An Ising Machine based on an Improved Parallel Annealing Algorithm for Solving Traveling Salesman Problems

by

Qichao Tao

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Integrated Circuits and Systems

Department of Electrical & Computer Engineering
University of Alberta

Abstract

Annealing-based Ising machines have shown promising results in solving combinatorial optimization problems. As a typical class of these problems, however, traveling salesman problems (TSPs) are very challenging to solve due to the constraints imposed on the solution. This thesis proposes a parallel annealing algorithm for a fully connected Ising machine that significantly improves the accuracy and performance in solving constrained combinatorial optimization problems such as the TSP. Unlike previous parallel annealing algorithms, this improved parallel annealing (IPA) algorithm efficiently solves TSPs using an exponential temperature function with a dynamic offset. Compared with digital annealing (DA) and momentum annealing (MA), the IPA reduces the run time by 44.4 times and 19.9 times for a 14-city TSP, respectively. Larger scale TSPs can be more efficiently solved by taking a k-medoids clustering approach that decreases the average travel distance of a 22-city TSP by 51.8% compared with DA and by 42.0% compared with MA. This approach grourps neighboring cities into clusters to form a reduced TSP, which is then solved in a hierarchical manner by using the IPA algorithm on each cluster.

Furthermore, an Ising machine that implements this annealing algorithm is designed by using a half-precision floating-point representation of the coefficients in the Ising model. This design reuses adders in the local field accumulator units (LAUs) for variable calculation and approximates the momentum scaling factor by a linear increment to save hardware. Approximate arithmetic is considered in the design for improving the speed of accumulation in the LAUs. A hybrid approximation method using lower-part-OR truncated adders (LOTAs) is applied and shown to reduce the

delay of the circuit by 3.8% at the cost of an increase in average travel distance by 1.9%. Finally, an IPA-based Ising machine prototype with 64 spins is built and synthesized by using the Synopsys Design Compiler. Using a 28-nm CMOS process with a supply voltage of 1.0 V, a temperature of 25 °C, and a clock frequency of 200 MHz, the total area of the circuit is 6.262 mm^2, the power dissipation is 41.108 mW, and the delay is 3.82 ns. Compared with other designs, the area and power dissipation of this circuit are larger due to the implementation of a half-precision floating-point representation. However, the IPA-based Ising machine is expected to solve more complicated problems and improve the solution quality.

Preface

In Chapter 3, an improved parallel annealing algorithm (IPA) is proposed. This work was published as Qichao Tao and Jie Han, "Solving Traveling Salesman Problems via a Parallel Fully Connected Ising Machine," in the Design Automation Conference (DAC), San Francisco, California, USA, July 10-14, 2022. I developed the MATLAB and VHDL codes for the IPA and IPA-based Ising machine, performed the simulations, analyzed the results, and drafted the article. The synthesis of circuits is finished with the help of PhD candidate Tingting Zhang. I also recorded a presentation video for the conference and attended the online meeting for the live Q&A. Ryan Holash, developed the original codes in C++ for generating the VHDL testbench that was partially used in Chapter 4. Tingting Zhang and Bailiang Liu provided comments and suggestions for the thesis. Dr. Jie Han supervised this work and revised the thesis.

Acknowledgements

Foremost, I would like to pay my great gratitude to my supervisor Dr. Jie Han, who made this work possible. His guidance and encouragement carried me through all the stages of my M.Sc. study. He always gave invaluable advice and insights which helped to boost my progress of writing the thesis.

I would also like to express my gratitude to great researchers Tingting Zhang and Bailiang Liu, with whom I had excellent collaboration. They provided useful feedback throughout my research. Thanks to the University of Alberta and Huawei Technologies Co., Ltd who supported this work.

Last but not the least, I would like to thank my family and my girlfriend for giving me endless amount of love, support and inspiration.

Table of Contents

List of Tables

List of Figures

Abbreviations

ACA almost correct adder.

ADB acceptance decision block.

aSB adiabatic simulated bifurcation.

ASU annealing schedule unit.

bSB ballistic simulated bifurcation.

CIM coherent Ising machine.

CLA carry look-ahead adder.

CMOS complementary metal-oxide-semiconductor.

CPU central processing unit.

DA digital annealing.

DDSS delta-driven simultaneous spin update.

dSB discrete-simulated bifurcation.

ED error distance.

ER error rate.

ESA equal-segmented adder.

FPGA field-parogrammable gate array.

GPU graphics processing unit.

IPA improved parallel annealing.

LAU local field accumulator unit.

LOA lower-part-OR adder.

LOTA lower-part-OR truncated adder.

LSBs least significant bits.

MA momentum annealing.

MC Monte Carlo.

MRED mean relative error distance.

NED normalized error distance.

NME normalized mean error.

NMED normalized mean error distance.

NP-hard non-deterministic polynomial-time hard.

OPO optical parametric oscillator.

PTNO phase-transition nano-oscillator.

RCA ripple-carry adder.

RED relative error distance.

RNG random number generator.

SA simulated annealing.

SB simulated bifurcation.

SCA stochastic cellular automata annealing.

SIGU self-interaction generating unit.

SIL software intervention layer.

SQA simulated quantum annealing.

SRAM static random-access memory.

STATICA stochastic cellular automata annealer.

SUU spin update unit.

TruA truncated adder.

TSP traveling salesman problem.

VHDL VHSIC Hardware Description Language.

VR violation rate.

Chapter 1

Introduction

1.1 Motivation

Combinatorial optimization problems exist in many applications, such as data mining, drug discovery, Internet of Things technology, chip design, and machine learning [1]. However, combinatorial optimization problems are time-consuming to solve with enumeration methods on a conventional computer [2]. Because it is non-deterministic polynomial-time hard (NP-hard), the number of such problems' candidate solutions dramatically increases when the problems' size increases. For example, $(M-1)!$ candidate solutions need to be traversed to solve a traveling salesman problem (TSP) with M cities [3]. The traversal time will substantially increase as the value of M becomes large, making finding the shortest path in a TSP difficult. However, getting an optimal solution at the cost of energy and time is infeasible in industrial applications. Offering a solution with acceptable accuracy in a short time and at high power efficiency is more attractive. Approximation methods are often used to obtain a suboptimal or good enough solution in industrial applications. Recently, an efficient approximate system, called an Ising machine, has emerged for solving combinatorial optimization problems [4].

Ising machines can broadly be classified into three categories: oscillator-based Ising machines, bifurcation-based Ising machines, and annealing-based Ising machines. The oscillator-based Ising machines are of two kinds. One consists of physical oscillators,

such as the coherent Ising machines (CIM) implemented with degenerate optical parametric oscillators (OPO) [5, 6], coupled lasers [7, 8], and the design in [9] built with phase-transition nano-oscillators (PTNO). The other branch emulates the characteristic of oscillators on the complementary metal-oxide-semiconductor (CMOS) circuits, like the simulated CIMs in [10, 11]. The bifurcation-based Ising machines follow the adiabatic and chaotic (ergodic) evolutions of nonlinear Hamiltonian systems, such as the adiabatic simulated bifurcation (aSB) [12, 13], ballistic SB (bSB) [14, 15] and the discrete SB (dSB) [16]. The annealing-based Ising machines belong to two categories. One is implemented on the superconducting circuits using a quantum annealing algorithm [17]. The other is executed on conventional digital CMOS circuits using simulation-based algorithms, such as simulated quantum annealing (SQA) [18–20] and simulated annealing (SA) [21–24]. This paper focuses on CMOS Ising machines that use SA because CMOS circuits are easy to manufacture and scale [25].

SA was first proposed in [21] and it can solve combinatorial optimization problems. Based on SA, the CMOS annealing [25–29] uses a majority-voter circuit to flip the state of each spin (as a model of magnetic spins). These CMOS annealing machines can achieve significant speed improvement in solving large-scale combinatorial optimization problems than conventional computers. A 20k-spin three-dimensional adjacently connected Ising prototype chip based on 65-nm CMOS technology was developed in [25], and the power efficiency is 1800 times higher than a general central processing unit (CPU). A more complex FPGA-based model was proposed in [26]. Four spins consist of a complete graph as a unit placed in a grid pattern, and spins in the same position in every neighborhood unit are connected. They reduced 90% of the logic elements and increased the solution accuracy. Then in [27], a 2×30k spin multichip CMOS annealing processor that expands the coefficients' bit-width and scalability was developed based on 40-nm CMOS technology. Its annealing speed is 26,000 times faster than the conventional CPU. The latest CMOS annealing-based Ising machine is constructed with 9 ASIC annealer chips that can scale up to 144k

spins [29]. The CMOS annealing Ising machines use a limited number of interactions between spins as they only implement the sparsely-connected topology of Ising models.

An embedding process is required for mapping a complex combinatorial optimization problem into a sparsely-connected Ising model, which is not always straightforward. However, a combinatorial optimization problem can be mapped directly to a fully-connected Ising model. Digital annealing (DA) [24, 30–32] realized the fully-connected Ising model structure on the field-programmable gate array (FPGA). In [24], researchers implemented the fully-connected architecture on a 28-nm CMOS technology and built a 512-spin Ising machine with 4-bit interaction accuracy. A 1024-spin fully-connected DA Ising machine with a parallel trial scheme and a dynamic offset was implemented on FPGAs in [32]. This design can be 12,000 times faster than the conventional processor and 6000 times faster than conventional SA.

However, only one spin can be flipped per Monte Carlo (MC) step as the spins connected are not allowed to simultaneously update in the Ising model. This increases the average time for an Ising machine to find a solution. The momentum annealing (MA) was proposed in [33] to update all spins of the fully-connected Ising model in parallel. The MA uses a two-layer-spin structure and self-interactions for ensuring two layers be equivalent in the ground state. It is implemented on a graphics processing unit (GPU), the annealing time for a fully-connected Ising machine with 100,000 spins is 250 times faster than the SA on a CPU. The stochastic cellular automata annealer (STATICA) [1] can simultaneously update all spins in the fully-connected Ising model. This Ising machine is based on a stochastic cellular automata annealing (SCA) algorithm. It also introduces duplication for each spin, and self-interactions for making each spin have the same state as its duplication at the end of annealing. Unlike [33] implementing MA on GPU, the STATICA is built on a 3mm-by-4mm 512-spin fully-connected Ising machine chip with the 65-nm CMOS technology. The speed of STATICA is 3.8 times faster than simulated bifurcation (SB) and 500 times

faster than SA for solving combinatorial optimization problems with 512 vertices.

However, the MA and SCA are dedicated to solve unconstrained combinational optimization problems, such as the max-cut problems. No Ising machine with parallel spin update has been proposed for solving constrained combinational optimization problems such as the TSP. So in this work, we propose an annealing algorithm that can efficiently solve constrained and unconstrained combinational optimization problems, and at the same time, achieve parallel spin update on the fully-connected Ising model. Furthermore, the hardware of an Ising machine that implements this annealing algorithm is developed.

1.2 Thesis Objectives

This research aims to design a parallel annealing Ising machine with high performance in solving combinational optimization problems. In particular, the research objectives are addressed as follows.

The existing parallel annealing algorithms have restrictions in solving constrained combinatorial optimization problems. Thus, we aim to propose an improved parallel annealing (IPA) algorithm that efficiently solves constrained and unconstrained combinatorial optimization problems. This improved annealing algorithm is expected to implement a full-connected topology while simultaneously updating all spins. Therefore, it can solve Ising problems without an embedding process and achieve a high annealing speed. Furthermore, it can easily escape from a local minimum, thus improving the quality of solutions.

Then an Ising machine that implements the improved parallel annealing algorithm is designed. It aims for the best tradeoffs between accuracy and hardware efficiency. Furthermore, the number of spins in an Ising model is fixed when the chip is fabricated. It is impossible to solve a larger scale problem using a fixed scale Ising machine. Therefore, the circuit design must support applications in a multi-chip system, and achieve high scalability.

1.3 Thesis Contribution

First, an IPA algorithm is proposed that applies an exponential temperature function with a dynamic offset and a clustering approach. It reduces the runtime by 44.4 times compared with a conventional parallel annealing algorithm in solving a 14-city TSP. The parameter setting for producing the optimal solution quality is discussed. The performance evaluation and comparison between the IPA and other annealing algorithms are presented. Second, a circuit is designed to realize the function of the IPA. In order to improve hardware efficiency, the circuit units in the design are reused for variable calculations. Furthermore, approximate arithmetic circuits are used to balance the hardware efficiency and the solution quality. Two approximate methods are considered and the performance of the Ising machine using the approximation methods is demonstrated. Finally, a hybrid approximation method using both a lower-part-OR adder and a truncated adder is applied.

1.4 Thesis Outline

This thesis is organized as follows. Several typical CMOS circuit-based Ising machines are reviewed and classified in Chapter 2. Then an improved parallel annealing algorithm, which improves the solution quality in solving constrained combinational optimization problems, is proposed in Chapter 3. In Chapter 4, a circuit design of the improved parallel annealing machine is presented. Finally, this thesis is concluded in Chapter 5.

Chapter 2

Background

2.1 The Ising Model

An Ising model mathematically describes the ferromagnetic interactions of magnetic spins. In the Ising model, each spin can be in either an upward (+1) or downward (−1) state. The interactions among the spins and external magnetic fields affect the states of spins. In an N-spin system, the Hamiltonian in an Ising model is defined as [34]:

$$H(\sigma_1, ..., \sigma_N) = -\sum_{i,j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i, \tag{2.1}$$

where $\sigma_i \in \{-1, +1\}$ ($i \in \{1, 2, ..., N\}$), denotes the state of the ith spin, J_{ij} indicates the interaction between the ith spin and the jth spin, and h_i is the external magnetic field for the ith spin.

Some designs of Ising machines implement a sparse spin-to-spin structure, such as the 2-D lattice topology, the 3-D lattice topology, and King's graph topology. As shown in Fig. 2.1 (a), each spin in a 2-D lattice topology has four connections with its neighbor spins (i.e., the connections with the north, east, south, and west spins). The dotted lines indicate the connections between the spins at the edge with those not shown in this figure. A 3-D lattice topology consists of two (or more) layers of 2-D lattices. As shown in Fig. 2.1 (b), each spin in a 3-D lattice topology has five connections with its neighbor spins (i.e., the connections with the north, east, south, west, and the front/back spins). King's graph topology is also a planar

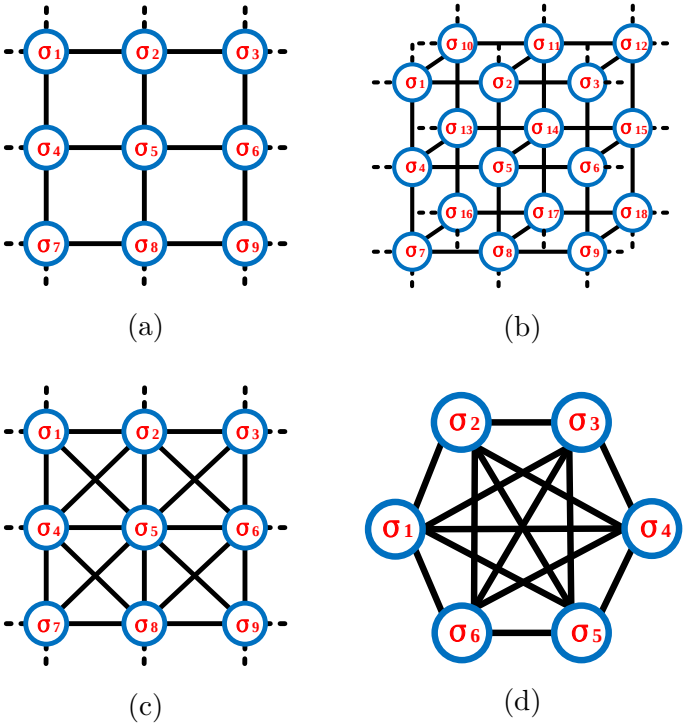


Figure 2.1: (a) A 2-D lattice topology, (b) a 3-D lattice topology, (c) a King's graph topology, and (d) a complete graph topology of the Ising model.

structure but it is more complex than the 2-D lattice topology. Fig. 2.1 (c) shows the structure of the King's graph topology. Each spin in a King's graph topology has eight connections with its neighbor spins (i.e., the connections with the north, east, south, west, northeast, northwest, southeast, and southwest spins).

However, spins for solving Ising problems can have more interactions than those in sparse spin-to-spin structures. Therefore, Ising machines with sparse spin-to-spin structures need an embedding process to convert general Ising problems to the available physical Ising models [3]. Several spins in physical Ising models are used to represent one spin in Ising problems in an embedding process. All Ising problems can be mapped to a complete graph. Therefore, some Ising machines use a complete graph topology, as shown in Fig. 2.1 (d). In the complete graph topology, all the spins are connected. The number of a spin interactions depends on the scale of an Ising machine.

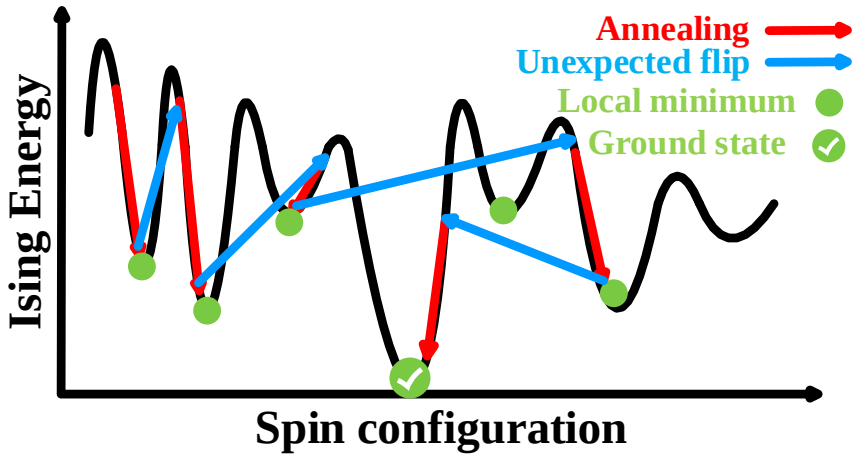


Figure 2.2: The energy profile of the Ising model during annealing with random flips [35].

2.2 Review and Classification of Annealing Algorithms and Ising Machines

Solving a combinatorial optimization problem via an Ising machine is to find the ground state of the energy (or the Hamiltonian of the Ising model). Annealing-based Ising machines search for the ground state using annealing algorithms. As shown in Fig. 2.2, the Ising energy tends to converge to a minimum value during the annealing process. Furthermore, the unexpected flips help the Ising machine escape from a local minimum.

Simulated annealing (SA) serves as the basis of various annealing algorithms, such as CMOS annealing, digital annealing (DA), momentum annealing (MA), and stochastic cellular automata annealing (SCA). SA mimics the thermal annealing in metallurgy. The spin-flip probability decreases as the temperature decreases, and the temperature decreases during the annealing process. As shown in Algorithm 1, the spin-flip probability is calculated by the energy variation and temperature. It will always be 1 if the energy variation value is negative. Thus, spin-flips tend to decrease the total energy of an Ising model. The spin-flip probability is high with a large

temperature value even if the energy variation is positive. Therefore, unexpected spin-flips can be caused at high temperatures. This unexpected spin-flip helps an Ising model escape from local minima and improves the solution quality. At the end of an annealing process, the spin configuration is stable as the temperature is low enough.

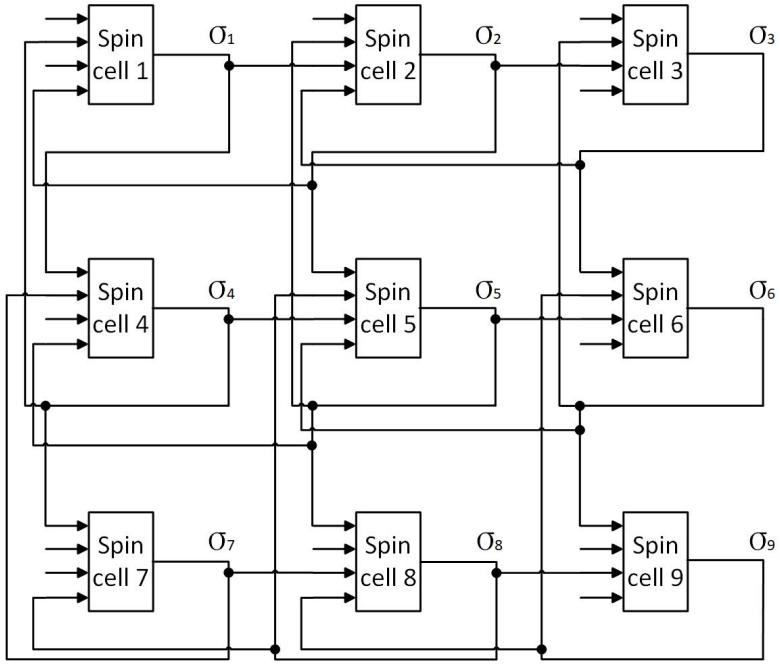
Algorithm 1 Simulated annealing

```
1: Initialize spin states
2: for each temperature T do
3:    for each MC sweep at this temperature do
4:       for i = 1 to N do
5:          Calculate the energy variation ΔE_i
            ΔE_i ⇐ 2σ_i(h_i + Σ_j J_ij σ_j)
6:          Calculate the spin-flip probability P_i
            P_i ⇐ min{1, exp(−ΔE_i/T)}
7:          Update the spin state
            if P_i > rand then
            σ_i ⇐ −σ_i
            end if
8:       end for
9:    end for
10:   Update the temperature T
11: end for
```

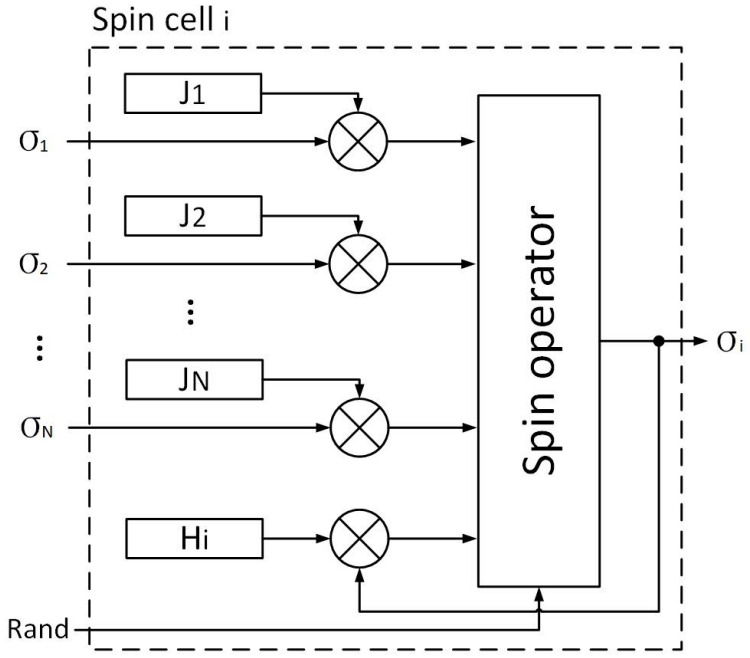
Then, three typical types of simulated annealing-based algorithms and their circuit designs are introduced.

2.2.1 CMOS Annealing Machines

The research group that proposed CMOS annealing machines used SRAMs to cause random flips of spins in the first-generation design. Thus, their designs are called CMOS annealing machines, although the newer generations do not use SRAMs for causing random flips. The architecture of a CMOS annealing machine is shown in Fig. 2.3(a) [28]; it consists of nine spin cells, and the structure of a generalized spin cell is shown in Fig. 2.3(b). Each spin cell has one output for the state of the spin (σ_i), and N inputs that connect to the outputs of adjacent spin cells (from σ_1 to σ_N).



(a)



(b)

Figure 2.3: (a) The architecture of a CMOS Ising machine [28], (b) the structure of a generalized spin cell.

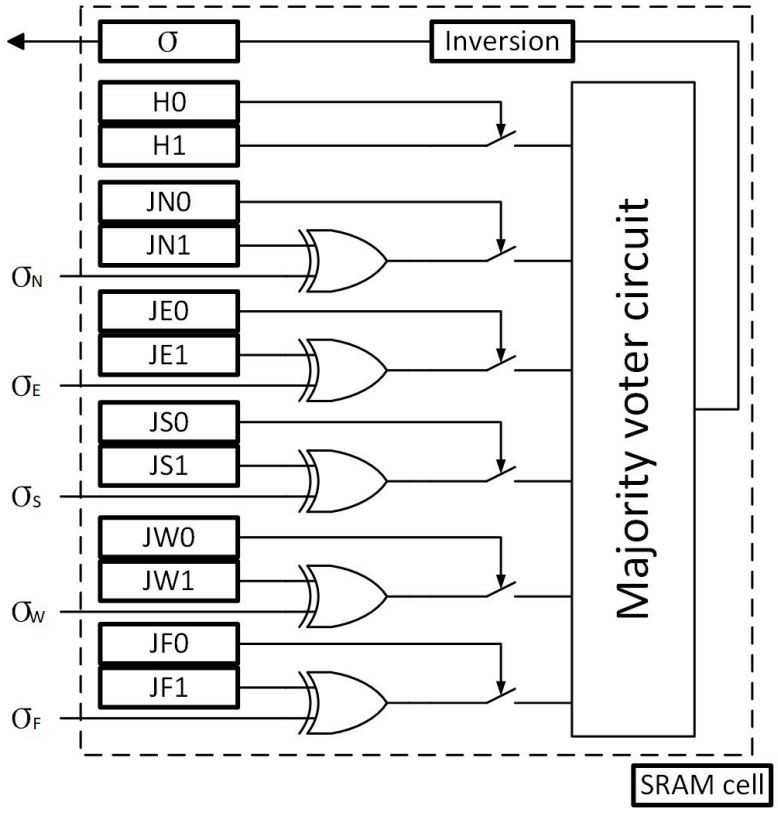


Figure 2.4: The architecture of the first-generation CMOS Ising machine [25].

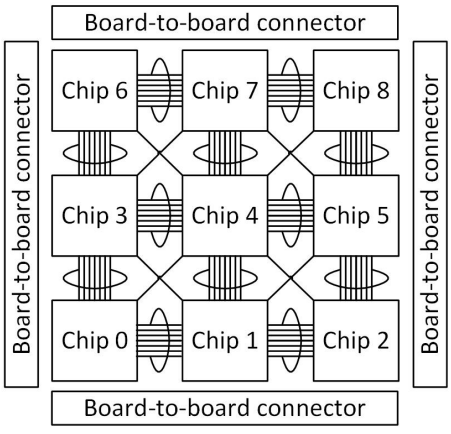


Figure 2.5: The architecture of the third-generation CMOS Ising machine [29].

$N+1$ memory cells store N interactions (from J_1 to J_N) and one external field (H_i) in the spin cell. The interactions are multiplied with the corresponding spin states, and the H_i is multiplied with σ_i. All the products and a random number ($Rand$), for causing unexpected spin-flips, are sent as inputs to a spin operator to determine the new state of the current state σ_i.

There are four generations of the CMOS annealing machines so far. The first-generation [25] applies a $3D$ lattice topology and achieves a 2-bit precision for the coefficients. Therefor, the inputs for neighbor spin states are described in σ_X ($X = N(north), E(east), S(south), W(west), F(front)$), the interactions stored in static random-access memory (SRAM) cells are described in $JX0$ and $JX1$ (the first bit and the second bit of an interaction), and the external field is described in $H0$ and $H1$. Its interaction and external field can only be $+1$, -1, or 0 and are stored SRAM cells. The structure of a first-generation's spin cell is shown in Fig. 2.4 [25]. The spin operator is a majority voter circuit: the new spin state is $+1$ when the number of $+1$ is more than the number of -1 among the inputs, and vice versa. The multiplication is realized by an XOR gate and a switch. Instead of causing unexpected spin-flips via a high temperature, this design directly uses a random number sequence of 1 and 0 to invert the state of a spin. The new spin state is inverted if the random number is 1. Furthermore, the random number is generated by utilizing the variability of SRAM cells (the SRAM cell called $Inversion$). A low supply voltage of the memory cells can cause random error bits in SRAMs, which are used to change the states of spins. To not change the interactions and external fields, only SRAM cells that store the states of spins are activated when lowering the supply voltage.

The second-generation [27] improves the precision for the coefficients to 3-bit and implements King's graph topology. The interactions and external fields can take integers from -3 to $+3$. Thus, the maximum voting strength is $55(= 9 \times 6 + 1)$, where the 1 is caused by the $Rand\_F$. $Rand\_F$, which is used to avoid repeatedly getting the same vote result, is also added to the external field and has the same

probability of being zero and one. The new spin's state is +1 if the input of the decision logic circuit $n \geq 28$, or -1 if the input of the decision logic circuit $n < 28$. Another random variable $Rand\_G$, for the temperature, is added to the interactions to cause unexpected spin-flips. It is regarded as an approximation of the SA algorithm because the new spin's state is related to the temperature.

The third-generation [29] achieves 5-bit precision for the coefficients and also implements King's graph topology. Because the approximated SA in the second-generation is for improving the hardware efficiency when the bit-width for the coefficients is small, it deteriorates the solution quality as the coefficients' bit-width increases in the third-generation. Thus, a flip-flop-based spin cell circuit that emulates the Metropolis algorithm is proposed in the third-generation to improve the accuracy of solutions. As shown in Fig. 2.5 [29], the third-generation is a multi-chip design. Every single chip contains 128×128 spins and communicates with its adjacent eight chips. Furthermore, the new spin's state is determined by a comparator. The state is +1 when the energy variation is larger than a random variable for the temperature, and vice versa. There are 9-chip $\times$ $16k$ spins in total. The fourth-generation [36] is a large scale design with 9-board $\times$ 9-chip $\times$ $16k$ spins based on the third-generation.

CMOS annealers only implement sparsely connected topologies, such as the 3D lattice topology and King's graph topology. Mapping combinatorial optimization problems to sparsely connected Ising machines needs extra embedding steps and the number of required spins rapidly increases if the problem is complex. However, an advantage of CMOS annealing machines, scalability, benefits from its sparse connection. It can achieve high-speed multi-chip communication. Thus building a CMOS annealing machines with an arbitrary number of spins is theoretically possible.

2.2.2 Digital Annealing Machines

Digital annealing machines are developed as an architecture that can be implemented in digital circuits because they can achieve a high precision for the coefficients. A dig-

ital annealing machine with N spins consists of N ΔE (energy variation) calculation units, N acceptance decision blocks (ADB), an update selector, an E_{off} generator, and a state variable update block, as shown in Fig. 2.6 [32]. Only one spin is selected via the update selector and updated in each annealing step. An index that indicates which spin is updated is sent to the state variable update block, then the new state of the flipped spin is sent back to ΔE calculation units for new ΔE computations. In each ΔE calculation unit, an interaction value corresponding to the update-bit index is selected and multiplied with the new state of the flipped spin. Then the product is accumulated with the former local field value. The summation is multiplied by the local spin state, and a new ΔE value is obtained. The new ΔE value is used in the ADB for the new spin's state processing. Metropolis-Hastings or Gibbs criterion can be used for determining the new spin's state. A random number between 0 and 1 and the dynamic offset variable E_{off} are required in ADB. The E_{off} decreases the ΔE so that the spin flip probability is increased when there is no spin flipped for several iterations (stuck in a local minimum). The digital annealing machines implement a fully connected topology. Thus, each ΔE calculation unit needs a large memory block to store all the interaction values.

There have been three generations of the digital annealing machines so far. The architecture of the first-generation [32] and the second-generation [31] are the same. The scale of the second-generation is eight times larger than the first-generation (1024 spins for the first-generation and 8192 spins for the second-generation), and the second-generation achieves higher coefficient precision (16 bits for the first-generation and 64 bits for the second-generation).

The third-generation of the digital annealing machine [37] improves the speed and the solution quality by integrating software and hardware. The software intervention layer (SIL) first searches for near-optimal solutions. Then these spin configurations are used in the hardware search core as the start points for optimal solution search. The architecture of the hardware search core is the same as the second-generation.

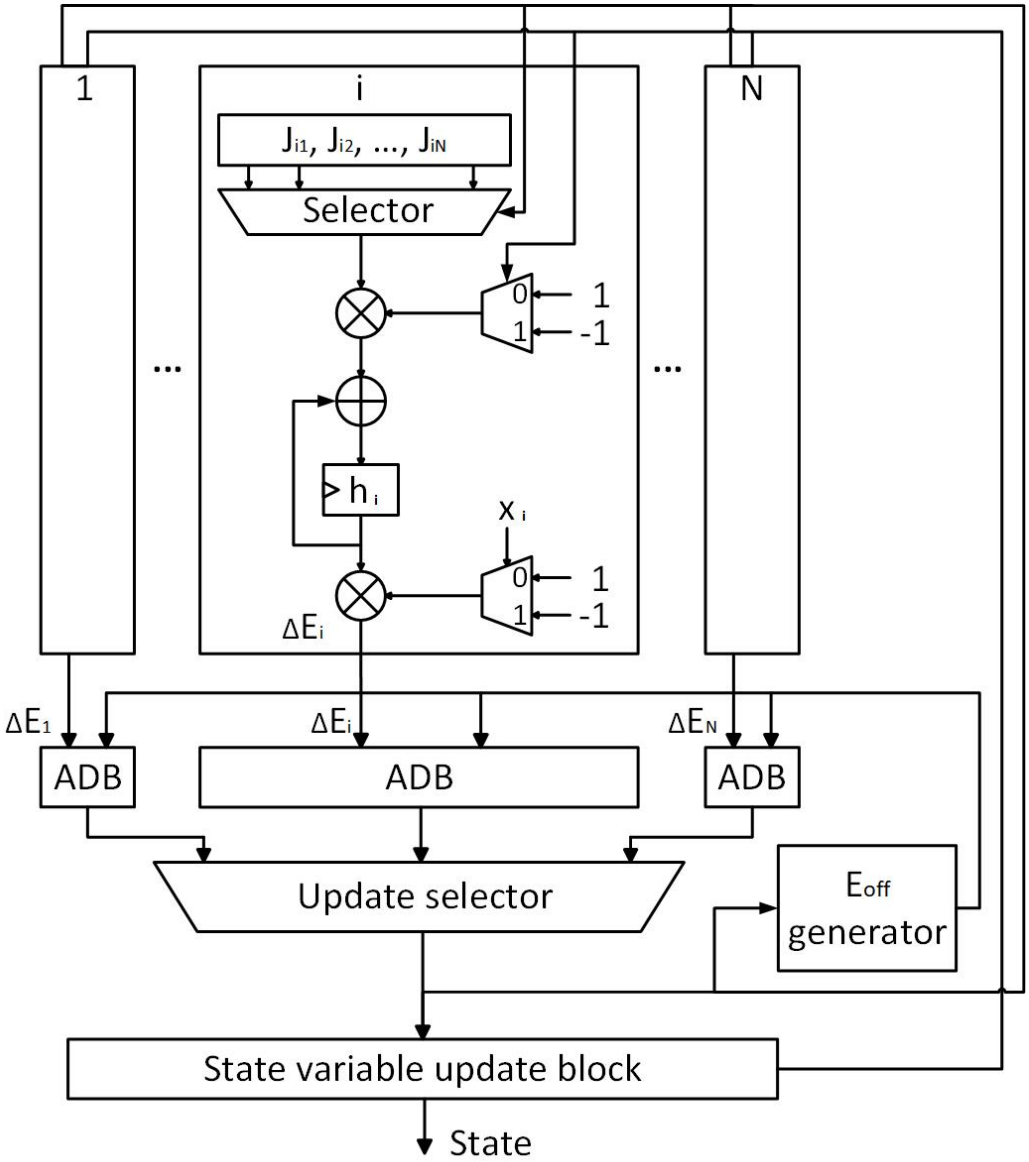


Figure 2.6: The architecture of a digital Ising machine [32].

Unlike the CMOS annealing machines, that only utilize sparsely connected topologies, digital annealing machines implement fully connected topology. It benefits from the implementation of sequential processing for the ΔE calculations. Calculating ΔE sequentially using ΔE calculation units and then deciding on new spin states in the ADB decreases the hardware overhead of the circuit. Therefore, it is possible for digital annealing machines to process a large number of interaction coefficients. Although solving combinatorial optimization problems using a digital annealing machine does not require embedding steps, the multi-chip processing of digital annealing machines faces a data communication problem. Because for CMOS annealing machines, multi-chip processing only needs to transmit a predictable number of bits that represent the states of spins placed at the edge of chips. However, the number of bits that need to be transmitted increases with the total number of spins in a digital annealing machine. Thus, the required data communication between chips changes when one more chip is added. Moreover, the area for memory block exponentially grows with the increasing number of spins. Another disadvantage of the digital annealing machines is that, it can only update one spin's state in each annealing step. Therefore, the processing speed is the bottleneck for digital annealing machines.

2.2.3 Parallel Annealing Machines

A two-layer structure for fully connected Ising models is proposed to realize the parallel spin update and thereby speed up the annealing process for momentum annealing (MA) [33] and stochastic cellular annealing (SCA) [1]. They are called parallel annealing in this thesis as they can update all spins simultaneously. A parallel annealing machine with N spins consists of N local field accumulation units (LAUs), N spin update units (SUUs), a delta-driven simultaneous spin update (DDSS) unit, and a memory block (SRAM), as shown in Fig. 2.7 [1]. The DDSS unit sequentially outputs indexes that represent flipped spins to SRAMs. When the SRAM receives an index from the DDSS, the corresponding interaction values in the same row are output to

LAUs to update the local field values. The new values are obtained after the DDSS traverses all the flipped spins. Then new spins states are determined by SUUs. For determining the new spins' states, the Gibbs criterion is used in the SCA, and the Metropolis criterion is used in the MA. The new spin state and flag signal Δ_i that indicates whether the spin is flipped are sent to DDSS for the next update cycle. In SCA, a sigmoid function is approximated by a linear function, to reduce the hardware cost.

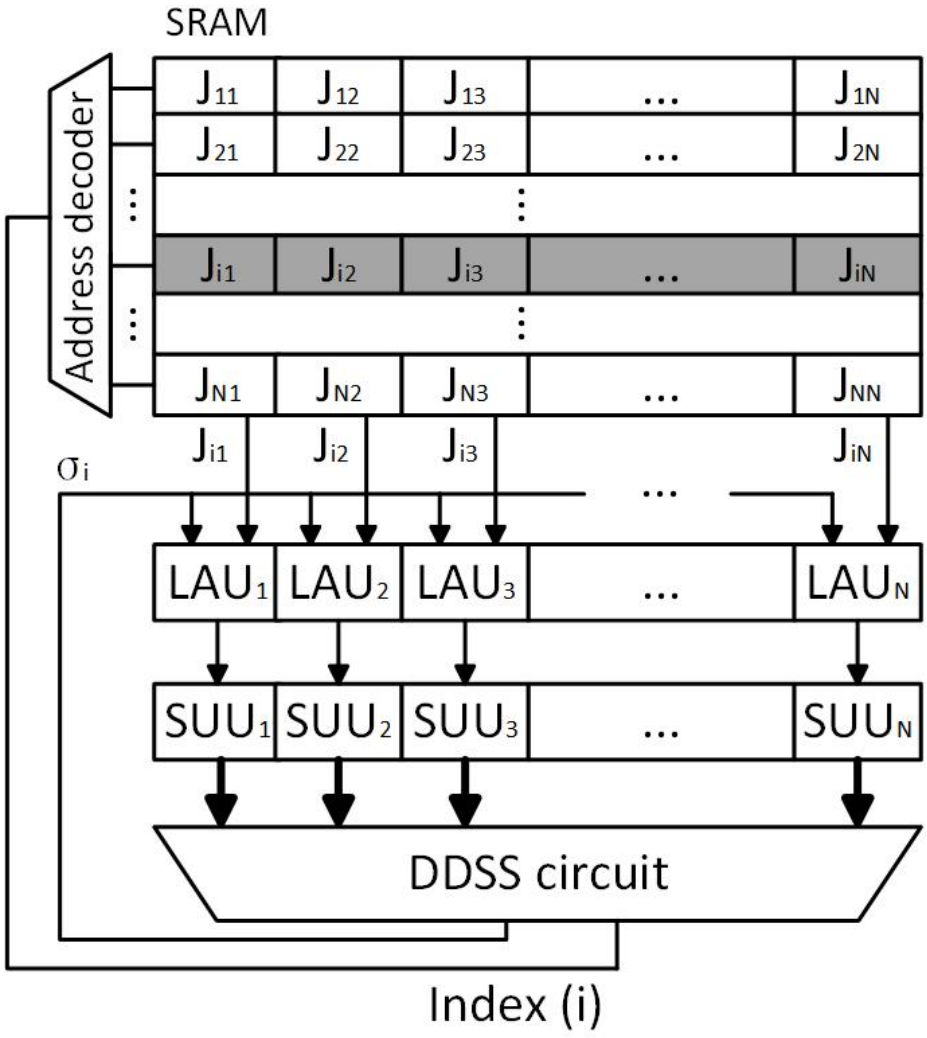


Figure 2.7: The architecture of a parallel Ising machine [1].

The parallel annealing machine improves the annealing speed compared with the

digital annealing machine. It also achieves a high precision. Besides the parallel spin update, the SCA significantly increases the annealing speed by utilizing a DDSS unit. Although the energy can converge quickly in parallel annealing, it is difficult to escape from local minimum states when solving constrained combinatorial optimization problems. Thus, the parallel annealing machines do not perform well in solving constrained combinatorial optimization problems, such as TSPs.

2.3 Combinatorial Optimization Problems

Combinatorial optimization problems can be divided into two categories, constrained and unconstrained [38]. The constrained problems are usually more complex than the unconstrained ones. Max-cut problems (as unconstrained problems) and traveling salesman problems (as constrained problems) are introduced as two typical types of combinatorial optimization problems.

2.3.1 Max-Cut Problems

By dividing a given graph $G = [\boldsymbol{V},\ \boldsymbol{E}]$ into two subsets, where $\boldsymbol{V}$ is the set of all vertices $(v_0, v_1, v_2, ..., v_n)$, and $\boldsymbol{E}$ is the set of all edges between the vertices, a max-cut problem is to find a division that makes the summation of the edges' cut weights maximum [39]. The Hamiltonian of an n-vertex max-cut problem can be expressed as:

$$H_{Max-cut} = \sum_{i<j} W_{ij}(a_i + a_j - 2a_i a_j), \tag{2.2}$$

where $a_i \in \{0, +1\}$ indicates that v_i is divided into subset $\boldsymbol{V_1}$ $(+1)$ or not (0), and W_{ij} denotes the weight of the edge between v_i and v_j. The Hamiltonian represents the weight summation of the cut edges.

Then the max-cut problem can be mapped to the Ising model by converting a_i $(\in \{0, +1\})$ to σ_i $(\in \{-1, +1\})$ as follows [40]:

$$H_{Max-cut} = \frac{1}{2}\sum_{i<j} W_{ij} - \frac{1}{2}\sum_{i<j} W_{ij}\sigma_i\sigma_j. \tag{2.3}$$

The first term in (2.3) is a constant. The second term correspond to the interaction term in (2.1), and it does not have a term for the external field.

2.3.2 Traveling Salesman Problems

A traveling salesman problem (TSP) is to find the shortest route to visiting all cities. Each city can only be visited once, and the salesman should return to the starting point [41]. The Hamiltonian of an n-city TSP can be expressed as [4]:

$$\begin{aligned} H_{TSP} &= A\sum_{k\neq l}^{n}\sum_{i}^{n} W_{kl}a_{ik}a_{(i+1)l} + B\sum_{i}^{n}(\sum_{k}^{n} a_{ik} - 1)^2 \\ &\quad + C\sum_{k}^{n}(\sum_{i}^{n} a_{ik} - 1)^2 , \end{aligned} \tag{2.4}$$

where a_{ik} ($\in \{0, +1\}$) indicates whether the kth city is visited ($+1$) or not (0) at the ith step, and W_{kl} denotes the distance between the kth city and the lth city. The first term in (2.4) is the objective function of the TSP, which computes the total distance of the route. The second and the third terms in (2.4) implement the constraints that prevent visiting multiple cities in one step and visiting a city more than once, respectively. These two terms take the minimum value 0 when $\sum_k a_{ik} = \sum_i a_{ik} = 1$. A, B and C are the parameters (with positive values) that balance the weights between the objective function and the constraints.

The TSP can be mapped to the Ising model by converting $a_{ik}(\in \{0, +1\})$ to $\sigma_{ik}(\in \{-1, +1\})$ as follows [4]:

$$\begin{aligned} H_{TSP} &= \frac{A}{4}\sum_{k\neq l}^{n}\sum_{i}^{n} W_{kl}\sigma_{ik}\sigma_{(i+1)l} + \frac{A}{2}\sum_{k\neq l}^{n}\sum_{i}^{n} W_{kl}\sigma_{ik} \\ &\quad + \frac{B}{4}\sum_{i}^{n}\sum_{k}^{n}\sum_{l}^{n}\sigma_{ik}\sigma_{il} + \frac{(n-2)B}{2}\sum_{i}^{n}\sum_{k}^{n}\sigma_{ik} \\ &\quad + \frac{C}{4}\sum_{i}^{n}\sum_{k}^{n}\sum_{j}^{n}\sigma_{ik}\sigma_{jk} + \frac{(n-2)C}{2}\sum_{i}^{n}\sum_{k}^{n}\sigma_{ik} \\ &\quad + \frac{A}{4}\sum_{k\neq l}^{n}\sum_{i}^{n} W_{kl} + (\frac{n^3}{4} - n^2 + n)(B + C). \end{aligned} \tag{2.5}$$

The last two constant terms in (2.5) that are unrelated to the states of spins are ignored when minimizing H_{TSP}. The first, third, and fifth terms correspond to the interaction term in (2.1), and the other terms correspond to the external field term in (2.1). Fig. 2.8 shows an example for mapping a 3-city TSP to the Ising model.

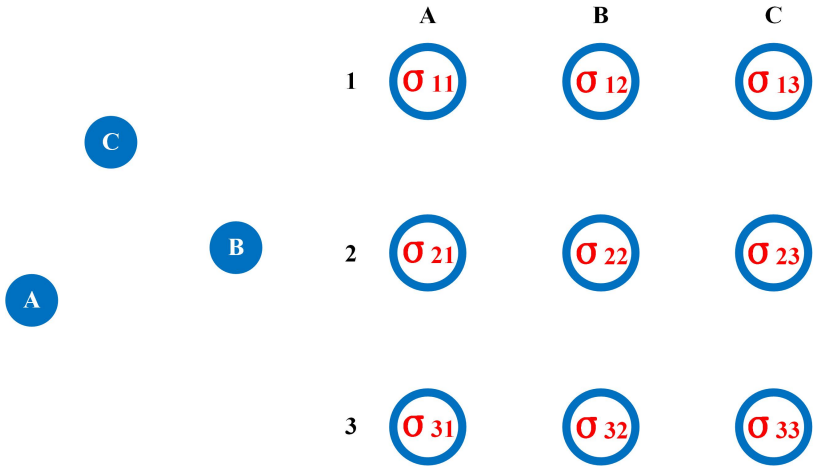


Figure 2.8: Mapping a 3-city TSP to the Ising model.

The σ_{22} equals to 1 when city B is visited at the second step, and the other spins in the same row and column (σ_{12}, σ_{21}, σ_{23}, and σ_{32}) should be -1 to conform the constraints.

2.4 A Clustering Approach

Clustering is a process of dividing a set of objects into several groups in terms of their characteristics [42], so that all objects in a group are similar and the objects in different groups are different. It has been implemented in many research areas, such as computational statistics and data mining [43].

In the TSPs, we call a solution that conforms to the constraints a feasible solution and a solution that does not conform to the constraints an infeasible solution. The feasible solution with the lowest Ising energy is the optimal solution. Every solution corresponds to a spin configuration in the Ising model, and the adjacent spin configurations of a feasible solution are infeasible solutions. The penalty terms make the

Ising energy of the infeasible solutions extremely high. Thus, every feasible solution corresponds to a local minimum in the Ising energy. The number of local minima is large for solving large-scale TSPs. It decreases the probability of an Ising machine finding the optimal or a near-optimal solution. However, the traveling distances of many feasible solutions are very long, and the efficiency of the problem solving can be increased by avoiding looking for those feasible but long-distance solutions. A clustering approach can be used for this purpose, as briefly explained below.

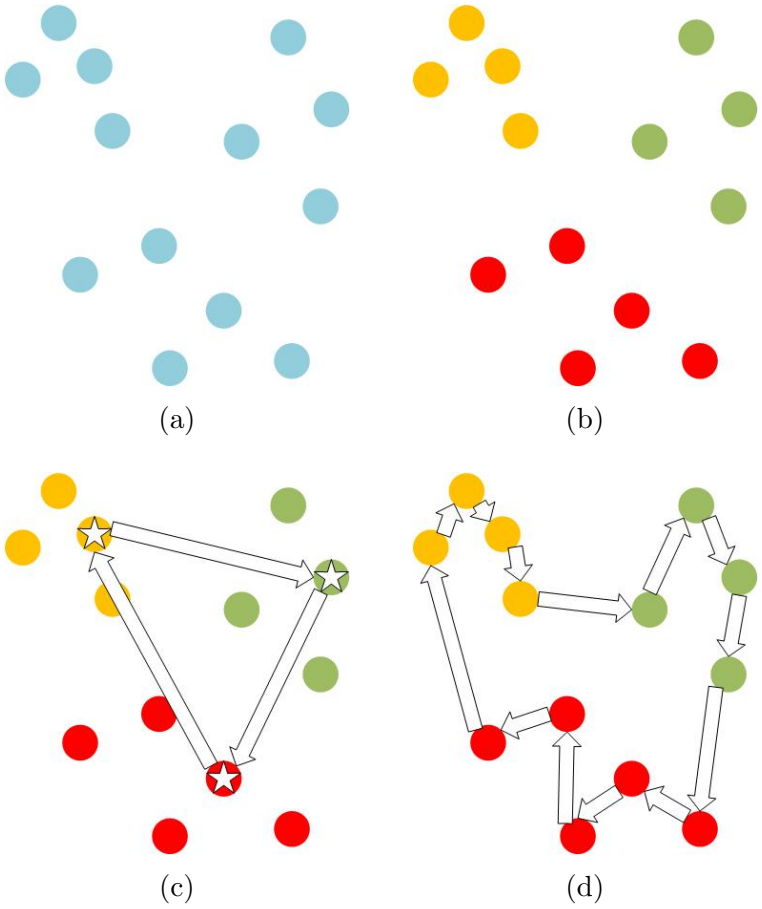


Figure 2.9: Solving a TSP with a clustering approach: (a) the original TSP, (b) after clustering, (c) solving the TSP consisting of the central points, (d) solving the original TSP.

First, the clustering approach divides cities into several groups with respect to their locations, as shown in Fig. 1 (b). Nearby cities are divided into the same group. Second, a smaller-scale TSP consisting of the central points of each group is solved,

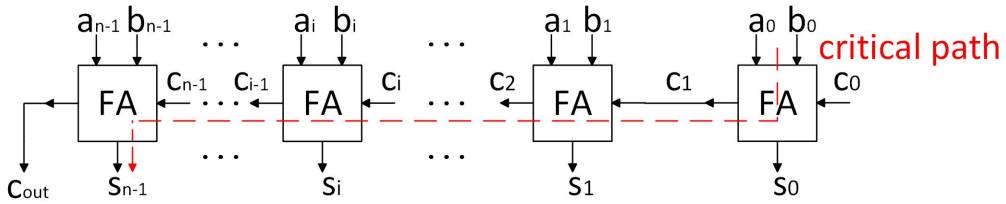


Figure 2.10: The ripple-carry adder [44].

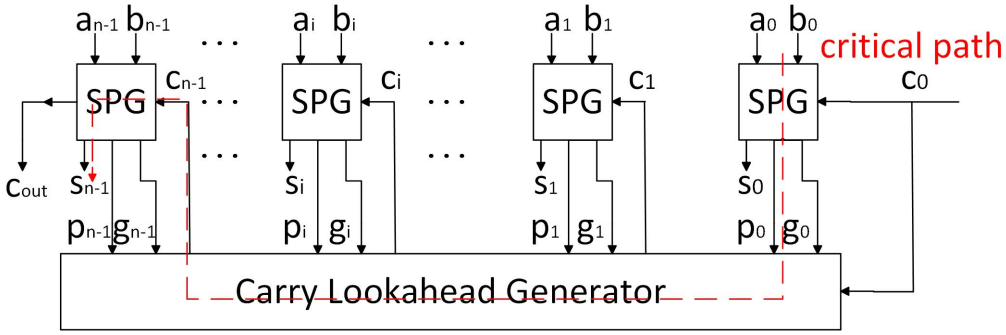


Figure 2.11: The carry look-ahead adder [44]

as shown in Fig. 1 (c). After obtaining a visiting order of these groups, a constraint dictates that the cities in the next group can not be visited until all cities in the current group have been visited. Finally, solving the original TSP with this visiting order constraint, as shown in Fig. 2(d).

2.5 Approximate Arithmetic

Approximate arithmetic is considered in this design to improve the hardware efficiency. The accumulation of local fields is the most time-consuming process for the Ising machine and does not require a high accuracy. Thus, approximation is applied to the adders in local field accumulation units.

An adder is a fundamental component in digital circuits. The ripple-carry adder (RCA) and the carry look-ahead adder (CLA) are the two best-known types. The carry in an n-bit RCA is propagated from one full adder to the next, while n full adders are cascaded, as shown in Fig. 2.10. Thus, the critical path delay of an n-bit

RCA is $O(n)$. An n-bit CLA generates in parallel, as shown in Fig. 2.11. It is much quicker than an RCA, because it's critical path is shorter. However, the CLA is more hardware-consuming than the RCA. Many approximate adders have been proposed to reduce the design's delay and hardware cost. There are four approximate adder methodologies, i.e., speculative adders, segmented adders, carry select adders, and approximate full adders [45].

Most carry chains do not propagate through an adder's entire length [46]. Thus, an almost correct adder (ACA) [47] that is based on the speculative adder [48] is proposed. It generates the carry utilizing the k least significant bits (LSBs) [44]. This method reduces the critical path delay to $O(log(k))$. Moreover, compared with conventional speculative adders, the ACA saves hardware by sharing the circuits in sub-carry generators.

As a basic structure of the segmented adders, the equal segmented adder (ESA), consists of several smaller sub-adders that work in parallel [44]. The carry inputs of each sub-adder are fixed. Therefore, its critical path delay is also $O(log(k))$ (k is the length of the segmented adders), as there is no carry propagation between sub-adders. Furthermore, the hardware cost of an ESA is much lower than an ACA because it does not require sub-carry generators.

Similar to the segmented adders, a carry select adder is divided into several blocks, and each block calculates the summation of k bits [49]. However, except the first block, all the others have two k-bit adders. One is for the summation with carry-in = 0, and the other for carry-in = 1. These two summations are selected as a part of the final result by the carry-out signal from the former block. The critical path delay of this adder is a little bit longer than the ESA's, as it has an additional multiplexer delay. Furthermore, the hardware cost is much higher because it requires two sub-adders and multiplexers. However, it achieves a high accuracy.

An approximate full adder modifies the least significant bits (LSBs) of an accurate adder to shorten the critical path and reduce the hardware cost. The lower-part-OR

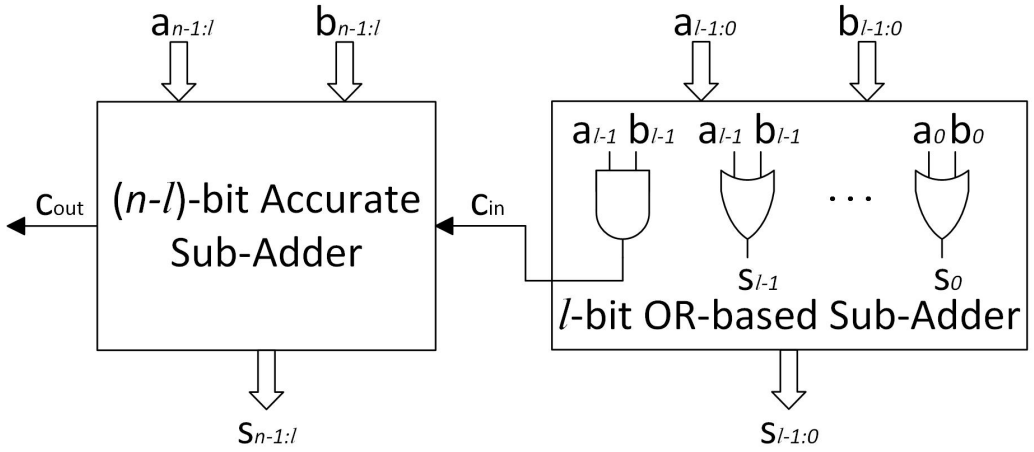


Figure 2.12: The lower-part-OR adder [44].

adder (LOA) [50] uses a simple approximate full adder. As shown in Fig. 2.12 [44], the lower part of the result is directly generated by using OR gates, and one AND gate is applied to generate the carry-out bit to the more significant part. The more significant part of the result is calculated via an accurate sub-adder. Therefore, its critical path delay decreases to $O(log(n-l))$, where l is the length of the lower part. Furthermore, the truncated adder (TruA), which directly prunes LSBs, is a baseline design of approximate full adders. It is efficient in a system that does not require a high accuracy.

The Ising machine is an approximate system. An Ising machine's speed and power dissipation are critical performance metrics. Therefore, the LOA and TruA are applied in this design for hardware efficiency improvement.

Chapter 3

Algorithm Design

3.1 Introduction

This chapter presents an improved parallel annealing (IPA) algorithm to solve constrained combinatorial optimization problems, such as the TSP, using parallel fully connected Ising machines. To the best of the authors' knowledge, this work is the first attempt to do so. The contributions lie in the following novelties aimed at improving the performance of fully connected Ising machines in solving TSPs: (1) using an exponential temperature function with a dynamic offset and (2) using k-medoids clustering in Ising model-based TSP solvers to preprocess data for improving the quality of solutions.

The remainder of this chapter is organized as follows. Section 3.2 reviews the background of parallel annealing algorithms. The IPA algorithm and the k-medoids are discussed in Section 3.3 for solving the TSPs. Section 3.4 reports the experimental results on TSP benchmarks. Section 3.5 concludes the chapter.

3.2 Parallel Annealing (PA)

In the two-layer spin structure for the Ising model, the couplings between σ_i^L and σ_j^R are denoted as $J_{ij}(i \neq j)$, whereas the couplings between σ_i^L and σ_i^R are called self-interactions (denoted as ω_i). Thus, the Hamiltonian for PA, H_P, is given by [1,

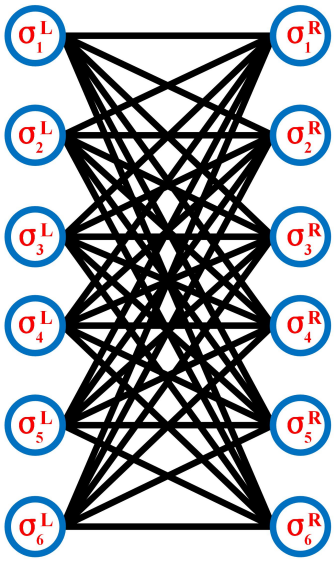


Figure 3.1: A two-layer spin structure for the Ising model [1, 33].

33]:

$$H_P = -\sum_{i,j} J_{ij}\sigma_i^L\sigma_j^R - \frac{1}{2}\sum_i h_i(\sigma_i^L + \sigma_i^R) + \omega_i \sum_i (1 - \sigma_i^L\sigma_i^R). \tag{3.1}$$

Only when the self-interactions ω_i are sufficiently large, are the spin configurations in both layers the same, i.e., $\sigma_i^R = \sigma_i^L$. Thus, the third term in (3.1) can be eliminated, so (3.1) becomes the same as (2.1) [1, 33]. ω_i is given by [33]:

$$\omega_i = \begin{cases} \sum_{s_j \in S} |J_{ij}| - \frac{1}{2}\sum_{s_j \in C} |J_{ij}| & (s_i \in C) \\ \frac{\lambda}{2} & (s_i \notin C) \end{cases}, \tag{3.2}$$

where λ is the largest eigenvalue of $-\boldsymbol{J}$ ($\boldsymbol{J}$ is a matrix of J_{ij}), s_i is the ith spin, C is a subset of the set of all spins S and C satisfies $C = \{s_i | \lambda \geq \sum_{s_j \in S} |J_{ij}|\}$.

The spin-flip probability is calculated using the Metropolis algorithm [51]. If σ_i^L is flipped, the total energy will be increased by

$$\Delta E_i = 2\sigma_i^L(\frac{h_i}{2} + \sum_j J_{ij}\sigma_j^R + \omega_i\sigma_i^R). \tag{3.3}$$

Then, the new spin-flip probability is $\min\{1, exp(-\Delta E_i/T)\}$, where T is the temperature.

To improve the efficiency of annealing, dropout and momentum scaling are introduced in [33]. The dropout sets each ω_i to be "0" with a decreasing probability. The momentum scaling multiplies every interaction ω_i by an increasing factor from 0 to 1. Thus, at the end of momentum annealing, every interaction ω_i will return to the value computed in (3.2) to ensure $\sigma_i^R = \sigma_i^L$ in (3.1).

3.3 Improved Parallel Annealing for TSPs

3.3.1 Improved Parallel Annealing (IPA)

The IPA for solving TSPs is shown in Algorithm 2. An exponential temperature function is used in the IPA and a dynamic offset is applied to the temperature function. First, the state of a spin is randomly initialized to "-1" or "$+1$", and the temperature increment (ΔT) due to the dynamic offset is initialized to "0". In Fig. 1(c), spins in the left layer are updated when the current step *step* is odd; otherwise, the spins in the right layer are updated. In each iteration ($step \in [1, iternum]$), the dropout rate (p_{step}) and the momentum scaling factor (c_{step}) are updated, where the *iternum* is the total number of iterations. The temperature (T_{step}) is then recalculated, where r in Algorithm 1 is the cooling coefficient. During the annealing, the self-interaction (ω_{ik}) is set to "0" with the probability p_{step} or decreased to $c_{step} \cdot \omega_{ik}$. Then, the energy variation (ΔE_{ik}) when σ_{ik} is flipped is evaluated using the spin interaction (J_{ikjl}) and the updated ω_{ik}. Subsequently, the spin-flip probability (P_{ik}) is calculated using the Metropolis algorithm. If P_{ik} is larger than a randomly generated number within $(0, 1)$, the spin will be flipped. Otherwise, the spin will remain unchanged. After each iteration, if no spin is flipped, ΔT will increase. Otherwise, ΔT will be reset to "0". Finally, the spin configuration (σ) at the end of iterations is output as the solution to the combinatorial optimization problem found by the IPA.

Algorithm 2 Improved Parallel Annealing for TSPs [35]

Input: spin interaction: $\boldsymbol{J}$; external magnetic field: $\boldsymbol{h}$;
the number of cities: M; self-interaction: $\boldsymbol{\omega}$;
hyperparameters: $iternum$, T_{init}, T_{inc}, r
Output: spin configuration (σ)

Initialize spin configurations
$T_s \Leftarrow T_{init}$
$\Delta T \Leftarrow 0$
for $step = 1$ to $iternum$ **do**
  if $step$ is odd **then**
    $A \Leftarrow L, B \Leftarrow R$
  else
    $A \Leftarrow R, B \Leftarrow L$
  end if
  Update p_{step} and c_{step}
  $T_{step} \Leftarrow (T_{step} + \Delta T) \cdot r^{step-1}$
  for $i = 1$ to M **do**
    for $k = 1$ to M **do**
      Temporarily set $\omega_{ik} \Leftarrow 0$ with the probability p_{step}, and temporarily decrease $\omega_{ik} \Leftarrow c_s \cdot \omega_{ik}$
      $\Delta E_{ik} \Leftarrow 2\sigma_{ik}^A(\frac{h_{ik}}{2} + \sum_{j,l} J_{ikjl}\sigma_{jl}^B + \omega_{ik}\sigma_{ik}^B)$
      $P_{ik} \Leftarrow \min\{1, exp(-\Delta E_{ik}/T_{step})\}$
      if $P_{ik} > rand$ **then**
        $\sigma_{ik}^A \Leftarrow -\sigma_{ik}^A$
      end if
    end for
  end for
  if no spin is flipped **then**
    $\Delta T \Leftarrow \Delta T + T_{inc}$
  else
    $\Delta T \Leftarrow 0$
  end if
end for

3.3.2 A Temperature Function

The classical annealing algorithm with parallel spin-update uses a logarithmic function as the temperature function to solve the max-cut problem as [33]:

$$T_{step} = \frac{1}{\beta_0 \ln(1+s)}, \tag{3.4}$$

where β_0 is a scaling factor for the inverted value of temperature and T_{step} denotes the temperature in the *step*th iteration. When the Ising model reaches a local minimum or ground state and T_{step} is sufficiently small, the flip probability for each spin, P_{ik}, is considered to be close to 0 (see lines 15 and 16 in Algorithm 2). With a proper β_0, however, the temperature only decreases to a value that results in low flip probabilities for all spins. Hence, it is possible for the Ising model to escape from local minima when solving max-cut problems.

To solve a TSP, the temperature required for maintaining low spin-flip probabilities is larger because ΔE_{ik} is larger due to the constraints. However, an Ising model cannot reach a local minimum or ground state, or meet the constraints at such a temperature. Thus, the temperature needs to be sufficiently low at the end of annealing when solving TSPs. It will, therefore, be difficult for the Ising model to escape from a local minimum. Moreover, the temperature using a logarithmic function rapidly decreases, so it will prevent the Ising model from traversing additional local minima, thereby reducing the quality of solutions. Hence, an exponential function is used as the temperature function to solve the TSP, as

$$T_s = T_{init} \cdot r^{s-1}, \tag{3.5}$$

where T_{init} is the initial temperature and r is the cooling rate. The slower decreasing rate of the exponential function makes the Ising model stay longer at a high temperature, therefore improving the quality of solutions.

Considering that the number of local minima increases with increasingly large TSPs, the Ising model is prone to be stuck in a local minimum during annealing.

Reducing the time spent in a local minimum can improve efficiency. Thus, we consider introducing a dynamic offset, as in [30–32], into the temperature function. To increase the probability of escaping from a local minimum, the temperature T_{step} needs to be very large. Therefore, ΔT is added to T_{step}, where ΔT is increased by T_{inc} if the spin configuration is unchanged. Lastly, ΔT is reset to zero after a change of the spin state has occurred.

The improvement of the solution quality after using an exponential function with a dynamic offset and the details for setting a proper T_{inc} are discussed in Section 3.4.

3.3.3 A Clustering Approach

The solution quality drastically deteriorates when the number of cities in the TSP is large. We further consider a clustering approach [4] to improve the quality of the solution. The basic idea is to group the nearby cities into one cluster and use the central point to represent each cluster. Then the TSP consisting of those central points can be solved by using the IPA. After learning the visiting order of each cluster, the original TSP can be more efficiently solved. This Ising machine can avoid producing solutions that conform to the constraints but with very long travel distances. For example, if cluster A contains three cities and is first visited among the clusters, then the visiting order of these three cities will be confined to the first three steps.

The k-medoids and k-means are two typical clustering approaches. To divide M vertices into k clusters, the first step of the k-means approach is to randomly generate k new vertices as the central points of the k clusters, whereas the k-medoids method chooses k vertices from the original set as the central points. In the second step, after the k central points are obtained, the other $(M - k)$ vertices in the set are assigned to the closest central point and form a cluster. In the third step, the k-means method generates a new central point for each cluster according to the mean value of the coordinates of the vertices in the cluster, whereas the k-medoids method chooses the

vertex with the smallest sum of distances from the other vertices in the same cluster as the new central point. Then, the second and the third steps are repeated until there is no change in any cluster.

Algorithm 3 The k-medoids clustering [35]

Input: distance matrix: $\boldsymbol{W}(M \times M)$; the number of clusters: k
Output: vertex indexes (with cluster labels)

Step 1
for $i = 1$ to M **do**
  $D_i = \sum_{j=1}^{M} W_{ij}$
end for
Choose k vertices (v) with the first k smallest D as the central points
Step 2
for $i = 1$ to M **do**
  Assign v_i to the closest central point and mark the label of v_i with the index of the corresponding central point
end for
Step 3
for each cluster **do**
  for each v_i in the cluster **do**
    $d_i = \sum_j W_{ij}$
  end for
  Choose v with the smallest d to be the new central point of the current cluster
end for
Repeat Step 2 and Step 3 until there is no change of elements in each cluster

Compared with the k-means, the disadvantage of k-medoids is the computation time for each cluster in the third step, $O(m^2)$, where m is the number of vertices in one cluster. However, there are $M \times M$ accumulators in the circuit of an Ising model for $M \times M$ spins that solves an M-city TSP, where $M = \sum_{i=1}^{k} m_i$ and m_i is the number of vertices in the ith cluster. Thus, this computation time can be reduced to $O(m)$ as it can be calculated in parallel with m accumulators. Furthermore, the calculation of the distances between the vertices is not required in an Ising machine as the distance values are included in the system's input, i.e., in the spin interaction matrix $\boldsymbol{J}$. In contrast, the k-means method needs extra arithmetic units to compute the distances between the vertices and the central points. Therefore, using k-medoids

for clustering can achieve a higher hardware efficiency than using k-means with no performance trade-off in an Ising machine.

The k-medoids clustering was proposed in [52], but it is first applied in an Ising model-based TSP solver in this work, as shown in Algorithm 3. A strategy for choosing the k vertices at the center of the map as initial central points is applied to improve the efficiency of k-medoids. The k vertices with the k smallest sums of distances from all the other vertices are selected as initial central points.

To implement the visiting restrictions, an M-by-M matrix $\boldsymbol{h_p}$ is added to the external magnetic field matrix, $\boldsymbol{h}$. For example, if the first three cities are confined to be visited at the first three steps, $\boldsymbol{h_p} = \begin{pmatrix} \boldsymbol{a} & \boldsymbol{b} \\ \boldsymbol{c} & \boldsymbol{d} \end{pmatrix}$, where $\boldsymbol{a}$ and $\boldsymbol{d}$ are 3-by-3 and $(M-3)$-by-$(M-3)$ zero matrices, respectively; $\boldsymbol{b}$ and $\boldsymbol{c}$ are 3-by-$(M-3)$ and $(M-3)$-by-3 matrices of $M \cdot max\{abs(\boldsymbol{J})\}$ (as a large value), respectively. The $max\{abs(\boldsymbol{J})\}$ returns the entry in matrix $\boldsymbol{J}$ with the largest absolute value.

3.4 Experimental Results

Seven benchmark datasets are used in the experiments, including $burma14$, $ulysses16$, $ulysses22$, and four other sub-datasets randomly selected from the TSPLIB benchmark ($n = 6$ and 12 from $gr431$ and $n = 7$ and 10 from $ali535$). The average (Ave), maximum (Max), minimum (Min), and standard deviation (Std) of the travel distances are obtained after performing annealing by 100 times with an iteration number of $10k$. The simulation was run in MATLAB on an AMD processor Ryzen 5 $3600X$ (3.8 GHz).

We evaluate the effect of the penalty parameters B and C on the performance of the IPA with six benchmark datasets, including $burma14$, $ulysses16$, $n = 6$ and 12 from $gr431$ and $n = 7$ and 10 from $ali535$. The results are obtained with $T_{inc} = max\{abs(\boldsymbol{J})\}$, $r = 0.97$, and $T_{init} = 1 \times 10^7$. Here, T_{init} is an arbitrarily large value and r is selected to ensure $T_{inc} \cdot r^{step-1}$ close to 0 at the end of annealing.

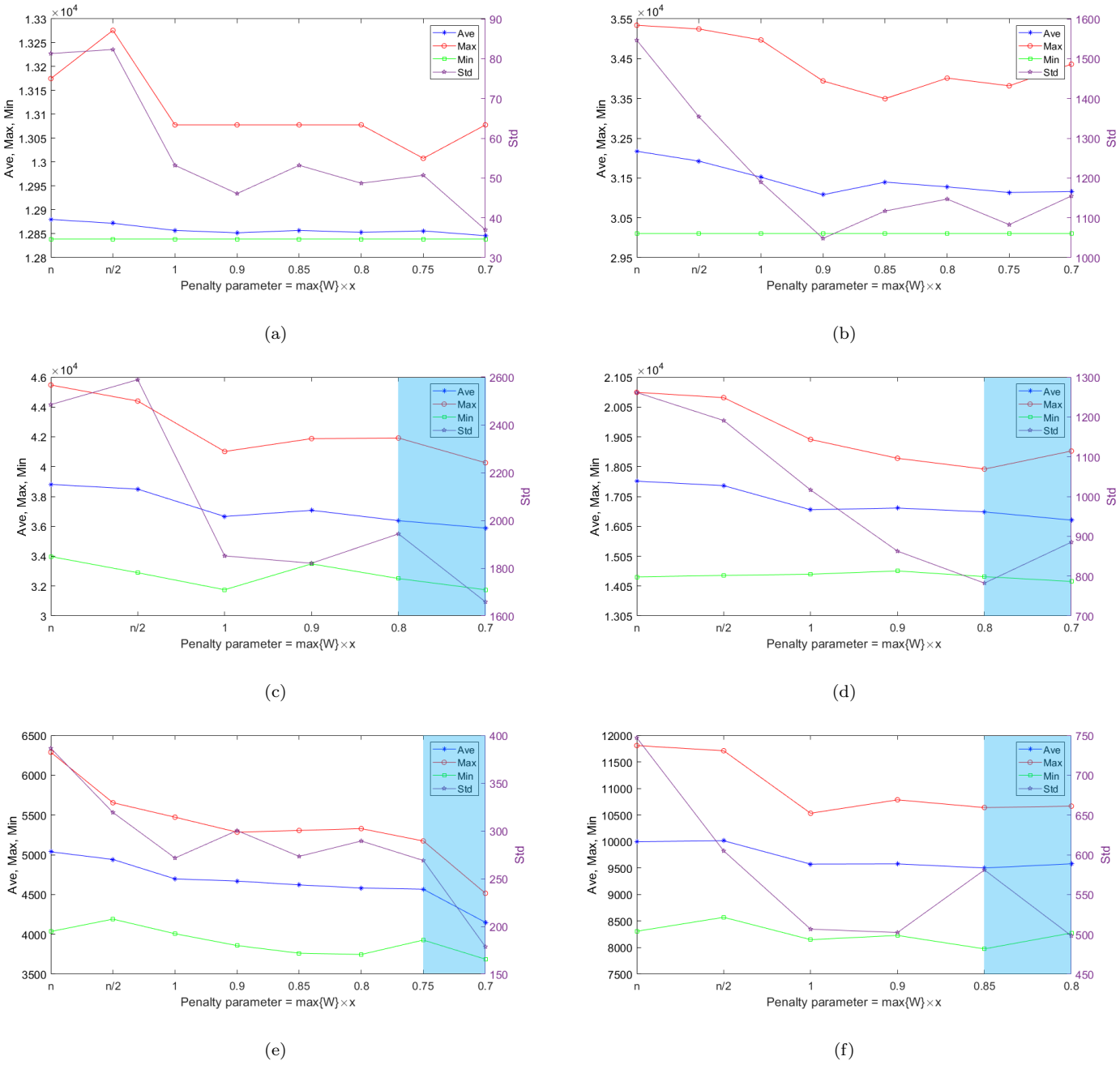
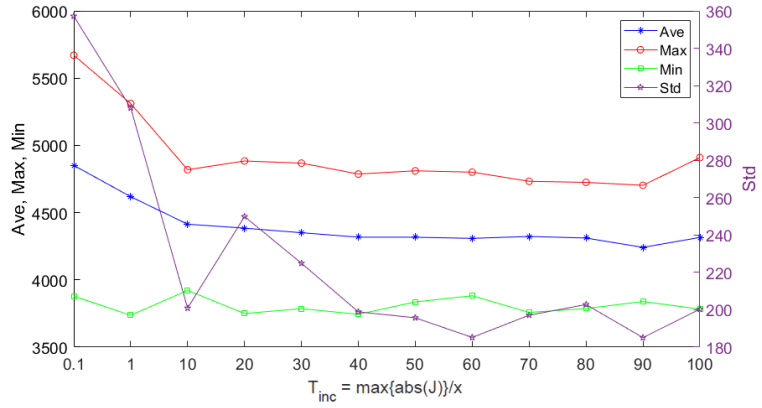
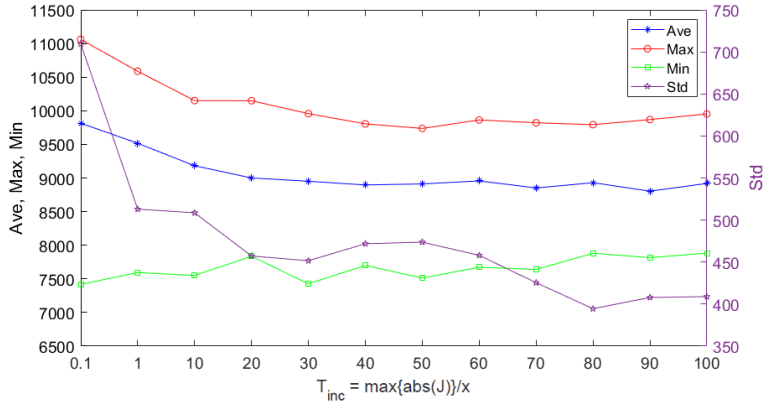


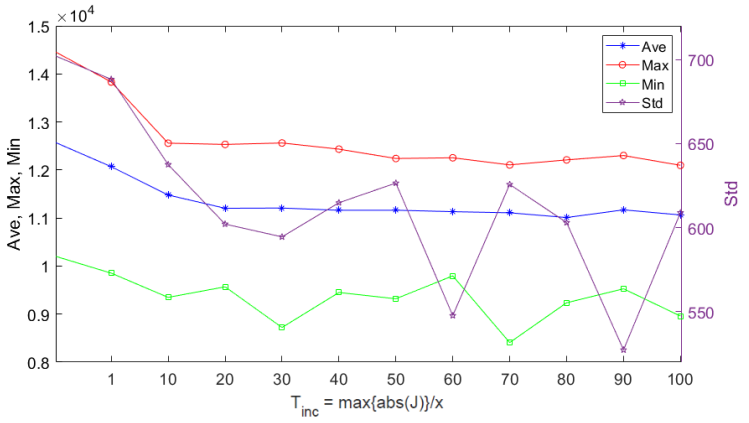
Figure 3.2: The effect of the penalty parameters B and C ($B = C$) on the quality of solutions: (a) for $n = 6$ from $gr431$, (b) for $n = 7$ from $ali535$, (c) for $n = 10$ from $ali535$, (d) for $n = 12$ from $gr431$, (e) for the benchmark $burma14$, and (f) for the benchmark $ulysses16$. The blue shadow area indicates the results that do not meet constraints [35].



(a)



(b)



(c)

Figure 3.3: The effect of T_{inc} on the quality of solutions: (a) for the benchmark *burma*14, (b) for the benchmark *ulysses*16, and (c) for the benchmark *ulysses*22 [35].

As shown in Fig. 3.2, for all six TSPs, more stable solutions with smaller Ave can be obtained when the penalty parameters decrease. It is due to the fact that for lower penalty parameter values, the Ising model escapes from the local minima with a higher probability. However, the parameter values must be large enough to ensure all solutions meet the constraints, which are violated when B and C are smaller than $0.85 \times \max\{\boldsymbol{W}\}$. The improvement in solution quality is less significant when B and C are smaller than $1 \times \max\{\boldsymbol{W}\}$. Therefore, we choose $B = C = 1 \times \max\{\boldsymbol{W}\}$ as the penalty parameter setting in our further experiments.

As a key to increasing the probability of escaping from a local minimum, an appropriate setting of T_{inc} can optimize the efficiency of an Ising model. Therefore, we investigated the effect of different T_{inc} values on the Ave and Std of solutions found by the IPA. The results for the three benchmarks, *burma*14, *ulysses*16, and *ulysses*22, are obtained with penalty parameters $B = C = 1 \times \max\{\boldsymbol{W}\}$, $r = 0.97$, and $T_{init} = 1 \times 10^7$. As shown in Fig. 3.3, the Ave, Max, and Std of the travel distances found by the Ising models decrease when T_{inc} decreases from $10 \times max\{abs(\boldsymbol{J})\}$ to $max\{abs(\boldsymbol{J})\}/10$. Furthermore, the Ave, and Max tend to be stable when T_{inc} is between $max\{abs(\boldsymbol{J})\}/10$ and $\frac{max\{abs(\boldsymbol{J})\}}{90}$. The further decrease of T_{inc} results in an increase of Ave, so it degrades the quality of solutions. A small T_{inc} reduces the chance of the Ising models escaping from local minima, and thus a larger iteration number is required for finding a suboptimal solution.

To evaluate the performance of the proposed methods, MA [33] and DA [32] are considered for comparison. The MA implements parallel spin-update but using a logarithmic temperature function, while the DA employs a dynamic offset but without parallel spin-update. Three benchmarks, *burma*14, *ulysses*16, and *ulysses*22, are used. The results are obtained with $r = 0.97$, $T_{inc} = \frac{max\{|\boldsymbol{J}|\}}{90}$ and $T_{init} = 1 \times 10^7$ for the IPA and DA. For the MA, $\beta_0 = 9 \times 10^{-4}$ for *burma*14, $\beta_0 = 8 \times 10^{-4}$ for *ulysses*16, and $\beta_0 = 5 \times 10^{-4}$ for *ulysses*22. These β_0 values are chosen to produce the best solution quality in the experiment. The penalty parameters in (2.5) are set as $A = 1$,

Table 3.1: (Unitless) Travel distances by using the IPA, MA, and DA for solving the TSP [35]

Metrics	$iternum = 10k$			$iternum = 50k$		
	IPA	MA	DA	IPA	MA	DA
burma14						
Ave	4241.6	5322.4	8832.9	4018.5	5133.3	6451.8
Max	4703.0	7524.0	9655.0	4423.0	7443.0	8009.0
Min	3839.0	4178.0	7507.0	3580.0	4099.0	4945.0
Std	185.1	683.7	379.8	159.9	547.7	696.4
ulysses16						
Ave	8804.2	11513.0	12722.0	8387.6	11451.0	12040.0
Max	9869.0	13992.0	15454.0	9218.0	14366.0	14669.0
Min	7816.0	8859.0	9827.0	7554.0	9242.0	8815.0
Std	407.9	1113.6	1141.5	303.3	1057.4	1240.4
ulysses22						
Ave	11170.0	13811.0	16619.0	10389.0	13367.0	16435.0
Max	12301.0	18914.0	20316.0	11167.0	16799.0	18862.0
Min	9527.0	11154.0	13224.0	9163.0	9363.0	13000.0
Std	527.3	1622.3	1425.6	433.7	1284.8	1156.4

$B = C = 1 \times \max\{\boldsymbol{W}\}$. Table 3.1 shows the performance of the IPA, MA, and DA for solving the TSP. The algorithms with parallel spin-update (IPA and MA) obtain lower Ave than DA for all three benchmarks. However, the Ave obtained by MA can hardly be improved by increasing the number of iterations. This occurs because the annealing algorithm using a logarithmic temperature function is easily stuck in a local minimum when solving TSPs. On the contrary, the algorithms that employ a dynamic offset, such as the IPA and DA, can find shorter distances when the iteration number increases. For solving *burma*14, when the iteration number is $10k$, the IPA decreases the Ave by 52.0% compared with DA and by 20.3% compared with MA. The required iterations for the DA, MA, and IPA to produce an Ave around 4920 are $250k$, $20k$, and $1k$, respectively, while the runtimes are 4.44 seconds, 1.99 seconds, and 0.10 seconds, respectively. The IPA achieves a 44.4× speed-up in runtime compared with DA, and 19.9× compared with MA.

Further decrease in Ave is obtained by using the clustering approach. We applied k-medoids twice for each benchmark. The original TSP is clustered into a second-level TSP with k_1 centric points, and the second-level TSP is clustered into a third-level TSP with k_2 centric points. The iteration number for solving the third-level TSP is 1000; it is 2500 for the second-level TSP and 3000 for the original TSP, so the total iteration number is 6500. As shown in Table 3.2, the reduction in Ave is 51.8% or 42.0% compared to DA or MA for *ulysses*22 ($iternum = 10k$), 39.4% or 33.1% for

Table 3.2: (Unitless) Travel distances by using the k-medoids clustering in the IPA for solving the TSP [35]

Metrics	burma14 ($k_1 = 7, k_2 = 4$)	ulysses16 ($k_1 = 8, k_2 = 4$)	ulysses22 ($k_1 = 10, k_2 = 6$)
Ave	3813.8	7705.0	8011.4
Max	4334.0	8923.0	9371.0
Min	3345.0	6686.0	7219.0
Std	268.9	440.9	433.2

*ulysses*16, and 56.8% or 28.3% for *burma*14, respectively.

Furthermore, the performance of the IPA in solving unconstrained combinatorial optimization problems, such as max-cut problems, was tested. Two max-cut benchmarks, $G11$ and $G12$, were used. The parameter settings of IPA and DA were the same as those for the TSP tests. However, for MA, $\beta = 0.1$ in the max-cut tests because the bit-with of interactions is 2 [33]. Fig. 3.4 shows the distribution of the results after performing annealing 100 times. The x-axis is the cut value, while the y-axis is the distribution frequency of the cut values. The iteration number for IPA and MA is $1k$, and it is $5k$ for DA. IPA obtains a higher average cut value and a lower standard deviation than MA and DA. The improvement is 2.6% for $G11$ and 1.7% for $G12$ compared with MA, 21.7% for $G11$ and 12.6% for $G12$ compared with DA. The performance gap between IPA and MA is insignificant because MA has already been optimized for solving unconstrained problems. Moreover, these results are obtained by using DA with a five times larger iteration number. Thus, the improvements in solution quality and performance after realizing the parallel spin update are significant.

3.5 Conclusions

This work attempts to solve constrained combinatorial optimization problems, such as the TSP, using parallel fully connected Ising machines. Specifically, an IPA algorithm is proposed to leverage an exponential temperature function with a dynamic offset and a k-medoids clustering approach. The exponential temperature function with a dynamic offset can alleviate the problem of being stuck in local minima, while the k-medoids clustering significantly reduces the average travel distance. The IPA is at least an order of magnitude faster than DA and MA to find a similar average travel distance. A shorter average travel distance can further be found by the IPA with a smaller number of iterations due to the use of k-medoids clustering. Moreover, the proposed IPA also excels in solving unconstrained combinatorial optimization prob-

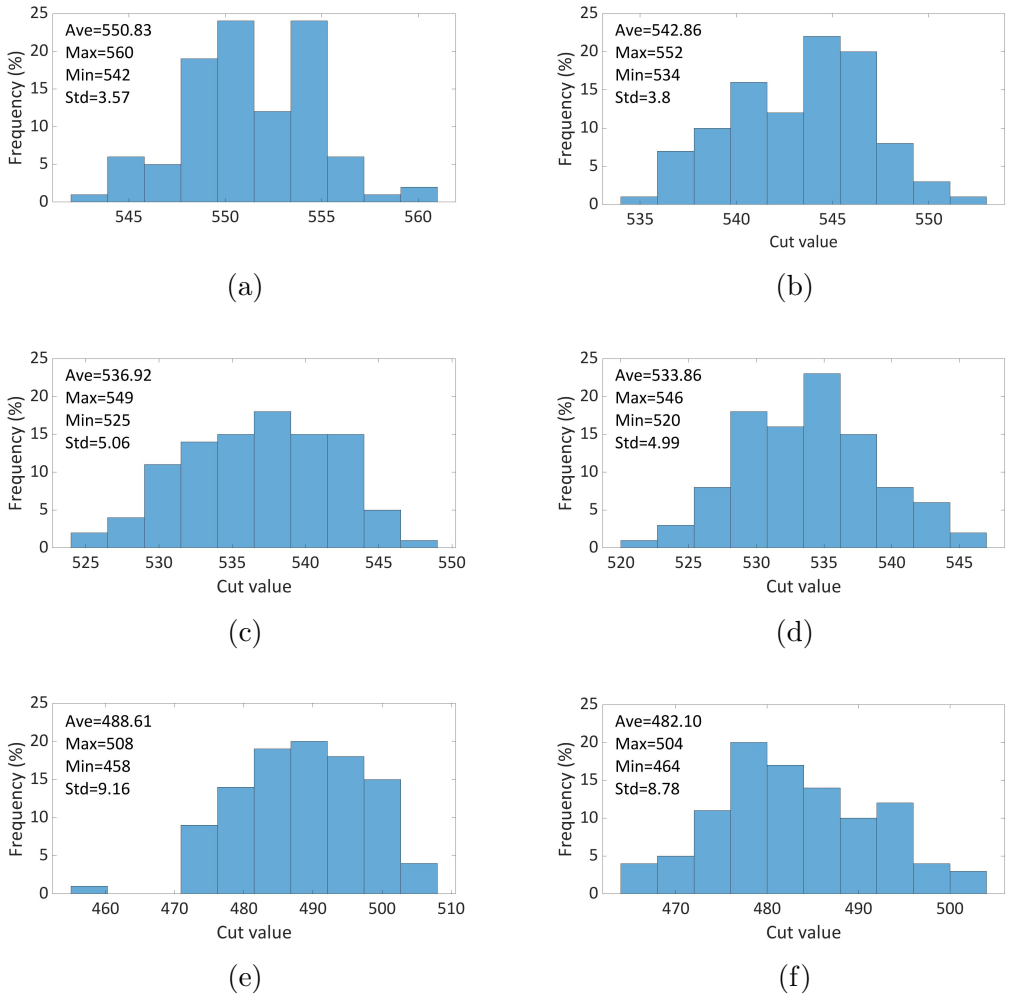


Figure 3.4: Max cut results obtained using IPA, MA, and DA: (a) solving G11 using IPA, (b) solving G12 using IPA, (c) solving G11 using MA, (d) solving G12 using MA, (e) solving G11 using DA, (f) solving G12 using DA.

lems, such as max-cut problems. These results pave the way for the development of energy-efficient circuit architectures for solving combinatorial optimization problems using the Ising model. The hardware design of this improved parallel annealing machine will be carried out in the next chapter.

Chapter 4

Hardware Design

4.1 Introduction

The circuit design of an improved parallel annealing Ising machine is presented in this chapter. Our architecture is inspired by the parallel annealing machine in [1]. However, a half-precision floating-point representation is implemented in this design to obtain a more extensive range for the coefficients, thus solving more complicated problems. The contributions lie in the following improvements aimed at the balance of speed and hardware efficiency: (1) the design of a new local field accumulator, which reuses adders in a pipeline for variable calculation; (2) approximating the momentum scaling factor to a linear increment and the proposal of a new self-interaction generating unit (SIGU), (3) the design of a new buffer-based energy calculation unit, and (4) using approximate arithmetic to improve the hardware efficiency. A model with 64 spins is shown as an example to demonstrate the function and performance of the circuit.

The remainder of this chapter is organized as follows. Section 4.2 introduces the components' circuits in the proposed improved parallel annealing machine. The approximate schemes are discussed in Section 4.3 for improving the hardware efficiency and evaluates the function of the optimized circuit design. Section 4.4 concludes this chapter.

4.2 Architecture and Circuit Design

4.2.1 Architecture of the Ising Machine

Fig. 4.1 shows the hardware architecture of the improved parallel annealing-based Ising machine with 64 spins. It is inspired by the parallel annealing machine in [1]. It consists of nine components, i.e., a local field accumulator unit (LAU), a spin update unit (SUU), a delta-driven simultaneous spin update unit (DDSS), a controller, an annealing schedule unit (ASU), a self-interaction generating unit (SIGU), a random number generator (RNG), a solution update unit, and a memory block. The controller, LAU, SIGU, and solution update unit are newly designed; while others are inspired by [1].

An Ising machine with 64 spins is implemented in this design. Thus, 64 LAUs, SUUs, RNGs, SIGUs are required in the architecture. The local field (lf_i), dropout rate (p_{step}), momentum scaling factor times self-interaction ($c_{step} \cdot \omega_{i,original}$, $\omega_{i,original}$ is the value in (3.2)), and dynamic offset (ΔT) are calculated in the LAUs. In particular, each LAU generates one lf_i and one $c_{step} \cdot \omega_{i,original}$. The SIGUs calculate 64 new self-interactions ($\omega_{i,step}$) with p_{step}, $c_{step} \cdot \omega_{i,original}$, and 64 random variables generated by RNGs. The SUUs use random numbers from RNGs, lf_i from the LAUs, and $\omega_{i,step}$ from the SIGUs to compute the new states of spins, $lf_i \cdot \sigma_i$ values, and Δ_i that indicates whether the ith spin is flipped. Two 64-bit vectors, $sigmas$ and Δs, consist of the 64 σ_i^{new} and Δ_i, respectively, while $lf_i \cdot \sigma_i$ consists of an $inout\_array$ type signal, called $energies$. The $inout\_array$ signal type is defined by an array (63 downto 0) of std_logic_vector (15 downto 0). The $sigmas$ and Δs are sent to the DDSS to obtain $index$ and σ_j^{old}, while the $sigmas$ and $energies$ values are used in the solution update unit to compute the solution. The $index$ is sent to the memory block to select corresponding $2 \cdot J_{0,j}$ to $2 \cdot J_{63,j}$. The $2 \cdot J_{0,j}$ to $2 \cdot J_{63,j}$ and σ_j^{old} are used in the LAUs for new lf_is' calculation. Moreover, the controller determines the behaviors of the entire system by coordinating the circuit timing with a 12-bit $instruction$ signal.

The details of each unit are illustrated in the following subsections.

4.2.2 A Local Field Accumulator (LAU)

A LAU is newly designed for reusing its adder in a pipeline. The energy variation (ΔE_i), shown in (3.3), is required to calculate the spin-flip probability. The states of spins on the left and right layers, denoted by σ_i^L and σ_i^R, respectively, can be represented by σ_i^{new} and σ_i^{old}, where σ_i^{new} denotes σ_i^L (or σ_i^R) and σ_i^{old} denotes σ_i^R (or σ_i^L) for updating σ_i^L (or σ_i^R). A larger ΔE_i value leads to a lower spin-flip probability, according to [32]:

$$P(\Delta E_i) = \begin{cases} \min\left[1, exp(\frac{-\Delta E_i}{T})\right] & (Metropolis) \\ \frac{1}{1+exp(\frac{\Delta E_i}{T})} & (Gibbs) \end{cases}. \tag{4.1}$$

The last term in (3.3), $2\omega_{i,step} \cdot \sigma_i^R \cdot \sigma_i^L$, becomes $2\omega_{i,original}$ when $\sigma_i^R = \sigma_i^L$. It decreases the spin-flip probability and makes σ_i^L the same as σ_i^R. Thus, the last term can be simplified to $2\omega_{i,step}$ to decrease the spin-flip probability at the end of annealing when the value of $\omega_{i,step}$ is large. Then, the energy variation function implemented on the circuit is shown in (4.2):

$$\Delta E_{i(hardware)} = 2\sigma_i^{new}(\frac{h_i}{2} + \sum_j J_{ij}\sigma_j^{old}) + 2\omega_{i,step}. \tag{4.2}$$

The LAU calculates the local field, $(\frac{h_i}{2} + \sum_j J_{ij}\sigma_j^{old})$ in (4.2). This unit is idle when the other units are working. Thus, it can be utilized to accumulate other values for the system. Here, the LAU is also used for the calculation of the dropout rate (p_{step}), momentum scaling factor times self-interaction ($c_{step} \cdot \omega_{i,original}$), and dynamic offset ($\Delta T$). As shown in Fig. 4.2, it consists of a multiplier, four registers, two multiplexers, and a demultiplexer. Only one LAU needs four registers in a system and others just require two, as p_{step} and ΔT are two variables that are shared among all spins. The demultiplexer and two multiplexers share one select signal (se), while four different signals control the four registers. When the LAU is working for local

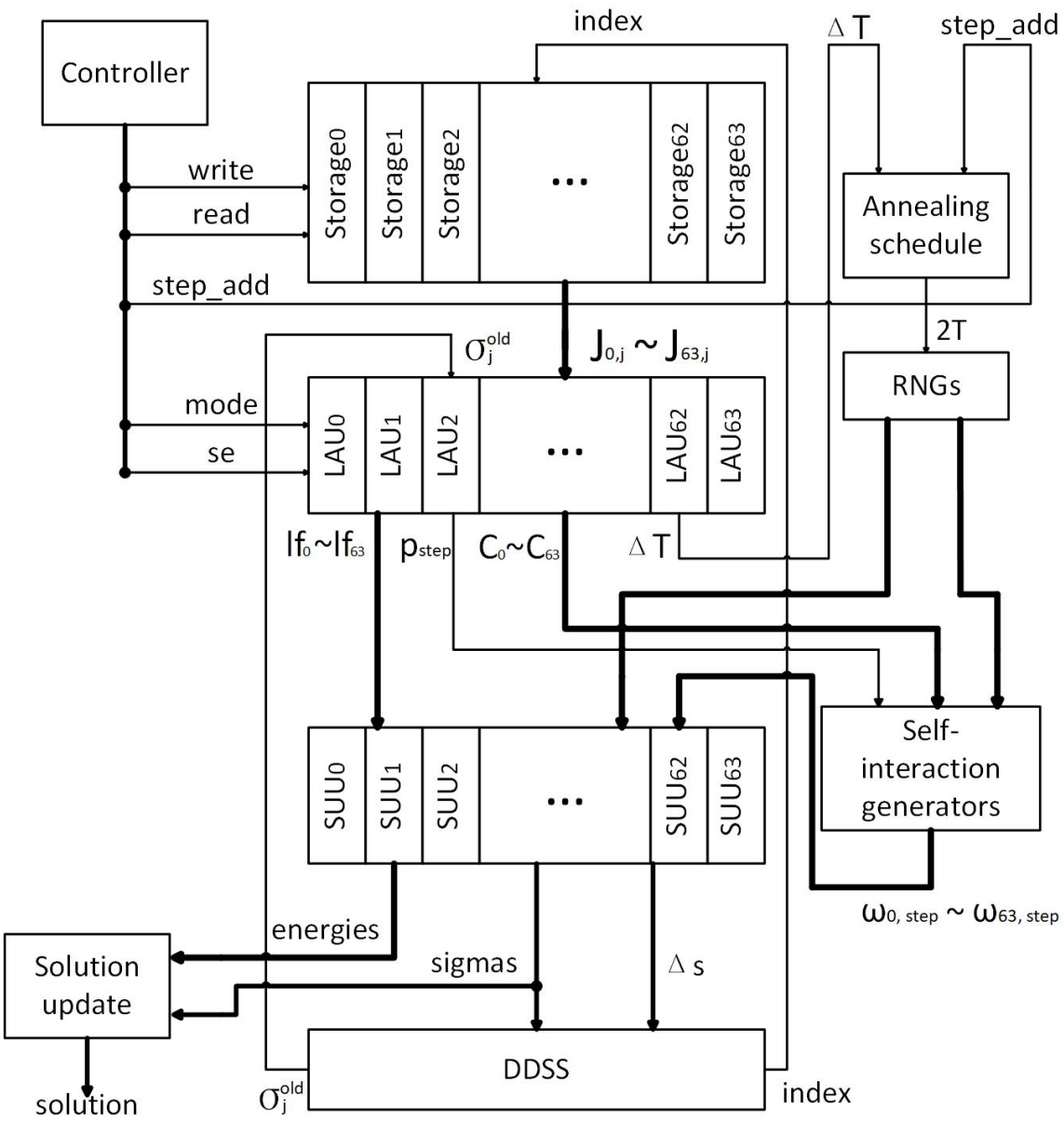


Figure 4.1: The architecture of the Ising machine [1].

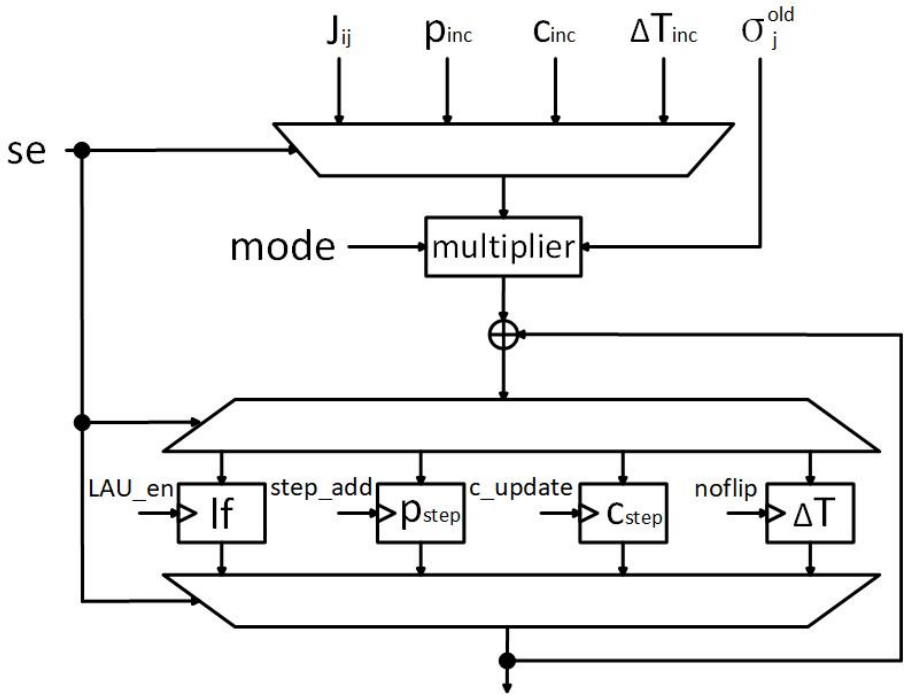


Figure 4.2: The local field accumulator unit (LAU).

field accumulation, $mode = 1$. se selects $2 \cdot J_{ij}$ and σ_j^{old} is multiplied with $2 \cdot J_{ij}$ by the multiplier. Then the adder accumulates the product to the value in the register of lf. This multiplier only performs $\times(+1)$ or $\times(-1)$, where $+1$ and -1 are represented by 1 and 0 in this Ising machine, respectively. Therefore, it inverts the sign bit of the half-precision floating point input when $\sigma_j^{old} = 0$, and does no operation to the input if $\sigma_i^{new} = 1$. Furthermore, the multiplier has a $mode$ input, the input from multiplexer will be output without any change when $mode = 0$, as the same as when the $\sigma_j^{old} = 1$. The spin states are initialized to "-1"s. Thus, the initial accumulation result of the local field value and external field value (with all spin states being "-1") is stored in the register. The products from the multiplier need to be multiplied by 2 as the energy is increased or decreased by $2 \cdot J_{ij} \cdot \sigma_j^{old}$ when σ_j^{old} is changed. Therefore, $2 \cdot J_{ij}$ is stored in the memory block.

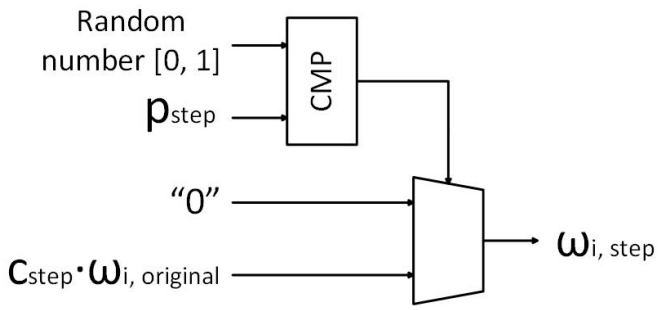


Figure 4.3: The self-interaction generating unit (SIGU).

4.2.3 A Self-Interaction Generating Unit (SIGU)

The p_{step} ($p_{step} = \max[0, 0.5 - \frac{step}{2step_{max}}]$) decreases from 0.5 to 0 and c_{step} ($c_{step} = \min[1, \sqrt{(\frac{step}{step_{max}})}]$ increases from 0 to 1, respectively, where $step$ is the number of the current step and $step_{max}$ is the number of total steps in the annealing process. They are used in the self-interaction generating unit for calculating self-interaction (ω_i), as shown in Fig. 4.3. It is a new design for generating self-interactions. The ω_i is multiplied with the value of c_{step}. Then the output of a SIGU can be zero at a probability p_{step}, or be $c_{step} \cdot \omega_{i,original}$ at a probability $(1 - p_{step})$. It consists of a comparator and a two-to-one multiplexer. The p_{step} value is calculated in the LAU, and decreases from 0.5 by $\frac{1}{(2step_{max})}$ every iteration. It is compared with a random number between 0 and 1. $\omega_{i,step}$ is 0 if $p_{step} > rand$, and $c_{step} \cdot \omega_{i,original}$ otherwise.

The circuit for implementing the function, $\sqrt{(\frac{step}{step_{max}})} \cdot \omega_{i,original}$, is hardware-consuming. To save the area of the circuit, the momentum scaling factor c_{step} is approximated by $\min[1, \frac{step}{step_{max}}]$. Thus, an approximate linear function, $\frac{step}{step_{max}}\omega_{i,original}$, is implemented. This value is also calculated in the LAU and increases from 0 by $\frac{\omega_{i,original}}{step_{max}}$ every iteration.

4.2.4 A Spin Update Unit (SUU)

Taking lf_i and $\omega_{i,step}$ as the inputs, the SUU determines the new state of the ith spin. As presented in Fig. 4.4, the SUU consists of a multiplier, an adder, a compara-

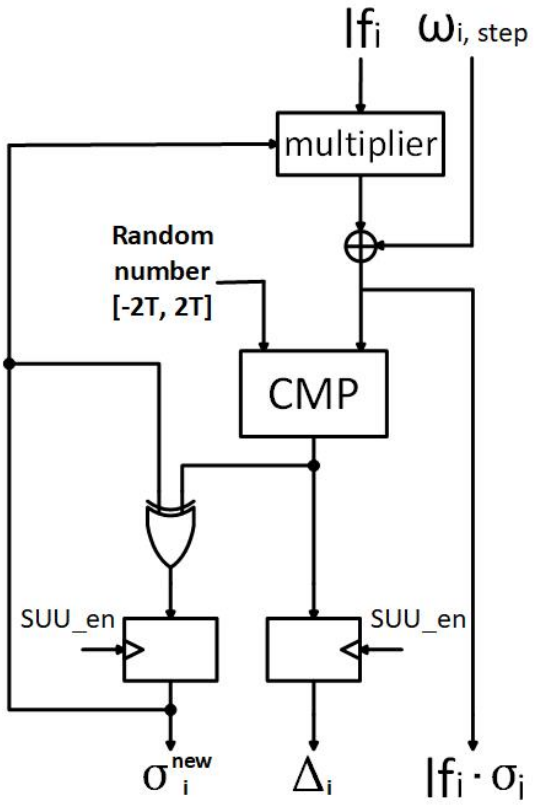


Figure 4.4: The spin update unit (SUU) [1].

tor, an exclusive-or (XOR) gate, and two registers [1]. First, lf_i is multiplied with σ_i^{new}. The multiplier inverts the sign bit of lf_i if $\sigma_i^{new} = 0$ and does no operation on lf_i if $\sigma_i^{new} = 1$. Then the product is added to $\omega_{i,s}$ to obtain the energy variation ($\Delta E_{i(hardware)}$, shown in (4.2)). The Gibbs criterion in (4.1) is used to calculate the spin-flip probability. It is equivalent to the sigmoid function, as shown in (4.3). To reduce the hardware cost, the sigmoid function is approximately implemented by using a linear function, as shown in (4.4) [1]. The spin-flip probability is 1 when ΔE_i is larger than $2T$, 0 when ΔE_i is smaller than $-2T$, or $\frac{\Delta E_i}{4T} + 0.5$ when ΔE_i is between $-2T$ and $2T$. Thus, ΔE_i is compared with a random number between $-2T$ and $2T$ by using a comparator. Its output, Δ_i, indicates whether the state of the ith spin is flipped (by 1) or not (by 0). Then, the new spin state is generated by XORing the

old spin state and Δ_i. Furthermore, the summation of the adder is output as $lf_i \cdot \sigma_i$ to the solution update unit for total energy calculation.

$$S(t) = \frac{1}{1 + exp(-t)}. \tag{4.3}$$

$$P = \begin{cases} \dfrac{\Delta E_i}{4T} + 0.5 & (-2T < \Delta E_i < 2T) \\ 1 & (\Delta E_i > 2T) \\ 0 & (\Delta E_i < -2T) \end{cases}, \tag{4.4}$$

4.2.5 A Random Number Generator (RNG)

Two kinds of random numbers are required following uniformly distributions from zero to one and $-2T$ to $2T$, respectively. It is inspired by [1] but implementing a half-precision floating-point representation. The random number generator (RNG) circuit is shown in Fig. 4.6. An XORshifter can generate a maximum length state sequence if the switches states are following the primitive polynomial. Fig. 4.5 shows a 4-bit XORshifter, and its primitive polynomial is $X^4 + X + 1$. The power of the variable X value indicates the corresponding switch is on. A 22-bit XORshifter is used to generate a 22-bit pseudo-random sequence in this design, whose primitive polynomial is $X^{22} + X + 1$. The 21th down to 12th bits and 11th down to 2st bits are used to generate two random numbers between $[1, 2)$, which subtract 1 to obtain random numbers between $[0, 1)$. The random number between $[-2T, 2T]$ are

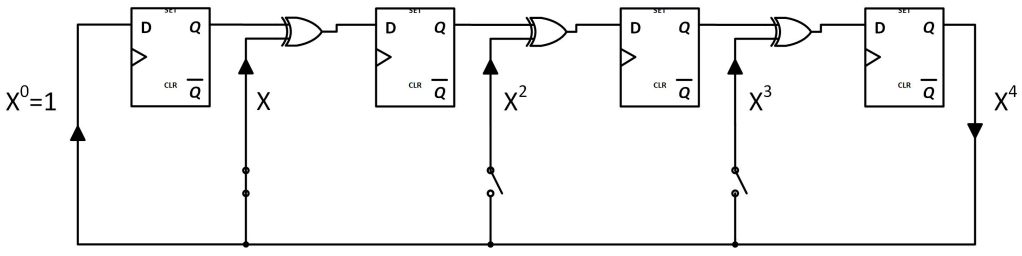


Figure 4.5: A 4-bit XORshifter.

generated by a similar method but using the 22nd bit and 1st bit as the sign, and then multiplied with the value of $2T$.

4.2.6 A Delta-Driven Simultaneous Spin Update (DDSS) Circuit

The DDSS circuit receives all the new spin states (σ_0^{new} to σ_{63}^{new}) and Δs (Δs indicates which spins are flipped) from the SUUs, and then outputs the state of flipped spin σ_j^{old} to LAUs and the corresponding index to the memory block [1]. It consists of 64 one-bit registers, a 64-to-1 multiplexer, a priority encoder, and a 6-to-64 decoder, as shown in Fig. 4.7 [1]. The Δs from SUUs are stored in registers and updated with the *ddss_en* signal. To identify the flipped spins and calculate new local field values, a DDSS circuit works as follows. The priority encoder outputs an index that indicates the position of the first "1" in Δs. Meanwhile, the multiplexer chooses the bit at the same position in *sigmas*, which indicates the state of the flipped spin σ_j^{old}. Then the decoder returns a 64-bit *rsts* signal to the 64 registers (each register receives one bit). It will reset the detected first "1" in registers to "0". When all "1" signals are reset to "0", the *empty* signal becomes 0, which means all states of flipped spins have been output to LAUs and the system is ready for the next spin update cycle.

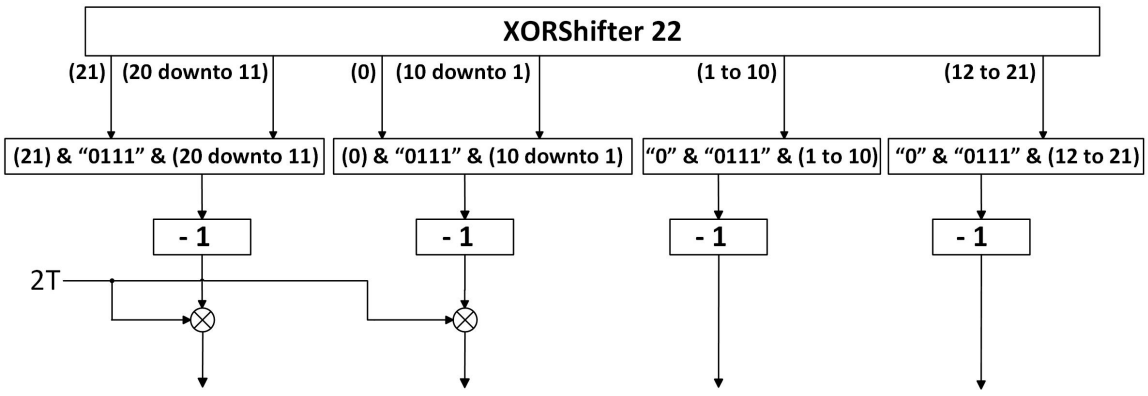


Figure 4.6: The random number generator (RNG) design.

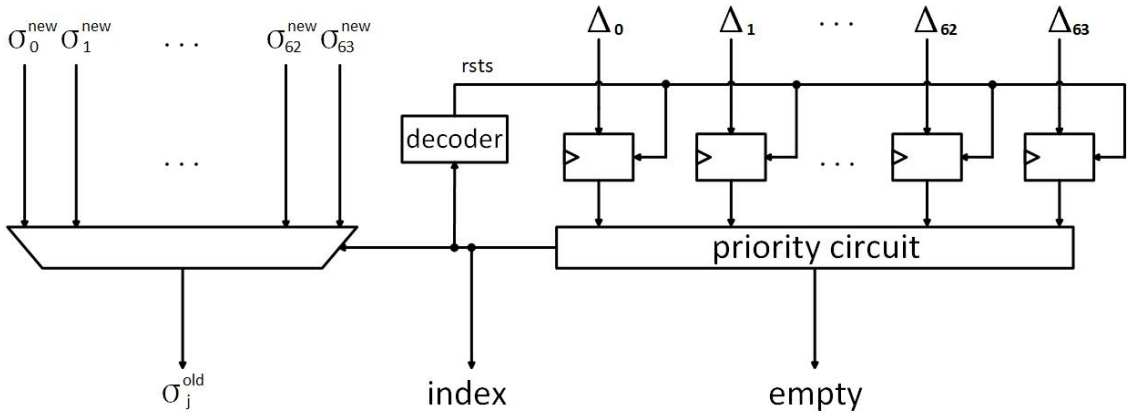


Figure 4.7: The delta-driven simultaneous spin update (DDSS) circuit [1].

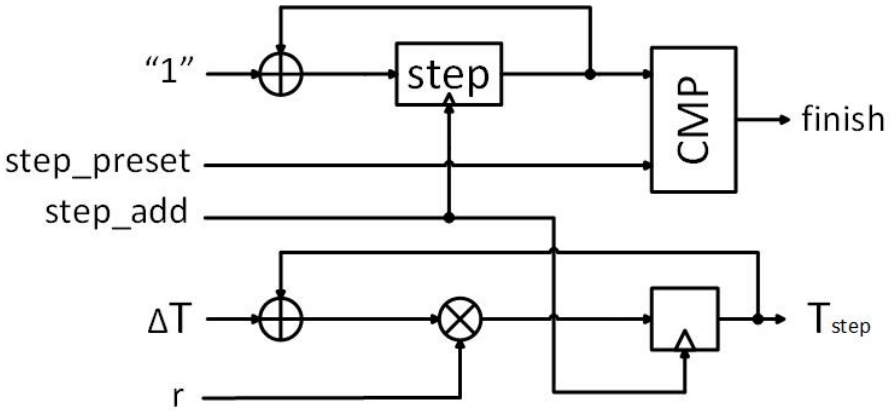


Figure 4.8: The annealing schedule unit (ASU).

4.2.7 An Annealing Schedule Unit (ASU)

The annealing schedule unit counts the annealing step and calculates the temperature. It is different from the design in [1] as the temperature in this work has a dynamic offset. It consists of two adders, two registers, one multiplier, and one comparator, as shown in Fig. 4.8. The annealing process stops when the *step* value is larger than the *step_preset* value, and the $finish$ signal becomes "1". The new T_{step+1} is calculated as:

$$T_{step+1} = (T_{step} + \Delta T) \times r, \tag{4.5}$$

where r is the decreasing rate of temperature ($r \leq 1$). The updates of step value and T_{step} are controlled by the *step_add* signal.

4.2.8 A Solution Update Unit

In order to select a solution with the lowest total energy among the found results, a solution update unit is newly designed. Its structure is shown in Fig. 4.9. This solution update unit consists of a candidate generator, an energy calculator, and an update unit.

Calculating the total energy of the Ising model at each iteration is inefficient. The total energy is calculated by accumulating the $lf_i \cdot \sigma_i$ values from SUUs. Thus, the accumulation cycle increases when the size of the Ising model increases. On the other hand, not all *energies* during the annealing need to be calculated because feasible solutions only show up at local minima or the ground state for TSPs. Therefore, the solution update unit only calculates the total energy when the system moves to *no* $flip$ state. The $noflip$ signal pulses every clock cycle if there is no change in the spins' configuration. An RS flip-flop and an AND gate are used to generate a candidate signal that only pulses one time when the spins' configuration stays with no change, as shown in Fig. 4.10.

The buffer in the candidate generator is for making the candidate hold for a delay

period. Suppose a new *candidate* signal pulse comes, but the accumulation of the last total energy is not finished: in that case, the Ising machine needs to stop annealing and wait for the accumulation process to finish. It decreases the speed of the Ising machine. Therefore, registers are used to store the data waiting for accumulation. Fig. 4.11 shows a structure of the energy calculator unit. For this Ising machine with 64 spins, 64 storage blocks are required, and each storage block has 16 16-bit registers. The Ising machine can store at most 16 candidates for accumulation, reducing the effect of energy calculation to the annealing speed. All registers are controlled by the *candidate* signal. In each storage block, a *read_address* signal determines which candidate is output to the accumulator, and a *write_address* signal determines which new candidate data are written into which registers. These two signals are 5-bit, and only the lower four bits are connected to the multiplexer and demultiplexer. The *write_address* and *read_address* are added by 1 at the rising edge of *candidate* and *accum_done*, respectively. The *accum_done* signal becomes 1 when $count = 64$, which indicates all the 64 values have been accumulated. The *count* signal is added by 1 at the rising edge of *clk_accum*, while the *clk_accum* is generated by ANDing the *clk* and *accum_en* signals. The *count* is reset to 0 after the *accum_done* signal becomes 1. An *annealing_en* signal changes from 1 to 0 when all data in the 16 registers have not been accumulated but a new *candidate* signal arrives. In this situation, the Ising

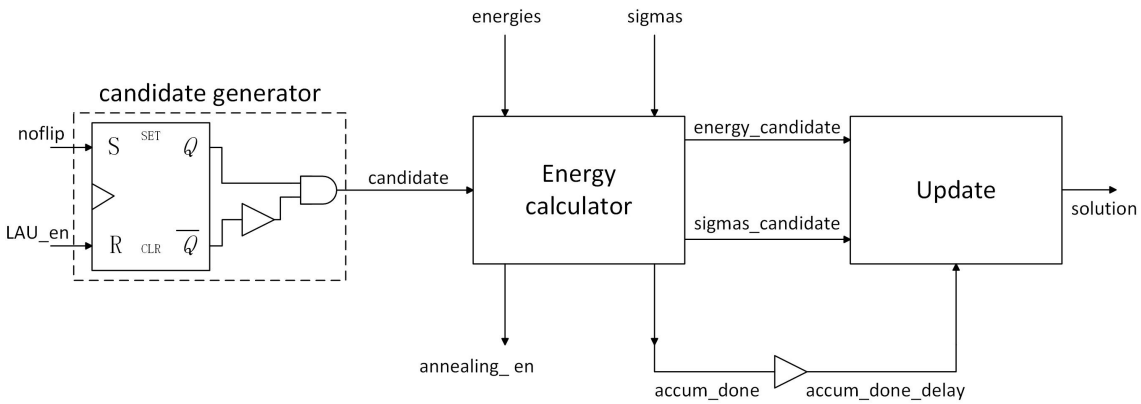


Figure 4.9: The solution update.

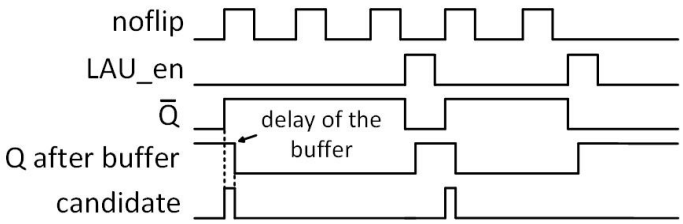


Figure 4.10: The waveform of the candidate signal.

machine needs to wait until storage space is released and then continues the annealing process. An *accum_en* signal changes from 1 to 0 when all data in the 16 registers have been accumulated but no new *candidate* signal arrives. The Ising machine needs to wait the next rising edge of *candidate* and then continues the accumulation process. A finite state machine (FSM) is used to realize the change of *annealing_en* and *accum_en*. The FSM has three states: state 1 ($annealing\_en = 1$, $accum\_en = 0$), state 1 ($annealing\_en = 1$, $accum\_en = 1$), and state 1 ($annealing\_en = 0$, $accum\_en = 1$). In state 1, FSM is waiting for *candidate* becomes 1, and then moves to state2. In state 2, annealing and accumulation continue working until $w\_addr - 15 = r\_addr$ (w_addr represents $write\_address$ and r_addr represents $read\_address$ in the figure of FSM) and *candidate* becomes 1. For example, when the $candidate = 1$, $w\_addr = 16(10000)$ and $r\_addr = 1(00001)$ (w_addr indicates the $register0$ and r_addr indicates the $register1$ in the storage block, as only the lower four bits are connected to the multiplexer and demultiplexer), the next input data will be written into the $register1$ ($w\_addr + 1$) but the accumulation of the old data in the $register1$ has not been finished. Therefore, FSM moves to state 3 and waits the accumulation to be finished. When an accumulation is finished, *accum_done* becomes 1, and the FSM moves back to state 2. Furthermore, when $w\_addr = r\_addr$, which means all data in registers have been accumulated, the FSM moves back to state 1 and waits *candidate* becomes 1. The initial state of this FSM is state 1. Another similar storage blocks array in the energy calculator unit is used for storing sigmas' states. The $write\_address$ and $read\_address$ are shared with those in storage blocks for

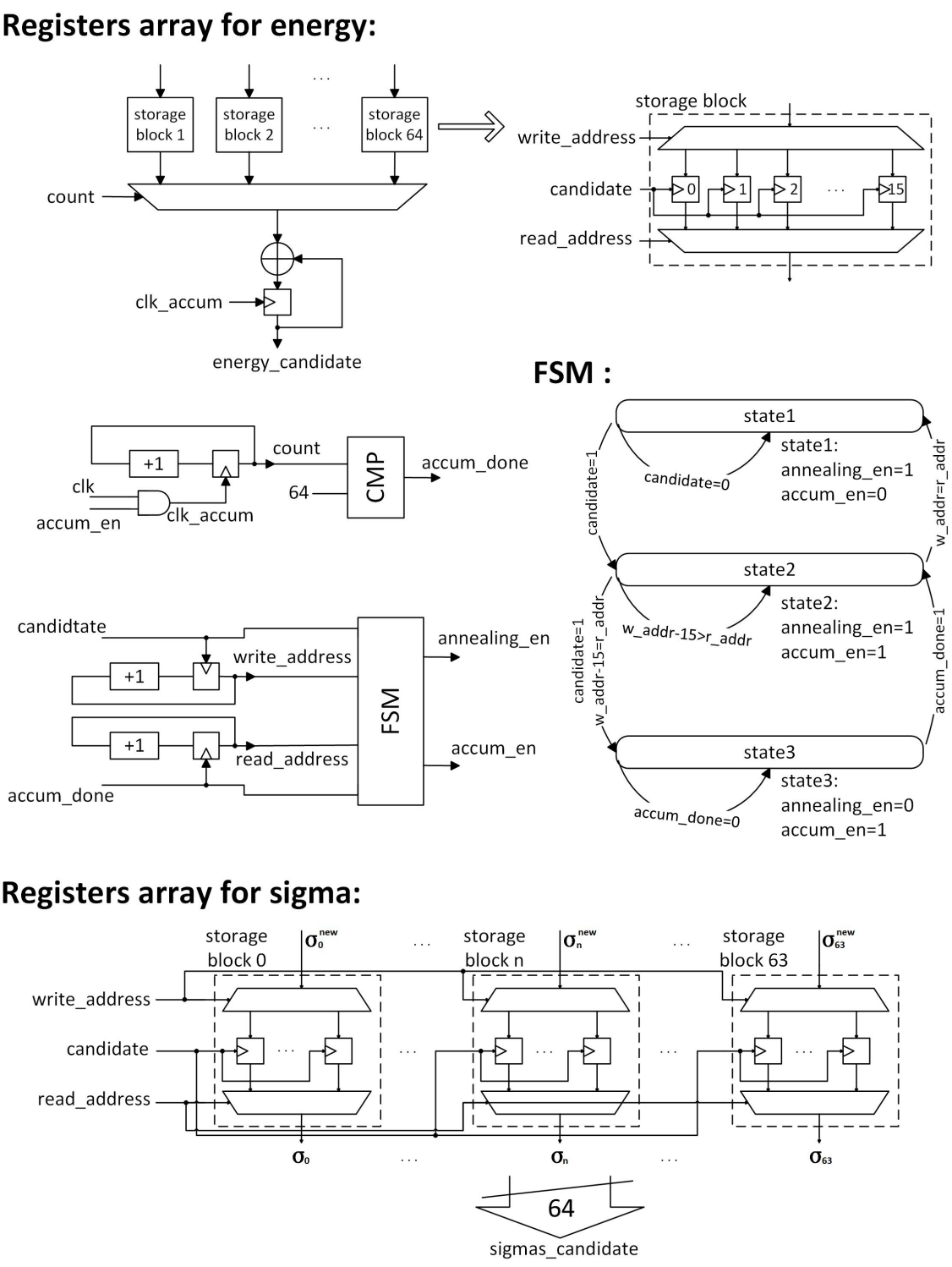


Figure 4.11: The energy calculator unit.

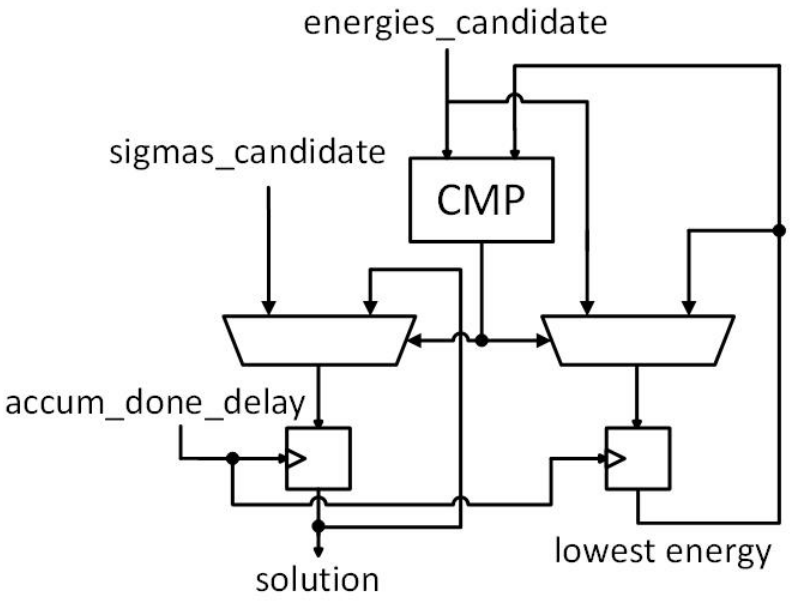


Figure 4.12: The update unit.

energies. In the update unit, the new summation of energy (*energy_candidate*) is compared with the lowest energy that the Ising machine found before, as shown in Fig. 4.12. If the new energy is lower, the lowest energy is updated by this *energy_candidate* and the solution is updated by the corresponding sigmas' states (*sigmas_candidate*). Otherwise, the lowest energy and the solution do not change. Furthermore, the registers are controlled by the *accum_done_delay* signal. The delay of a buffer is applied for ensuring the outputs of multiplexers and the comparator are stable before the sampling.

4.2.9 Control Unit

The control unit is a finite state machine. No details of this unit have been disclosed in the previously published papers. As shown in Fig. 4.13, it has fifteen states. All the delay states, i.e., *step adding delay*, *DDSS enable*0 *delay*, *LAU enable delay*1, *LAU enable delay*2, *DDSS enable*1 *delay*, and *no flip delay* are used for making the instruction signal hold for one clock cycle to ensure the accuracy of the sampled signals. All the state transitions are controlled by clock and corresponding signals.

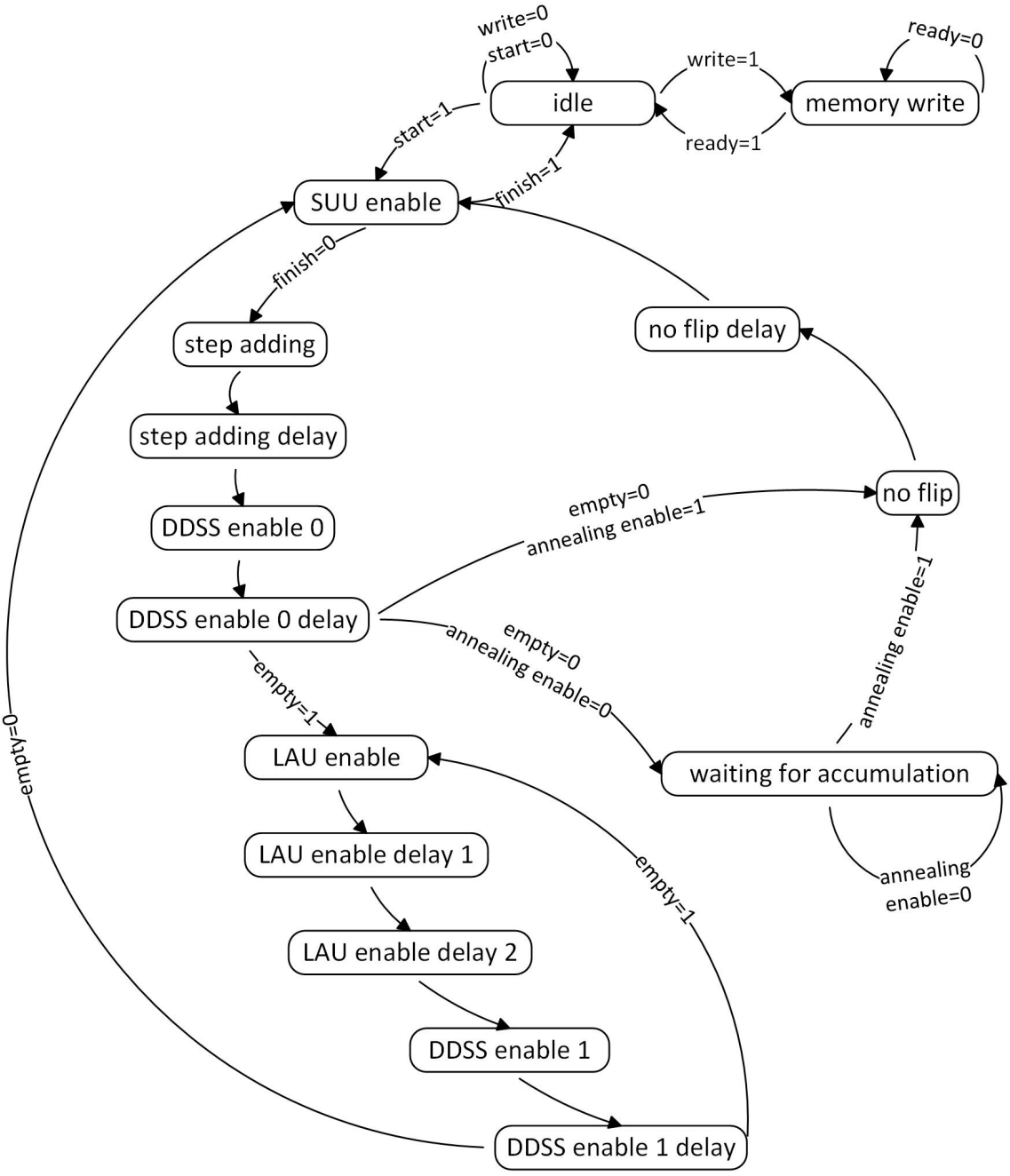


Figure 4.13: The state transition of control unit.

If there is no label on the arrow, it means the transition is only controlled by clock signal. The *idle* and *memory write* states are used for the system initialization. The circuit does not work at *idle* state, and data is written into the memory block at *memory write* state. The state returns to *idle* when *ready* is 1, which indicates all data have been written. The system starts annealing when *start* signal is 1 and stops annealing if $finish$ becomes 1. The *SUU enable*, *step adding*, *DDSS enable*0, *LAU enable*, *DDSS enable*1, and *no flip* are used for coordinating the annealing process as follows. *SUU enable* is the start point (also the end point) of the annealing; SUUs are working at this state and generate new spins states. Then the system adds an annealing step by one at *step adding*, if $finish = 0$. After adding the step, the system goes through *DDSS enable*0, *LAU enable*, and *DDSS enable*1. The DDSS outputs index and spin state at *DDSS enable*0 and *DDSS enable*1 states. The *DDSS enable*0 and *DDSS enable*1 are for distinguishing whether still no spin is flipped ($empty = 0$) after the spin update. If $empty = 0$, state moves to *no flip* and then back to *SUU enable* for updating the spins again. The dynamic offset increases at *no flip* state. Otherwise, the state moves to *LAU enable* and performs the accumulation of lf_i in LAUs. Meanwhile, if *annealing enable* = 0, which means there is no space in the solution update unit, the system needs to stop annealing and waiting for the accumulation of total energy.

The output of the control unit is a twelve-bit *instruction* signal, as shown in Fig. 4.14. The 11th bit is *mode* that controls the calculation model of LAU; the 10th bit is $rst_{dynamic}$ that resets the value of dynamic offset to zero when a spin is flipped; the 9th bit is $noflip$; the 8th bit is C_s *update* that controls the adding of C_s; the 7th and 6th bits are $se1$ and $se0$ that used in LAU; the 5th and 4th bits are *write*

11	10	9	8	7	6	5	4	3	2	1	0
mode	rstdynamic	noflip	C_update	se1	se0	write	read	DDSS_en	LAU_en	SUU_en	step_add

Figure 4.14: The instruction generated by the control unit.

Table 4.1: Instructions and the corresponding states

state	*Instruction*
idle	000000000000
memory write	000000100000
SUU enable	000000000010
step adding	000001000000
step adding delay	000001000000
*DDSS enable*0	000010001000
*DDSS enable*0 *delay*	000110001000
LAU enable	110000010000
*LAU enable delay*1	110000010000
*LAU enable delay*2	110000010100
*DDSS enable*1	000000001000
*DDSS enable*1 *delay*	000000001000
waiting accumulation	000000000000
no flip	000011000000
no flip delay	001011000000

and *read* signals for memory block; the 3rd, 2nd, 1st, and 0th bits are $DDSS\_en$, LAU_en, SUU_en and *step_add* signals that control the update of registers in DDSS, LAU, SUU and ASU, respectively. Table 4.1 shows the instruction signals of different states.

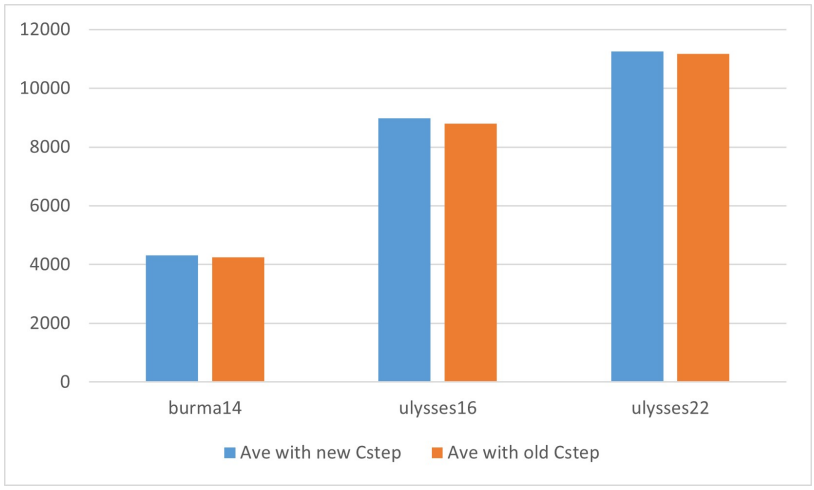
4.3 Results and Discussion

4.3.1 Approximation of the Momentum Scaling Factor

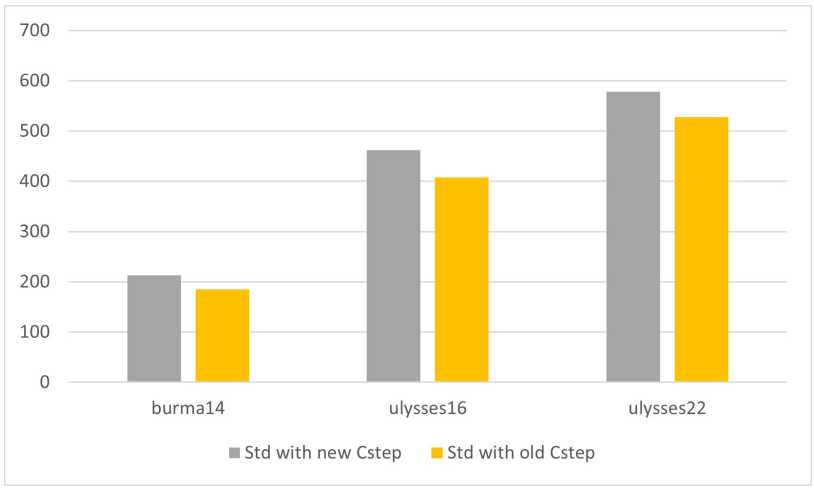
The momentum scaling factor is $c_{step} = \sqrt{\frac{step}{step_{max}}}$ in the algorithm simulation, where *step* is the current step number and $step_{max}$ is the step number at the end of annealing. Because the self-interaction (ω_i) is different for every spin, calculating $c_{step} \cdot \omega_i$ requires a multiplier, and the number of multipliers depends on the number of spins. It incurs a high cost when the number of spins in the system is large. However, with a linear c_{step}, the adders in LAUs can be reused to calculate $c_{step} \cdot \omega_i$ because the increment, $(c_{step} \cdot \omega_i - c_{step-1} \cdot \omega_i)$, is constant. Therefore, a linear momentum scaling factor $c_{step} = \frac{step}{step_{max}}$ is applied in the circuit implementation for hardware efficiency. Fig. 4.15 shows the effect after applying a linear momentum scaling factor. The increase on average is 1.6% for *burma*14, 1.9% for *ulysses*16, and 0.7% for *ulysses*22, while the increase in the standard deviation is 15.2% for *burma*14, 13.3% for *ulysses*16, and 9.7% for *ulysses*22.

4.3.2 Approximation of Floating-point Adders

A half-precision floating-point is implemented in this design. As shown in Fig. 4.16, it contains a sign bit, 5 bits for the exponent, and 10 bits for the mantissa. The architecture of a floating-point adder consists of circuits for the larger-exponent detection, mantissa alignment, mantissa summation, normalization, and rounding, as shown in Fig. 4.17 [53]. First, two operands are unpacked, and the hidden 1 before the decimal point is added to each mantissa. The mantissa with a smaller exponent is right-shifted for alignment. Then two mantissas are added by using an adder. After



(a)



(b)

Figure 4.15: The (a) average and (b) standard deviation of travel distance after applying a linear momentum scaling factor (new c_{step}). (the old c_{step} is the conventional momentum scaling factor)

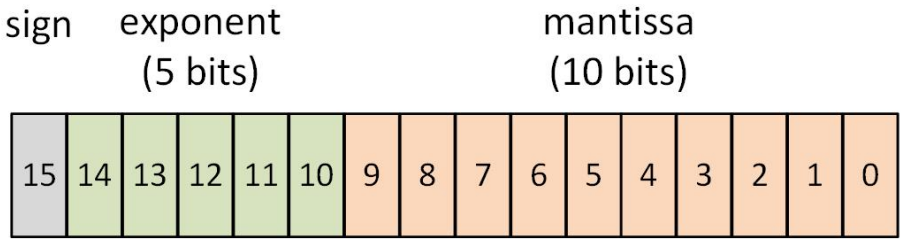


Figure 4.16: The IEEE 754 half-precision floating point format.

the normalization and rounding of the summation, the result is ready for the IEEE standard representation. The mantissa adder requires 12 bits in the accurate design. In our Ising machine circuit design, the mantissa adders in the floating-point adders in the LAUs are replaced by TruAs and LOAs.

The truncated adders (TruA) and lower-part-OR adders (LOA) are considered to improve the hardware efficiency of the circuit design. The mean relative error distance (MRED), error rate (ER), normalized mean error (NME), and normalized mean error distance (NMED) of truncated adders and lower-part-OR adders are shown in Table 4.2 and Table 4.3, respectively. The k indicates the least significant bits (LSBs) truncated in TruA and the length of the lower part in LOA. The error, error distance (ED), and the relative error distance (RED) are calculated as [44]:

$$error = R' - R, \tag{4.6}$$

$$ED = |R' - R|, \tag{4.7}$$

$$RED = \frac{|R' - R|}{R}, \tag{4.8}$$

where R' is the approximate result, and R is the accurate result. The normalized error (NE) and normalized error distance (NED) are normalizations of error and error distance by the maximum output of the half-precision floating-point adder (65504 for 16-bit). The MRED, NME, and NMED are the average values of RED, NE, and NED, respectively.

Table 4.2: The error characteristics of k-LSB truncated adders

k	MRED (10^{-3})	ER (%)	NME (10^{-3})	NMED (10^{-3})
3	3.1	96.93	-2.9	2.9
4	6.7	99.21	-6.2	6.2
5	13.9	99.81	-12.9	12.9
6	28.3	99.96	-26.3	26.3

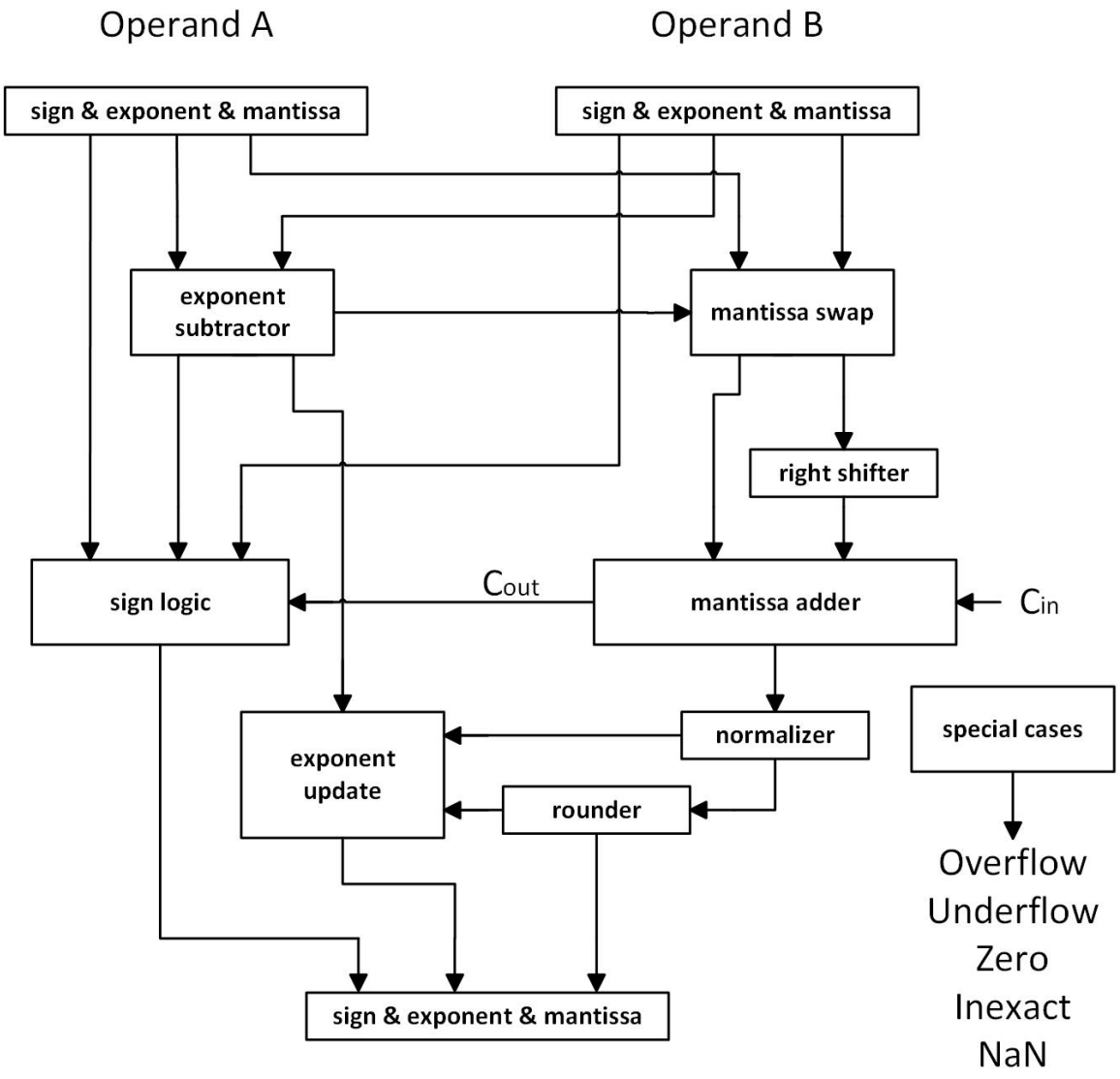


Figure 4.17: The architecture of a floating-point adder [53].

Table 4.3: The error characteristics of k-bit approximated lower-part-OR adders

k	MRED (10^{-3})	ER (%)	NME (10^{-6})	NMED (10^{-3})
4	1.4	68.14	7.68	1.3
5	2.7	76.35	7.14	2.5
6	5.4	82.33	6.83	5.0
7	10.7	86.50	8.92	10.0
8	21.1	89.95	8.68	20.1

The NMED and MRED of $LOA-8$ are close to those of $TruA-6$. The number after the name of an approximate adder is the number of truncated LSBs in a TruA and the number of approximated LSBs (using OR gates) in an LOA. However, the ER of $LOA-8$ is much lower than that of $TruA-6$ because truncation leads to a loss in precision. Furthermore, the NME of $LOA-8$ is close to zero as the error of an LOA can be positive or negative, whereas the error of a $TruA$ can only be negative. The error introduced by the approximated bits is compensated by itself and averaged to zero during the accumulation. Therefore, LOA is a better approximation scheme for accumulators than $TruA$ considering accuracy albeit with increased circuit area.

The error in a half-precision floating-point adder can be significant when the exponent is large or marginal when the exponent is small. Therefore, NMED and NME are unreliable metrics in a performance judgement of an approximate floating-point adder, and the relative error distance (RED) is first considered in the experiment. By observation, the distribution of REDs resembles a gamma distribution. Thus, we used gamma distributions to approximate the distribution of RED. The probability density of the gamma distribution is given by [54]:

$$f(y|\alpha,\beta)=\frac{\beta^{\alpha}}{\Gamma(\alpha)}y^{\alpha-1}e^{-\beta y}, \tag{4.9}$$

where $\Gamma(.)$ is the gamma function, α is the shape parameter, and β is the inverse scale parameter. The α and β can be calculated by using the expected value (Exp) and standard deviation (D) of the gamma distribution. The relationships between α, β, Exp and D are as follows:

$$Exp=\frac{\alpha}{\beta}, \tag{4.10}$$

$$D=\frac{\alpha}{\beta^2}. \tag{4.11}$$

The random variable that follow a gamma distribution ($rand\_gamma$) is applied to the accumulation process in the IPA machine to test the system's fault tolerance. In particular, when calculating $a+b$, the result after applying the $rand\_gamma$ is $(a+b)\times(1-rand\_gamma)$ for simulating RED of a $TruA$ (the error is always

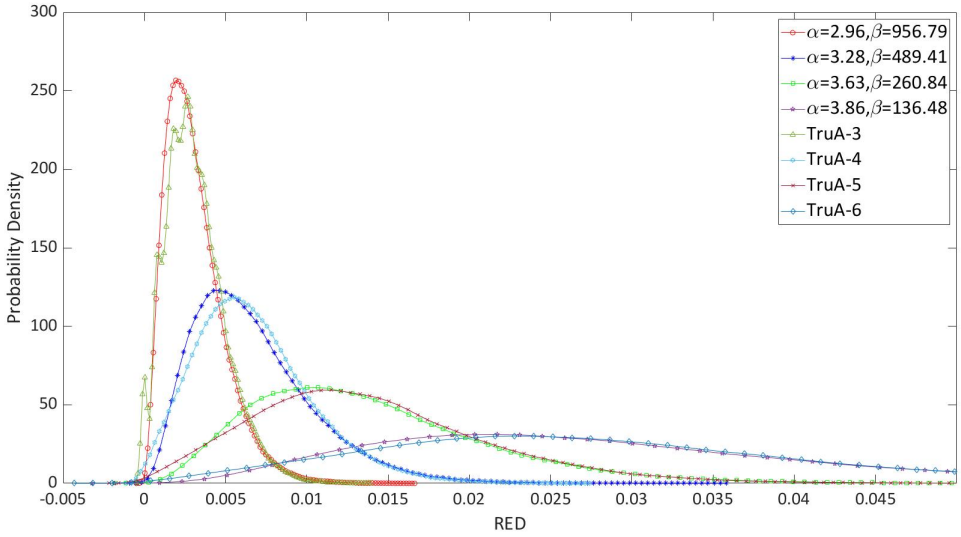


Figure 4.18: Fitting the probability density of TruAs' errors using gamma distributions.

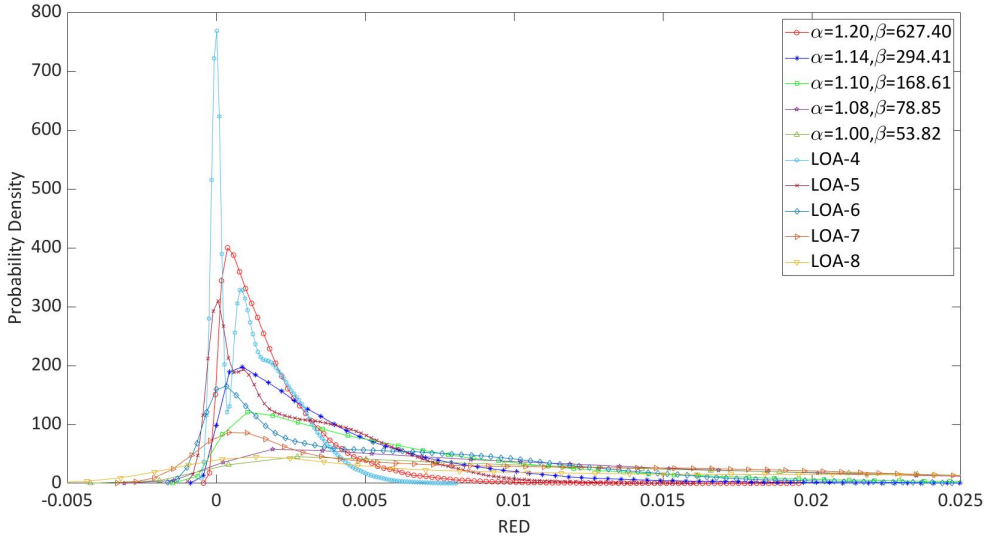


Figure 4.19: Fitting the probability density of LOAs' errors using gamma distributions.

negative for a $TruA$); and is $(a+b) \times (1 \pm rand\_gamma)$ for simulating RED of an LOA (the error can be positive or negative with equivalent probability for an LOA). As shown in Fig. 4.18, the RED of $TruA-3$, $TruA-4$, $TruA-5$, and $TruA-6$, are approximated by gamma distribution with $\alpha = 2.96$, $\beta = 956.79$; $\alpha = 3.28$, $\beta = 489.41$; $\alpha = 3.63$, $\beta = 260.84$; and $\alpha = 3.86$, $\beta = 136.48$; respectively. For LOAs, as shown in Fig. 4.19, the RED of $LOA-4$, $LOA-5$, $LOA-6$, $LOA-7$, and $LOA-8$, are approximated by gamma distribution with $\alpha = 1.20$, $\beta = 627.40$; $\alpha = 1.14$, $\beta = 294.41$; $\alpha = 1.10$, $\beta = 168.61$; $\alpha = 1.08$, $\beta = 78.85$; and $\alpha = 1.00$, $\beta = 53.82$; respectively.

With the decrease in ER, the probability of having a zero in RED is increased. Thus, the probability distribution of RED is changed and a pinnacle shows up at 0 in the RED's curve. The gamma distribution do not fit the actual RED distributions well in this scenario. For simplicity, however, gamma distributions are used for an error tolerance test. All fitted gamma distributions take higher values than the actual RED distributions when RED is larger than 0. Therefore, $rand\_gamma$ has a higher probability to take a value large than RED and cause more errors than an approximate adder.

Table 4.4 shows the average (Ave), maximum (Max), minimum (Min), and standard deviation (Std) of the travel distances obtained by the IPA machine after applying the random variables that simulated the error due to the approximate adders. The violation rate (VR) indicates the probability of getting a result that does not conform to the constraint. As our model only implements 64 spins, an 8-city TSP is used as the benchmark. Furthermore, the distances between each city are scaled to $[0, 1]$ to ensure the solution update unit works well. The VR increases significantly when the number of truncated bits or approximated bits is larger than 4 for TruAs and larger than 6 for LOAs. Therefore, the Ising machine can tolerate errors with an MRED around 0.0067. $TruA-3$, $TruA-4$, $LOA-4$, $LOA-6$, and $LOA-6$, are considered and implemented in the circuit design for the hardware simulation.

Table 4.5 shows the average travel distance (Ave), maximum travel distance (Max), minimum travel distance (Min), standard deviation of the distance (Std), and VR of solutions found by the IPA machine after applying LOAs or TruAs.

In the actual implementations, both $TruA-3$ and $TruA-4$ achieve 0% VR, and the Aves are smaller than the results obtained in the fault tolerance test. This occurs because the difference between the augend and addend is not large due to the small scale of the problem, but a significant relative error distance shows up when the difference between the augend and addend is extremely large. Therefore, a lower VR is obtained in the actual implementations. The same is observed in the actual implementations of $LOA-4$, $LOA-5$, and $LOA-6$. Both VR and Ave decreases in actual implementations, but the decreasing rate is smaller than after using a $TruA$, because the RED of a LOA is much smaller than that of a $TruA$, which leads to smaller gaps between the results from fault tolerance tests and actual implementations. The solution quality obtained by using $TruA-3$, $TruA-4$, $LOA-4$, and $LOA-5$, are similar. Using any of them can find the shortest travel distance (2.39) while keeping a low Std, which means the quality of solutions is stable. However, the Ising machine can not find the shortest travel distance after using $LOA-6$ and the Ave becomes much larger.

The circuit area ($Area$), power dissipation ($Power$), and $Delay$ of the IPA machine after implementing a TruA or a LOA are shown in Table 4.6. Simulated results are obtained by synthesizing circuits using the Synopsys Design Compiler. The CMOS 28 nm technology is applied with asupply voltage of 1.0 V, temperature of 25 $°C$, and clock frequency of 200 MHz. Using approximate adders in Ising machines decreases the hardware cost. Implementing $LOA-6$ results in the smallest circuit area (6.259 mm^2) and the shortest delay (3.79 ns). However, it shows a 4% probability of getting a solution that violates the constrains of TSPs and leads to a much higher average traveling distance. The circuits by using $LOA-5$ and $TruA-4$ result in similar average travel distance, VR, area, and delay time. However, the power consumption

Table 4.4: The travel distances after applying randomly generated numbers that follow gamma distributions (for TruAs and LOAs) as noise.

	Ave	Max	Min	Std	VR
$Gamma1(TruA-3)$	2.87	3.54	2.39	0.25	0%
$Gamma2(TruA-4)$	3.07	4.63	2.39	0.30	7%
$Gamma3(TruA-5)$	4.42	6.67	2.39	0.97	72%
$Gamma4(TruA-6)$	6.92	11.05	3.37	1.64	98%
$Gamma5(LOA-4)$	2.82	3.18	2.39	0.22	0%
$Gamma6(LOA-5)$	2.91	3.91	2.39	0.31	2%
$Gamma7(LOA-6)$	3.16	4.33	2.39	0.42	8%
$Gamma8(LOA-7)$	4.29	8.10	2.51	1.14	63%
$Gamma9(LOA-8)$	6.59	12.00	2.90	1.71	96%

Table 4.5: The travel distances after using TruAs and LOAs.

	Ave	Max	Min	Std	VR
$No\ Approximate$	2.67	3.08	2.39	0.17	0%
$TruA-3$	2.69	3.14	2.39	0.18	0%
$TruA-4$	2.74	3.58	2.39	0.24	0%
$LOA-4$	2.71	3.06	2.39	0.18	0%
$LOA-5$	2.77	3.17	2.39	0.21	0%
$LOA-6$	3.08	3.82	2.65	0.26	4%

of using $TruA-4$ is slightly lower and it is the most power-efficient scheme among the five approximate adders.

In order to gain more improvement in hardware efficiency, we propose an approximate scheme that combines both $TruA$ and LOA, thus referred to as a lower-part-OR truncated adder ($LOTA$). It truncates the k LSBs of an l-bit LOA. Thus, OR gates process $(l-k)$ bits, and k can not be larger than $(l-1)$. Table 4.7 shows the error characteristics of this adder. The NME of a $LOTA$ is mainly determined by its truncated bits. However, the ER of a $LOTA$ is related to the number of approximated bits. The ER of $LOTA-5\&3$ is close to that of $LOTA-5\&4$, and the ER of $LOTA-6\&3$ is close to that of $LOTA-6\&4$. ($LOTA-l\&k$ means a LOTA with $l-bit$ approximation and $k-bit$ truncation.) Furthermore, when the number of approximated bits is only one more than the number of truncated bits, the NMED and MRED of a $LOTA$ are between that of the LOAs with the same approximated bit and TruAs with the same truncated bit. For example, the NMED and MRED of $LOTA-4\&3$ and $LOTA-5\&4$ are lower than that of $TruA-3$ and $TruA-4$ but higher than that of $LOA-4$ and $LOA-5$. The REDs of $LOTA-4\&3$, $LOTA-5\&3$, $LOTA-5\&4$, $LOTA-6\&3$, and $LOTA-6\&4$ are fitted by gamma distributions with $\alpha=2.17$, $\beta=712.62$; $\alpha=2.63$, $\beta=622.54$; $\alpha=2.18$, $\beta=310.75$; $\alpha=2.15$, $\beta=325.93$; and $\alpha=2.62$, $\beta=324.00$; respectively. As shown in Table 4.8, the VR

Table 4.6: The hardware overhead after applying LOAs or TruAs

	$Area(mm^2)$	$Power(mW)$	$Delay(ns)$
No Approximate	6.300	41.289	3.97
$TruA-3$	6.275	41.035	3.90
$TruA-4$	6.266	40.931	3.89
$LOA-4$	6.273	41.263	3.84
$LOA-5$	6.266	41.223	3.82
$LOA-6$	6.259	41.080	3.79

Table 4.7: The error characteristics of approximate adders that combine LOA and TruA (LOTAs)

l	4	5		6	
k	3	3	4	3	4
NMED (10^{-3})	2.5	3.6	5.3	6.1	7.5
MRED (10^{-3})	2.8	3.9	5.7	6.6	8.1
ER (%)	93.40	97.06	97.16	98.42	98.88
NME (10^{-3})	-2.1	-2.1	-4.6	-2.1	-4.7

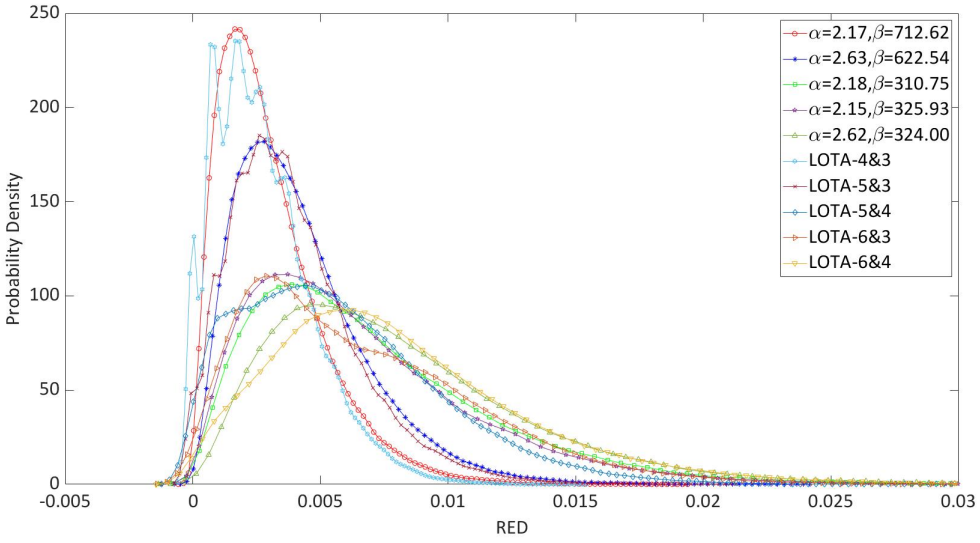


Figure 4.20: Fitting the probability density of LOTAs' errors using gamma distributions.

of solutions becomes higher than 10%, while both the Ave and Std increase when applying gamma4, gamma5, and gamma6 to the Ising machine as noise. Therefore, $LOTA-4\&3$ and $LOTA-5\&3$ are chosen and implemented for hardware simulations. Table 4.9 shows the average travel distance (Ave), maximum travel distance (Max), minimum travel distance (Min), standard deviation of the distance (Std), and VR of solutions found by the IPA machine after applying LOAs or TruAs. Table 4.10 show the Ising machines' hardware overhead after implementing LOTAs. Both $LOTA-4\&3$ and $LOTA-5\&3$ can achieve a 0% VR and find the shortest travel distance. The delays by using $LOTA-4\&3$, and $LOTA-5\&3$ are the same as those using $LOA-4$, and $LOA-5$, respectively, because the delay is determined by the number of approximate bits. The circuit area by using the $LOTA$ is lower than using the LOA and $TruA$ with the same LOA or $TruA$ bits. While the power consumption after using $LOTA$ is between those of using the LOA and $TruA$ with the same LOA or $TruA$ bits (higher than $TruA$ but lower than LOA). Furthermore, the circuit area, delay, and power dissipation of using $LOTA-5\&3$ are slightly lower than using $LOTA-4\&3$, while they can find a similar average travel distance. Compared with TruAs and LOAs, the average travel distance obtained via $LOTA-5\&3$ is similar to $TruA-4$ and $LOA-4$ but the circuit area and delay are reduced. Furthermore, the power dissipation after using $LOTA-5\&3$ is close to that after using $TruA-4$, while $TruA-4$ results in lowest power dissipation. Therefore, $LOTA-5\&3$ is chosen and implemented in the IPA machine for hardware efficiency.

Compared with the circuit without approximate adders, the circuit area is reduced by 0.6%, the power consumption is reduced by 0.4%, and the delay time is reduced by 3.8%. The decrease in circuit area and power consumption is not significant, because the area of LAUs is only 11% of the total circuit area. As a trade-off, the average travel distance is increased by 1.9%. Therefore, the circuit area of the 64-spin IPA-based Ising machine is 6.262 mm^2, power consumption is 41.108 mW, and delay is 3.82 ns.

Table 4.8: The travel distances after applying randomly generated numbers that follow gamma distributions (for LOTAs) as noise.

	Ave	Max	Min	Std	VR
$Gamma10(LOTA-4\&3)$	2.83	3.50	2.39	0.23	0%
$Gamma11(LOTA-5\&3)$	2.90	3.99	2.39	0.29	2%
$Gamma12(LOTA-5\&4)$	3.11	4.60	2.39	0.44	11%
$Gamma13(LOTA-6\&3)$	3.18	5.47	2.39	0.59	15%
$Gamma14(LOTA-6\&4)$	3.21	4.51	2.39	0.50	17%

Table 4.9: The travel distances obtained by using LOTAs.

	Ave	Max	Min	Std	VR
$LOTA-4\&3$	2.71	3.17	2.39	0.20	0%
$LOTA-5\&3$	2.72	3.06	2.39	0.21	0%

Table 4.10: The hardware overhead due to LOTAs

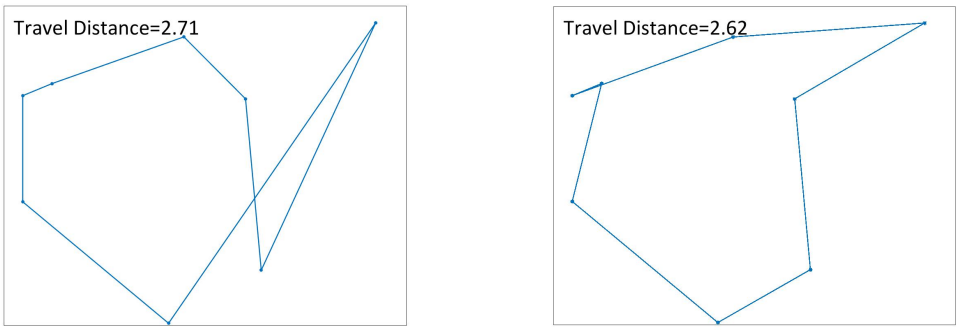
	$Area(mm^2)$	$Power(mW)$	$Delay(ns)$
$LOTA-4\&3$	6.269	41.130	3.84
$LOTA-5\&3$	6.262	41.108	3.82

4.3.3 Circuit Evaluation

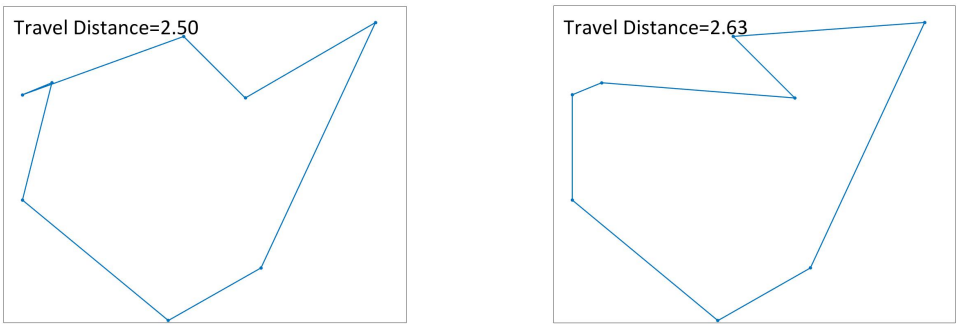
The circuit of an IPA machine with 64 spins was implemented by VHDL in Vivado. Validating the function of the circuit requires a test benchmark. However, the values in memory blocks must be preset for the IPA machine. Manually writing the testbench code is inefficient. Therefore, we used MATLAB to generate a testbench code in a text file and then copied and pasted it to Vivado. We also used the same 8-city TSP as before, and the distances between cities were scaled to $[0, 1]$. The parameter settings are: $T_{init} = 400$, $r = 0.97$, and $step_{max} = 1000$. The output of the circuit is a 64-bit binary sequence indicating the spin configuration. After interpreting the binary sequence to the visiting order of cities, we obtained the travel route. Fig. 4.21 presents some travel routes obtained by using the IPA machine. The route of 2.39 (unitless) long is the shortest distance and optimal solution of this 8-city TSP.

4.4 Conclusions

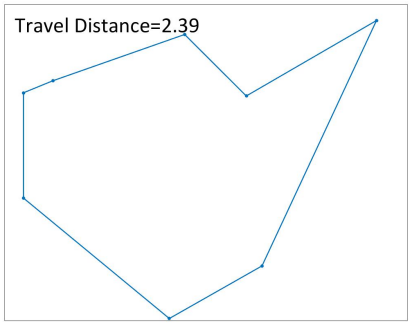
In this chapter, a circuit design is proposed to realize the function of the improved parallel annealing algorithm. Although the architecture is inspired by the structure of STATICA, many improvements were introduced for solving TSPs. First, this system implements a half-precision floating-point representation with a precision sufficient for solving large-scale TSPs. Second, reusing accumulators in the LAUs and approximating the momentum scaling factor to a linear increment save hardware costs. Using a linear momentum scaling factor only results in a marginal reduction in solution quality. Third, the solution update unit improves the quality of the solutions, and it is essential to introduce a dynamic offset. Moreover, a $LOTA - 5\&3$ is applied in the design for higher hardware efficiency. It decreases the area, power, and delay of the circuit while only incurring a 1.9% increase in the average travel distance. Although the decrease in circuit area and power dissipation is not significant, the reduction in delay improves the speed of the design.



(a) (b)



(c) (d)



(e)

Figure 4.21: Travel routes of an 8-city TSP obtained using the improved parallel annealing machine.

Chapter 5

Conclusions & Future Work

5.1 Conclusions

This dissertation begins with reviewing and classifying CMOS-based Ising machines and the underlying annealing algorithms. Then an improved parallel annealing algorithm (IPA) is proposed for solving TSPs. The proposed IPA performs better than MA and DA in solving unconstrained and constrained combinatorial optimization problems, such as TSPs and max-cut problems. Finally, circuits are designed for a high-performance and area-efficient implementation of the IPA.

In Chapter 2, simulated-annealing-based Ising machines are reviewed, including CMOS annealing machines, digital annealing (DA) machines, momentum annealing (MA) machines, and stochastic cellular annealing (SCA) machines. Furthermore, MA and SCA are classified as parallel annealing (PA) because they simultaneously update all spins. CMOS annealing-based Ising machines can achieve high-speed multi-chip communication and great scalability that benefit from the sparsely-connected Ising topology. However, solving complicated combinatorial optimization problems via CMOS annealing machines requires an embedding process. DA-based Ising machines implement a fully-connected Ising topology and are thus adaptive for solving combinatorial optimization problems without embedding. Moreover, DA requires a dynamic offset for solution quality improvement. However, only one spin's state is updated per iteration, and thus the speed of annealing is the performance bottleneck of DA-

based Ising machines. PA-based Ising machines overcome the speed limitation while implementing the fully-connected Ising topology. However, PA is inefficient in solving constrained combinatorial optimization problems such as TSPs. Moreover, the background of Ising models, combinatorial optimization problems, clustering approaches, and approximate arithmetic are also reviewed in this chapter.

An IPA for solving TSPs is proposed in Chapter 3. It applies an exponential temperature function with a dynamic offset and prevents the Ising energy from getting stuck in local minima. Therefore, it is more efficient than a conventional PA for solving TSPs. Furthermore, the k-medoids clustering significantly reduces the average travel distance. In addition to improving the solution quality, the IPA improves the speed of annealing in solving a 14-city TSP compared with other annealing algorithms. Moreover, larger cut values are obtained by using the IPA in max-cut problems than using MA and DA. It is promising in solving both constrained and unconstrained combinatorial optimization problems.

In Chapter 4, the circuit design of IPA is proposed. A half-precision floating-point representation is implemented in this design. Floating-point adders in the local field accumulator units (LAUs) are reused, and a momentum scaling factor is approximated to a linear increment for improving the hardware efficiency. A solution update unit is used to output the solution with the lowest Ising energy found during the annealing process. Furthermore, the circuit delay is shortened by using lower-part-OR truncated adders (LOTAs) in the LAUs. Meanwhile, the circuit area and power dissipation are also saved. As a trade-off, the solution quality suffers a marginal deterioration. However, a drawback is that the area and power dissipation of this circuit are larger than other designs due to the implementation of a half-precision floating-point representation. The IPA machine is expected to solve more complicated problems and improve the solution quality.

5.2 Future Work

Although the proposed IPA machine performs well when solving 8-city TSPs, it has limitations in scalability. First, the IPA machine does not have input and output (IO) ports for communication between chips. Therefore, it can not be implemented in a multi-chip system for constructing a larger-scale Ising machine. Second, the design's random number generators require multipliers to calculate $T_{step} \times rand$ (T_{step} is the temperature at $step$th iteration and $rand$ is a random number between $[0, 1)$), resulting in a relatively large circuit area and power consumption. Furthermore, the hardware costs increase with the size of the IPA machine as each spin needs a random number generator. Therefore, the scalability of the IPA machine will be investigated in our future work. The IPA Ising chip will be enhanced with a speedy IO port and be embedded into a multi-chip system like the CMOS annealing machines. However, a challenge is that the bandwidth of the IO port and the number of required memory blocks will be significant because this design implements a fully-connected topology. Moreover, a lightweight random number generator will be developed. The objective is to generate $T_{step} \times rand$ without using multipliers, where the $rand$ is a random number between 0 to 1. Additionally, the clustering approach is not integrated into the circuit as solving 8-city TSPs does not require it. In future work, a unit for preprocessing data via a clustering approach will be added to the system. It will reuse the accumulators in the LAUs, thus limiting the hardware overhead.

Bibliography

[1] K. Yamamoto *et al.*, "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*, 2020.

[2] K. Someya, R. Ono, and T. Kawahara, "Novel Ising model using dimension-control for high-speed solver for Ising machines," *2016 14th IEEE International New Circuits and Systems Conference (NEWCAS)*, 2016.

[3] T. Zhang, Q. Tao, B. Liu, and J. Han, "A review of simulation algorithms of classical Ising machines for combinatorial optimization," *ISCASS*, 2022.

[4] A. Dan, R. Shimizu, T. Nishikawa, S. Bian, and T. Sato, "Clustering approach for solving traveling salesman problems via Ising model based solver," *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

[5] Z. Wang, A. Marandi, K. Wen, R. L. Byer, and Y. Yamamoto, "Coherent Ising machine based on degenerate optical parametric oscillators," *Physical Review A*, 2013.

[6] T. Inagaki *et al.*, "A coherent Ising machine for 2000-node optimization problems," *Science*, 2016.

[7] S. Utsunomiya, K. Takata, and Y. Yamamoto, "Mapping of Ising models onto injection-locked laser systems," *Optics express*, 2011.

[8] S. Utsunomiya, N. Namekata, K. Takata, D. Akamatsu, S. Inoue, and Y. Yamamoto, "Binary phase oscillation of two mutually coupled semiconductor lasers," *Optics Express*, 2015.

[9] S. Dutta *et al.*, "An Ising Hamiltonian solver using stochastic phase-transition nano-oscillators," *arXiv preprint arXiv:2007.12331*, 2020.

[10] A. D. King, W. Bernoudy, J. King, A. J. Berkley, and T. Lanting, "Emulating the coherent Ising machine with a mean-field algorithm," *arXiv preprint arXiv:1806.08422*, 2018.

[11] E. S. Tiunov, A. E. Ulanov, and A. Lvovsky, "Annealing by simulating the coherent Ising machine," *Optics express*, 2019.

[12] K. Tatsumura, A. R. Dixon, and H. Goto, "FPGA-based simulated bifurcation machine," *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*, 2019.

[13] K. Tatsumura, M. Yamasaki, and H. Goto, "Scaling out Ising machines using a multi-chip architecture for simulated bifurcation," *Nature Electronics*, 2021.

[14] T. Zhang, Q. Tao, and J. Han, "Solving traveling salesman problems using Ising models with simulated bifurcation," *2021 18th International SoC Design Conference (ISOCC)*, 2021.

[15] T. Zhang and J. Han, "Efficient traveling salesman problem solvers using the Ising model with simulated bifurcation," *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.

[16] H. Goto *et al.*, "High-performance combinatorial optimization based on classical mechanics," *Science Advances*, 2021.

[17] M. W. Johnson *et al.*, "Quantum annealing with manufactured spins," *Nature*, 2011.

[18] E. Crosson and A. W. Harrow, "Simulated quantum annealing can be exponentially faster than classical simulated annealing," *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, 2016.

[19] H. M. Waidyasooriya and M. Hariyama, "A GPU-based quantum annealing simulator for fully-connected Ising models utilizing spatial and temporal parallelism," *IEEE Access*, 2020.

[20] H. M. Waidyasooriya and M. Hariyama, "Highly-parallel FPGA accelerator for simulated quantum annealing," *IEEE Transactions on Emerging Topics in Computing*, 2019.

[21] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, 1983.

[22] R. A. Rutenbar, "Simulated annealing algorithms: An overview," *IEEE Circuits and Devices magazine*, 1989.

[23] D. Henderson, S. H. Jacobson, and A. W. Johnson, "The theory and practice of simulated annealing," *Handbook of metaheuristics*, 2003.

[24] S. Kitamura, R. Iimura, and T. Kawahara, "AI chips on things for sustainable society: A 28-nm CMOS, fully spin-to-spin connected 512-spin, multi-spin-thread, folded halved-interaction circuits method, annealing processing chip," *2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI)*, 2020.

[25] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, "A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing," *IEEE Journal of Solid-State Circuits*, 2015.

[26] C. Yoshimura, M. Hayashi, T. Okuyama, and M. Yamaoka, "Implementation and evaluation of FPGA-based annealing processor for Ising model by use of resource sharing," *International Journal of Networking and Computing*, 2017.

[27] T. Takemoto, M. Hayashi, C. Yoshimura, and M. Yamaoka, "A 2 × 30k-spin multi-chip scalable CMOS annealing processor based on a processing-in-memory approach for solving large-scale combinatorial optimization problems," *IEEE Journal of Solid-State Circuits*, 2019.

[28] C. Yoshimura, M. Hayashi, T. Takemoto, and M. Yamaoka, "CMOS annealing machine: A domain-specific architecture for combinatorial optimization problem," *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[29] T. Takemoto *et al.*, "4.6 a 144kb annealing system composed of 9× 16kb annealing processor chips with scalable chip-to-chip connections for large-scale combinatorial optimization problems," *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, 2021.

[30] M. Aramon, G. Rosenberg, E. Valiante, T. Miyazawa, H. Tamura, and H. G. Katzgraber, "Physics-inspired optimization for quadratic unconstrained problems using a digital annealer," *Frontiers in Physics*, 2019.

[31] S. Matsubara *et al.*, "Digital annealer for high-speed solving of combinatorial optimization problems and its applications," *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020.

[32] S. Tsukamoto, M. Takatsu, S. Matsubara, and H. Tamura, "An accelerator architecture for combinatorial optimization problems," *Fujitsu Sci. Tech. J*, 2017.

[33] T. Okuyama, T. Sonobe, K.-i. Kawarabayashi, and M. Yamaoka, "Binary optimization by momentum annealing," *Physical Review E*, 2019.

[34] A. Lucas, "Ising formulations of many NP problems," *Frontiers in physics*, 2014.

[35] Q. Tao and J. Han, "Solving traveling salesman problems via a parallel fully connected Ising machine," *Design Automation Conference (DAC)*, 2022.

[36] K. Yamamoto, T. Takemoto, C. Yoshimura, M. Mashimo, and M. Yamaoka, "A 1.3-Mbit annealing system composed of fully-synchronized 9-board x 9-chip x 16-kbit annealing processor chips for large-scale combinatorial optimization problems," *2021 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, 2021.

[37] N. Hiroshi, K. Junpei, Y. Noboru, and M. Toshiyuki, "Description: Third generation digital annealer technology," *FUJITSU LIMITED*, 2021.

[38] G. A. Kochenberger, F. Glover, B. Alidaee, and C. Rego, "A unified modeling and solution framework for combinatorial optimization problems," *OR Spectrum*,

[39] M. Grötschel and W. R. Pulleyblank, "Weakly bipartite graphs and the max-cut problem," *Operations research letters*, 1981.

[40] C. Cook, H. Zhao, T. Sato, M. Hiromoto, and S. X.-D. Tan, "GPU-based ising computing for solving max-cut combinatorial optimization problems," *Integration*, 2019.

[41] K. L. Hoffman, M. Padberg, G. Rinaldi, *et al.*, "Traveling salesman problem," *Encyclopedia of operations research and management science*, 2013.

[42] E. Schubert and P. J. Rousseeuw, "Fast and eager k-medoids clustering: O(k) runtime improvement of the PAM, CLARA, and CLARANS algorithms," *Information Systems*, 2021.

[43] A. K. Tung, J. Han, L. V. Lakshmanan, and R. T. Ng, "Constraint-based clustering in large databases," *International Conference on Database Theory*, 2001.

[44] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proceedings of the IEEE*, 2020.

[45] H. Jiang, C. Liu, L. Liu, F. Lombardi, and J. Han, "A review, classification, and comparative evaluation of approximate arithmetic circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 2017.

[46] V. Camus, J. Schlachter, and C. Enz, "A low-power carry cut-back approximate adder with fixed-point implementation and floating-point precision," *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016.

[47] A. K. Verma, P. Brisk, and P. Ienne, "Variable latency speculative addition: A new paradigm for arithmetic circuit design," *Proceedings of the conference on Design, automation and test in Europe*, 2008.

[48] S.-L. Lu, "Speeding up processing with approximation circuits," *Computer*, 2004.

[49] K. Du, P. Varman, and K. Mohanram, "High performance reliable variable latency carry select addition," *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2012.

[50] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient VLSI implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2009.

[51] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The journal of chemical physics*, 1953.

[52] H.-S. Park and C.-H. Jun, "A simple and fast algorithm for k-medoids clustering," *Expert systems with applications*, 2009.

[53] W. Liu, L. Chen, C. Wang, M. O'Neill, and F. Lombardi, "Inexact floating-point adder for dynamic image processing," *14th IEEE International Conference on Nanotechnology*, 2014.

[54] M. Khodabina and A. Ahmadabadib, "Some properties of generalized gamma distribution," *Mathematical Sciences*, 2010.