Approximate Parallel Annealing Ising Machines (APAIMs): Controller and Arithmetic Design

Qichao Tao
Department of Electrical & Computer Engineering
University of Alberta
Edmonton, Canada
qichaotao@mail.tsinghua.edu.cn

Tingting Zhang
Department of Electrical & Computer Engineering
University of Alberta
Edmonton, Canada
ttzhang@ualberta.ca

Jie Han
Department of Electrical & Computer Engineering
University of Alberta
Edmonton, Canada
jhan8@ualberta.ca

***Abstract*—The demand for solving complex combinatorial optimization problems (COPs) in commercial and industrial applications motivates the development of efficient solvers. Ising model-based computers, or Ising machines, have emerged as high-performance solvers. Recently, an approximate parallel annealing Ising machine (APAIM) has been developed for solving constrained COPs such as the traveling salesman problem (TSP). To provide additional detail about the APAIM, this paper presents the designs of its controller and arithmetic units, especially that of the approximate adders in the local field accumulator units (LAUs) required for computing the Hamiltonian in the Ising model. The controller is implemented as a finite state machine and generates an instruction to determine the system operation. To improve hardware efficiency, the so-called lower-part-OR and truncated adder is used for the mantissa addition of floating-point numbers in the LAUs. Although the solution quality is slightly reduced, the use of approximate adders improves the hardware efficiency of a 64-spin APAIM.**

***Index Terms*—Simulated annealing, parallel annealing, Ising model, approximate adders, Ising machines.**

I. Introduction

Combinatorial optimization plays a key role in such applications as drug discovery, chip design, and machine learning [1]. Many combinatorial optimization problems (COPs) are computationally intensive to solve, which has motivated the development of efficient solvers to obtain a suboptimal solution within a reasonable time. One of the promising solvers is an Ising model-based computer, or an Ising machine. The Ising model mathematically describes the ferromagnetism of magnetic spins [2]. It solves COPs by finding the Hamiltonian at the ground state [3].

Conventional simulated annealing (SA) models the physical process of thermal annealing in metallurgy [2]. Various SA algorithms have been developed for ultrafast energy convergence or efficient hardware implementations of the Ising model [4]. A parallel annealing algorithm realizes parallel spin update by leveraging a two-layer spin structure [5], [6]. An improved parallel annealing (IPA) uses an exponential temperature function with a dynamic offset for efficiently solving a constrained COP, such as the traveling salesman problem (TSP) [7]. To implement the IPA, an approximate parallel annealing Ising machine (APAIM) approximately computes the momentum scaling factor using a linear function and applies approximation in addition for hardware efficiency [8].

Complementary to [8], this paper presents the controller design of the APAIM and explores the use of approximate adders for accumulation. The controller is implemented as a finite state machine with fifteen states. It generates a 12-bit instruction signal to determine the system operation. To save hardware, approximation is applied in the mantissa addition of 16-bit floating-point (FP) numbers. Specifically, the computation of less significant bits (LSBs) in the addition is approximated by using OR gates and truncation. The synthesis result shows that using approximate adders in the 64-spin APAIM reduces circuit area and delay, but maintains a high solution quality for the TSP.

The remainder of this paper is organized as follows. Section II introduces the IPA and the APAIM. The designs of the controller and approximate adders in the APAIM are discussed in Sections III and IV, respectively. Section V presents the experiment results. Section VI concludes the paper.

II. The APAIM

The Hamiltonian (H) of an Ising model is given by [1]

$$H = -\sum_{i,j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i, \tag{1}$$

where σ_i (or σ_j) is the state of the ith (or jth) spin with the value of $+1$ or -1, J_{ij} describes the interaction between the ith and jth spins, and h_i is the external magnetic field for the ith spin.

An n-city TSP is to find the shortest route that visits each of the n cities exactly once and returns to the starting point. The Hamiltonian of solving a TSP using an IPA with $N(=n^2)$ spins in a lattice and their replicas in a two-layer Ising model is given by [7]

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada (Project Number: RES0051374) and the University of Alberta (Project Number: RES0049590). T. Zhang was supported by the China Scholarship Council (CSC).

978-1-6654-5707-1/22/$31.00 ©2023 IEEE

$$H_{TSP} = -\sum_{i,k,j,l} J_{ikjl}\sigma^L_{ik}\sigma^R_{jl} - \frac{1}{2}\sum_{i,k} h_{ik}(\sigma^L_{ik} + \sigma^R_{ik}) + \omega_{ik}\sum_{i,k}(1 - \sigma^L_{ik}\sigma^R_{ik}), \quad (2)$$

where J_{ikjl} is the interaction between the spin with the index (i,k) on the left layer (σ^L_{ik}) and the spin with the index (j,l) on the right layer (σ^R_{jl}), h_{ik} is the external magnetic field for σ^L_{ik} and σ^R_{ik}. J_{ikjl} and h_{ik} are determined by the distances between cities. ω_{ik} is the self-interaction factor that indicates the coupling strength between σ^L_{ik} and σ^R_{ik}. It increases with annealing to guarantee $\sigma^R_{ik} = \sigma^L_{ik}$.

Algorithm 1 presents the IPA for solving the TSP based on (2). The spin configuration, the temperature (T) for annealing and the dynamic offset (ΔT) for increasing the temperature are first initialized (Line 1). Then, the spins on the left layer are updated when the current iteration (s) is odd; otherwise, the spins on the right layer are updated (Lines 3-5). In each iteration, ω_{ik} is reset to "0" with the dropout rate p; otherwise, it is decreased to $c \cdot \omega_{ik}$, where c is the momentum scaling factor (Line 9). The local field (lf_{ik}) determined by J_{ikjl} and h_{ik} (Line 10) is used to compute the energy variation (ΔE_{ik}) when σ_{ik} is flipped (Line 11). When the spin-flip probability (P_{ik}) is larger than the random number ($rand$), some randomness is introduced to help the system jump out of the local minima by flipping σ_{ik} (Lines 12-13). Subsequently, ΔT will increase by an increment value (T_{inc}) if no spin is flipped or be reset to "0" otherwise (Lines 17-19). When s reaches the predefined number of iterations for updating spin states, s_{max}, the spin configuration provides the solution found by the IPA.

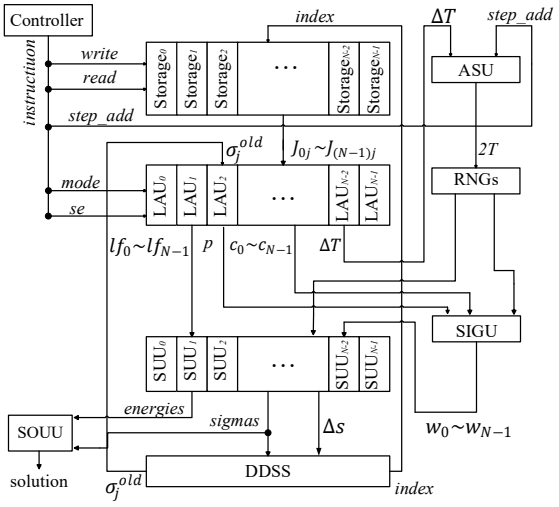

Fig. 1. The architecture of the APAIM [8]. The 2-D model in (2) is converted to a 1-D Ising model by reformulating σ_{ik} to $\sigma_{(i-1)\cdot n+k}$.

As shown in Fig. 1, an APAIM with N spins designed for the IPA is constructed by N local field accumulator units (LAUs), N self-interaction generating units (SIGUs), N spin update units (SUUs), $N/2$ random number generators (RNGs), an annealing schedule unit (ASU), a delta-driven simultaneous spin update unit (DDSS), a solution update unit (SOUU), a controller, and a memory block. These units are introduced as follows:

Algorithm 1 Improved Parallel Annealing for TSPs [7]

1: Initialize spin configurations, $T \Leftarrow$ a large value, $\Delta T \Leftarrow 0$
2: **for** $s = 1$ to s_{max} **do**
3: **if** s is odd **then**
$A \Leftarrow L, B \Leftarrow R$
4: **else**
$A \Leftarrow R, B \Leftarrow L$
5: **end if**
6: Update p and c, $T \Leftarrow (T + \Delta T) \cdot r^{s-1}$
7: **for** $i = 1$ to n **do**
8: **for** $k = 1$ to n **do**
9: $\omega_{ik} \Leftarrow 0$ with p, or $\omega_{ik} \Leftarrow c \cdot \omega_{ik}$ with $1 - p$
10: $lf_{ik} \Leftarrow (\frac{h_{ik}}{2} + \sum_{j,l} J_{ikjl}\sigma^B_{jl})$
11: $\Delta E_{ik} \Leftarrow (2lf_{ik}\sigma^A_{ik} + 2\omega_{ik}\sigma^A_{ik}\sigma^B_{ik})$
12: $P_{ik} \Leftarrow \min\{1, exp(-\Delta E_{ik}/T_s)\}$
13: **if** $P_{ik} > rand$ **then**
$\sigma^A_{ik} \Leftarrow -\sigma^A_{ik}$
14: **end if**
15: **end for**
16: **end for**
17: **if** no spin is flipped **then**
$\Delta T \Leftarrow \Delta T + T_{inc}$
18: **else**
$\Delta T \Leftarrow 0$
19: **end if**
20: **end for**

- LAU: It calculates lf_i (Line 10 in Algorithm 1), p, $c \cdot \omega_i$ (Line 9), and ΔT (Line 17).
- SIGU: It updates ω_i depending on p and $c \cdot \omega_i$ from the LAU (Line 9).
- SUU: It updates the spin states ($sigmas$ in Fig. 1) by using $rand$ from the RNGs (Line 13), lf_i from the LAUs (Line 10), and ω_i from the SIGUs (Line 9). It also generates Δs (in Fig. 1) for the DDSS as the index to indicate whether the spin sate is changed and computes $lf_i \cdot \sigma_i$ values ($energies$) for the SOUU.
- RNG: It generates a random number, $rand$ (Line 13).
- ASU: It counts the iterations and computes T (Line 6).
- DDSS: It receives all the new spin states ($sigmas$) and Δs from the SUUs, and then outputs the state of the flipped spin σ^{old}_j to LAUs and the corresponding index to the memory block.
- SOUU: It is a unit designed to improve the solution quality. It first computes the energy of the Ising model based on $energies$ and $sigmas$. Then, it selects a solution with the lowest energy among the found results.
- Controller. It determines the system operation by using an $instruction$ signal.
- Storage: the memory block. It uses the $index$ from the DDSS to select one value from $\{J_{0j}, J_{1j}, ..., J_{(N-1)j}\}$ and then sends it to the LAUs for computing new lf_i.

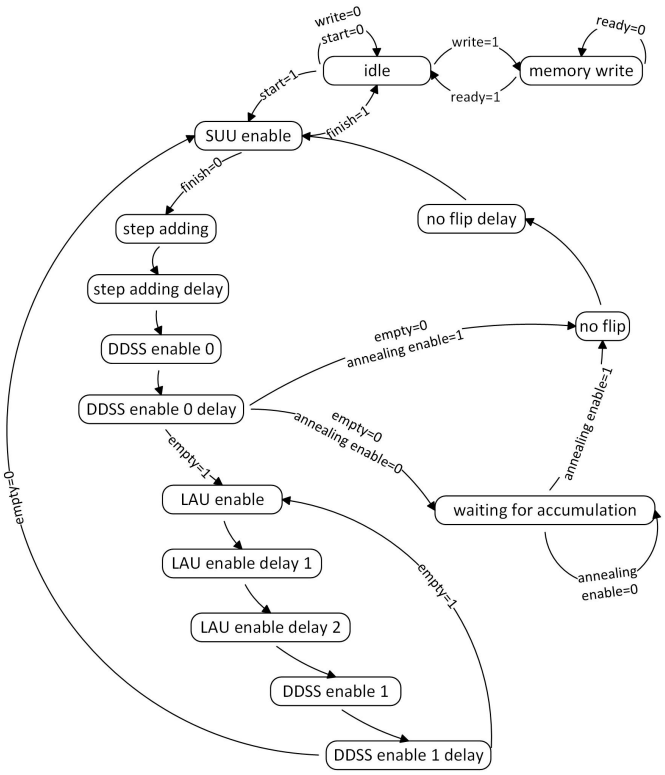

Fig. 2. The state transition in the FSM for the control unit.

III. The Controller Design in the APAIM

A. The Design of the Finite State Machine

In the computing architecture of the APAIM, the control unit is implemented as a finite state machine to ensure a proper sequence in the operation. As shown in Fig. 2, 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 held for one clock cycle to ensure the accuracy of the sampled signals. All the state transitions are controlled by clock and status signals. If there is no label on the arrow, it means the transition is only controlled by a clock signal. The $idle$ and $memory\ write$ states are used for system initialization. The circuit does not work at the $idle$ state, and data are written into the memory block at the $memory\ write$ state. The state returns to $idle$ when $ready$ is 1, which indicates that all data have been written.

The system starts annealing when the $start$ 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 (and also the end point) of the annealing; SUUs are working at this state and will generate new spin 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 indexes and spin states at $DDSS\ enable\ 0$ and $DDSS\ enable\ 1$ states. The $DDSS\ enable\ 0$ and $DDSS\ enable\ 1$ are for distinguishing whether no spin is flipped ($empty = 0$) after the spin update. If $empty = 0$, the system 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 system moves to $LAU\ enable$ and performs the accumulation of lf_i in the LAUs. Meanwhile, if $annealing\ enable = 0$, the system stops annealing to wait for the accumulation of total energy done.

B. The Instructions

The output of the control unit is a 12-bit $instruction$ signal, as shown in Fig. 3. The 11th bit is $mode$ that controls the computation mode of the LAU; the 10th bit is $rst_{dynamic}$ that resets ΔT to zero when a spin is flipped; the 9th bit is $noflip$; the 8th bit is $C_{s_{update}}$ to control updating c (Line 6 in Algorithm 1); the 7th and 6th bits are $se1$ and $se0$ used in the LAU; the 5th and 4th bits are $write$ and $read$ signals for the 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 I shows the instructions at different states.

TABLE I
Instructions and the corresponding states

States	$Instruction$
$idle$	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

IV. Approximate Adders in the APAIM

A. Using Truncated Adders or Lower-part-OR Adders

For arithmetic operation, a 16-bit FP number representation is used for the coefficients to obtain a more extensive range for solving complicated COPs. It contains a sign bit, 5 bits

11	10	9	8	7	6	5	4	3	2	1	0
$mode$	$rst_{dynamic}$	$noflip$	$C_{s_{update}}$	$se1$	$se0$	$write$	$read$	$DDSS\_en$	LAU_en	SUU_en	$step\_add$

Fig. 3. The instruction generated by the control unit.

for the exponent, and 10 bits for the mantissa. Therefore, the truncated adders (*TruA*s) [9] and lower-part-OR adders (*LOA*s) [10] are considered, respectively, in the LAUs to improve the hardware efficiency of mantissa addition.

The k LSBs in the mantissa adder are truncated in *TruA*s, while the l LSBs in the mantissa adder are computed by using OR gates in *LOA*s. The results in error metrics, including mean relative error distance (*MRED*), error rate (*ER*), normalized mean error (*NME*), and normalized mean error distance (*NMED*) [11] of *TruA*s and *LOA*s are shown in Table II.

Let *TruA*-k denote a *TruA* with k-bit truncation and *LOA*-l denote a *LOA* with l-bit approximation. Evaluated by four error metrics, *TruA* is less accurate than *LOA* when $k = l$ and even when l is slightly larger than k. For example, the *MRED*, *ER*, *NME*, and *NMED* of *LOA*-8 are lower than that of *TruA*-6 as the addition result produced by a *LOA* can be overestimated or underestimated, whereas the *TruA* always underestimates the result. Therefore, *LOA* is a better approximation scheme for accumulators than *TruA* considering accuracy albeit with an increased circuit area.

TABLE II
THE ERROR CHARACTERISTICS OF *TruA*S, *LOA*S AND *LOTA*S

Approximate Adders		*MRED* (10^{-3})	*ER* (%)	*NME* (10^{-3})	*NMED* (10^{-3})
TruA	$k = 3$	3.1	96.93	-2.9	2.9
	$k = 4$	6.7	99.21	-6.2	6.2
	$k = 5$	13.9	99.81	-12.9	12.9
	$k = 6$	28.3	99.96	-26.3	26.3
LOA	$l = 4$	1.4	68.14	7.68	1.3
	$l = 5$	2.7	76.35	7.14	2.5
	$l = 6$	5.4	82.33	6.83	5.0
	$l = 7$	10.7	86.50	8.92	10.0
	$l = 8$	21.1	89.95	8.68	20.1
LOTA	$l = 4, k = 3$	2.8	93.40	-2.1	2.5
	$l = 5, k = 3$	3.9	97.06	-2.1	3.6
	$l = 5, k = 4$	5.7	97.16	-4.6	5.3
	$l = 6, k = 3$	6.6	98.42	-2.1	6.1
	$l = 6, k = 4$	8.1	98.88	-4.7	7.5

$MRED = \frac{1}{N}\sum\frac{|R'-R|}{R}$, $ER = \frac{N_{err}}{N}$, $NME = \frac{\sum R'-R}{R_{max}\times N}$, and $NMED = \frac{\sum|R'-R|}{R_{max}\times N}$, where R' and R are the approximate and the accurate result, respectively, N_{err} and N are the numbers of the input cases that lead to erroneous result and all the possible input combinations, respectively, and R_{max} is the absolute value of the maximum result.

B. Using Both Lower-part-OR and Truncated Adders

In order to gain further improvement in hardware efficiency, we propose an approximate scheme that uses both *TruA* and *LOA*, thus it is referred to as a lower-part-OR and truncated adder (*LOTA*). It truncates k LSBs in an l-bit *LOA*. Thus, OR gates process $(l - k)$ bits, and k cannot be larger than l. Let *LOTA*-$l\&k$ denote a *LOTA* with l-bit approximation and k-bit truncation. Table II 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. When the number of approximated bits is only one more than the number of truncated bits, the *NMED* and *MRED* of a *LOTA* are close to that of *TruA*s with the same truncated bits, and larger than that of the *LOA*s with the same approximated bits.

C. Accelerating the Simulation of Error Characteristics of Approximate Adders

To accelerate the fault tolerance test, the error characteristics of approximate adders are simulated by using an injected noise. By observations from the experimental results, the distribution of the relative error distance ($RED = \frac{|R'-R|}{R}$) resembles a Gamma distribution. Its probability density is given by $f(y|\alpha,\beta) = \frac{\beta^\alpha}{\Gamma(\alpha)}y^{\alpha-1}e^{-\beta y}$, where $\Gamma()$ is the Gamma function, α is the shape parameter, and β is the inverse scale parameter.

We use Gamma distributions to approximate the distribution of *RED*s. The random variable that follows the Gamma distribution, denoted by r, is applied to the accumulation process in the LAUs. In particular, when calculating the addition of two input operands a and b, $(a + b)$ is multiplied by $(1 - r)$ in a *TruA* (the error is always negative) and multiplied by $(1 \pm r)$ in a *LOA* and a *LOTA* (the error can be positive or negative with a similar probability). Although not shown, due to space limitation, our experiments indicate the fitting Gamma distributions produce similar trends as the *RED* distributions but with higher values. Moreover, the Gamma distribution fits best to simulate the *RED*s for *TruA*s, compared to the other two approximate adders.

V. THE PERFORMANCE OF THE APAIM

A. Solution Quality of Solving TSPs

The solution quality is evaluated from the average (*Ave*), maximum (*Max*), minimum (*Min*), and standard deviation (*Std*) of the travel distances obtained by the APAIM. Moreover, the violation rate (*VR*) is used to indicate 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 considered as the benchmark and the distances between cities are scaled to $[0, 1]$. Table III shows the results when simulating the error due to approximate adders by random variables. The *VR* increases significantly when the number of truncated bits or approximated bits is larger than 4 for *TruA*s and larger than 6 for *LOA*s. As shown in Table II, the MRED results for *TruA*-4 and *LOA*-6 are 0.0067 and 0.0054, respectively. Therefore, the APAIM can tolerate errors with an *MRED* of around 0.0067. Moreover, the *VR* becomes higher than 10%, while both the *Ave* and *Std* increase when applying Gamma distributed noise for *LOTA*-5&4, *LOTA*-6&3, and *LOTA*-6&4 to the APAIM. Thus, *TruA*-3, *TruA*-4, *LOA*-4, *LOA*-5, *LOA*-6, *LOTA*-4&3, and *LOTA*-5&3 are considered to evaluate the solution quality obtained from their actual implementations. As shown in Table IV, both *VR* and *Ave* decrease in actual implementations. Except *LOA*-6, all the approximate adders lead to a similar solution quality with 0% *VR*s and small *Ave*s. A low *Std* indicates that the solution quality is stable.

TABLE III
THE TRAVEL DISTANCES OBTAINED FROM THE APAIM AFTER APPLYING RANDOMLY GENERATED NUMBERS THAT FOLLOW GAMMA DISTRIBUTIONS (FOR *TruA*S, *LOA*S AND *LOTA*S) AS NOISE

APAIMs with Noise			*Ave*	*Max*	*Min*	*Std*	*VR*
Adders	l/k	(α, β)					
TruA ($l = k$)	3	$(2.96, 956.79)$	2.87	3.54	2.39	0.25	0%
	4	$(3.28, 489.41)$	3.07	4.63	2.39	0.30	7%
	5	$(3.63, 260.84)$	4.42	6.67	2.39	0.97	72%
	6	$(3.86, 136.48)$	6.92	11.05	3.37	1.64	98%
LOA ($k = 0$)	4	$(1.20, 627.40)$	2.82	3.18	2.39	0.22	0%
	5	$(1.14, 294.41)$	2.91	3.91	2.39	0.31	2%
	6	$(1.10, 168.61)$	3.16	4.33	2.39	0.42	8%
	7	$(1.08, 78.85)$	4.29	8.10	2.51	1.14	63%
	8	$(1.00, 53.82)$	6.59	12.00	2.90	1.71	96%
LOTA	4/3	$(2.17, 712.62)$	2.83	3.50	2.39	0.23	0%
	5/3	$(2.63, 622.54)$	2.90	3.99	2.39	0.29	2%
	5/4	$(2.18, 310.75)$	3.11	4.60	2.39	0.44	11%
	6/3	$(2.15, 325.93)$	3.18	5.47	2.39	0.59	15%
	6/4	$(2.62, 324.00)$	3.21	4.51	2.39	0.50	17%

TABLE IV
THE TRAVEL DISTANCES OBTAINED FROM THE APAIM WITH THE USE OF *TruA*S, *LOA*S, AND *LOTA*S

APAIMs		*Ave*	*Max*	*Min*	*Std*	*VR*
AccA	$l, k = 0$	2.67	3.08	2.39	0.17	0%
TruA ($l = k$)	$k = 3$	2.69	3.14	2.39	0.18	0%
	$k = 4$	2.74	3.58	2.39	0.24	0%
LOA ($k = 0$)	$l = 4$	2.71	3.06	2.39	0.18	0%
	$l = 5$	2.77	3.17	2.39	0.21	0%
	$l = 6$	3.08	3.82	2.65	0.26	4%
LOTA	$l = 4, k = 3$	2.71	3.17	2.39	0.20	0%
	$l = 5, k = 3$	2.72	3.06	2.39	0.21	0%

AccA: accurate adder.

TABLE V
THE CIRCUIT MEASUREMENTS OF THE APAIM

Ising Machines		Spin	Precision	Area (mm^2)	Power (mW)	Delay (ns)
APAIM	$l, k = 0$	64	16-bit	6.300	41.289	3.97
	$l = k = 3$	64	16-bit	6.275	41.035	3.90
	$l = k = 4$	64	16-bit	6.266	40.931	3.89
	$l = 4, k = 0$	64	16-bit	6.273	41.263	3.84
	$l = 5, k = 0$	64	16-bit	6.266	41.223	3.82
	$l = 6, k = 0$	64	16-bit	6.259	41.080	3.79
	$l = 4, k = 3$	64	16-bit	6.269	41.130	3.84
	$l = 5, k = 3$	64	16-bit	6.262	41.108	3.82
STATICA [5]		512	5-bit	12	629	-

-: not reported. Simulation results are obtained by using the Synopsys Design Compiler. A CMOS 28 nm technology is applied with a supply voltage of 1.0 V, a temperature of $25°C$, and a clock frequency of 200 MHz. Some data in the table are reported in [8].

B. Hardware Performance

The hardware performance of the 64-spin APAIM with 16-bit FP coefficients implemented with or without approximate adders is measured and reported in Table V. The circuit measurements for STATICA [5] are also presented. However, a fair comparison between the APAIM and the STATICA cannot be conducted because the STATICA was designed to solve simple unconstrained COPs and synthesized using a 65-nm technology.

Using approximate adders in the APAIM reduces the hardware cost. 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 mostly reduced by the number of approximate bits. The circuit area by using the *LOTA* is lower than using the *LOA* with the same approximate bits, whereas the power for using *LOTA* is between those of using the *LOA* and *TruA* with the same approximate bits (higher than *TruA* but lower than *LOA*). Furthermore, the circuit area, delay, and power dissipation by using *LOTA*-5&3 are slightly lower than using *LOTA*-4&3, however they can find a similar average travel distance. Compared with TruAs and LOAs, the average travel distance by using *LOTA*-5&3 is similar to those using *TruA*-4 and *LOA*-4 but the circuit area and delay are smaller. Finally, although *TruA*-4 results in the lowest power dissipation, the power by using *LOTA*-5&3 is close to that by using *TruA*-4. Therefore, the APAIM with *LOTA*-5&3 achieves a good trade-off between hardware efficiency and solution quality for solving TSPs.

VI. CONCLUSION

This paper presents the controller design and explores the use of approximate arithmetic in a recently developed Ising machine, namely, the APAIM, for solving TSPs. The controller implements a finite state machine with fifteen states and produces a signal to instruct the system operation. To reduce the hardware for the mantissa addition of 16-bit FP numbers, the less significant bits are processed by using different approximation schemes. The hardware performance and the solution quality show the potential of the 64-spin APAIM with the use of lower-part OR and truncated adders for efficiently solving TSPs.

REFERENCES

[1] A. Lucas, "Ising formulations of many NP problems," *Front. Phys.*, p. 5, 2014.

[2] R. A. Rutenbar, "Simulated annealing algorithms: An overview," *IEEE Circuits Syst. Mag.*, vol. 5, no. 1, pp. 19–26, 1989.

[3] N. Mohseni, P. L. McMahon, and T. Byrnes, "Ising machines as hardware solvers of combinatorial optimization problems," *Nat. Rev. Phys*, vol. 4, no. 6, pp. 363–379, 2022.

[4] T. Zhang, Q. Tao, B. Liu, and J. Han, "A review of simulation algorithms of classical Ising machines for combinatorial optimization," in *ISCAS*. IEEE, 2022, pp. 1877–1881.

[5] K. Yamamoto, K. Kawamura, K. Ando, N. Mertig, T. Takemoto *et al.*, "STATICA: A 512-spin 0.25 m-weight annealing processor with an all-spin-updates-at-once architecture for combinatorial optimization with complete spin–spin interactions," *IEEE JSSC*, vol. 56, no. 1, pp. 165–178, 2020.

[6] T. Okuyama, T. Sonobe, K.-i. Kawarabayashi, and M. Yamaoka, "Binary optimization by momentum annealing," *Phys. Rev. E*, vol. 100, no. 1, p. 012111, 2019.

[7] Q. Tao and J. Han, "Solving traveling salesman problems via a parallel fully connected Ising machine," in *DAC*, 2022, pp. 1123–1128.

[8] Q. Tao, T. Zhang, and J. Han, "An approximate parallel annealing ising machine for solving traveling salesman problems," *IEEE Embed. Syst. Lett*, 2023.

[9] N. Zhu, W. L. Goh, W. Zhang, K. S. Yeo, and Z. H. Kong, "Design of low-power high-speed truncation-error-tolerant adder and its application in digital signal processing," *IEEE TVLSI*, vol. 18, no. 8, pp. 1225–1229, 2009.

[10] 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 TCAS1*, vol. 57, no. 4, pp. 850–862, 2009.

[11] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," *Proc. IEEE*, vol. 108, no. 12, pp. 2108–2135, 2020.