pulse [28], or the polarity of a capacitor [3]);
(2) Identify the mechanism of coupling and how to program the coefficients;
(3) Demonstrate the problem solving capability showing both the theory of its operation (reveal the "invisible hand" of nature) and satisfactory results of practice;
(4) Demonstrate superior machine metrics (solution time, energy consumption, and construction costs).

It is important to note that different approaches may offer different fundamental tradeoffs. Each of them may go through varying gestation speeds. Thus, it would be premature to evaluate a general approach based on observed instances of prototypes. Nevertheless, we provide a broad-brushed characterization[1], which can help researchers get a basic sense of the landscape – as long as the caveats are properly understood.

[1]This characterization is by no means comprehensive. In particular, for conceptual clarity, we treat the numerous designs that accelerate a von Neumann algorithm (simulated annealing or a variant) using GPU, FPGA, or an ASIC not as a physical Ising machine, but an accelerated simulated annealer [17, 22, 23, 40, 47–49, 57, 58].

2.2 The three (and a half) generations of Ising machines

One of the earliest and perhaps the best known Ising machines are the quantum annealers marketed by D-Wave. Quantum annealing (QA) is different from adiabatic quantum computing (AQC) in that it relaxes the adiabaticity requirement [12]. QA technically includes AQC as a subset, but current D-Wave systems are not adiabatic. In other words, they do not have the theoretical guarantee of reaching ground state. Without the ground-state guarantee, the Ising physics of qubits has no other known advantages over alternatives. And it can be argued that using quantum devices to represent spin is perhaps suboptimal. First, the devices are much more sensitive to noise, necessitating cryogenic operating condition that consumes a lot of power (25KW for D-Wave 2000q). Second, it is perhaps more difficult to couple qubits than to couple other forms of spins, which explains why current machines use a local coupling network. The result is that for general graph topologies, the number of nodes needed on these locally-coupled machines grow quadratically and a nominal 2000 nodes on the D-Wave 2000q is equivalent to only about 64 effective nodes [24, 25].

Coherent Ising machines (CIM) can be thought of as a second-generation design where some of the issues are addressed [11, 28, 37, 45, 55]. In [28], all 2000 nodes can be coupled with each other, making it apparently the most powerful physical Ising machine today. CIM uses special optical pulses serving as spins and therefore can operate under room temperature and consumes only about 200W power. However, besides their own technical challenges, the current CIMs all use computed rather than physical coupling. Such a design is essentially a hybrid physical-simulated Ising machine, and is unlikely to be energy-efficient due to fundamental reasons. A recent experiment shows that a 2000 node CIM requires more computation than a computational accelerator [48].

Since the operating principle of CIM can be viewed with a Kuramoto model [46], using other oscillators can in theory achieve a similar goal. This led to a number of electronic oscillator-based Ising machines (OIM) which can be considered as a third-generation [16, 51, 52]. These systems use LC tanks for spins and (programmable) resistors as coupling units.[2] These electronic oscillator-based Ising machines are a major improvement over earlier designs in terms of machine metrics. To be sure, their exact power consumption and operation speed depend on the exact inductance and capacitance chosen and can thus span a range of orders of magnitude. But it is not difficult to target a desktop size implementation with around 1-10W of power consumption – a significant improvement over cabinet-size machines with a power consumption of 200W-25KW. However, for on-chip integration, inductors are often a source of practical challenges. They are area intensive, have undesirable parasitics with reduced quality factor and increased phase noise all of which pose practical challenges in maintaining frequency uniformity and phase synchronicity between thousands of on-chip oscillators.

[2]Technically, the phase of the oscillators is equivalent to spin with two degrees of freedom, spanning an XY-plane (rather than the 1 degree of freedom of up or down in the Ising model). Consequently, an additional mechanism is needed to impose a constraint – such as the Sub-Harmonic Injection Locking (SHIL) [15] – to solve Ising formula problems.

Another electronic design with a very different architecture (think of it as generation 3.5) is the Bistable Resistively-coupled Ising Machine (BRIM) [3]. In BRIM, the spin is implemented as a capacitor whose voltage is controlled by a feedback circuit making it bistable. The design is CMOS-compatible and since it uses voltage (as opposed to phase) to represent spin, it enables a straightforward interface to additional architectural support for computational tasks. We therefore use a baseline substrate similar to BRIM. Note that the same principle discussed in the paper could directly apply to all Ising machines with different amount of glue logic.

2.3 The common issue

For most state-of-the-art (physical) Ising machines today, when the problem fits the hardware, significant speedups and energy efficiency gains can be expected compared to von Neumann computing. However, little is discussed on what happens when the problem is beyond the capacity of the machine. It is understandable to assume that the machine can still accelerate proportional to the fraction of the problem that can be mapped. As we will show next, the reality is that for problems beyond the capacity of the machine, we can expect little to no benefit at all.

3 ON THE DIVIDE AND CONQUER STRATEGY

We first discuss the approach adopted by D-Wave's system (Sec. 3.1). As their systems are the only commercially available Ising machine platforms, their solution is both the state-of-the-art and a baseline for any comparison. We then discuss the details of a divide-and-conquer strategy, starting with the basic principle (Sec. 3.2) and then show that the sub-problems to be solved are not independent of each other, making parallelization challenging (Sec. 3.3).

3.1 Practice

With D-Wave's tool [13], one can use any Ising machine to solve a problem larger than its hardware capacity. To see the performance of such a system (for the reader's convenience, it is replicated in the appendix as Algorithm 1), we will use a model of BRIM [3] as the Ising machine. Even though the general strategy should work with any Ising machine, we note that BRIM offers a number of practical advantages for our study. First, it offers all-to-all coupling. This means that an n-node machine can map any n-node arbitrary graph.[3] Second, as we explore architectural support for scalable Ising machine later, the CMOS compatibility of BRIM gives us significant design flexibility.

In Fig. 1, we show the speedup of an 500-node Ising machine as the problem size increases past the machine capacity. We omit the measurement details as they do not affect the qualitative lessons. From the figure, we see two things. First, when the problem gets bigger but still fits within the machine, the speedup increases. Clearly, larger hardware capacities are desirable.

However, the figure also shows a second point, and perhaps a more important point: as soon as the problem is bigger than what

[3]Many machines offer a large number of *nominal* nodes but only local coupling [27, 47, 57]. A general graph of n nodes has $O(n^2)$ coupling parameters. Mapping such a graph therefore requires $O(n^2)$ nodes for local connection machines.

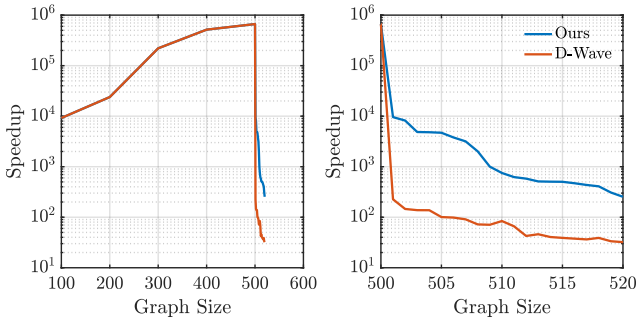

Figure 1: Speedup of two divide-and-conquer algorithms (D-Wave and ours) using a 500-spin BRIM plus a sequential computer for support. [Left]: Speedup for all graphs tested. [Right] Magnified segment for graph sizes from 500 to 520.

the hardware can hold, the speedup crashes precipitously (Fig. 1-right). The fact that the speedup reduces is not surprising. What is surprising is how much and how quickly it does. Of course, some of this is due to the specific implementation of the D-Wave tool. We have thus created an alternative (Algorithm 2 in the appendix), shown as "Ours" in the figure, and the result (shown in Fig. 1) is only a small improvement. The sharp drop is due to a fundamental reason, which we delve into next.

3.2 Principle

We first start with the principle of divide and conquer in Ising optimization problems. This part may be obvious to some readers who should skip to Sec. 3.3. The problem of minimizing Eq. 1 is often described as navigating a high-dimensional energy landscape to find the lowest valley. We can imagine keeping some dimensions fixed (*e.g.*, longitude) and navigate along the remaining dimensions in search of a better spot. Many solvers can be described using this analogy. This is the essence of the divide and conquer strategy. This point (as well as its problem) can be shown clearly and explicitly with some straightforward math. Here, we think the matrix notation is more helpful. We rewrite Eq. 1 as:

$$H = -\sigma^\top J\sigma - \mu h^\top \sigma \tag{2}$$

where $\sigma = [\sigma_1, ..\sigma_n]^\top$, $J = |J_{ij}|_{n\times n}$, and $h = [h_1, .., h_n]^\top$. Here J is a symmetric matrix with the diagonal being 0. If we divide the n-node problem into two sub-problems of k and $n-k$ nodes, we can rewrite Eq. 2 as follows.

$$\begin{aligned} H &= -[\sigma_u^\top, \sigma_l^\top] \left[\begin{array}{c|c} J_u & J_\times \\ \hline J_\times^\top & J_l \end{array}\right] \begin{bmatrix}\sigma_u \\ \sigma_l\end{bmatrix} - \mu[h_u^\top, h_l^\top]\begin{bmatrix}\sigma_u \\ \sigma_l\end{bmatrix} \\ &= -\sigma_u^\top J_u\sigma_u - \sigma_l^\top J_\times^\top\sigma_u - \sigma_u^\top J_\times\sigma_l - \sigma_l^\top J_l\sigma_l - \mu h_u^\top\sigma_u - \mu h_l^\top\sigma_l \\ &= -\left(\sigma_u^\top J_u\sigma_u + \sigma_l^\top J_\times^\top\sigma_u + \mu h_u^\top\sigma_u\right) - \left(\sigma_l^\top J_l\sigma_l + \sigma_u^\top J_\times\sigma_l + \mu h_l^\top\sigma_l\right) \\ &= \underbrace{-\left(\sigma_u^\top J_u\sigma_u + g_u^\top\sigma_u\right)}_{sub\text{-}problem\ (J_u, g_u)} \underbrace{-\left(\sigma_l^\top J_l\sigma_l + g_l^\top\sigma_l\right)}_{sub\text{-}problem\ (J_l, g_l)} \\ &\quad where\ g_u = \mu h_u + J_\times\sigma_l,\ g_l = \mu h_l + J_\times^\top\sigma_u \end{aligned} \tag{3}$$

With this rewrite, we show that the bigger square matrix can be decomposed into the upper and lower sub-matrixes J_u and J_l (both square), and the "cross terms" ($J_\times$ and its transpose). The effect of the cross terms can be combined with the original biases (h_u and h_l respectively) into new ones (g_u and g_l respectively). From this point of view, an Ising optimization problem with n nodes can always be decomposed into sub-problems of k and $n-k$ nodes, and by transitivity into a combination of sub-problems of any sizes.

3.3 Issues of decomposition

Eq. 3 not only shows the principle of decomposition, it also clearly shows the issue with it. In the original problem, J and h are parameters and do not change. After decomposition, the bias of the upper partition (g_u) is now a function of the *state* of the lower partition. This means that the two partitions are not independent. In other words, strictly speaking, the sub-problems have to be solved sequentially: when search changes the current state of the upper partition, we need to update the parameters of the lower partition to reflect the change before starting the search in the lower partition. Partitioning also does not reduce total workload. Thus, it is not surprising that there is no parallel version of canonical simulated annealing through divide-and-conquer.

In the case of trying to solve a bigger problem than a machine's capacity, the issue may seem irrelevant: After all, if we can decompose a bigger problem into two parts (say, A and B), and A now fits into an Ising machine; we can expect to enjoy speedup from the processing of A even if processing of A and B can not overlap. The reasoning is correct. But in reality, there are multiple subtle problems with severe consequences. We will discuss two that are relevant here:

(1) As we already saw, with decomposition, the sub-problem's formulation changes constantly, which requires reprogramming. For many Ising machines, reprogramming is a costly operation and can take more time than solving the problem. To cite a perhaps extreme example, D-Wave's programming time is 11.7 ms, compared to a combined 240 μs for the rest of the steps in a typical run [19]. Keep in mind, a common (if not universal) usage pattern of these Ising machines is to program once and anneal many (e.g., 50) times from different initial conditions and take the best result. In such a usage pattern, long programming time is amortized over many annealing runs. In a decomposed problem, reprogramming may have to occur many times *within* one annealing run.
(2) Even if the cost of reprogramming is somehow addressed, we still have the familiar Amdahl's law. Using a concrete example of BRIM (Fig. 1), the speedup of solving a 500-node problem over a sequential computer is on the orders of 10^5. Consider using our algorithm to decompose a 510-node problem into a 500-node sub-problem mapped to the hardware and the remaining portion plus glue computation left for software. The workload for software measured in instruction amounts to about 0.13% of the software workload for the original 510-node problem. Much of the remaining software workload is the glue (calculating the new biases $g_u = \mu h_u + J_\times \sigma_l$ and $g_l = \mu h_l + J_\times^\top \sigma_u$), different from original solver. As a result, Amdahl's law does not directly apply. Nonetheless, we can (ab)use it to roughly estimate a speedup upper-bound on the order of 700x.

There are certainly some nuances to the simplified analyses above, but the bigger point is crystal clear and recapped below.

3.4 Recap

In principle, the problem formulation clearly allows decomposition of larger problems. But the smaller component problems are not independent. As a result, relying on von Neumann computing to glue together multiple Ising machines is a fundamentally flawed strategy, as it severely limits the acceleration of problems even marginally larger than a machine's capacity. The machines need to be designed from ground up to be used in collaboration and address the decomposition bottleneck.

4 SCALING OF ISING MACHINES

Our general approach is conceptually straightforward: make a physically bigger Ising machine (spread across multiple chips). We first discuss generalities in Sec. 4.1 and a more direct incarnation of a macro chip and its problems in Sec. 4.2.

4.1 General analysis of scaling Ising machines

4.1.1 Ising machine basics. The core of an Ising machine contains two types of components: nodes and coupling units. As already discussed in Sec. 2, the coupling units need to be programmable to accept the coupling strengths J_{ij}[4] as the input to the optimization problem, and the dynamical system will evolve based on some annealing control. Finally, all spins are read out as the solution to the problem. In a generic problem, any spin can be coupled with any other spin. Thus, there are far more coupling parameters than spins ($O(N^2)$ vs $O(N)$). A number of existing Ising machines, however, adopt a machine architecture where only nearby spins are allowed to couple, resulting in a system with $O(N)$ coupling units and $O(N)$ spins [27, 47, 52, 56]. A special software tool [18] is used to first convert the original problem into a form that follows the constraint placed by the machine's architecture. Loosely speaking, these $O(N)$ coupling units can therefore map a problem of the scale of $O(\sqrt{N})$ spins. This is confirmed by observation of actual problems [3, 25]. For the rest of the paper, we will focus only on architectures with all-to-all connections.

4.1.2 Electronic Ising machine baseline. A number of electronic Ising machines have been recently proposed [16, 51, 52]. The primary operating principle is similar, though at a deeper level there are significant technical differences. We will discuss the relevant operating principle here. Readers familiar with these machines can skip forward. Our baseline is BRIM [3] where an array of N bi-stable nodes are interconnected by an array of $N \times N$ resistive coupling units. The coupling units are programmed by an array of DACs before the system starts annealing. The coupling resistor value between nodes i and j is set to $1/J_{ij}$: strong coupling meaning lower resistance. And the sign of coupling strength can be implemented with either parallel or antiparallel connection. Specifically, if the coupling parameter J_{ij} is negative, then the two nodes are

[4]The bias term $\mu h_i \sigma_i$ can be viewed as a special coupling term $J_{i,n+1}\sigma_i\sigma_{n+1}$ ($J_{i,n+1} \triangleq \mu h_i \sigma_i$) which coupled σ_i with an extra, fixed spin ($\sigma_{n+1} = +1$).

connected in an antiparallel fashion (positive plate of i connected to negative plate of j and vice versa). This encourages the two nodes to be in opposite polarities, so that the contribution of this pair's coupling ($-J_{ij}\sigma_i\sigma_j$) lowers the overall energy. Conversely, if J_{ij} is positive, the plates of the same polarity will be coupled through the resistor. Fig. 2 shows the machine characteristics.

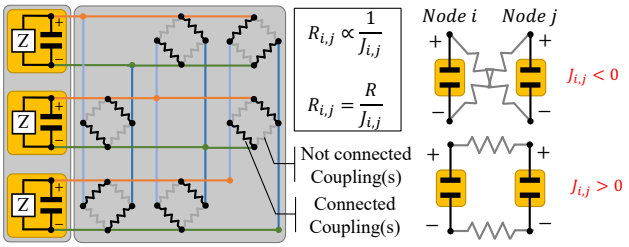

Figure 2: A simplified diagram of BRIM showing nodal capacitors, coupling resistors, and parallel/antiparallel connections.

All these electronic Ising machines can be analyzed as a dynamical system and Lyapunov stability analysis can show why they tend toward low-energy state in a more theoretical fashion. But a more intuitive discussion with an example situation suffices for our purpose. Let us imagine the system is in a particular state, and one spin (say, $\sigma_k = -1$) is "wrong" – meaning if we flip it ($\sigma_k = +1$), energy will improve/decrease. A little algebra will show that this means

$$\sigma_k \left(\sum_{j \neq k} J_{jk}\sigma_j \right) < 0 \tag{4}$$

If we recall J_{jk} is represented by the coupling resistor between nodes j and k and substitute σ_j with the representation of it (V_j, the voltage of node j), the term $\sum_{j\neq k} J_{jk}\sigma_j$ is thus approximated by $\sum_{j\neq k} \frac{V_j}{R_{jk}}$, which describes the current coming into node k. According to Eq. 4, this value is of the opposite sign of σ_k. This shows that if node k is wrong, the combined current input to it will be in the opposite polarity and thus has the effect of trying to correct/flip it. A similar exercise can show that when node k is right (flipping it would increase/deteriorate energy), the current input from outside node k will agree with the current polarity of k, and thus keeping it in the current state. Given this baseline, we now analyze one conceptually straightforward design of an Ising machine with a large capacity.

4.2 A macrochip architecture

Fig. 3 shows a k-by-k array of chips each with N^2 coupling units that together form a larger machine with $(kN)^2$ coupling units. We simply connect the wires of a row of coupling units to the corresponding wires of the same row from the left and/or right neighbor chip. Similarly, the wires of the same column from upper and lower neighbors are joined. If we ignore the packaging and wire loading issue for a moment, we can treat the entire circuit area as one (bigger) macrochip Ising machine with kN nodes.

A single macrochip machine is not necessarily limited to solving one problem at a time. Given an Ising machine of kN nodes, without any system support, a user can already solve multiple smaller problems simultaneously simply by manipulating the coupling matrix – so long as the sum of the number of nodes from each problem does not exceed kN. This can be seen in the illustration shown in Fig. 4.

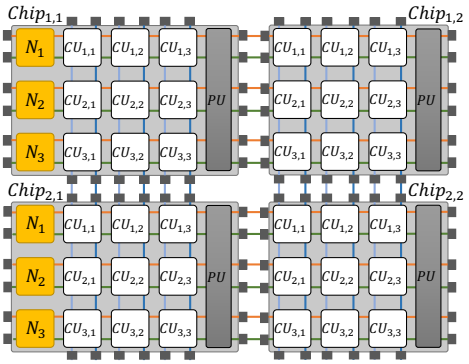

Figure 3: Block diagram of a macrochip of k-by-k array (k=2) of chips. Only the leftmost chips need nodes. *CU*'s are Coupling Units and *PU*'s are Programming Units.

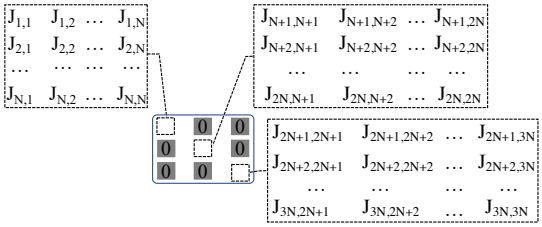

Figure 4: Conceptual view showing wasted resource when a single macrochip solves multiple smaller problems.

If we keep the shaded area of the coupling matrix all zero, the matrix is effectively isolated into several smaller submatrixes. This is obviously not resource-efficient: it takes k^2 chip to form a macrochip of kN nodes. If we use this macrochip to solve smaller problems of size N, we can only accommodate k such problems. Indeed, in that specific case, only the coupling arrays of the chips in the diagonal of the $k \times k$ array are being used.

Such waste is not difficult to avoid. One approach is to introduce some reconfigurability, so that the chips can either work together to solve a bigger problem or independently to solve multiple smaller problems. Fig. 5 demonstrates this functionality. Three types of units need to be reconfigurable in such a design: the nodes, the diagonal couplers, and the interface pins in each chip. In independent operation, each chip is isolated from others and operate just like a single-chip Ising machine: the nodes are in regular mode, the pins are disconnected from rows and columns of wires, and the diagonal couplers are in cross-over mode (where the wires for row i and column i are connected at the diagonal coupler (i,i)). In collective operation, corresponding rows and columns of wires are connected through the pins to neighboring chips; all the diagonal couplers except on the main diagonal of the entire macrochip will be switched to regular coupler mode (the main diagonal will remain

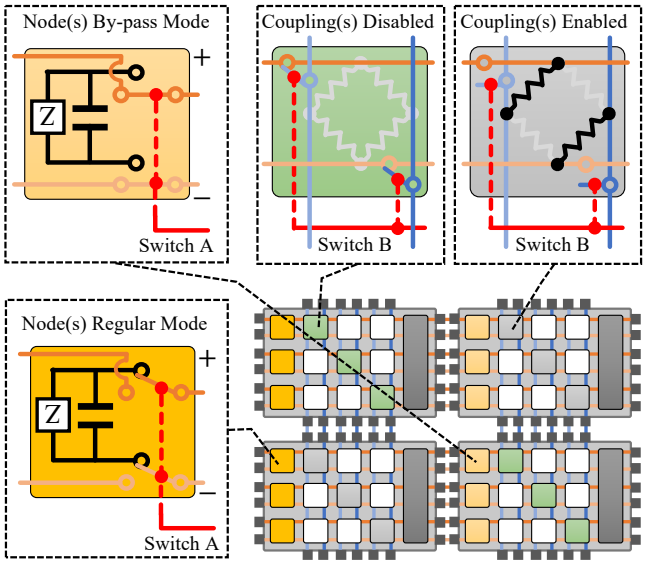

Figure 5: Macrochip with switching support to allow either independent or joint operation. In this specific configuration, each chip operates jointly to solve one large problem.

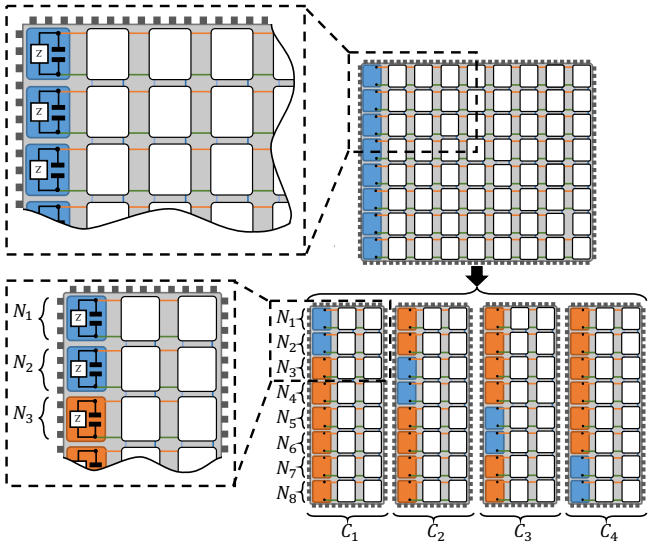

Figure 6: Logical view of a single BRIM chip (top) split into multiple chips (bottom). In the multi-chip setup, nodes are labelled N_1 to N_8 while chips are labelled C_1 to C_4. Nodes with orange color are shadow copies, while regular nodes are colored blue. The zoomed in logical views are also shown. The inter-node digital interconnect is not shown.

in cross-over mode); and only nodes in the leftmost chips are in regular mode while other nodes on other chips will be in by-pass mode. Note that it is also possible that a subset of the chips will be working collectively while the rest work independently.

While this macrochip design is conceptually straightforward, there are a number of issues concerning its implementation. The primary concern is the chip-interface. Depending on whether the chips are integrated via PCB or interposers, the chip-to-board interface may become an engineering challenge. As the interface carries fast-changing analog signals between multiple chips, they certainly make analysis of system behavior less straightforward. For this reason, the next section will focus on an entirely digital interface. In a sense, we use multiple chips plus a digital interconnect to make a multiprocessor Ising machine.

5 A MULTIPROCESSOR ARCHITECTURE

By having all coupling coefficients embodied in physical units, the macrochip just discussed fundamentally avoids any glue computation to support multi-chip operation. While we keep this essential feature, the multiprocessor architecture addresses the interface issues of the macrochip.

5.1 Basic architecture

Fig. 6 shows this design. On the top, we have the logical system layout: N nodes with N^2 coupling units. Physically, they can be split over, say, 4 chips, each one with a complete set of nodes. Some of these nodes (shown in orange) are merely "shadow copies" of the real nodes on some other chip. For example, node 3 on chip C_1 is just a buffer. The real node 3 is on C_2. When node 3 changes polarity (say to -1), C_2 will communicate this information to other chips through a digital fabric (not shown). All other chips will use a buffer to maintain a -1 value for node 3 until C_2 notifies them of further changes. In other words, compared to the real node, the shadow copies are approximate in time (a bit delayed) and in value (always at a voltage rail).

5.2 Reconfigurable chip architecture

With this design, the logical structure of a single chip captures a long slice of the overall coupling matrix. This logical structure is still implemented based on a typical square baseline chip architecture. All we needed to do is to build it from a modular, re-configurable array. As shown in Fig. 7, a single chip can be made into a square array of $k \times k$ modules ($k = 4$ in the example). Each module consists of an array of n configurable nodes, and $n \times n$ coupling units. The general idea is that these modules can then be strung together differently for different purposes. For instance, the three configurations are: ① $2n \times 8n$; ② $4n \times 4n$; and ③ $1n \times 16n$. In this way, this chip can be used as a part of a four-chip multiprocessor of $8n$ nodes total, as a single machine of $4n$ nodes, or part of a 16-chip multiprocessor of $16n$ nodes total.

Let us take the configuration of $2n \times 8n$ as a concrete example. When combined with 3 others chips of the same configuration, the system forms a complete $8n \times 8n$ coupling matrix. In Fig. 7 (bottom right) we show the desired logical organization of the 16 modules. Among these modules, only 2 (providing $2n$ nodes) are configured as regular nodes (module 1 and 2 in blue); 6 are configured as shadow copies (3, 4, and 9 to 12 in orange); the rest are configured to pass through (green). In addition to different configuration of the nodes, wire connections need to change too. For instance, modules 1 and

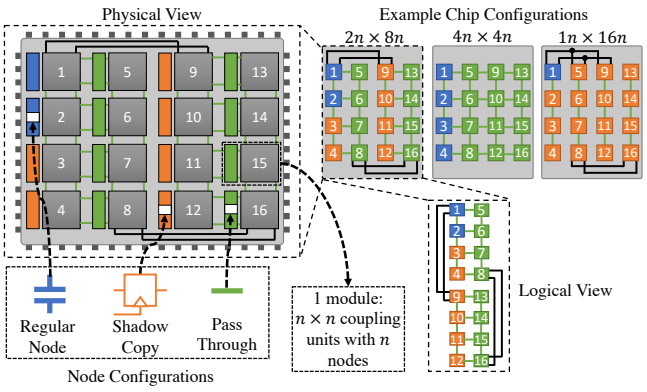

Figure 7: Illustration of a reconfigurable chip made of 4×4 modules each with n nodes and an array of $n \times n$ coupling units. The nodes can operate in three different modes: regular (blue), shadow copy (orange), or pass-through (green). These modules can be configured in three ways, corresponding to different coupling (sub)matrices. ① $2n \times 8n$: as can be seen in the logical view (bottom right), the 16 modules are effectively connected as 2 columns each with 8 modules. This can be used in a 4-chip multiprocessor with a total of $8n$ nodes. ② $4n \times 4n$: the chip is an independent machine with $4n$ nodes and the $16n^2$ coupling units organized as a $4n \times 4n$ matrix. These $4n$ nodes are in the first column (blue). The nodes in the rest of the modules are set in pass-through mode (green). ③ $1n \times 16n$: similar to the first example, this chip is used in a 16-chip multiprocessor with a total of $16n$ nodes.

9 have to be connected so that modules 1 to 4 and 9 to 12 can act as one 8-module tall column.

5.3 Communication demand and technological solutions

The basic idea is that when a spin changes polarity, one chip needs to communicate to other chips in order for them to update their shadow copies. The communication demand is, to a first approximation, $f_s N log(N)$, where N is the total number of spins in a system and f_s is the frequency of spin flips. Take a concrete example of our baseline Ising substrate: on average, one spin/node flips every 10 ns, depending on problems being solved. Assuming the same spin flip frequency, if we take sixteen 8,000-spin chips to form a multiprocessor Ising machine, the total system would offer 32,000 spins ($\sqrt{16} \times 8000$) and would require at least 50 Tb/s (broadcast) bandwidth. In fact, due to the annealing schedule, the system has a higher spin flip frequency at the beginning of the schedule and thus would demand even more peak bandwidth.

Note that such communication is also needed for any multi-threaded von Neumann solver. The difference is that, compared with a state-of-the-art physical Ising machine, a von Neumann solver is orders-of-magnitude slower and thus has a correspondingly lower bandwidth demand.

Given this significant, intrinsic bandwidth demand, a number of technological solutions immediately come to mind. Optical communication and 3D integration are both appealing options. Indeed, 3D integration is a very convenient solution to the proposed architecture. Fig. 8 shows an example 4-layer 3D IC. We can see nodes and their shadow copies are conveniently located on top of each other and thus can be easily connected with through-silicon vias (TSV). In fact, due to the short distance of the TSVs, shadow nodes are no longer necessary architecturally. They may still be a convenient circuit solution for improving driving capabilities though.

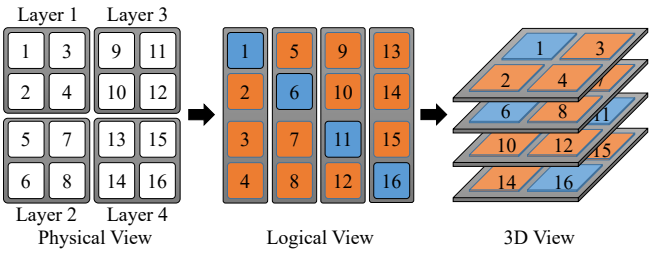

Figure 8: Illustration of 3D-integrated multiprocessor. Each layer (physical view) is shown as four modules (each module is an $n \times n$ array). When four layers are connected to form a multiprocessor, every layer operates as an $1n \times 4n$ slice (logical view). Together, they form the $4n \times 4n$ machine. In the logical view, within each slice, the blue box indicates the module of regular spins while the orange ones are their shadow copies (*e.g.*, block 6's shadow copies are 2, 10, and 14, which are conveniently on top of each other in the 3D view to be connected by TSV).

Finally, we can always slow down the physics of the Ising machine so that the communication demand matches the supply of the fabric. In the case of BRIM, this can be achieved in a combination of (at least) two ways. First, the machine's RC constant can be increased – higher coupling resistors will be used to slow down charging. Second, the system can be stopped altogether, for instance, to wait out a congestion. No matter how we combine these mechanisms the math is simple: to reduce bandwidth demand by 2x, we need to slow down the machine by 2x. There are things that we can do to reduce the bandwidth demand without a corresponding reduction in performance. We discuss these next.

5.4 Concurrent mode operation

Concurrent operation of multiple Ising machines (solving the same problem) can be roughly described as a combination of each machine performing local searches independently and exchanging information about the state of spins with each other. A surprisingly consequential design parameter is how long to wait before communicating a change of spin to others. Sending any change immediately seems the natural choice as the multiprocessor functions most closely to a monolithic, large physical Ising machine. However, waiting has its merit too. During a window of time, a spin may flip back and forth multiple times. With some waiting, we avoid wasting bandwidth on unnecessary updates. In this regards,

the longer the waiting, the more we can save on bandwidth. In reality, however, the wait time has implications on the solution quality, as we explain next.

5.4.1 Impact of global state ignorance. In a concurrent operation, every solver is always having some "ignorance" of the true state of spins mapped on other solvers. Take a 4-solver system as an example. At any point, the system's global state can be represented by $S_g = [A, B, C, D]^\top$ where each letter represents the spin vector of each machine. Due to communication delays as well as waiting mentioned above, the first solver's *belief* of the current state is thus $S_1 = [A, B', C', D']^\top$. In its local search, it is essentially trying to optimize the energy of this believed state $E(S_1)$. A low $E(S_1)$ does not necessarily mean that the energy of the system's true state $E(S_g)$ is also low. Imagine that solver 1 is now given the true state of other solvers and recalculates the energy. We can define the difference as the the *energy surprise*: $E_{surprise} = E(S_1) - E(S_g)$.[5] Fig. 9 shows the empirical observations of the energy surprises.

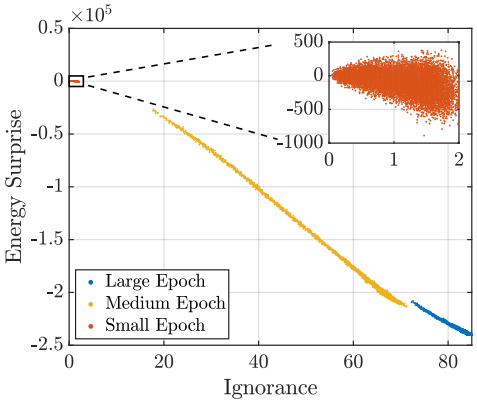

Figure 9: Degree of ignorance and the corresponding *energy surprise* for different epoch sizes. The graph has 8000 nodes which is partitioned and mapped onto 8 solvers with each solving 1000 nodes. The figure shows the communication for small, medium and large *epochs*. A magnified segment near the origin is also shown.

This particular experiment is obtained by solving an 8000-node problem divided into 8 sub-problems each solved by a simulated annealing (SA) solver. After initialization, each solver uses the state of the other 7000 nodes to compute the biases. Then they perform local searches for a fixed amount of time (called an epoch) before communicating the state with each other. At the epoch boundary, we can show the amount of ignorance measured by the percentage of external spins that have changed. We also calculate the energy surprise. Each dot represents the data of one epoch and the figure shows the results of all epochs from 20 runs.

We see that when the epoch time is long, more spin changes occur from other solvers. As a result, any single solver is under a higher degree of ignorance of the external state, leading to a higher degree of misjudgement and a larger magnitude of the energy surprise. When the epoch is longer than a certain value, the surprise is highly correlated with the degree of ignorance. In this regime, one can say that the parallel solvers are clearly doing a poor job (also reflected in very poor final solution quality not shown in the graph). So far, this message is consistent with earlier analysis that decomposed sub-problems are not independent from each other. However, when the epoch time goes below a certain threshold (small epoch, also shown in the magnified segment), the situation seems to have gone through a phase change: now, the energy surprise is no longer uniformly negative. At any rate, the magnitude of surprise gets lower. In other words, despite having *some* ignorance, the solvers can still find reasonable solutions. In fact, the overall solution quality is no worse (and often statistically better) than running the solvers sequentially (*i.e.*, without any ignorance). This means that it is practical to operate multiple Ising machines concurrently so long as they can communicate sufficiently swiftly to prevent building up too much ignorance.

5.4.2 Coordinated induced spin flips. Another important aspect of the design is about spin flips introduced to the system to prevent it from being stuck at a local minimum. (We will refer to them as *induced* spin flips.) These spin flips are generally applied stochastically, similar to an accepted proposal in the Metropolis algorithm. In a practical implementation, randomness is often of a deterministic, pseudo-random nature. As a result, if we properly synchronize the pseudo random number generator (PRNG) on each chip, we can guarantee that each chip will generate the same output everywhere at about the same time. In this way, we can apply induced spin flips without explicit communication. In other words, instead of randomly choosing, say, spin node 3 to flip and then sending an explicit message from the chip that contains the node to other chips informing them of the flip, the PRNG on all chips would be programmed to induce node 3 (and its shadow copies) to flip and update the nodal capacitor or the shadow register at about the same time.

To sum: phenomenologically multiple solvers *can* operate concurrently and seems to offer the highest solution quality, provided they keep each other informed "sufficiently promptly". This means, we have to choose a short epoch time, which generally means high communication demands. One opportunity to reduce the demand is using properly synchronized PRNG for induced spin flips.

5.5 Batch mode operation

While careful design of concurrent operation can achieve noticeable bandwidth savings (about 1.5x as we shall see), a completely different mode of operation – batch mode – can allow a more substantial savings (about 5x). This mode leverages the fact that a common, if not universal, mode of using an annealer is to perform a batch of annealing jobs of the same problem with different initial states and take the best solution from the batch. Each job is thus independent of each other. Knowing that we have a batch of jobs *of the same setup*, we can stagger them in a fairly straightforward manner to reduce the necessary communication.

The key idea is illustrated in Fig. 10. Viewed vertically, a single job (from one initial state, say Job 1) is still spread out over multiple solvers (on Chip 1 in epoch 1, on Chip 2 in epoch 2, etc.) like in concurrent mode. So each chip is still just annealing for its part of

[5] Defined this way, a positive energy surprise means that the current state has lower (better) energy than what the solver believed prior to the update: in other words, it is a good/positive surprise.

the problem. But the solvers now work sequentially. At the end of Epoch 1, Chip 1 passes on the updated spin state to others (indicated in the figure by the change to a darker red for the first quarter of spins in all chips). Chip 2 then picks up Job 1 and continues exploration of the second quarter of the spins.

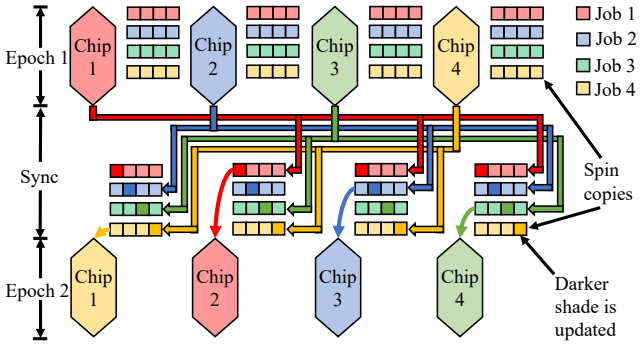

Figure 10: Illustration of batch mode operation where four staggered job runs of the same problem starting from different initial conditions are solved by four solvers. Each job run is depicted with a different color. Each chip has a copy of spins for every job. At the end of the epoch, all the chips broadcast its spins via dedicated channels which updates all the spin copies (shown by darker shades). Then each chip loads the next job's spin states and starts the next epoch of annealing of that job.

Viewed horizontally, in every epoch each of the 4 chips works on a different job (indicated by different colors). In the synchronization phase, they exchange the updated state and afterward start annealing on a different job. The key advantage of this approach is that each epoch can be much longer in time – without creating any ignorance. As already discussed, with a longer epoch, the total communication bandwidth needed can be much less than that needed to communicate every single event of spin flip.

To exploit parallelism, in batch mode, n different jobs (from different initial states) are performed simultaneously across n solvers. As a result, the system as a whole needs to carry n copies of states instead of just one in the concurrent mode. To support this, we need a modest increase in storage ($n \times N$ bits per solver) to keep the states for different jobs.

Finally, we note that it is tempting to think that a good way to run batch mode is just like in a von Neumann system where every machine runs an independent job. This is decidedly less efficient in a multiprocessor Ising machine: If an entire problem is solved by one machine, we need to context-switch in the new parameters at the end of every epoch. The data volume is $O(bN^2)$ bits, where b is the bit width of coupling weight – not to mention the effort it takes to reprogram the Ising machine. In contrast, in our proposed batch mode, the data volume is $O(N)$ bits.

6 EXPERIMENTAL ANALYSIS

6.1 Experimental methodology

Because we are in the early stage of Ising machine development, access to physical systems is difficult. Most of our comparisons will be performed with a mixture of modeling, using results reported in literature, and measuring time of simulated annealing (SA). In all the experiments, SA is natively executed, while BRIM's dynamical system evolution is modeled by solving differential equations using 4th-order Runge-Kutta method. When comparing to reported results, we are obviously limited by the type of benchmarks that were used in literature for direct comparison.[6] Fortunately, a few benchmarks (K-graphs) have been commonly used. One such graph that we will use for comparison is known as K16384 [49] and contains 16,384 spins with all-to-all connections. Simulating dynamical systems with differential equations can be orders-of-magnitude slower than cycle-level microprocessor simulation. Simulating $1\mu s$ of dynamics in K16384 takes about 3 days on a very powerful server. Therefore, we only use it for direct performance comparison. Smaller K-graphs (*e.g.*, K2000 [28]) are used for some additional analyses.

While the execution time of SA is generally the closest thing to a standard performance yardstick, there are actually quite some subtleties. First, different versions have vastly different performance. We choose Isakov's algorithm [29] as it is the fastest version as far as we know. Second, we apply an optimization using dense matrix representation. This exploits fully connected graphs like the K-graphs to improve performance. Finally, researchers have tuned the annealing schedules for these specific graphs. This tuning turns out to have significant impact on execution time. Similar tuning on our hardware annealing schedule could potentially also improve performance. We can not yet perform such tuning as the simulation cost is prohibitive.

6.2 Single solver baseline

To get a sense of the landscape of physical Ising machines and digital accelerators for simulated annealing, in Fig. 11 we show results of a few uniprocessor Ising machines running K2000: a BRIM chip (simulated), simulated annealing (measured), and the reported results for STATICA [54], CIM [28], and two variants of simulated bifurcation machine (SBM) [22]. Many other machines are missing from this comparison because they simply cannot map the graph. Despite its seemingly modest size (2000 nodes), K2000 is a fully connected graph and will take millions of nodes for machines with only local couplings.

For this graph, BRIM could reach the best known solution of 33,337 in 11 μs. The only other machine that could reach similar solution quality is dSBM in 2 ms, about 180x slower. Even if we are willing to accept lower solution quality, a single-chip BRIM is still at least an order of magnitude faster.

In summary, a properly designed physical Ising machine can be 6 orders of magnitude faster than a conventional simulated annealer (SA) and about 2 orders of magnitude faster than the state-of-the-art computational annealer. The only disadvantage of a physical Ising machine over a computational annealer is that the latter can more easily scale to solve bigger problems. We now look at how the proposed multiprocessor architecture addresses this issue. We will narrow our focus to comparing against just SA and SBM [49] as the latter is the fastest system at the moment.

[6] Cross-benchmark comparison is full of pitfalls (if not meaningless altogether) as there is no easy way to compare solution quality for different problems.

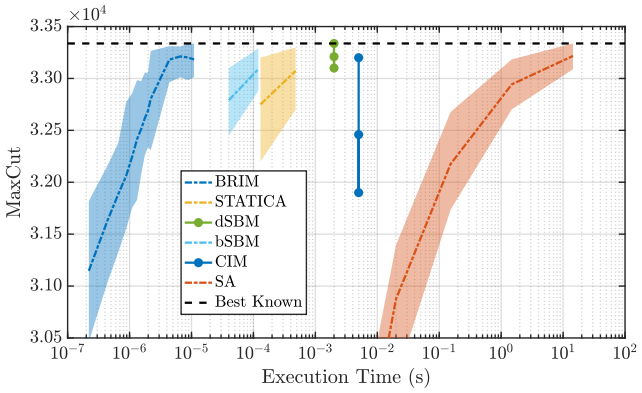

Figure 11: Performance of K2000 graph on diverse machines shown as solution cut value on y-axis (higher is better) and execution time in x-axis. At every time scale, the machine goes through 100 runs. In machines where multiple time scale results are available, the average results for each time scale are shown in a dashed line and the range is shown as a shaded region. Results with only one time scale are shown as bars with the dots indicating the range and the average cut values. Data for bSBM [22], dSBM [22], STATICA [54] and CIM [28] are obtained from the references.

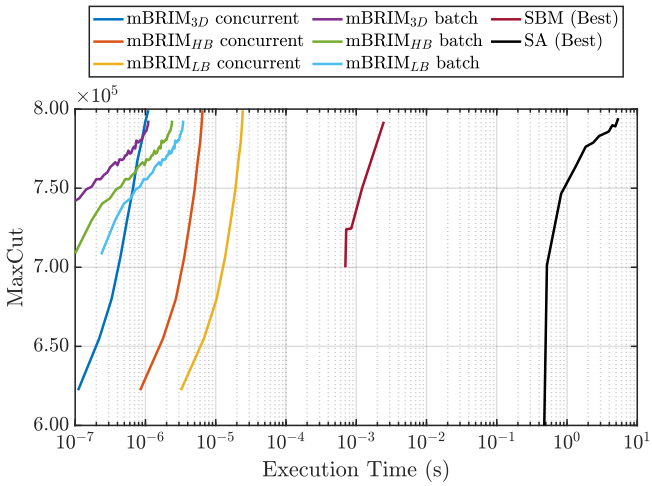

Figure 12: Solution quality as annealing proceeds for K16384 graph on 4-chip BRIM multiprocessors compared with an 8-FPGA version of SBM [49] and simulated annealing (SA).

6.3 High-level comparison

We compare our proposed multiprocessor BRIM (mBRIM) with SBM using the larger K16384 graph as the benchmark. This allows us to directly compare solution quality and performance with reported results in the literature. We assume a 4-chip multiprocessor. Each chip is a BRIM-style electronic Ising machine with 8192 nodes. Such a chip should have a smaller die size (about 80 mm^2 in a 45 nm technology) and consume much less power (less than 10 W) than a single FPGA used in SBM. We use three incarnations of this multiprocessor as proxies for different implementation choices:

(1) $mBRIM_{3D}$: A 3D-integrated version where communication is essentially instantaneous and without bandwidth limit;
(2) $mBRIM_{HB}$: A system with high communication bandwidth. Each chip is provided with three dedicated channel each of 250 GB/s. The total bandwidth is thus close to that of HBM [38].
(3) $mBRIM_{LB}$: A system with low communication bandwidth (4x less than $mBRIM_{HB}$).

Fig. 12 shows the best solution quality and time obtained by different mBRIMs, by an 8-FPGA implementation of SBM [49] and by SA. For clarity, only the results of the best-quality run are shown in the graph. If we compare the highest performing mBRIM ($mBRIM_{3D}$ concurrent mode), with SBM, we see that mBRIM gets to a much better solution quality (793,423 to 799,292 vs SBM's best result of about 792,000) and is about 2200x faster (1.1 μs vs 2.47 ms). Even the bandwidth constrained configuration ($mBRIM_{LB}$), which we would operate in batch mode, is more than 700x faster than SBM, also with a higher solution quality.

Next we look at the impact of bandwidth limitation. As already discussed in Sec. 4, if the communication bandwidth between chips is insufficient, we can resort to slowing down the Ising machines to cope. The impact, of course, is we need to wait longer to obtain the solution. Both $mBRIM_{HB}$ and $mBRIM_{LB}$ are slower than $mBRIM_{3D}$ due to congestion-induced stalling. In these bandwidth limited situations, our proposed batch mode operation is a reasonably effective tool and can improve execution speed. Specifically, batch mode allows the same amount of annealing to be finished by 2.8x and 7x faster for $mBRIM_{HB}$ and $mBRIM_{LB}$, respectively. With batch mode, $mBRIM_{HB}$ is only about 2x slower than $mBRIM_{3D}$ and $mBRIM_{LB}$ is another 1.4x slower. However, the solution quality is reduced to 792,728 (which is still better than SBM's result). We will take a more detailed look as to how the speed improvement is achieved and other issues in the next section.

Finally, we compare mBRIM to SA. We see that to get to the same solution quality, mBRIM is about 4.5×10^6 faster. This compares to the 1.3×10^6 speedup in K2000. Note here that there is an extraordinary difference (about 140x) for SA's performance due to tuning the annealing schedule.

6.4 In-depth analysis

6.4.1 First-principle approximations. It may be beneficial to understand how different types of solver work from first principles. No matter what solver is used, we need to explore the high-dimensional energy landscape sufficiently to achieve a good solution. As an example, for the K800 graph (800 nodes, all-to-all), simulated annealing (SA) and BRIM explored 148K and 115K different states respectively to arrive at comparable solution quality. In BRIM, on average, there is a spin flip every 20ps. In (sequential) SA, flipping individual spins is achieved computationally: the energy of an alternative configuration (with a particular spin flipped) is calculated and based on the energy, the new state is probabilistically accepted. Roughly speaking, we count 140,000 instructions executed per spin flip running SA [29].

Simulated bifurcation (SB) is an entirely new computational approach. It can be thought of as simulating a dynamical system. Thanks to its algorithm design, it is easier to parallelize. Thus, despite having similar workload, it can be faster.[7] Nevertheless, accelerating SB to the level of BRIM would require about 1000x more computational throughput or about 2 Peta Ops per second. We can see why physical Ising machines are more attractive even compared to the best computational accelerator.

6.4.2 Effects of bandwidth optimizations. As already discussed in Sec. 5.4, the degree of global state ignorance can have significant impact on solution quality. Thus, in concurrent mode, we need to let solvers frequently update each other. In batch mode, we can tolerate much longer epochs and only need to communicate cumulative state changes between the beginning and end of the same epoch (which we call *bit change* to differentiate from spin flips). If a spin flips, say, 4 times in an epoch, its state will end up the same as at the beginning of the epoch, and we do not need to communicate anything. In other words, there are 4 spin flips but 0 bit change. Intuitively, the longer the epoch, the higher the fraction of spin flips will result in no bit change. Fig. 13 confirms this intuition quantitatively.

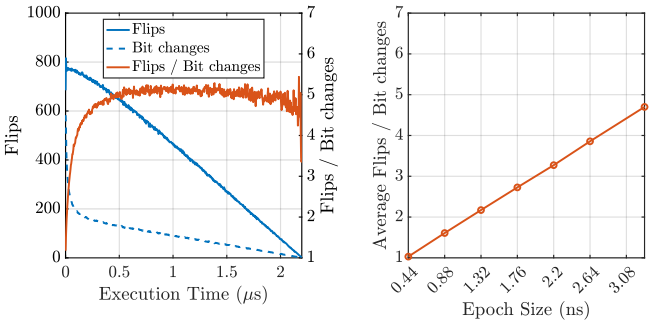

Figure 13: [Left] Evolution of flips and bit changes over time for a 4 chip BRIM with a fixed epoch size of 3.3 ns. The left vertical axis corresponds to flips (solid blue line) and bit changes (dashed blue line). The right vertical axis corresponds to the ratio of flips to bit changes shown in red. [Right] Correlation of the average ratio of flips to bit changes with epoch size. The ratio increases almost linearly with increasing epoch size.

We measure the number of spin flips during an epoch and count the number of bit changes. In Fig. 13 (left), we show both numbers and their ratio as the annealing proceeds. We see that for a fixed epoch size, the ratio is rather stable after an initial period. Fig. 13 (right) shows the ratio as a function of different epoch sizes. Not surprisingly, the longer the epoch the higher the ratio. As we can see, if we can use an epoch size of about 3 ns, we can reduce traffic demand by around 4-5x compared to using sub-nanosecond epochs.

Increasing epoch size does degrade solution quality as can be seen in Fig. 14, where we show the solution quality as a function of epoch size. We can see the best solution quality is achieved

[7] A non-trivial portion of SA is also massively parallel. However, there is no effort for custom-hardware implementation of parallel SA.

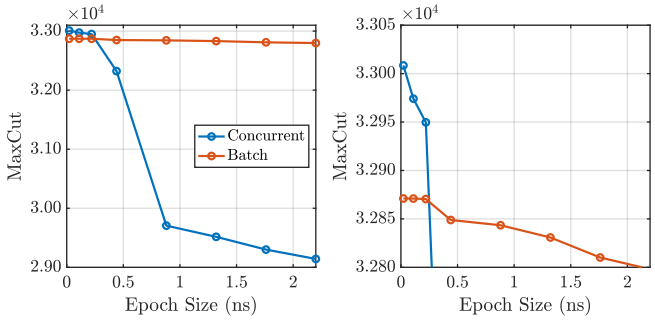

Figure 14: [Left] Average MaxCut solution for different epochs in concurrent and batch mode. [Right] Magnified y axis.

with concurrent mode with a small epoch size. When bandwidth is sufficient, this is the best mode to use. In a bandwidth-bound system, the dynamical system needs to be slowed down. In that case, 4-5x traffic reduction means the dynamical system can run about 4-5x faster. Achieving traffic reduction in turn requires longer epochs. In such a case, we see that the concurrent mode does not tolerate longer epochs well and the solution quality drops quickly and significantly. Batch mode, on the other hand, is much more tolerant to longer epochs as the solution quality reduces but only very slightly. Thus, in a bandwidth-bound system, batch mode will be very useful in not sacrificing execution speed while maintaining high solution quality.

Finally, we look at the bandwidth reduction when coordinating induced spin flips (Sec. 5.4.2). Fig. 15 (left) shows the amount of bit changes and induced spin flips with the evolution of time. The percentage of bit changes due to induced spin flips is also plotted. Of course, the value is a function of epoch size. Fig. 15 (right) shows the average percentage with different epoch sizes. Clearly a non-trivial amount of communication (30-38%) can be saved with the optimization of coordinating induced flips for both concurrent and batch modes. In a bandwidth constrained system, a corresponding improvement in execution time (about 1.5x) can be expected.

6.5 Contrast with other parallel processing

Finally, we note that communication among distributed agents is clearly a common component and performance bottleneck in parallel processing. Thus, in exploring solutions for Ising machines, we may have reinvented some wheels. For instance, using shadow copies is a necessity for us the same way keeping copies (ghosts) of non local neighbors is in parallel algorithms [7, 20, 34, 39]. Also, techniques of reducing communication while limiting performance consequences have been explored in different contexts: sending lower precision data [5] – sometimes just 1 bit [10, 42], using lossy compression [1], reducing the number of elements transmitted [4, 21, 44, 50, 53] or even skipping rounds [8, 35, 43]. Compared to these situations, two key differences can be highlighted specifically for the case of Ising machines:

(1) Scale of the problem: Ising machines are dynamical systems and can evolve extremely quickly. They thus present enormous raw communication demand without optimizations.

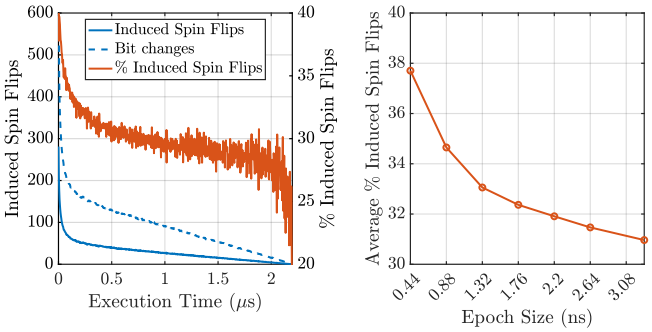

Figure 15: [Left] Evolution of induced spin flips and bit changes over time for a 4 chip BRIM with a fixed epoch size of 3.3 ns. The left vertical axis corresponds to induced spin flips (solid blue line) and bit changes (dashed blue line). The right vertical axis corresponds to the percentage of bit changes that are induced spin flips (shown in red). [Right] Correlation of the average percentage of induced spin flips with epoch size.

For instance our assumed 4 BRIM chips (each about 80 mm^2 in 45 nm technology) would need about 4 TB/s.

(2) Design flexibility: Optimizations such as batch mode and coordinating induced spin flips are possible because we can exploit the freedom to orchestrate the evolution process of the underlying Ising machine. As a result, there is no additional logic such as compression and yet we can maintain solution quality while reducing communication demand to only 218 GB/s (20x reduction).

7 CONCLUSIONS

Physical Ising machines can solve Ising formula optimization problems with extreme speeds and energy efficiencies, even when compared with special-purpose (von Neumann) accelerators. However, existing Ising machines have a fixed capacity. If we simply use a divide-and-conquer strategy, the benefit of using Ising machine reduces quickly when the problem is even slightly bigger than the machine capacity. Instead we need machines that are fundamentally designed to cooperate with other machines to solve a larger problem. In this paper, we have presented the architectural design and optimizations of a *multiprocessor Ising machine*. Our analysis of the design can be summarized into a few key takeaway points:

(1) Even under conservative conditions we can see that the multiprocessor can achieve about 2200x speedup compared to the state-of-the-art computational accelerator.

(2) With the proposed multiprocessor architecture, a physical Ising machine can now also scale up and solve bigger problems. While it is difficult to compare speedups over different problems, it is safe to say that the performance advantage of a multiprocessor BRIM over its von Neumann counterpart is as significant as in the case of single-chip BRIM.

(3) Given the extreme speed of a physical Ising machine, communication bandwidth is likely the performance bottleneck and the machine dynamics need to slow down correspondingly. In these cases, our proposed batch mode operation can lead to about 4-5x reduction in communication demand, translating to about 4-5x improvement in processing throughput.

ACKNOWLEDGMENTS

This work has been supported by a University Research Award from the University of Rochester, by New York State Center of Excellence in Data Science at the University of Rochester under Project 1689dC13, and also in part by DARPA under Agreement No. HR00112090012.

Appendix A DIVIDE-AND-CONQUER ALGORITHMS

Users interact with D-Wave systems using the Solver API (SAPI) over network. A job is submitted to a SAPI server queue. Jobs are then assigned to workers, which run on a conventional processor and are responsible for submitting instructions to the quantum processor, receiving results from the quantum processor, post-processing results when necessary, and sending results back to user.

Algorithm 1 D-WAVE's d-n-c algorithm [13]

```
1: Input: QUBO instance
2: # Vbest, lowest value found to date
3: # Qbest, solution bit vector corresponding to the lowest value so far
4: # index, indices of the bits in the solution
5:
6: # Get initial estimate of minimum value and backbone
7: Qtmp ← random 0/1 vector
8: (Vbest, Qbest) ← TabuSearch(QUBO, Qtmp)
9: index ← OrderByImpact(QUBO, Qbest)
10: passCount ← 0
11: Qtmp ← Qbest
12: total ← fraction * size(QUBO)
13:
14: while passCount < numRepeats do
15:     for i = 0; i < total; i = i + subQuboSize do
16:         # select subQubo with other variables clamped
17:         subQubo ← Clamp(QUBO, Qtmp, index[i : i + subQuboSize − 1])
18:         (subV, subQ) ← DWaveSearch(subQubo)
19:         # project onto full solution
20:         Qtmp[index[i : i + subQuboSize − 1]] ← subQ
21:     end for
22:     (V, Qnew) ← TabuSearch(QUBO, Qtmp)
23:     index ← OrderByImpact(QUBO, Qnew)
24:     if V < Vbest then
25:         Vbest ← V; Qbest ← Qnew
26:         passCount ← 0
27:     else if V == Vbest then
28:         Qbest ← Qnew
29:         passCount + +
30:     else
31:         passCount + +
32:     end if
33:     Qtmp ← Qnew
34: end while
35: Output: Vbest, Qbest
```

The general idea of the algorithm (shown as Algorithm 1 here, replicated from [13]) is straightforward: If part of the state remains fixed, then the original QUBO problem is converted into a smaller sub-problem (*subQubo* in line 15) that can be launched on a solver (line 18). Repeating this action over different portions of the state vector (lines 15 to 21) constitutes one pass of the algorithm. Multiple passes are performed (while loop starting in line 14) to achieve a better result. Shown in Algorithm 2 is our approach which is a bit more efficient.

Algorithm 2 Our d-n-c algorithm

Input: $Graph$
\# $V \leftarrow$ random 0/1 spin vector
\# $numSolvers \leftarrow$ number of solvers
\# $numRepeats \leftarrow$ number of times to repeat
\# $epoch \leftarrow$ Epoch times for each solver
\# $ratio \leftarrow$ ratio in which the graph is to be partitioned

$(subG, subV) \leftarrow RandPartition(Graph, V, ratio)$
$ene \leftarrow IsingEnergy(subG, subV)$
for $r = 0; r < numRepeats; ++r$ **do**
  for $i = 0; i < numSolvers; ++i$ **do**
    # Launch solvers (Can be parallel)
    $Solver(subG[i], subV[i], epoch[i], ene[i])$
  end for
  $Synchronise(subG, subV, ene)$
end for

$V \leftarrow copy(subV)$
$FinalEne \leftarrow IsingEnergy(Graph, V)$
Output: $V, FinalEne$

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Gregory S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian J. Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Józefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Gordon Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul A. Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda B. Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *CoRR* abs/1603.04467 (2016). arXiv:1603.04467 http://arxiv.org/abs/1603.04467

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

[3] Richard Afoakwa, Yiqiao Zhang, Uday Kumar Reddy Vengalam, Zeljko Ignjatovic, and Michael Huang. 2021. BRIM: Bistable Resistively-Coupled Ising Machine. *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)* (2021), 749–760. https://doi.org/10.1109/HPCA51647.2021.00068

[4] Dan Alistarh, Torsten Hoefler, Mikael Johansson, Sarit Khirirat, Nikola Konstantinov, and Cédric Renggli. 2018. The Convergence of Sparsified Gradient Methods. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 5977–5987.

[5] Dan Alistarh, Jerry Li, Ryota Tomioka, and Milan Vojnovic. 2016. QSGD: Randomized Quantization for Communication-Optimal Stochastic Gradient Descent. *CoRR* abs/1610.02132 (2016). arXiv:1610.02132 http://arxiv.org/abs/1610.02132

[6] Rami Barends, Alireza Shabani, Lucas Lamata, Julian Kelly, Antonio Mezzacapo, Urtzi Las Heras, Ryan Babbush, Austin G Fowler, Brooks Campbell, Yu Chen, et al. 2016. Digitized adiabatic quantum computing with a superconducting circuit. *Nature* 534, 7606 (2016), 222–226.

[7] William J. Barry, Mark T. Jones, and Paul E. Plassmann. 1998. Parallel adaptive mesh refinement techniques for plasticity problems. *Advances in Engineering Software* 29, 3 (1998), 217–225. https://doi.org/10.1016/S0965-9978(98)00040-4

[8] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. 2019. *Qsparse-Local-SGD: Distributed SGD with Quantization, Sparsification, and Local Computations*. Curran Associates Inc., Red Hook, NY, USA.

[9] Natalia G Berloff, Matteo Silva, Kirill Kalinin, Alexis Askitopoulos, Julian D Töpfer, Pasquale Cilibrizzi, Wolfgang Langbein, and Pavlos G Lagoudakis. 2017. Realizing the classical XY Hamiltonian in polariton simulators. *Nature materials* 16, 11 (2017), 1120–1126.

[10] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. 2018. signSGD: compressed optimisation for non-convex problems. *CoRR* abs/1802.04434 (2018). arXiv:1802.04434 http://arxiv.org/abs/1802.04434

[11] Fabian Böhm, Guy Verschaffelt, and Guy Van der Sande. 2019. A poor man's coherent Ising machine based on opto-electronic feedback systems for solving optimization problems. *Nature Communications* 10, 1 (2019), 3538. https://doi.org/10.1038/s41467-019-11484-3

[12] Sergio Boixo, Troels F Rønnow, Sergei V Isakov, Zhihui Wang, David Wecker, Daniel A Lidar, John M Martinis, and Matthias Troyer. 2014. Evidence for quantum annealing with more than one hundred qubits. *Nature physics* 10, 3 (2014), 218–224.

[13] Michael Booth, Steven P. Reinhardt, and Aidan Roy. 2017. Partitioning Optimization Problems for Hybrid Classical/Quantum Execution. *Technical Report* (2017). https://docs.ocean.dwavesys.com/projects/qbsolv/en/latest/_downloads/bd15a2d8f32e587e9e5997ce9d5512cc/qbsolv_techReport.pdf

[14] Paul I Bunyk, Emile M Hoskinson, Mark W Johnson, Elena Tolkacheva, Fabio Altomare, Andrew J Berkley, Richard Harris, Jeremy P Hilton, Trevor Lanting, Anthony J Przybysz, et al. 2014. Architectural considerations in the design of a superconducting quantum annealing processor. *IEEE Transactions on Applied Superconductivity* 24, 4 (2014), 1–10.

[15] KyungHyun Cho, Alexander Ilin, and Tapani Raiko. 2011. Improved learning of Gaussian-Bernoulli restricted Boltzmann machines. In *International conference on artificial neural networks*. Springer, 10–17.

[16] Jeffrey Chou, Suraj Bramhavar, Siddhartha Ghosh, and William Herzog. 2019. Analog Coupled Oscillator Based Weighted Ising Machine. *Scientific Reports* 9, 1 (2019), 14786. https://doi.org/10.1038/s41598-019-49699-5

[17] Chase Cook, Hengyang Zhao, Takashi Sato, Masayuki Hiromoto, and Sheldon X. D. Tan. 2019. GPU Based Parallel Ising Computing for Combinatorial Optimization Problems in VLSI Physical Design. arXiv:1807.10750 [physics.comp-ph]

[18] D-WAVE. 2014. minorminer. https://github.com/dwavesystems/minorminer

[19] D-WAVE. 2022. Operation and Timing. https://docs.dwavesys.com/docs/latest/c_qpu_timing.html

[20] Zachary DeVito, Niels Joubert, Francisco Palacios, Stephen Oakley, Montserrat Medina, Mike Barrientos, Erich Elsen, Frank Ham, Alex Aiken, Karthik Duraisamy, Eric Darve, Juan Alonso, and Pat Hanrahan. 2011. Liszt: A domain specific language for building portable mesh-based PDE solvers. In *SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12. https://doi.org/10.1145/2063384.2063396

[21] Melih Elibol, Lihua Lei, and Michael I. Jordan. 2020. Variance Reduction with Sparse Gradients. *CoRR* abs/2001.09623 (2020). arXiv:2001.09623 https://arxiv.org/abs/2001.09623

[22] Hayato Goto, Kotaro Endo, Masaru Suzuki, Yoshisato Sakai, Taro Kanao, Yohei Hamakawa, Ryo Hidaka, Masaya Yamasaki, and Kosuke Tatsumura. 2021. High-performance combinatorial optimization based on classical mechanics. *Science Advances* 7, 6 (2021), eabe7953. https://doi.org/10.1126/sciadv.abe7953 arXiv:https://www.science.org/doi/pdf/10.1126/sciadv.abe7953

[23] Hidenori GYOTEN, Masayuki HIROMOTO, and Takashi SATO. 2018. Area Efficient Annealing Processor for Ising Model without Random Number Generator. *IEICE Transactions on Information and Systems* E101.D, 2 (2018), 314–323. https://doi.org/10.1587/transinf.2017RCP0015

[24] Ryan Hamerly, Takahiro Inagaki, Peter L McMahon, Davide Venturelli, Alireza Marandi, Tatsuhiro Onodera, Edwin Ng, Carsten Langrock, Kensuke Inaba, et al. 2018. Scaling advantages of all-to-all connectivity in physical annealers: the Coherent Ising Machine vs. D-Wave 2000Q. *D-Wave 2000Q arXiv* (2018).

[25] Ryan Hamerly, Takahiro Inagaki, Peter L McMahon, Davide Venturelli, Alireza Marandi, Tatsuhiro Onodera, Edwin Ng, Carsten Langrock, Kensuke Inaba, Toshimori Honjo, et al. 2019. Experimental investigation of performance differences between coherent Ising machines and a quantum annealer. *Science advances* 5, 5 (2019), eaau0823.

[26] R Hamerly, A Sludds, L Bernstein, M Prabhu, C Roques-Carmes, J Carolan, Y Yamamoto, M Soljačić, and D Englund. 2019. Towards Large-Scale Photonic Neural-Network Accelerators. In *2019 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 22–8.

[27] R. Harris, M. W. Johnson, T. Lanting, A. J. Berkley, J. Johansson, P. Bunyk, E. Tolkacheva, E. Ladizinsky, N. Ladizinsky, T. Oh, F. Cioata, I. Perminov, P. Spear, C. Enderud, C. Rich, S. Uchaikin, M. C. Thom, E. M. Chapple, J. Wang, B. Wilson, M. H. S. Amin, N. Dickson, K. Karimi, B. Macready, C. J. S. Truncik, and G. Rose. 2010. Experimental investigation of an eight-qubit unit cell in a superconducting optimization processor. *Phys. Rev. B* 82 (Jul 2010), 024511. Issue 2. https://doi.org/10.1103/PhysRevB.82.024511

[28] Takahiro Inagaki, Yoshitaka Haribara, Koji Igarashi, Tomohiro Sonobe, Shuhei Tamate, Toshimori Honjo, Alireza Marandi, Peter L McMahon, Takeshi Umeki, Koji Enbutsu, et al. 2016. A coherent Ising machine for 2000-node optimization problems. *Science* 354, 6312 (2016), 603–606.

[29] S.V. Isakov, I.N. Zintchenko, T.F. Rønnow, and M. Troyer. 2015. Optimised simulated annealing for Ising spin glasses. *Computer Physics Communications* 192 (Jul 2015), 265–271. https://doi.org/10.1016/j.cpc.2015.02.015

[30] Richard M. Karp. 1972. *Reducibility among Combinatorial Problems*. Springer US, Boston, MA, 85–103. https://doi.org/10.1007/978-1-4684-2001-2_9

[31] Kihwan Kim, M-S Chang, Simcha Korenblit, Rajibul Islam, Emily E Edwards, James K Freericks, G-D Lin, L-M Duan, and Christopher Monroe. 2010. Quantum simulation of frustrated Ising spins with trapped ions. *Nature* 465, 7298 (2010), 590–593.

[32] Andrew D King, Juan Carrasquilla, Jack Raymond, Isil Ozfidan, Evgeny Andriyash, Andrew Berkley, Mauricio Reis, Trevor Lanting, Richard Harris, Fabio Altomare, et al. 2018. Observation of topological phenomena in a programmable lattice of 1,800 qubits. *Nature* 560, 7719 (2018), 456–460.

[33] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680. https://doi.org/10.1126/science.220.4598.671 arXiv:https://science.sciencemag.org/content/220/4598/671.full.pdf

[34] Orion S. Lawlor, Sayantan Chakravorty, Terry L. Wilmarth, Nilesh Choudhury, Isaac Dooley, Gengbin Zheng, and Laxmikant V. Kalé. 2006. ParFUM: A Parallel Framework for Unstructured Meshes for Scalable Dynamic Physics Applications.

Eng. with Comput. 22, 3 (dec 2006), 215–235.

[35] Tao Lin, Sebastian U. Stich, and Martin Jaggi. 2018. Don't Use Large Mini-Batches, Use Local SGD. *CoRR* abs/1808.07217 (2018). arXiv:1808.07217 http://arxiv.org/abs/1808.07217

[36] Andrew Lucas. 2014. Ising formulations of many NP problems. *Frontiers in Physics* 2 (2014), 5.

[37] Peter L. McMahon, Alireza Marandi, Yoshitaka Haribara, Ryan Hamerly, Carsten Langrock, Shuhei Tamate, Takahiro Inagaki, Hiroki Takesue, Shoko Utsunomiya, Kazuyuki Aihara, Robert L. Byer, M. M. Fejer, Hideo Mabuchi, and Yoshihisa Yamamoto. 2016. A fully programmable 100-spin coherent Ising machine with all-to-all connections. *Science* 354, 6312 (2016), 614–617. https://doi.org/10.1126/science.aah5178 arXiv:https://science.sciencemag.org/content/354/6312/614.full.pdf

[38] Chris Mellor. 2021. DRAM, it stacks up: SK hynix rolls out 819 GB/s HBM3 tech. https://www.theregister.com/2021/10/20/sk_hynix_hbm3/

[39] Misbah Mubarak, Seegyoung Seol, Qiukai Lu, and Mark S. Shephard. 1900. A Parallel Ghosting Algorithm for The Flexible Distributed Mesh Database. *Scientific Programming* 21 (01 Jan 1900), 654971. https://doi.org/10.3233/SPR-130361

[40] Saavan Patel, Lili Chen, Philip Canoza, and Sayeef Salahuddin. 2020. Ising Model Optimization Problems on a FPGA Accelerated Restricted Boltzmann Machine. arXiv:2008.04436 [cs.AR]

[41] D Pierangeli, G Marcucci, and C Conti. 2019. Large-scale photonic Ising machine by spatial light modulation. *Physical review letters* 122, 21 (2019), 213902.

[42] Frank Seide, Hao Fu, Jasha Droppo, Gang Li, and Dong Yu. 2014. 1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs. In *Interspeech 2014* (interspeech 2014 ed.). https://www.microsoft.com/en-us/research/publication/1-bit-stochastic-gradient-descent-and-application-to-data-parallel-distributed-training-of-speech-dnns/

[43] Sebastian U. Stich. 2019. Local SGD Converges Fast and Communicates Little. In *International Conference on Learning Representations*. https://openreview.net/forum?id=S1g2JnRcFX

[44] Sebastian U. Stich, Jean-Baptiste Cordonnier, and Martin Jaggi. 2018. Sparsified SGD with Memory. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 4452–4463.

[45] Kenta Takata, Alireza Marandi, Ryan Hamerly, Yoshitaka Haribara, Daiki Maruo, Shuhei Tamate, Hiromasa Sakaguchi, Shoko Utsunomiya, and Yoshihisa Yamamoto. 2016. A 16-bit Coherent Ising Machine for One-Dimensional Ring and Cubic Graph Problems. *Scientific Reports* 6, 1 (2016), 34089. https://doi.org/10.1038/srep34089

[46] Y Takeda, S Tamate, Y Yamamoto, H Takesue, T Inagaki, and S Utsunomiya. 2017. Boltzmann sampling for an XY model using a non-degenerate optical parametric oscillator network. *Quantum Science and Technology* 3, 1 (nov 2017), 014004. https://doi.org/10.1088/2058-9565/aa923b

[47] T. Takemoto, M. Hayashi, C. Yoshimura, and M. Yamaoka. 2019. 2.6 A 2 by 30k-Spin Multichip Scalable Annealing Processor Based on a Processing-In-Memory Approach for Solving Large-Scale Combinatorial Optimization Problems. In *IEEE International Solid- State Circuits Conference*.

[48] Kosuke Tatsumura, Alexander R. Dixon, and Hayato Goto. 2019. FPGA-Based Simulated Bifurcation Machine. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 59–66. https://doi.org/10.1109/FPL.2019.00019

[49] Kosuke Tatsumura, Masaya Yamasaki, and Hayato Goto. 2021. Scaling out Ising machines using a multi-chip architecture for simulated bifurcation. *Nature Electronics* 4, 3 (01 Mar 2021), 208–217. https://doi.org/10.1038/s41928-021-00546-4

[50] Hongyi Wang, Scott Sievert, Zachary Charles, Shengchao Liu, Stephen Wright, and Dimitris Papailiopoulos. 2018. ATOMO: Communication-Efficient Learning via Atomic Sparsification. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 9872–9883.

[51] Tianshi Wang and Jaijeet Roychowdhury. 2019. OIM: Oscillator-Based Ising Machines for Solving Combinatorial Optimisation Problems. arXiv:1903.07163 [cs.ET]

[52] Tianshi Wang, Leon Wu, and Jaijeet Roychowdhury. 2019. New Computational Results and Hardware Prototypes for Oscillator-Based Ising Machines. In *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA) *(DAC '19)*. Association for Computing Machinery, New York, NY, USA, Article 239, 2 pages. https://doi.org/10.1145/3316781.3322473

[53] Jianqiao Wangni, Jialei Wang, Ji Liu, and Tong Zhang. 2018. Gradient Sparsification for Communication-Efficient Distributed Optimization. In *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2018/file/3328bdf9a4b9504b9398284244fe97c2-Paper.pdf

[54] Kasho Yamamoto, Kazushi Kawamura, Kota Ando, Normann Mertig, Takashi Takemoto, Masanao Yamaoka, Hiroshi Teramoto, Akira Sakai, Shinya Takamaeda-Yamazaki, and Masato Motomura. 2021. STATICA: A 512-Spin 0.25M-Weight Annealing Processor With an All-Spin-Updates-at-Once Architecture for Combinatorial Optimization With Complete Spin–Spin Interactions. *IEEE Journal of Solid-State Circuits* 56, 1 (2021), 165–178. https://doi.org/10.1109/JSSC.2020.3027702

[55] Yoshihisa Yamamoto, Kazuyuki Aihara, Timothee Leleu, Ken-ichi Kawarabayashi, Satoshi Kako, Martin Fejer, Kyo Inoue, and Hiroki Takesue. 2017. Coherent Ising machines—Optical neural networks operating at the quantum limit. *npj Quantum Information* 3, 1 (2017), 1–15.

[56] Masanao Yamaoka, Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, Hidetaka Aoki, and Hiroyuki Mizuno. 2015. A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing. *IEEE Journal of Solid-State Circuits* 51, 1 (2015), 303–309.

[57] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno. 2015. 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*. 1–3. https://doi.org/10.1109/ISSCC.2015.7063111

[58] Chihiro Yoshimura, Masato Hayashi, Takuya Okuyama, and Masanao Yamaoka. 2017. Implementation and Evaluation of FPGA-based Annealing Processor for Ising Model by use of Resource Sharing. *International Journal of Networking and Computing* 7, 2 (2017), 154–172. http://www.ijnc.org/index.php/ijnc/article/view/148

[59] G Zames, NM Ajlouni, NM Ajlouni, NM Ajlouni, JH Holland, WD Hills, and DE Goldberg. 1981. Genetic algorithms in search, optimization and machine learning. *Information Technology Journal* 3, 1 (1981), 301–302.