Ising Model Processors on a Spatial Computing Architecture

Yanze Wu
Cyber Security Engineering Department
George Mason University
Fairfax, VA, USA
yzwu42@gmu.edu

Md Tanvir Arafin
Cyber Security Engineering Department
George Mason University
Fairfax, VA, USA
marafin@gmu.edu

***Abstract*—Data-flow-driven spatial computing architectures are emerging to enable efficient acceleration of complex machine learning models at the edge devices. Interestingly, their potential in other domains of computing is yet to be thoroughly explored. Hence, this paper investigates the application of spatial architectures for designing reconfigurable Ising model processors on edge devices. We target AMD's Versal Adaptive SoCs platform and implement Ising model processors using the parallelization opportunities in the VLIW-supported vector processors arranged in a spatial configuration. Our experiments on a VCK-190 evaluation platform demonstrate that spatial computing implementation for the standard Metropolis algorithm achieves $3\times$ to $9\times$ speedup compared with a generic ARM Cortex A-72 processor for varying matrix sizes and precision. We also present implementation issues for design accelerators on Versal ASoCs. The code and artifacts for the work are available at https://github.com/SPIRE-GMU/Ising-AIE for reproducing the experiments and results presented in this work.**

***Index Terms*—Ising model, spatial computing architecture, hardware accelerator design, Versal Adaptive SoCs.**

I. Introduction

NP-hard combinatorial optimizations constitute some of the most critical problems in computer science, with implications in a broad spectrum of applications ranging from optimal routing for modern semiconductor design to explaining the physical properties of materials. Thus, accelerating the solutions to such problems is of great interest. Unfortunately, such problems are complex by definition, and therefore, conventional von Neumann architecture suffers from fundamental processing and memory bottlenecks even in running optimized heuristics for solving NP-hard problems [1]–[3].

One such problem is the Ising spin glass model, or, in short, the Ising model. The Ising model is primarily used to describe the ferromagnetic properties of materials. Interestingly, an Ising model is NP-complete [4] and provides a standard benchmark for understanding computing issues for complex combinatorial problems. Although there exists heuristics, such as the nearest neighbor interaction and Metropolis algorithm, to find the approximate solution for the 2-D Ising model, the overall structure of the problem provides a standard benchmark for understanding acceleration for complex combinatorial problems. As a result, there has been significant research on the accelerator designs for the Ising model using different hardware platforms, such as parallel processors [5], graphic processor units (GPU) [3], field-programmable gate arrays (FPGA) [2], [6], [7] and even quantum computers [1].

Some Ising model accelerators, such as quantum processors or massively parallel compute clusters, are designed for specific applications, *i.e.,* large-scale physics modeling and network simulation, and are not applicable in low-power compute nodes. Thus, finding solutions for combinatorial problems in real-time on a low-power budget-restricted edge-computing node remains a challenging problem [2], [6], [8]. There have been proposals for building ASICs and using them as a co-processor for such tasks [9]; however, such ASIC co-processors or accelerators are fixed in terms of the problem size and not reconfigurable with the changing problem size.

Interestingly, there has been a paradigm shift in accelerator design primarily driven by the increasing size and complexity of the machine learning (ML) workload. Although GPUs remain the *de-facto* driver for most of the ML tasks, more advanced architectures such as Groq's Tensor Streaming chips [10], Tenstorrent's GraySkull processors [11], and AMD's Versal AI Engines [12] have demonstrated better potential in ML compute acceleration for non-trivial complex models. These devices are built on a fundamental design concept: spatial computing architecture (SCA). The SCA-based designs utilize a 2-D grid of a large number of small processing elements to support temporal and spatial parallelization of the workload. In addition, SCA-based designs often provide low-power processing capabilities compared to their GPU counterparts. As a result, this data-flow-oriented design paradigm offers new opportunities for accelerating computation-intensive tasks, such as solving complex combinatorial optimization on edge devices.

Hence, in this work, we present a Versal Adaptive SoCs-based Ising model processor to study the effectiveness of using spatial computing architecture to solve combinatorial optimization problems. To our knowledge, *this is the first implementation* of the Ising model on a spatial computing architecture. The key contribution of the work can be summarized as follows:

C1. We demonstrate the detailed design process for efficiently implementing the Ising model on the AI engine tiles of the Xilinx VCK-190 platform. Our design uses

traditional parallelization techniques and exploits spatial parallelization capabilities available in the Ising model.

C2. We evaluate the performance of the optimized spatial accelerator in terms of latency and power consumption. Our experiments show that, compared with the embedded ARM cores, spatial accelerators can provide a significant performance boost with the increasing size of the Ising model.

C3. We also discuss the limitations of the current spatial architectures, specifically for AI engines in the Versal Adaptive SoCs in compute acceleration for combinatorial problems.

C4. We have made the code and artifacts for this work available at https://github.com/SPIRE-GMU/Ising-AIE. This repository can be used to reproduce our experiments and explore parallelization techniques presented in this work.

The following sections are presented in the given order. First, Section II discusses the Ising model and the basics of the Versal Adaptive SoCs. Then, Section III presents the detailed Ising model solver design on the Versal AI engine. Next, Section IV details our experimental setup along with the results. After that, critical implementation issues for such designs are presented in Section V. Finally, Section VI concludes the paper.

II. Background

A. The Ising Model

The Lenz-Ising model (also known as the Ising model) [13] describes the ferromagnetic properties, such as the phase transition mechanism of materials. A ferromagnetic material can be considered a collection of interacting spin units, as shown in Figure 1. In the Ising model, these spin units are arranged in a lattice structure – for example, a 2-dimensional configuration of the spin units constitutes the 2-D Ising model. For a grid of N-lattice points, a spin unit (σ_i) resides at each grid point, which can be either at a spin up ($+1$) or spin down (-1) configuration, *i.e.,* $\sigma_i \in \{-1, +1\}$, where $i = 1, \ldots, N$. Then, the energy for a spin unit configuration can be calculated using the local Hamiltonian,

$$H(\sigma_i) = \sum_{j=1}^{N} J_{i,j}\sigma_i\sigma_j - h_i(\sigma_i) \tag{1}$$

Here, $J_{i,j}$ is the interaction between i^{th} and j^{th} spin units and h_i represents the external magnetic field or bias the i^{th} spin unit. Now, by aggregating the local energy for all the lattice points in a structure, the total energy of the configuration is represented with the global Hamiltonian as,

$$H = \sum_{<i,j>} J_{i,j}\sigma_i\sigma_j - \sum_{i}^{N} h_i(\sigma_i) \tag{2}$$

For a given lattice structure or arrangement S, the Ising model finds the spin configurations, *i.e.,* values for $\sigma_1, \sigma_2, \ldots, \sigma_N$, that yield the minimum total energy (also known as the ground state) for the structure.

The Ising model is an NP-complete problem [4]. Interestingly, since it is an NP-complete problem, a class of combinatorial NP-hard problems can be reduced to the Ising model. For example, the max-cut problem can be reduced to the Ising model by setting $h = 0$ in Equation 3, and considering that the interaction parameter J represents the edges, and the σs represent the nodes of a graph G. Then, finding the ground state becomes the equivalent of finding the solution of the max-cut problem for G [3]. Similar reductions can also be performed for other NP-hard problems [14].

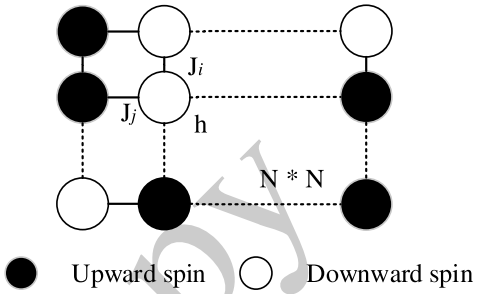

Fig. 1. The 2-D Ising model. Black blocks are the upward spin units, and white blocks are the downward ones.

Since the 2-D Ising model is NP-complete, finding a tractable solution for the Ising model requires approximations and heuristics. One such heuristic is the nearest-neighbor-based Metropolis algorithm, which can be described as follows. To solve for the ground state of a structure S, we can start with an initial random spin configuration of the structure and assume the nearest-neighbor interactions only. Then, we update each lattice point so that the total energy of S is reduced at each step. This can be done sequentially – by considering the interaction of each lattice point p_i with the others and then setting the spin σ_i so that the local Hamiltonian for p_i is reduced. To do so, we can take the negative of the first order derivative for the local Hamiltonian, *i.e.,*

$$-\frac{\partial H_i}{\partial \sigma_i} = -\sum_{j} J_{i,j}\sigma_j - h_i \tag{3}$$

If $-\frac{\partial H_i}{\partial \sigma_i} > 0$, then the spin of the unit (*i.e.,* σ_i) is flipped; otherwise it remain as before. This spin update is computed over the entire S to find the next configuration demonstrating lower energy than the previous configuration. A solution is found when this process converges to the lowest energy.

B. Versal Adaptive SoCs

AMD's Versal Adaptive SoCs present the newest generation of reconfigurable spatial computing products. This product line features a 2-D grid of up to 400 adaptive intelligent engines (AIEs). These AIEs are fundamentally different from the GPU SIMD cores since each of the AIEs operates individually and can be networked together for a complex task. Each AIE contains a 7-way VLIW vector processor, a scalar processor, and a local memory unit. In addition, the Versal devices also contain FPGA cores, reconfigurable network-on-chips (NoCs),

and two full-fledged ARM processors to support real-time operation and data movement, as shown in Figure 2. The overall design of the device targets edge applications and is optimized for power efficiency.

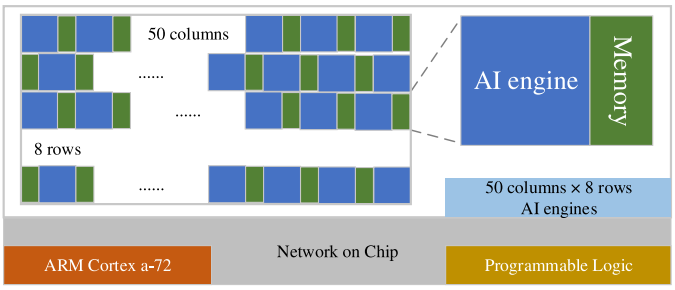

Fig. 2. The architecture of AMD-Xilinx Versal Adaptive System on Chip (ASoC). Each chip contains a grid of 8×50 adaptive intelligent engine (AIE) tiles, programmable FPGA logic, and two ARM processors.

III. Design of Ising Model Accelerators on a Spatial Architecture

To find a solution for the Ising model, we start with a random distribution of spins. Considering the nearest neighbor approximation, we notice that non-neighboring lattice points are independent, and their spin updates can be computed in parallel. This leads to a checkerboard formation [15] where the complete grid can be divided into black and white points, and computation of the spin updates for the same colored grid points can be done in parallel. We have exploited this parallelism using single and multiple AI engines.

First, consider the simplest case with a serial processor [16]. Let's consider a $n \times n$ checkerboard. We can update the non-neighboring $n^2/2$ lattice points serially for the first half of the iteration and then the rest in the second half. Assuming each spin update computation takes t time on a serial processor, it will take tn^2 for one iteration pass.

Algorithm 1 represents the serial implementation. It first initializes an M×M lattice with random value selected from $\{1, -1\}$, representing a random initial spin distribution in a system. Next, the system begins to update the state of atoms in n iterations. Every time updating the state, it traverses each grid point and accumulates energy and magnetization.

Now, considering the case of an l-way VLIW vector processor, we can vectorize the checkerboard algorithm. Since the same colored grid points are independent, one can issue spin updates for l-grid points in parallel. Assuming each spin update computation takes t' time on a vector processor, for this scenario, it will take $t'n^2/l$ time for one iteration pass. Assuming $t' \approx t$, this will provide linear speedup with the increasing number of l. Therefore, we also implemented the parallel checkerboard algorithm using the vector processors in a single AI engine. Details on the AIE-specific intrinsic-based codes for the implementation are presented in Figure 4.

Spatial acceleration using the 2-D arrangement of the AIE tiles is one of the critical features of the AI engine. To leverage this, we divide the complete grid into smaller sub-grids and

Algorithm 1 Ising Model

```
Input: Grid length (M)
Output: Energy, Magnetization
Vol = M × M
S[Vol] ← {−1, 1}                                  ▷ Initialize grid
Xup[Vol], Xdn[Vol], Yup[Vol], Ydn[Vol] ← S[Vol]
                          ▷ Initialize neighboring cell of each grid
double Magnetization, Energy = 0.0
while n < iteration do
  double mags, esum = 0.0
  for i = 0,...,Vol-1 do
    energy_current, energy_updated, delta = 0.0
    neighbor = Xup[i] + Xdn[i] + Yup[i] + Ydn[i]
    energy_current = - s[i]× neighbor
    energy_updated = s[i] × neighbor
    delta = energy_updated - energy_current
    if e^(−beta×delta) > erand48() then
      s[i] = -s[i]
      energy_current = energy_updated          ▷ Update the
      energy
    end if
    esum += energy_current
    mags += s[i]                                  ▷ Accumulate
  end for
  Energy += esum / Vol
  Magnetization += fabs(mags / Vol )
end while
Magnetization /= iteration
Energy /= iteration
```

then process each in separate tiles using the VLIW vector processors. We implement the computation for each sub-grid into individual tiles for our design. For example, we divide a 32×32 grid into four 16×16 sub-grids and use a 2×2 tile formation to compute the spin updates for the entire grid as shown in Figure 3.

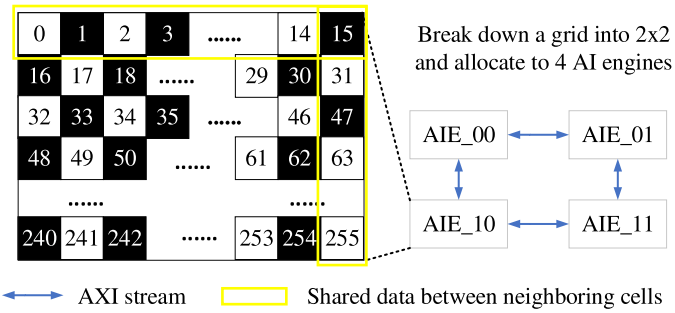

Fig. 3. Breaking down a grid into 2×2 formation and allocating the tasks to four AI engine tiles. For AIE_10, cells in the North and East sides are neighboring cells that are processed in AIE_00 and AIE_11 tiles. The data at the edge of each sub-grid is shared between the tiles to satisfy the boundary conditions at each sub-grid. All of the four AI engine tiles communicate via the AXI stream.

For this spatial distribution of the computation, let us assume the lattice is divided into black-and-white points.

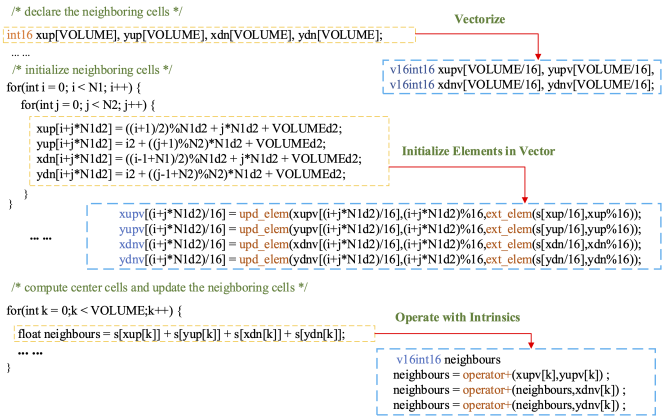

Fig. 4. Vectorizing the Metropolis algorithm for implementing Ising model solvers in the vector processor of a single AI engine tile. For this code segment, 16 variables are packed into a v16int16 vector. One can increase and decrease the bit-width of each variable based on the computational preferences. The computation of each vector is done in one single step, updating 16 grid points in parallel.

In each iteration step, all the same colored grid (say the black) points can be updated in parallel, although they are distributed in multiple AIE tiles. Next, to satisfy the boundary conditions, updated boundary values (for the black points) at each sub-grid are communicated to the neighboring grids via the AXI stream. Note that since the same colored grid points do not affect each other, these updates can be safely calculated and transmitted. Once all the neighboring updated values for the alternating color (*i.e.,* the black points) are available, the remaining same colored (i.e., white) grid points can be computed in parallel synchronously on all the tiles. Thus, one iteration of the Metropolis algorithm is completed. Our design uses the available AXI stream connection between the tiles to communicate the updates between neighbors.

Algorithm 2 presents the key idea of the computation. The most computationally expensive part of the algorithm, *i.e.*, updating each atom in a system, is operated in the vector processors with AI intrinsics **operator+/-**, **ext_elem**, and **mul**.

IV. Experimental Setup and Results

We have used the AMD-Xilinx VCK-190 reconfigurable computing platform that supports spatial accelerators using the AIE tiles. The details of the experimental setup are given in Table I. Since the Versal ASoCs provide reconfigurable tiles, we have explored the acceleration of the Ising model with varying matrix sizes and precision. We have also investigated the Metropolis algorithm in (1) serial, (2) vectorized, and (3) multi-tile configurations. Our detailed results are presented in Table II.

For the serial implementations, computing time increases in a polynomial fashion along with the matrix size. This is expected as discussed in Section III. In the vector implementation, our design adopts the checkerboard technique to separate the non-neighboring nodes. As a result, our design operates on multiple (up to 16) grid points simultaneously and achieves around 16× speedup. This observation agrees

Algorithm 2 Vectorized Ising Model Implementation

Input: Grid length (M)
Output: Energy, Magnetization
Vol = M × M / 16
▷ Assume a M*M grid, and divide into 16 group
v16int16 S[Vol] ← $\{-1, 1\}$ ▷ Vector initialization
v16int16 Xup[Vol], Xdn[Vol], Yup[Vol], Ydn[Vol] ← S[Vol]
double Magnetization, Energy = 0.0
while n < iteration **do**
  double mags, esum = 0.0
  v16int32 magsv, esumv
  for i = 0,...,Vol/16-1 **do**
    v16int32 energy_current, energy_updated, delta
    v16int16 neighbor = **operator+**(Xup[i] + Xdn[i] + Yup[i] + Ydn[i])
    energy_current = **mul**(-s[i], neighbor)
    energy_updated = **mul**(s[i], neighbor)
    delta = **operator-**(energy_updated - energy_current)
    if $e^{(-beta \times delta)}$ > erand48() **then**
      s[i] = -s[i]
      energy_current = energy_updated
      ▷ Update the energy
    end if
    esumv = **operator+**(esumv, energy_current)
    magsv = **operator+**(magsv, s[i]) ▷ Accumulate
  end for
  for i = 0,...,15 **do**
    mags = **ext_elem**(magsv,i)
    esum = **ext_elem**(esumv,i)
  end for
  Energy += esum / Vol
  Magnetization += fabs(mags / Vol)
end while
Magnetization /= iteration
Energy /= iteration

TABLE I
Experimental details for the AMD-Xilinx VCK-190 reconfigurable computing platform.

VCK-190	Part Number	XCVC1902-VSVA2179-2MP-ES
	Processor	Dual-core ARM Cortex-A72
	Memory	8 GB DIMM DDR4
	AI Engine	400 AIE tiles @1.25GHz

with our discussions in the previous section. For a larger problem size, such as the case of the 64×64 grid, the total memory requirements exceed the total memory available in each tile and, therefore, cannot be implemented on a single tile alone. In such cases, we need to leverage the multi-tile spatial architecture.

For our multi-tile spatial design, we process a 64 × 64 grid by dividing it into four 32 × 32 sub-grids and follow the Metropolis algorithm presented in Section III. We expected a 16 (from the vectorized implementation)×4 (from spatial

TABLE II
PERFORMANCE RESULTS AND COMPARISON FOR EXECUTING THE ISING MODEL WITH DIFFERENT MATRIX SIZES. THE ITERATION LIMIT IS SET TO 2000.

Matrix	Precision	Cortex A-72	Serial Implementation			Vector Implementation				
		Time(ms)	Dyn. Power(W)	Total(W)	Time(ms)	Dyn. Power(W)	Total(W)	Time(ms)	Speedup	Tiles
16×16	int32	31.4	0.875	2.678	179.9	0.820	2.623	11.4	15.78	1
32×32	int16	115.8	0.817	2.620	716.0	0.877	2.680	46.0	15.56	1
64×64	int8	456.7	0.817	2.620	2881.0	1.198	3.001	49.3	58.44	4

separation) $= 64\times$ acceleration. However, due to the communication overhead between the tiles to address the boundary conditions, we observe a speedup of around $58\times$ compared to a single-tile serial implementation. Compared with the ARM Cortex A-72 processor, the spatial design demonstrates a $9\times$ improvement in speed.

We have also recorded the power consumption for the serial, vectorized, and multi-tile implementations using the Power Design Manager (PDM) tool available in Vitis. The power consumption for different grid sizes on the serial and vectorized implementation remains relatively the same due to using a single tile. However, the spatial implementation with four tiles for the $64\times$ 64 grid consumes more dynamic power due to the increased number of AI engines engaged. Notice the total power consumption for even the larger grid size remains around 3 watts. These experimental results show that spatial computing architectures can provide promising performance gain in acceleration and power budgets for solving complex combinatorial problems in low-power edge nodes.

V. IMPLEMENTATION ISSUES

Although we have demonstrated that efficient implementation of Ising model processors is feasible in a spatial architecture, some constraints and design choices for the Versal ASoCs create significant bottlenecks in such computation. These implementation issues are discussed below.

- Small Stack Size: The stack size of an AIE tile can be prohibitively small (32KB) for more complex problems. For example, storing a sub-grid of 64×64 in each tile is not feasible; thus, the data must be fetched/stored from/to the global memory for large grid sizes. This will cause significant delays in processing.
- Limited Operation Sets: The vector processor in the AIE tile supports only a limited set of data types and operations. Even using the intrinsics, a designer is bound to use only certain operations, which affects the available design choices.
- Limited Connectivity: Each tile has a connection to only four streams, two INs and two OUTs. As a result, an AIE tile cannot support both-way stream communication with all four neighbors.

VI. CONCLUSIONS

This work presents a novel spatial acceleration-based Ising model processor on a Versal ASoC platform. These new coarse-grained reconfigurable architectures can significantly improve performance in solving complex combinatorial problems in budget-constrained edge devices. Given the novelty of spatial architectures, our design and the open-source implementation will help and inspire further research on compute acceleration using spatial computing architectures.

VII. ACKNOWLEDGEMENT

This material is based upon work supported by the National Science Foundation under Grant No. 2245156. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] S. Patel, L. Chen, P. Canoza, and S. Salahuddin, "Ising Model Optimization Problems on a FPGA Accelerated Restricted Boltzmann Machine," Oct. 2020. arXiv:2008.04436 [physics].
[2] A. Minamisawa, R. Iimura, and T. Kawahara, "High-speed Sparse Ising Model on FPGA," in *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, (Dallas, TX, USA), pp. 670–673, IEEE, Aug. 2019.
[3] C. Cook, H. Zhao, T. Sato, M. Hiromoto, and S. X.-D. Tan, "GPU Based Parallel Ising Computing for Combinatorial Optimization Problems in VLSI Physical Design," Mar. 2019. arXiv:1807.10750 [physics].
[4] B. A. Cipra, "The Ising Model Is NP-Complete,"
[5] G. T. Barkema and T. MacFarland, "Parallel simulation of the Ising model," *Physical Review E*, vol. 50, pp. 1623–1628, Aug. 1994.
[6] A. Mondal and A. Srivastava, "Ising-FPGA: A Spintronics-based Reconfigurable Ising Model Solver," *ACM Transactions on Design Automation of Electronic Systems*, vol. 26, pp. 1–27, Jan. 2021.
[7] C. Yoshimura, M. Hayashi, T. Okuyama, and M. Yamaoka, "FPGA-based Annealing Processor for Ising Model," in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, (Hiroshima, Japan), pp. 436–442, IEEE, Nov. 2016.
[8] A. Mondal and A. Srivastava, "Spintronics-based Reconfigurable Ising Model Architecture," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, (Santa Clara, CA, USA), pp. 134–140, IEEE, Mar. 2020.
[9] R. Iimura, S. Kitamura, and T. Kawahara, "Annealing Processing Architecture of 28-nm CMOS Chip for Ising Model With 512 Fully Connected Spins," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 68, pp. 5061–5071, Dec. 2021.
[10] L. Gwennap, "Groq rocks neural networks," *Microprocessor Report, Tech. Rep., jan*, 2020.
[11] J. Vasiljevic, L. Bajic, D. Capalija, S. Sokorac, D. Ignjatovic, L. Bajic, M. Trajkovic, I. Hamer, I. Matosevic, A. Cejkov, *et al.*, "Compute substrate for software 2.0," *IEEE micro*, vol. 41, no. 2, pp. 50–55, 2021.
[12] AMD, "Versal: The First Adaptive Compute Acceleration Platform," Sept. 2020.
[13] E. Ising, "Contribution to the theory of ferromagnetism," *Z. Phys*, vol. 31, no. 1, pp. 253–258, 1925.
[14] N. Mohseni, P. L. McMahon, and T. Byrnes, "Ising machines as hardware solvers of combinatorial optimization problems," *Nature Reviews Physics*, vol. 4, pp. 363–379, May 2022.
[15] L.-P. Henry, P. C. Holdsworth, F. Mila, and T. Roscilde, "Spin-wave analysis of the transverse-field ising model on the checkerboard lattice," *Physical Review B*, vol. 85, no. 13, p. 134427, 2012.
[16] "Introduction to parallel programming with mpi,." https://pdc-support.github.io/introduction-to-mpi/, Accessed May 24, 2024.