Trotter Based Parallel Processing of Quantum Annealing for FPGA

Sohei Shimomai Shinji Kimura

Waseda University
Graduate School of Fundamental Science and Engineering
3-4-1, Okubo, Shinjuku, Tokyo, 169-8555 JAPAN
sohei.shimomai@islab.cs.waseda.ac.jp shinji_kimura@waseda.jp

Abstract— The paper proposes a trotter-oriented parallel emulation method of quantum Monte Carlo method for FPGA. Random spin toggles in trotters are manipulated in parallel. By using the Mersenne twistor method to compute random numbers and by sharing spin information of adjacent trotters, the proposed parallel processing hardware can obtain the same accuracy as in the case of serial processing and gains more than 20 times speed-up compared with the serial hardware on 32 trotter case.

I. Introduction

Recently, the development of new computing devices and mechanisms has become active worldwide because the forthcoming end of the process is shrinking. Especially new computation mechanisms for specialized problems capture our attention. Quantum annealing is one of such new computation methods to solve combinatorial optimization problems [1]. The method uses the properties of quantum spins where spin changes their direction independently to minimize the total energy depending on the correlation and the self energy of spins. The property can be used to parallelize the annealing process [2].

Combinatorial optimization problems are common in daily life, such as the determination of substantial quantity in chemistry, delivery planning [3] and job scheduling in companies [4], [5]. Quantum annealing can be used to solve such problems in a short time [6].

Quantum annealing can be physically implemented using quantum spins (quantum bits) in superconducting chips working under near absolute zero temperature. Physical quantum annealers with the cooling unit are large and expensive and it is rather hard to implement a large number of spins. On the other hand, the simulation of quantum annealing process called Simulated Quantum Annealing (SQA) has been studied for executing on conventional CPU or GPU [7], [8]. SQA can manage a large number of spins for solving a variety of combinatorial problems [9]. Quantum Monte Carlo is a widely used SQA method, where spins are randomly and iteratively toggled to minimize the total energy of correlated spins. When a randomly selected spin is toggled, the energy difference is computed by the toggle and judged based on the difference in whether the toggle is accepted. Spin toggles are executed in series to manage the correlation between spins.

Hardware emulation of SQA has also studied on conventional chip or FPGA [9], [10], [11]. By parallelizing spin toggles, we can accelerate the Quantum Monte Carlo, but the correlation between spins should be carefully managed. Necessary hardware resources for parallel operations are another issue, and previous proposals [9], [10], [11] do not work efficiently because of the resource limitation of FPGA. These previous works [9], [10], [11] have focused on increasing the number of parallels and have used full parallel processing. Therefore, they may not be able to solve large problems due to hardware resources.

The paper proposes a parallelization method that takes the hardware resources of FPGA into account. In Quantum Monte Carlo, several spin sets called trotters are used to simulate the quantum effect, and the correlation between spins in different trotters is very limited compared with that of spins in the same trotter. In the proposed method, toggles of spins in different trotters are parallelized by caring for the limited correlation. An operational unit is designed to manage spin toggles of several trotters and the number of parallel operational units is decided to depend on the usable FPGA resource. By using the Mersenne twistor method to compute random numbers and by incorporating information about neighboring trotters, the proposed parallel processing method can obtain the same accuracy as in the case of serial processing. This method makes it possible to adjust the number of parallels according to the hardware resources and the problem scale.

The rest of the paper is organized as follows: Chapter 2 describes the emulation of SQA, Chapter 3 describes the proposed inter-trotter parallel processing method, and Chapter 4 evaluates and discusses the processing time of the proposed parallel processing method. Chapter 5 is a conclusion.

II. Emulation of Simulated Quantum Annealing

This chapter shows an emulation method of simulated quantum annealing on the Ising model using Quantum Monte Carlo method. We represent a mapping of the traveling salesman problem to the Ising model.

A. *Ising Model*

The Ising model is to describe the spin behavior of magnetic materials in statistical mechanics [7]. It consists of a set of spins that take two states: upward and downward. The entire spins are affected by an external magnetic field, and each spin interacts with the other. A spin can be represented by a spin variable s_i, which takes the value of +1 for upward state and -1 for downward state. The interaction coefficient betweens_i and s_j is denoted by$J_{i,j}$and the self-energy due to the magnetic field on the spin s_i is denoted by h_i. With these symbols, The energy of an n spin Ising model with a transverse field is expressed by Eq.(1)

$$H = -\sum_{i<j} J_{ij} s_i s_j - \sum_{i=1}^{n} h_i s_i \tag{1}$$

In actual magnetic materials, the spin direction changes to minimize the energy function H. Fig.1 shows an example of a 9-spin Ising model. QUBO (Quadratic Unconstrained Binary Optimization)is another optimization problem on binary variables, where the variables in QUBO take the binary value 1 or 0 and the quadratic expression of the variables represents the cost function to be minimized. The spin variable s_i in the Ising model and the binary variable x_i in QUBO can be converted by EQ.(2), and QUBO is equivalent to the Ising model.

$$x_i = \frac{(s_i + 1)}{2} \tag{2}$$

The choice of the Ising model and QUBO is up to us and an easier way can be used to formulate a combinatorial problem. From the point of view of the computation, QUBO may reduce the number of operations because product terms in a quadratic expression become 0 when a binary variable is 0.

B. *Simultated Quantum Annealing*

In the section, SQA based on a Quantum Monte Carlo method (QMC) is introduced. In QMC, the spin state with minimum energy is searched from an initial spin state by toggling a randomly selected spin one by one. In detail, a spin is randomly selected, the energy difference is computed when toggling the spin, and then its toggle is judged to be accepted or not. The spin toggle steps need a large number of iterations.

QMC uses random numbers for the initial spin state

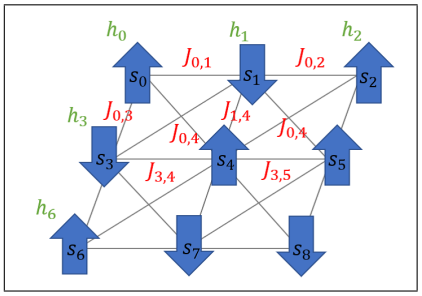

Fig. 1. Two-dimensional Ising model.

generation and the spin toggling steps. There are several ways to generate random numbers such as Linear congruential generators and the Mersenne-Twister method. In our QMC, we use Mersenne-Twister method. In addition to thermal fluctuations similar to one in Simulated Annealing, quantum fluctuations are used to search for the optimal solution. By using these fluctuations, QMC prevents falling into the locally optimal solution. In addition, we deal with multiple spin sets called "trotters". Trotters are multiple replicas of quantum-specific states. The entire system is searched from m different states. Between neighboring trotters, interaction occurs specified by the interaction coefficient corresponding to the transverse magnetic field [7]. In quantum annealing, a term for the transverse magnetic field between trotters is added to Eq(1). The Hamiltonian of the Ising model in QMC is expressed as follows [2].

$$\begin{aligned} H = &-\frac{1}{m}\sum_{k=1}^{m}\Big(\sum_{i<j} J_{ij} s_{k,i} s_{k,j} - \sum_{i=1}^{n} h_i s_{k,i}\Big) \\ &-\frac{1}{2\beta} logcoth\Big(\frac{\beta\Gamma}{m}\Big)\sum_{k=1}^{m}\sum_{i=1}^{n} s_{k,i} s_{k+1,i} \end{aligned} \tag{3}$$

m is number of trotters, $s_{i,k}$ is a spin i in trotter k. When mapping Ising models with SQA, it is necessary to consider trotters that represent multiple quantum states. Therefore, we treat a two-dimensional array of trotter numbers and spin numbers. We show the spin array with Trotter in Fig. 3. The first part corresponds to the mean of each trotter's energy. The second part corresponds to the correlation of trotters. The trotter index is denoted by k, the temperature is denoted β and the strength of the transverse magnetic field is denoted by Γ.

When Γ is large, the interaction between trotters is small and the spins search for the optimal solution without interference between trotters. As Γ is gradually decreased, the interaction between trotters becomes larger and the interference increases. As can be seen from equation (2), $\frac{\beta\Gamma}{m}$ becomes smaller and converges to the optimal solution. Fig.2 shows an image of the search in quantum annealing.

We show the procedure of QMC.

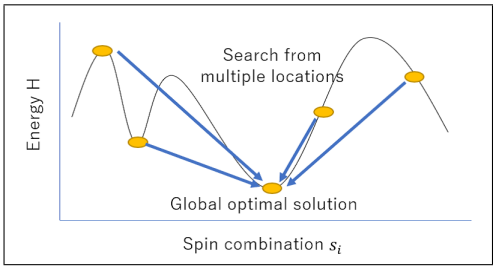

Fig. 2. Transition diagram of quantum annealing.

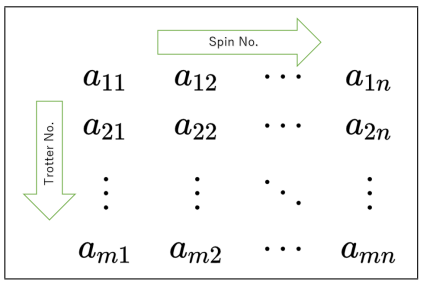

Fig. 3. Spin array introducing trotters .

1. Based on the input data, initial spins are generated for all trotters.
2. Initialize various variables such asβ and Γ.
3. A trotter and a spin in the trotter are randomly selected and toggled, and the energy changes ΔH are calculated before and after the toggling.
4. If ΔH is minus, spin toggling is accepted. Even when ΔH is plus, spin toggling is accepted based on appropriate transition probabilities.
5. Repeat steps 3. to 4. a specified number of times (inner loop).
6. Multiply the transverse magnetic field by 0.99.
7. Repeat steps 3. to 6. a specified number of times (outer loop).
8. Select the trotter with the minimum energy H as the optimal Ising model.
9. Transform the Ising model selected in 8. to the state corresponding to the combinatorial optimization problem.

QMC consists of two loops, 5. and 7., which are called the inner and outer loops.

C. Mapping of the Traveling Salesman Problem

To check our proposed parallelization method, we use the Traveling Salesman Problem. The TSP is the problem of finding the route with the shortest total distance among all routes visiting all cities only once, under a given distance between each city. When modeling the N-city traveling salesman problem, prepare an array with N^2 spins of N rows and N columns. The rows indicate "how many cities to visit" and the columns indicate "which cities to visit". Thus, the element in the ith row and jth column of the array represents "visit the city j at time i or not". The distance $d_{j1,j2}$ between cities $j1$ and $j2$ is included in the total distance if $j1$ is visited at time i and $j2$ is visited at time i+1. For representing the total distance by the multiplication of variables, binary variables are suitable and QUBO is used. The total distance is the sum of $d_{j1,j2} \times s_{i*N+j1} \times s_{(i+1)*N+j2}$.

III. Parallel Processing

In QMC, the calculation of the energy difference for toggling spins is repeated about 10^9 times. The repetition needs a lot of time on a computer and its acceleration is required. We introduce a parallel algorithm of QMC for FPGA. Spin toggles in QMC are parallelized with caring about the correlation between spins. If two correlated spins are toggled at the same time, the correlation should be cared for when computing the energy difference by the toggles. FPGA is a good platform to implement parallel processing units but its resource limitation should be considered.

In this section, we describe the interdependencies in SQA and then describe a parallel processing method that we can change the number of parallelisms depending on hardware resources in the design phase.

A. Inter Dependencies in Quantum Annealing

When the number of Trotter is m and the number of spins of the traveling salesman problem is$n$, the spin array of the entire Ising model is $m \times n$. In QMC, there are complex interdependencies. So, it is necessary for implementing parallel processing to consider them. We show the image of the dependencies in Fig. 4. As an example, suppose we make a toggle decision for $spin(i)$ spin at trotter number m. As a condition for judgment, we use the energy change ΔH before and after toggling the spin. When calculating ΔH, from the first entry of 3, we need information on all spins of trotter number k. From the second entry, we also need information on $spin(i)$ of the adjacent trotters with trotter numbers $k-1$ and $k+1$. Therefore, if multiple tasks read and write spin information from shared memory at the same time, there is a possibility of information discrepancies.

B. Parallel Processing Architecture

This section describes the proposed parallel processing method of trotters considering interdependencies.

The set of spins is arranged as shown in Fig. 3. To process spin toggles in different trotters in parallel, we divide

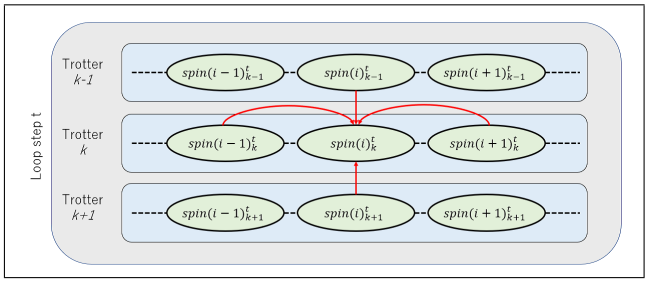

Fig. 4. Interdependencies among trotters.

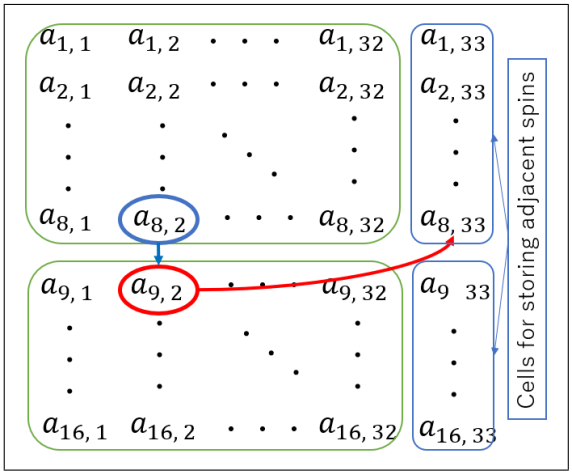

Fig. 5. Example of Sequence Division.

the spin array into trotter rows according to the number of parallelisms. When the number of trotters is m and the number of parallelisms is P, then the number of parallels P is $1 \leq P \leq m$. Each processing unit randomly selects one spin to be toggled, and P processing units are considered. They perform a pipeline processing of the toggle decision for the selected spin to reduce processing time. Parallel units might select the correlated spins in different trotters. At present, the energy differences of the selected spins are computed using the current state and do not care about the effect of the toggle. However the selection of correlated spins is very rare, and almost the same optimization results can be obtained by this method. When calculating ΔH, processing units add or subtract the correlation coefficient between the correlated spins according to the value of the spin variable in the same trotter. Processing units also need the spin information of the adjacent trotters. However, if the spins located in the rows at both ends of them are selected as the toggle targets, they contain only spin information in at most one adjacent trotter, resulting in insufficient data references. To solve this problem, we add a spin dedicated to adjacent spin information at the end within each processing unit. If need adjacent spin information during spin selection, they obtain the spin data from the adjacent processing unit and store it there. By referring to the information there, they can calculate accurate ΔH.

As an example, the image of processing the entire spin array in 2 parallel when the number of trotters is 16 and the number of spins is 32 is shown in Fig.5. One processing unit contains spins with trotter numbers 1-8; the other contains spins with trotter numbers 9-16. If we want to find ΔH when $a_{8,2}$ is toggled, the spin states information of $a_{7,2}$ and $a_{9,2}$ are needed for calculating. Since $a =_{9,2}$ is necessary, that is stored in $a_{8,33}$, in advance. By referring to the spin information of $a_{8,33}$ instead of $a_{9,2}$, it is possible to process without any difference in information among the processing units.

Next, we show a time chart of the execution in Fig.6. Each processing unit randomly selects a trotter and a spin and shares the information with neighboring processing units as needed. After all the processing in the loop is finished, the energy is calculated for each trotter to find the minimum spin state.

IV. Implementation and Evaluation

In this section, we describe the program implemented on FPGA and evaluate the processing time.

A. FPGA Implementation

For the FPGA implementation, the functions were described in C and converted to the hardware description language Verilog-HDL by high-level synthesis using Xilinx's Vitis HLS. For random number generation, we used the Mersenne-Twister method. We also tried the linear congruence method, but the remainder of the random number generated by the linear congruence method, when divided by an even number, was likely to alternate between even and odd numbers, resulting in a loss of accuracy during parallel execution.

For the evaluation, we used a Xilinx Alveo U250 FPGA board, which has only PL (Programmable Logic) and can exchange data with the CPU of the host PC. As an environment for running the emulator on the FPGA, we used PYNQ (Python for Zynq) running on the Jupyter notebook. Fig.7 shows the overall architecture of the emulator. The host computer and FPGA are connected by a PCIe bus, and the input data are stored in the global memory of the FPGA board. The kernel store the spin data in local memory and the PE contains adders and comparators for adding energies between spins and comparing them with random numbers. Each kernel runs in parallel. Single-precision floating-point kernels are used for the interaction coefficients between spins, and double-precision floating-point kernels are used for the energy difference calculations.

B. Evaluation

We prepared a 32 City (1024 spin), 64 City (4096 spins), and 96 City (9216 spins) TSP and measured the FPGA re-

TABLE I
PROCESSING TIME AND RESOURCE CONSUMPTION AND ITS RATIO AT EACH NUMBER OF PARALLELISM.

Instance Size	Trotters	Parallel	Time[s]	BRAM	DSP48E	FF	LUT
32 City 1024spin	32	1	207.84	123(2%)	135(1%)	13651(˜0%)	23683(1%)
		2	107.48	149(2%)	176(1%)	17920(˜0%)	31568(1%)
		4	54.46	199(3%)	268(2%)	26280(˜0%)	44658(2%)
		8	29.43	303(5%)	452(3%)	42976(1%)	70866(4%)
		16	17.68	503(9%)	820(6%)	76068(2%)	123496(7%)
		32	12.17	871(16%)	1556(12%)	141555(4%)	213613(12%)
64 City 4096spin	32	1	279.80	413(7%)	135(1%)	13750(˜0%)	23778(1%)
		2	143.67	494(9%)	176(1%)	18066(˜0%)	31714(1%)
		4	76.11	656(12%)	268(2%)	26528˜0%()	44834(2%)
		8	42.17	980(18%)	452(3%)	43428(1%)	70990(4%)
		16	25.20	1620(30%)	820(6%)	77092(2%)	123676(7%)
		32	17.24	2916(54%)	1556(12%)	142875(4%)	214051(12%)
96 City 9126spin	32	1	513.92	911(16%)	146(1%)	16163(˜0%)	25385(1%)
		2	276.41	1091(20%)	198(1%)	21078(˜0%)	34021(1%)
		4	159.12	1449(26%)	296(2%)	30386(˜0%)	48021(2%)
		8	100.45	2169(40%)	492(4%)	48995(1%)	76012(4%)
		16	71.16	3601(66%)	884(7%)	85800(2%)	129729(7%)
		32	39.25	1423(26%)/URAM 96(7%)	1567(12%)	179356(5%)	1316760(76%)

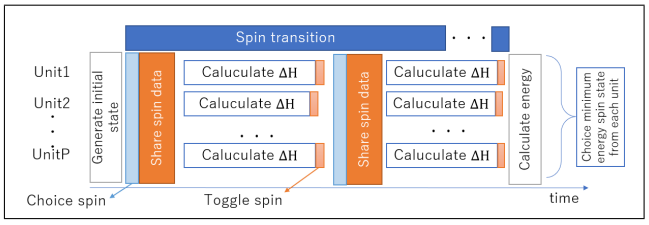

Fig. 6. Parallel Processing Time Chart.

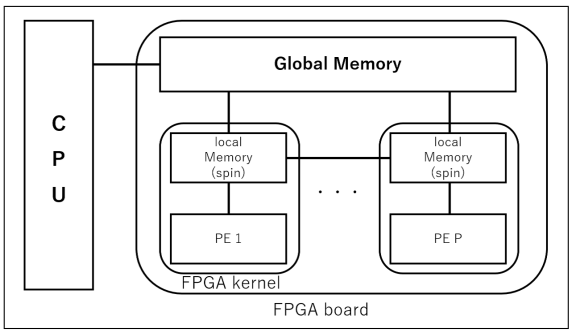

Fig. 7. Emulator architecture.

sources and processing time. In the evaluation, the number of trotters was fixed at 32, and the number of parallels varied from 1, 2, 4, 8, 16, and 32. To compare processing speeds between parallel numbers, we set the operating frequency for each parallel number to a constant 155 MHz. Tab.I shows the processing time and percentage of FPGA resources used for different numbers of parallelism and problem scales. We refer to serial processing without parallel processing as "one parallel". In Problem Scale 96 City, we used LUTRAM and URAM in addition to BRAM for storing input data because of the limited resources of BRAM.

Fig.8 shows the ratio of processing times (hereafter referred to as "ratio") when the number of parallels is changed from one to another. We calculated the ratio using $\frac{t_n}{t_1}$, where t_n is the processing time at a parallel number n. As can be seen from Fig.8, when the problem scale is 32 City, 32-parallel achieves up to 17.08 times faster than 1-parallel. In this parallel method, the growth of the ratio to one parallel slowed down as the number of parallels increased. The emulator reads the self-energy and interaction coefficient from the input data when making the spin toggle decision. However, the amount of data read from the input data varies depending on the spin selected. In addition, to prevent differences in spin information among processing units, the proposed method shares data every time all processing units complete one spin toggle decision. Due to these factors, we believe that the increase in the number of data parallels has increased the waiting time until all processing units are completed, resulting in a longer overall processing time. Furthermore, this feature became more pronounced as the problem scale increased. We believe that this is because the larger the problem scale, the larger the difference in the amount of data read from the input data due to the choice of spins.

Tab.II compares the processing time of this implementation with 32 parallelisms on FPGA (16 parallelisms in 96 cities) and CPU. The CPU is an Intel i9-7900X CPU@3.30GHz with 128 GB of main memory. Tab.II shows that when the problem scale is 64 City, the FPGA implementation is 2.80 times faster than The CPU. Then, by incorporating random number calculations based on the Mersenne-Twister method and information on neigh-

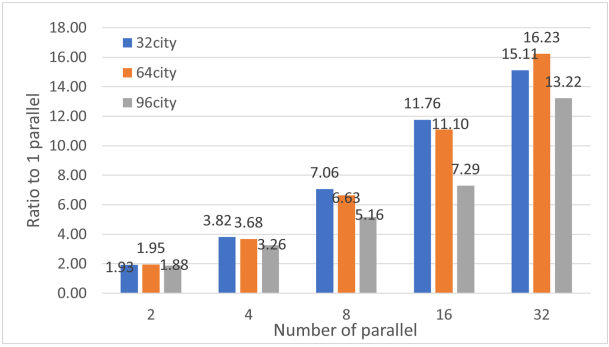

Fig. 8. Ratio of processing time at each parallel number to processing time at parallel number 1.

TABLE II
PROCESSING TIME ON CPU AND FPGA.

	Time[s]	
Instance Size	CPU(3.3GHz)	FPGA(155MHz)
32 City	29.03	12.17
64 City	48.34	17.24
96 City	80.66	39.25

boring trotters into the emulator, the interdependencies among spins are overcome, and parallel processing is performed according to the hardware resources.

In this implementation, 32-parallel achieved a speedup of up to 17.08 times faster than 1-parallel. The FPGA implementation achieved a speedup of up to 2.80 times faster than the CPU. However, as the number of parallels increased, the latency to complete all processing units increased, and the increase in the ratio to one parallel slowed down as the number of parallels increased.

We believe that this method can be applied to a variety of problems because it employs trotter partitioning for parallel processing. Future work is needed to incorporate parallel processing methods that use fewer BRAM resources.

V. CONCLUSION

This paper proposes a new parallel processing method of Simulated Quantum Annealing using Quantum Monte Carlo. The proposed method divides trotters into P processing units and each processing unit selects and computes energy difference when the spin is toggled. The correlation between trotters in different processing units is considered by synchronizing the necessary information. By using the Mersenne-Twister method for random number generation, the quality is almost the same as serial processing. The proposed method can set the parallelism considering the hardware resource of FPGA. The proposed method is implemented on FPGA and gains more than 20 times speed-up compared with a serial execution of hardware on 32 trotter case. The hardware is about 2 times faster compared with a serial software execution on 3.3 GHz CPU.

ACKNOWLEDGEMENTS

The authors would like to thank to Prof. Masao Yanagisawa, Prof. Youhua Shi, Prof. Toshihiko Yoshimasu, and members of Kimura laboratory of Waseda University for their daily discussions.

Thanks are also due to Dr. Yuichi Nakamura, Dr. Yuki Kobayashi and Dr. Masayuki Shirane for their supports and comments. This paper was based on results obtained from a project, JPNP16007, commissioned by the New Energyand Industrial Technology Development Organization (NEDO), Japan. This work is partly supported by a fund from NEC.

REFERENCES

[1] Yamaoka Masanao, Yoshimura Chihiro, Hayashi Masato, Okuyama Takuya, Aoki Hidetaka, Mizuno Hiroyuki, “Basic Research on AI -Ising Computer-”, Hitachi Vol.98, pp.272-273, Apr. 2016.

[2] T. Kadowaki, H. Nishimori, “Quantum annealing in the transverse Ising model,” Physical Review E, vol. 58, no. 5, p. 5355, 1998.

[3] F. Neukart, G. Compostella, C. Seidel, D. von Dollen, S. Yarkoni, and B. Parney, “Traffic flow optimization using a quantum annealer,” Frontiers in ICT, vol. 4, p. 29, 2017.

[4] R. Orus, S. Mugel, and E. Lizaso, “Quantum computing for finance: overview and prospects,” Reviews in Physics, p.100028, 2019.

[5] N. Elsokkary, F. S. Khan, D. La Torre, T. S. Humble, and J. Gottlieb, “Financial portfolio management using d-wave quantum optimizer: The case of Abu Dhabi securities exchange,” Oak RidgeNational Lab. (ORNL), Oak Ridge, TN (United States), Tech. Rep., 2017.

[6] K. Tadashi, N. Hidetoshi, “Study of optimization problems by quantum annealing [ph. d. thesis],” Tokyo: Department of Physics, Tokyo Institute of Technology, Dec.1998.

[7] H. M. Waidyasooriya, Y. Araki, and M. Hariyama, “Accelerator architecture for simulated quantum annealing based on resource utilization aware scheduling and its implementation using OpenCL,”, International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), pp. 336–340, Nov. 2018.

[8] M. Suzuki, S. Miyashita, and A. Kuroda, “Monte Carlo simulation of quantum spin systems. I,” Progress of Theoretical Physics, vol. 58, no. 5, pp. 1377–1387, 1977.

[9] Takuya Okuyama, Masato Hayashi, and Masanao Yamaoka, “An Ising computer based on simulated quantum annealing by path integral Monte Carlo method,” Proc. of IEEE International Conference on Rebooting Computing, pp. 1-6, Nov. 2017.

[10] Hasitha Muthumala Waidyasooriya, Yusuke Araki, and Masanori Hariyama, “Accelerator Architecture for Simulated Quantum Annealing Based on Resource Utilization Aware Scheduling and its Implementation Using OpenCL,” Proc. of 2018 International Workshop on Smart Info-Media Systems in Asia (SISA 2018), pp. 335-340, Dec. 2018.

[11] H. M. Waidyasooriya and M. Hariyama, ”Highly-Parallel FPGA Accelerator for Simulated Quantum Annealing,” in IEEE Transactions on Emerging Topics in Computing, vol. 9, no. 4, pp. 2019-2029, 1 Oct.-Dec. 2021.