The Journal of Supercomputing
https://doi.org/10.1007/s11227-021-03859-5

Design space exploration for an FPGA-based quantum annealing simulator with interaction-coefficient-generators

Chia-Yin Liu[1] · Hasitha Muthumala Waidyasooriya[1] · Masanori Hariyama[1]

Accepted: 30 April 2021

Abstract
Quantum annealing simulation attracts much attention recently for solving combinatorial optimization problems. FPGA acceleration is a promising way to reduce the huge processing time in quantum annealing simulations. However, the performance of FPGA accelerators is often restricted by the small external memory bandwidth. To solve this problem, we propose a data-transfer-bottleneck-less FPGA-based accelerator for quantum annealing simulation. The proposed architecture is implemented on an FPGA and achieved up to 179 times speed-up compared to single-core CPU implementation. The proposed accelerator is two times faster compared to previous FPGA accelerators, and process up to 262,144 spins, which is not possible in any existing FPGA accelerators due to limited external memory capacity.

Keywords Simulated quantum annealing · OpenCL for FPGA · Quantum Monte Carlo simulation · FPGA accelerator

1 Introduction

Combinatorial optimization is to find the optimal solution from a finite set of possible ones and used in many fields, for example, the traveling salesman problem [1], traffic flow problem [2], financial issue [3, 4], graph coloring [5], and graph partitioning [6]. Quantum annealing (QA) is an efficient way for solving combinatorial optimization [7, 8]. It is a meta-heuristic which relates simulated annealing and

✉ Hasitha Muthumala Waidyasooriya
hasitha@tohoku.ac.jp; hasitha@ecei.tohoku.ac.jp

Chia-Yin Liu
liu.chia-yin.r8@dc.tohoku.ac.jp

Masanori Hariyama
hariyama@tohoku.ac.jp

1 Graduate School of Information Sciences, Tohoku University, 6-3-09, Aramaki-Aza-Aoba, Aoba, Sendai, Miyagi 980-8579, Japan

Published online: 18 May 2021

quantum systems to find the minimum or maximum solution in discrete search space using quantum tunneling phenomena. There are quantum annealers such as D-Wave, a commercial quantum annealer performing computations relied on quantum effects is used in many fields [9]. It solves problems by a multi-qubit QPU (quantum processing unit) using quantum bits and operating quantum annealing in an adiabatic system, which proves higher speed than the conventional computer.

However, the problem size that can be handled by the existing quantum annealers is small due to the physical limitation of the small number of quantum bits. Even the D-Wave 2000Q [10], a state-of-the-art quantum annealer contains only 2048 nodes and 5600 coupler that is not enough to solve practical problems. Except for small problems that can be directly operated, many problems need to be mapped through minor embedding [11].

To apply the quantum annealing to a large practical problem, quantum annealing simulation (QAS) on conventional computers is important. Quantum annealing simulation can be implemented using Quantum Monte Carlo (QMC) simulation by mapping the problem to a transverse Ising model [7, 12, 13]. Quantum annealing simulation using CPUs takes a huge amount of processing time. To improve the efficiency, we process quantum annealing simulation in FPGAs.

FPGA is a low-power and reprogrammable architecture consisting of logic circuit, memory modules, and DSP units. It takes the advantage of parallel processing, flexible designs, and energy-efficient computation, which is suitable for designing custom accelerators, especially for the large-scale operations. The basic CPU-FPGA heterogeneous processing system is represented by Fig. 1. The work in [14] is one of the most recent FPGA accelerators for QAS. It exploits the spatial and temporal parallelism of the QMC algorithm to increase the processing speed drastically compared to that of CPUs. However, its parallelism is restricted by the external memory access bandwidth.

We proposed the basic idea of FPGA-based "data-transfer-bottleneck-less architecture" in [15]. It uses coefficient generators to generate the interaction coefficients internally so that external memory access is reduced drastically. Since all the coefficient data are no longer stored, the problem size is not limited by the external memory capacity. As a result, large-size problems such as 262,144 spins can be implemented, which is not possible in previous FPGA accelerators. However, it uses more internal memory that can limit the degree of parallelism. In this paper, we explore the design space of FPGAs to optimize the "data-transfer-bottleneck-less architecture." We demonstrate that the processing speed can be increased drastically by increasing the spatial parallelism. We conducted a comprehensive evaluation and

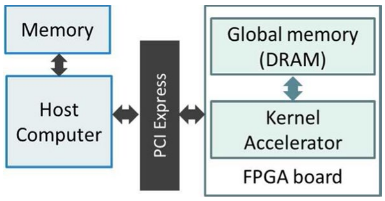


Fig. 1 The CPU-FPGA heterogeneous processing system

discussed the relationship between the FPGA hardware resources and the design parameters such as the number of spins, the number of Trotters, and the degree of spatial parallelism. The proposed FPGA-based architecture can achieve almost two times speed-up compared to the previous one [14]. The maximum degree of spatial parallelism of our architecture is 64, which is four times larger compared to that of the previous FPGA-based QAS. Moreover, the problem size is increased up to 262,144 spins, which is eight times larger than that under the previous work [14]. Compared to the CPU-based QAS, the proposed architecture gains at most 179 times speed-up.

2 Previous work

2.1 Ising model

Quantum annealing simulation can be implemented using Quantum Monte Carlo (QMC) simulation by mapping the problem to a transverse Ising model [7, 12]. Ising model is a mathematical model for a spin system. Figure 2 shows a lattice of spins in an Ising model where each spin in either spin-up or spin-down interacts with other spins [7, 12].

The additional "Trotter" dimension is used to describe the $(d + 1)$-dimensional classic Ising model [16]. As shown in Fig. 3, the Ising model with the "Trotter" dimension consisting of many Trotter slices uses multiple replicas of the original lattice. Each spin interacts with the neighbors in the same trotter and also in the trotter direction, including up and down layers. We use M to denote the number of Trotters.

To simulate quantum annealing, the combinatorial optimization problem is mapped into an Ising model and represented by the Hamiltonian function [17]. The energy function of the interaction of spins is estimated by Eq. (1). Each spin is denoted by s_i including $s_i = +1$ (spin-up) and $s_i = -1$ (spin-down). The local magnetic field is represented by h_i. We use J_{ij} to represent the interaction between every

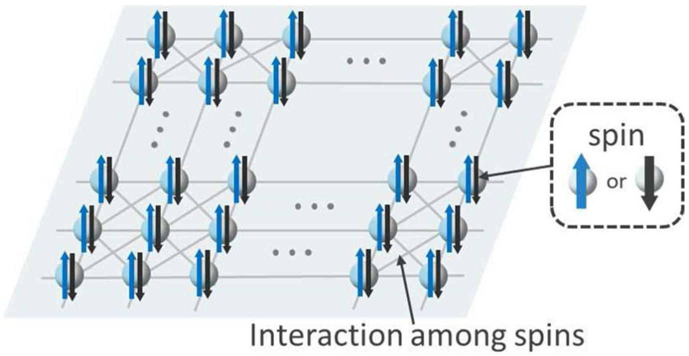


Fig. 2 A lattice of spins in an Ising model

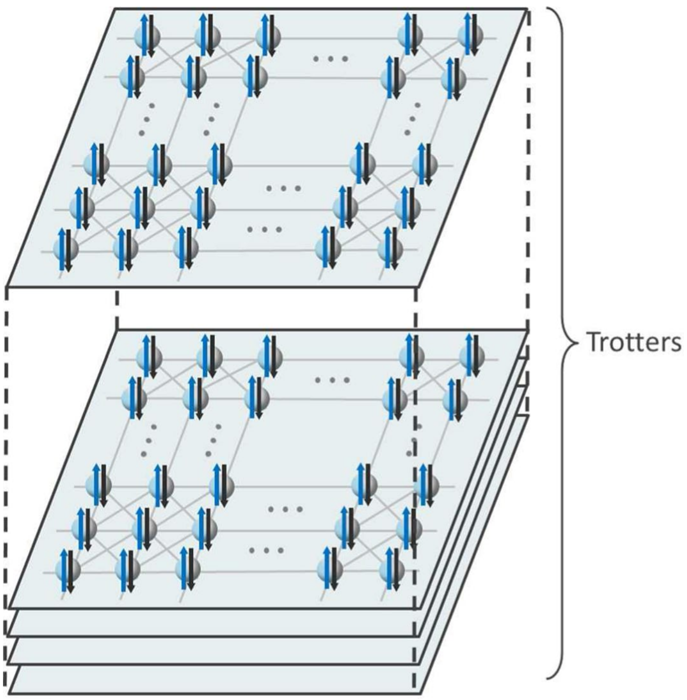


Fig. 3 A Ising model consisting of many Trotter slices

two spins in the model. Spins of all Trotter slices are related to others and probabilistically changed. As the transverse field decreases, the energy of the whole system reduces toward the ground state, and the spin values are finally decided.

$$H = -\sum_{i,j} J_{ij}s_is_j - \sum_{i} h_is_i. \tag{1}$$

2.2 Accelerating quantum annealing simulation

Accelerators for quantum annealing simulation using ASIC (application-specific integrated circuit) are proposed in [18, 19]. However, they cause high development costs. Also, the problem size and the number of interactions cannot be adjusted after fabrication. Therefore, FPGA and GPU accelerators that are more flexible for design are important. Studies in [20–22] use GPUs, and that in [14, 23–26] use FPGAs to design acceleration for QAS. The GPU-based quantum annealing simulators improve the processing time but cause higher power consumption [20–22]. Therefore, we discuss the FPGA-based accelerators for better power efficiency.

There are several types of spin models used in FPGA-based acceleration. The models used in [23–25] are not fully connected in which spins only interact with a few neighbors. Such architectures are not efficient to use with dense spin models where one spin interacts with many other spins. An FPGA-based accelerator for quantum annealing simulation proposed in [14] implements parallel computation on a fully connected spin model. Figure 4 shows the architecture discussed in [14]. It consists of multiple CUs (computing unis) belonging to multiple Trotters. FIFOs are used to transfer the interaction coefficients from one Trotter to the next. The energy of a spin is calculated in parallel using multiple CUs. The number of CUs per Trotter corresponds to the degree of spatial parallelism. The energies of the spins belonging to different Trotters are computed in a temporal parallel manner. To increase the processing speed of the accelerator proposed in [14], we have to increase the number of CUs in each Trotter. When the number of CUs increases, more data should be accessed in parallel and the required memory access bandwidth increases. However, it has to load data from global memory frequently and the access bandwidth between global memory and computing units is limited. As a result, the degree of spatial parallelism in this architecture is limited to 16 due to facing a data transfer bottleneck.

We propose the basic idea of “data-transfer-bottleneck-less architecture” in [15]. The data transfer bottlenecks caused by the global memory access are eliminated by generating interaction coefficients internally in parallel. Since interaction coefficients are not stored in the external memory, the required external memory capacity does not increase exponentially with the number of spins. As a result, we can implement large spin models up to 262,144 spins, which is not possible in the

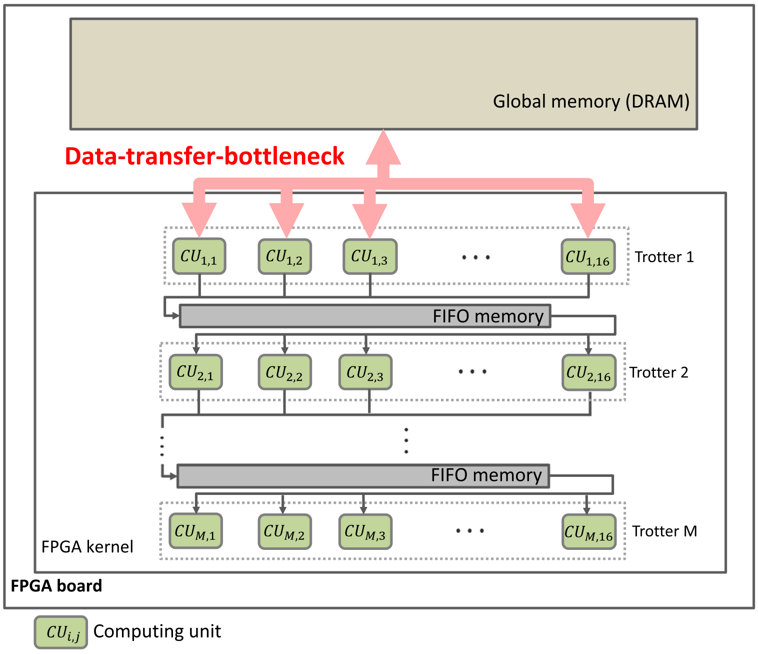

Global memory (DRAM)
Data-transfer-bottleneck
$CU_{1,1}$ $CU_{1,2}$ $CU_{1,3}$ ··· $CU_{1,16}$
Trotter 1
FIFO memory
$CU_{2,1}$ $CU_{2,2}$ $CU_{2,3}$ ··· $CU_{2,16}$
Trotter 2
FIFO memory
$CU_{M,1}$ $CU_{M,2}$ $CU_{M,3}$ ··· $CU_{M,16}$
Trotter M
FPGA kernel
FPGA board
$CU_{i,j}$ Computing unit

Fig. 4 The architecture facing data transfer bottleneck and limiting problem size

previous FPGA accelerators due to limited external memory capacity. However, this method could increase the internal memory and computational resources. Such problems could potentially decrease the problem size or processing speed, and this has not been discussed in [15]. In this paper, we explore the design space for various scaling factors of the proposed accelerator architecture. We also discuss how to utilize spatial and temporal parallelism efficiently to increase the number of spin when the hardware resources are limited. We found out that the spatial parallelism can be increased up to four times of that of [15].

3 Data-transfer-bottleneck-less FPGA accelerator architecture

3.1 Interaction coefficient generation

As discussed in Sect. 2.2, the processing speed of the FPGA-based accelerator proposed in [14] is restricted by the external memory access bandwidth. To solve this problem, we propose a method to generate the coefficient matrix J inside the FPGA, without storing the whole matrix in the external memory. For many combinatorial optimization problems, the input data required to generate the coefficient matrix J are extremely smaller than the total size of the matrix. Therefore, we can store the input data of the generator also in the local memory. As a result, we can reduce the external memory access drastically and avoid data transfer bottleneck even for large-scale problems.

Figure 5 shows the proposed FPGA accelerator architecture. It consists of a set of interaction coefficient generators and an array of computing units (CUs). An interaction coefficient generator is denoted by G_i. The set of interaction coefficient generators accesses the input data from the local memory and generates the required portion of coefficients J in parallel. The generated coefficients values J are transferred into the first Trotter and used in CUs in parallel. After finishing the computation of the current Trotter, J values are stored in FIFO memories and reused in other Trotters. The energy of the spins belonging to different Trotters is computed in a temporal parallel manner according to the method proposed in [26]. We generate and store a new portion of coefficient values in the FIFO while discarding the old values that are no longer required for further computation. Therefore, we only need a small capacity of memory in FIFOs for storing required coefficient data.

The interactions between spins differ according to the optimization problem. Therefore, the structure of the interaction coefficient generator is also decided by the problem. We explain the interaction coefficient generator of the "number partitioning problem" [27].

Number partitioning is a problem to divide a set of positive integers into two disjoint subsets A and B such that the sum of all integers of the subset A is similar as possible to the sum of all integers of the subset B. An example of number partitioning problem is shown in Fig. 6. The set {3, 2, 6, 8, 1} is divided into two

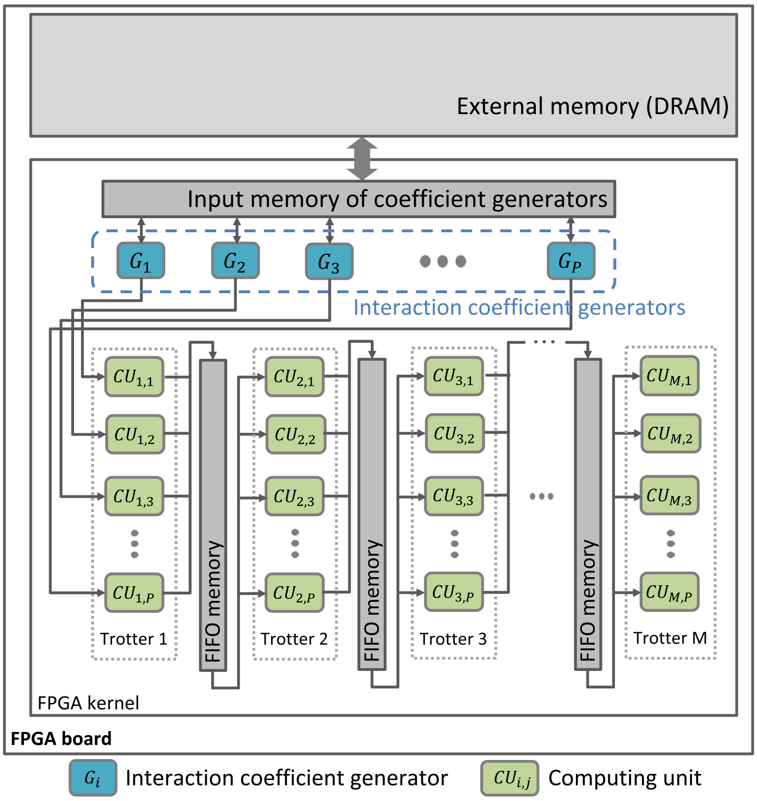
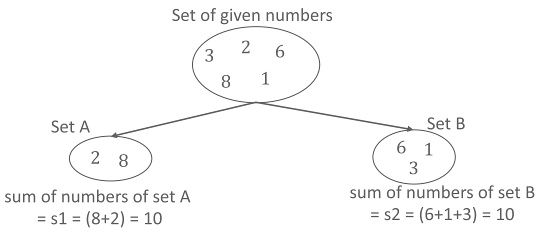


Fig. 5 Data-transfer-bottleneck-less and high computational parallelism architecture

Fig. 6 Example of a number partitioning problem



subsets A and B. The sum of integers of subset A is 10 and equals to that of the subset B.

$$H = \left(\sum_{i=1}^{N} e_i s_i \right)^2, \tag{2}$$

$$H = A \sum_{i,j=1}^{N} e_i e_j s_i s_j. \tag{3}$$

Number partitioning is an NP-complete problem. It can be mapped on to Ising model [27] as shown by Eq. (2). The set contains N positive integers denoted by e_i where $i \in 1, 2, ..., N$. Ising spin variable $s_i = \pm 1$ corresponds to integer e_i. The integers are divided into two sets according to the value of the corresponding Ising variable, that is $+1$ or -1. Therefore, the sum of $e_i s_i$ is the difference of the sums of two subsets. When the sums of two subsets are similar as possible, the energy H in Eq. (2) is minimized. We can derive Eq. (3) from the expansion of Equation (2) while describing all constant values as A. According to Eqs. (1) and (3), the interaction coefficient J_{ij} between each pair of spins i and j is $e_i \times e_j$. The size of J matrix is $N \times N$. The constant A is scaled to 1 and h_i is zero. Thus, we can design the interaction coefficient generator that computes the product of e_i and e_j, as shown in Fig. 7. When the degree of spatial parallelism is defined as P, we use P generators for parallel computing.

3.2 Increasing the degree of parallelism

The degree of computational parallelism is decided by the degree of spatial and the degree of temporal parallelism. For improving the processing speed, we have to increase both spatial and temporal parallelism. The degree of temporal parallelism is limited by the number of Trotters. Since the previous work [14] uses the maximum degree of temporal parallelism by processing all Trotters in parallel, we have to focus on the spatial parallelism to increase the processing speed further.

As shown in Fig. 5, the degree of spatial parallelism refers to the number of CUs in a Trotter. The number of interaction coefficient generators is also P, since CUs require interactions coefficients in parallel. After the interaction coefficients are generated, those are transferred to the first Trotter. Since all Trotters share the same coefficients, we need only P coefficient generators. In order to increase P, we have to

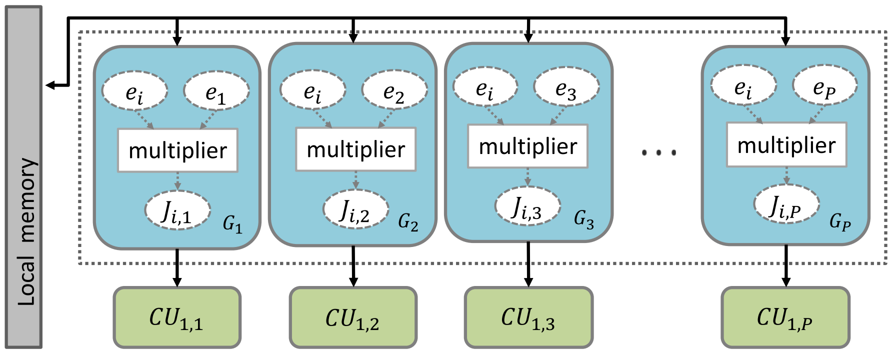


Fig. 7 Interaction coefficient generator for number partitioning problem

increase the number of coefficient generators and also the number of CUs per Trotter. Note that, the data required to generate coefficients are downloaded and stored in the internal memory. Therefore, we can increase *P* without utilizing the global memory bandwidth.

4 Evaluation

For the evaluation, we use Terasic DE10-Pro board [28] containing Intel Stratix 10 GX 1SG280HU FPGA [29]. FPGA kernels are compiled using Quartus 19.1 compiler with Intel FPGA SDK for OpenCL 19.1. The CPU is Intel Xeon Gold 6130 CPU, and the "C compiler" is gcc 7.4.0. Evaluations are done using the number partitioning problem.

4.1 Design space exploration

The proposed accelerator architecture is scalable in terms of number of spins, number of Trotters, and the degree of parallelism. The relationship between each scaling factor and FPGA resources is different. In this section, we explore the design space to optimize the accelerator architecture.

Table 1 shows the relationship between the resource utilization and the number of Trotters (the degree of temporal parallelism). In this case, the degree of spatial parallelism is a constant. When the degree of temporal parallelism is increased, we need more FIFOs to store the intermediate data between Trotters. As a result, the RAM block utilization is proportional to the number of Trotters. The DSP block utilization is also proportional to the number of Trotters, since more Trotters use more parallel computations. Although the logic block utilization is also increased with the number of Trotters, this increase is less significant than that of the other resources.

Table 2 shows the relationship between the resource utilization and the degree of spatial parallelism under the condition of the constant number of Trotters. Both DSP and RAM blocks are proportional to the degree of spatial parallelism. More DSP blocks are required to do more computations. RAM blocks are also duplicated to

Table 1 Resource utilization while implementing different numbers of spins and different numbers of Trotters. (The degrees of parallelism is set by 16)

Number of spins	Number of Trotters	Resource utilization		
(*N*)	(*M*)	Logic	DSP blocks	RAM blocks
16,384	4	162,338 (17%)	120 (2%)	883 (8%)
	8	200,115 (21%)	224 (4%)	1354 (12%)
	16	284,052 (30%)	432 (8%)	2358 (20%)
32,768	4	163,931 (18%)	120 (2%)	1345 (11%)
	8	204,429 (22%)	224 (4%)	2211 (19%)
	16	290,112 (31%)	432 (8%)	4023 (34%)

Table 2 Resource utilization while implementing different numbers of spins at different degrees of parallelism

Number of spins	Degree of parallelism	Resource utilization		
(N)	(P)	Logic	DSP blocks	RAM blocks
8,192	16	160,365 (17%)	120 (2%)	653 (6%)
	32	211,304 (23%)	236 (4%)	876 (7%)
	64	330,200 (35%)	464 (8%)	1421 (12%)
16,384	16	162,338 (17%)	120 (2%)	883 (8%)
	32	214,027 (23%)	236 (4%)	1162 (10%)
	64	339,631 (36%)	464 (8%)	1794 (15%)
32,768	16	163,931 (18%)	120 (2%)	1345 (11%)
	32	215,967 (23%)	236 (4%)	1818 (16%)
	64	342,520 (37%)	464 (8%)	2590 (22%)
65,536	16	167,999 (18%)	120 (2%)	2249 (19%)
	32	221,987 (24%)	236 (4%)	3109 (27%)
	64	348,850 (37%)	464 (8%)	4403 (38%)
131,072	16	174,317 (19%)	120 (2%)	4055 (35%)
	32	229,517 (25%)	236 (4%)	5687 (49%)
	64	361,601 (39%)	464 (8%)	8072 (69%)
196,608	16	188,219 (20%)	120 (2%)	6211 (53%)
	32	247,848 (27%)	236 (4%)	9383 (80%)
262,144	16	185,622 (20%)	120 (2%)	7665 (65%)

(The number of Trotters is fixed to 4)

increase parallel access to the memory. However, increase in the RAM block utilization is smaller compared to that of the case in Table 1.

As shown in Tables 1 and 2, the degree of temporal parallelism has a larger impact on RAM block utilization compared to that of the degree of spatial parallelism. According to this observation, it is better to increase the degree of spatial parallelism, rather than increasing the temporal parallelism, when the RAM block utilization is a bottleneck. Since we can increase the spatial parallelism without concerning the memory access bottleneck, it is an advantage of the proposed method compared to previous work [14]. The increase in the logic and DSP utilization is extremely small despite doubling the number of spins. On the other hand, RAM blocks are increased significantly with the number of spins. This is because, RAM blocks are required to store the intermediate data between Trotters and the amount of intermediate data increases with number of spins.

Figure 8 shows the largest number of spins that are implemented on the FPGA, when the number of Trotters is 4, 8, or 16, and the degree of spatial parallelism is 16, 32, or 64. In order to increase the degree of parallelism, we have to reduce the number of spins. However, the effect on the number of spins is larger when increasing the degree of temporal parallelism, compared to increasing the degree of spatial parallelism. Therefore, increasing the temporal parallelism is preferred.

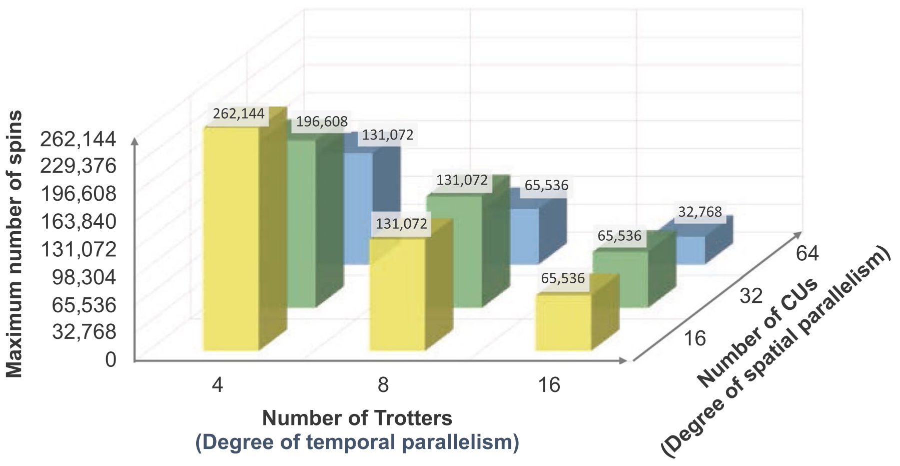


Fig. 8 The relationship among the number of spins, the degree of spatial parallelism, and the degree of temporal parallelism

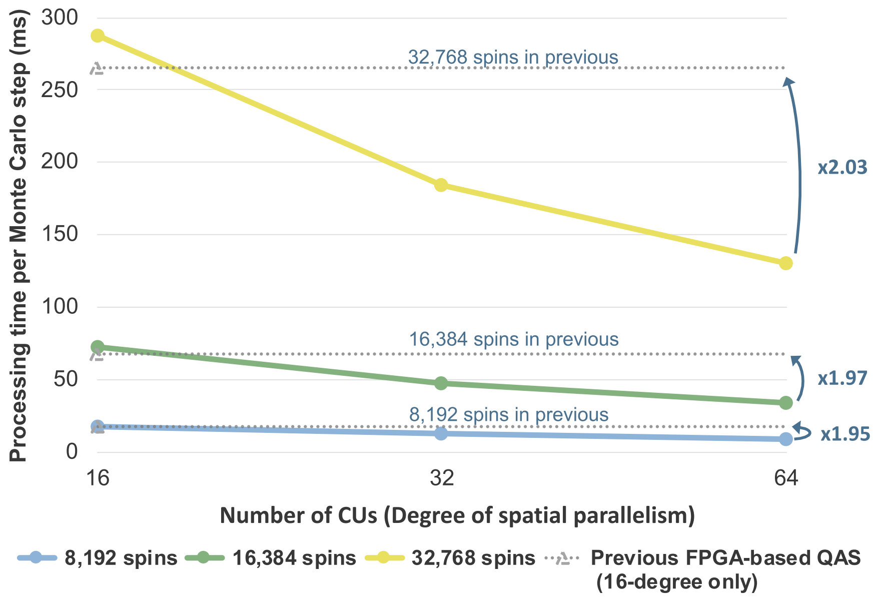
4.2 Comparison against previous work

Figure 9 shows the processing time comparison against previous work [14]. Figure 9a and b shows the processing times when the number of Trotters is 4 and 8, respectively. The degree of spatial parallelism of the previous work [14] is 16 and cannot be increased due to data transfer bottlenecks. The degree of spatial parallelism of the proposed method can be increased up to 64. However, the proposed method use internal memory to store interactions coefficients, so that the clock frequency is reduced.

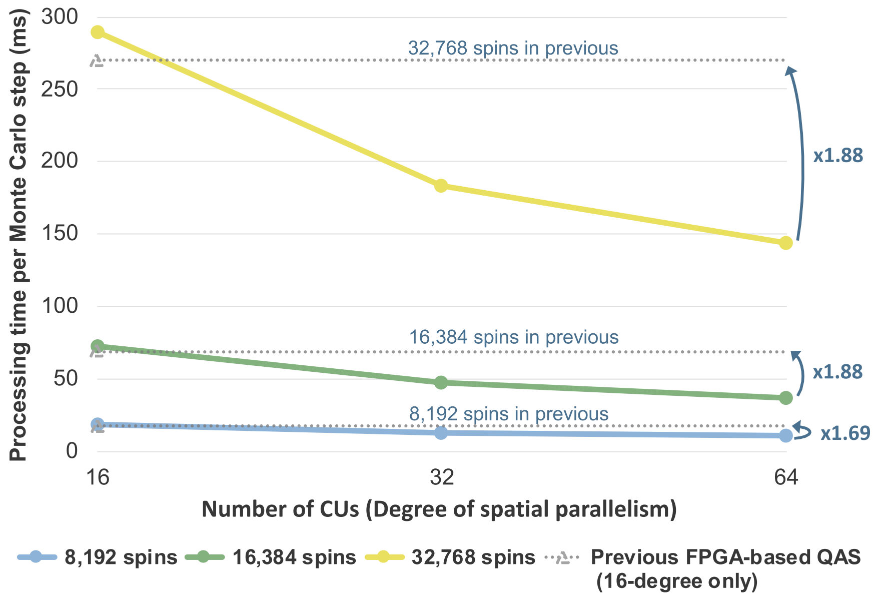
When the degree of spatial parallelism is increased, we achieved up to two times processing speed increase in the proposed method. Similar processing speed gain is achieved for different number of spins. Theoretically, the slope should be the same for 16~32 CUs and also 32~64 CUs, since the processing speed is linearly relative to the degree of spatial parallelism. However, we have seen a steeper slope for 16~32 CUs compared to that of 32~64 CUs. The reason for this is the difference of the clock frequency. We use interaction generators and internal memory to store the generated data temporally. When the degree of spatial parallelism increases, more internal memory and DSPs are used. As a result, the clock frequency is decreased.

When both methods use the same 16-degree spatial parallelism, processing speed of the proposed method is slightly smaller compared to that of the previous work. The reason is also the difference of the clock frequency, where the proposed method has a relatively smaller frequency due to more resource usage.

Figure 10 shows the number of spins of the proposed and the previous work [14]. For a fair comparison, we use the same Stratix 10 FPGA for both accelerators and use the same degrees of spatial and temporal parallelisms. The maximum number of spins in the previous work [14] is only 32,768 while the proposed accelerator can process up to 262,144 spins. When the number of spins is N, the



(a) The number of Trotters is 4.



(b) The number of Trotters is 8.

Fig. 9 Comparison of processing time against previous FPGA-based QAS. (The number of Trotters is fixed

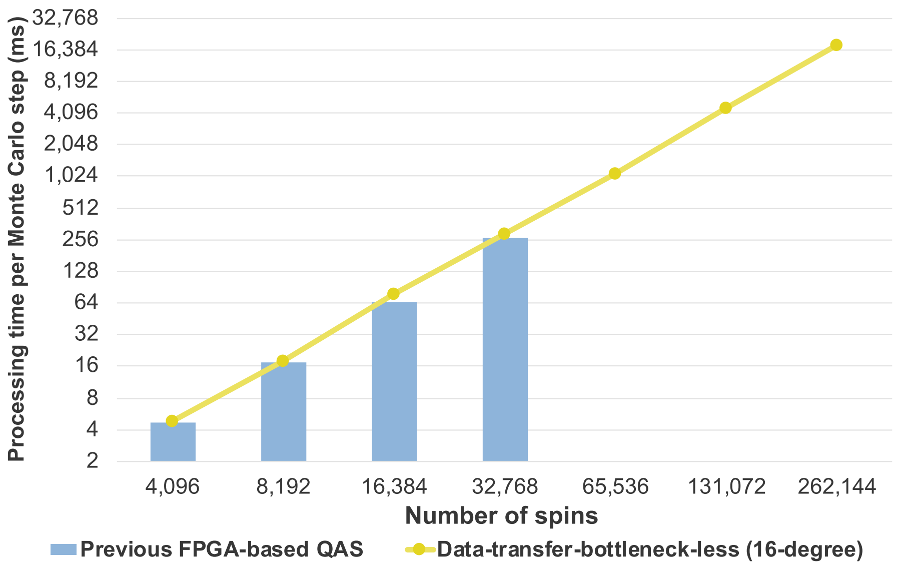
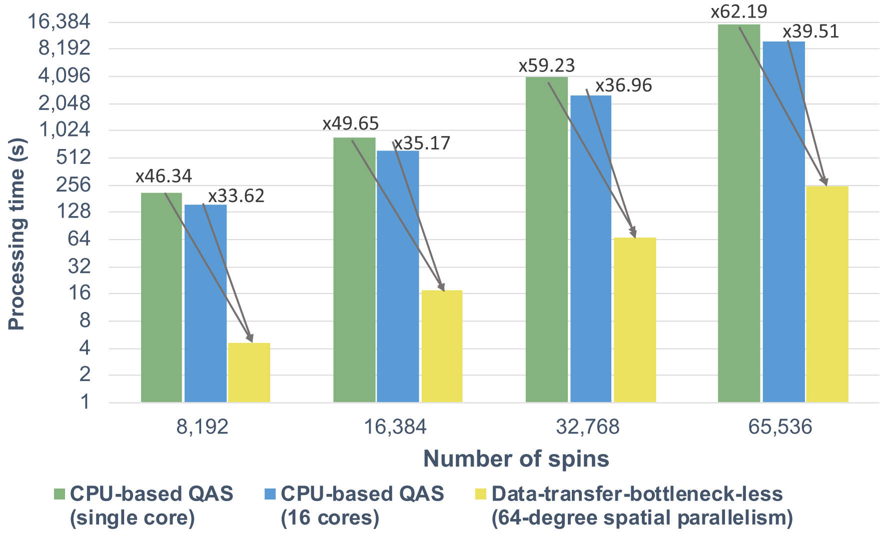


Fig. 10 Comparison of number of spins of the proposed and previous [14] work

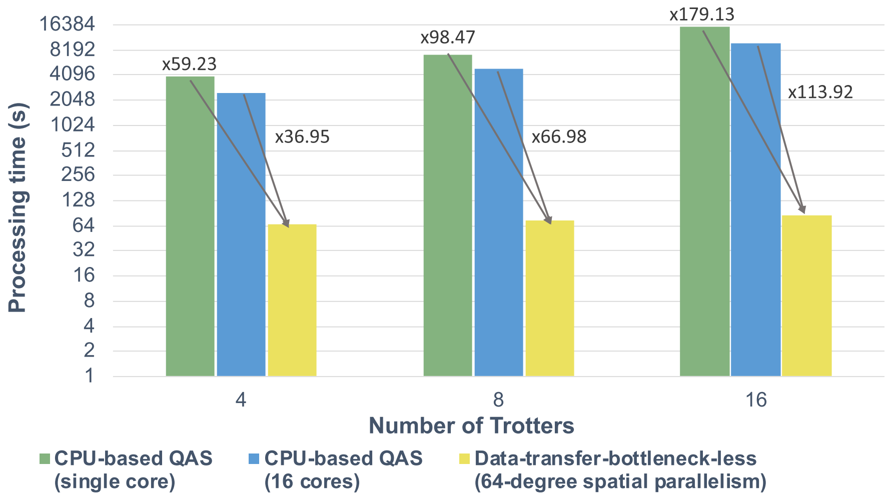
size of the coefficient matrix J is $N \times N$, so that the matrix size increases exponentially with the number of spins. Since the previous work [14] stores matrix J in the external memory, the number of spins is limited by the external memory capacity. In the proposed architecture, matrix J is generated internally, so that the spin size does not depend on the external memory capacity. We can also see that there is no noticeable difference in processing times for both accelerators. As a result, we can say that the proposed method significantly increases the number of spins without a processing speed penalty.

Figure 11 shows the comparison against CPU implementation. Figure 11a shows the processing time required to process different number of spins. The number of Trotters is a constant for all implementations. The degree of spatial parallelism in the proposed method is 64. The proposed architecture can achieve at least 46.34 times and at most 62.19 times speed-up compared to single-core CPU implementation. The speed-up compared to multicore CPU implementation using spatial parallelism is from 33.6~39.5 times. We can see a better speed-up for large spin models. One reason for this could be more efficient cache utilization for small spin models in the CPU. Therefore, the proposed accelerator architecture is more productive for large-scale problems.

Figure 11b shows the processing time required to process different number of Trotters. The number of spins is a constant for all implementations. The degree of spatial parallelism in the proposed method is 64. The proposed architecture achieves at least 59.23 times and at most 179.13 times speed-up compared to single-core CPU implementation. The speed-up compared to multicore CPU implementation is from 33.9~113.9 times. Since all Trotters are processed in parallel in



(a) Processing time vs. number of spins (The number of Trotters is fixed to 4).



(b) Processing time vs. number of Trotters. (The number of spins is set as 32,768).

Fig. 11 Comparison of processing time against CPU

the proposed method (similar to the previous work [14]), the speed-up increases with the Trotter size. Although there is no specific rule to define the Trotter size of an Ising model, a larger Trotter size is usually required for finding the solution in smaller steps. Therefore, the proposed accelerator architecture is more productive for larger number of Trotters.

When using a CPU for quantum annealing simulation, it is not possible to always achieve faster processing speed using more computing power (or more cores). Achieving Trotter-level temporal parallel processing on a CPU is an extremely hard problem. As explained in previous work [14], we can only perform parallel processing on independent operations. When there is a data dependency, we have to wait until the previous Trotter is computed. In CPU, we have to use explicit synchronization at very small granularity, so that the processing speed can be adversely affected by this overhead. On the other hand, the proposed method can predetermine how many clock cycles required for the previous operation to be completed, and we can start the next operation exactly after this number of clock cycles. It is possible to implement spatial parallelism in a CPU using parallel reduction. However, this does not give linear processing speed increase with the number of cores. Possible reasons for this could be the synchronization overhead and memory access bottlenecks. Even using the computing power of all 16 cores, we could only increase the processing speed by nearly two times that of single-core implementation.

5 Conclusion

We propose an FPGA-based accelerator architecture for quantum annealing simulation that solves the data transfer bottleneck between the external memory and the FPGA. We implement the number partitioning problem on the proposed architecture and compared the performance results with the CPU-based QAS and a previous FPGA-based accelerator for QAS. The proposed method outperforms both CPU and the previous work [14] in terms of processing time. The speed-up is achieved by using significantly more parallel computations compared to that of CPU. CPU can only use spatial parallelism, while the proposed method uses both spatial and temporal parallelism. The proposed architecture has achieved up to 179 times speed-up compared to a single-core CPU implementation. The speed-up is larger for large number of spins and Trotters. This shows that the proposed data-transfer-bottleneck-less architecture is more productive for large-size problems in terms of processing speed.

The proposed method uses four times more spatial parallel computations compared to that of the previous work [14]. In previous work, the spatial parallelism is restricted by the memory access bandwidth of the external memory. However, in the proposed method, the coefficient data are generated inside the FPGA, without accessing the external memory. Therefore, we can increase the degree of spatial parallelism until the resource constraint is met. The processing speed of the proposed architecture is almost twice larger than the previous work [14]. Since coefficient data are not stored in the external memory, the achievable number of spins is independent of the external memory capacity. As a result, we increase the number of spins up to eight times compared to that of the previous work [14]. This also shows that the proposed data-transfer-bottleneck-less architecture can be used for large-size problems.

Acknowledgements This research is partly supported by MEXT KAKENHI, grant numbers 19K11998 and 20H04197.

References

1. Lawler Eugene L (1985) The travelling salesman problem: a guided tour of combinatorial optimization. Wiley, New York
2. Neukart Florian, Compostella Gabriele, Seidel Christian, von Dollen David, Yarkoni Sheir, Parney Bob (2017) Traffic flow optimization using a quantum annealer. Frontiers in ICT 4:29–29
3. Orus Roman, Mugel Samuel, Lizaso Enrique (2019) Quantum computing for finance: Overview and prospects. Rev Phys 4:100028
4. Elsokkary N, Khan FS, La Torre D, Humble TS, Gottlieb J (2017) Financial portfolio management using d-wave quantum optimizer: The case of abu dhabi securities exchange. Technical report, Oak Ridge National Lab.(ORNL), Oak Ridge, TN (United States),
5. Titiloye Olawale, Crispin Alan (2011) Quantum annealing of the graph coloring problem. Discret Optim 8(2):376–384
6. Ushijima H, Negre CFA, Mniszewski SM (2017) Graph partitioning using quantum annealing on the d-wave system. In: Proceedings of the Second International Workshop on Post Moores Era Supercomputing, pp 22–29
7. Kadowaki T, Nishimori H (1998) Quantum annealing in the transverse Ising model. Phys Rev E 58:5355–5363
8. Tadashi K (June 1998) Study of optimization problems by quantum annealing. PhD thesis, Department of Physics, Tokyo Institute of Technology
9. D-wave (2019) D-wave systems. https://www.dwavesys.com
10. D-wave (2019) The D-Wave 2000Q Quantum Computer Technology. https://www.dwavesys.com/d-wave-two-system
11. Zaribafiyan Arman, Marchand Dominic J, Rezaei Seyed Saeed Changiz (2017) Systematic and deterministic graph minor embedding for cartesian products of graphs. Q Inf Process 16(5):1–26
12. Suzuki M, Miyashita S, Kuroda A (1977) Monte carlo simulation of quantum spin systems i. Prog Theor Phys 58(5):1377–1387
13. Crosson E, Harrow AW (Oct 2016) Simulated quantum annealing can be exponentially faster than classical simulated annealing. In: 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS), pp 714–723
14. Waidyasooriya H, Hariyama M Highly-parallel FPGA accelerator for simulated quantum annealing. In: IEEE transactions on emerging topics in computing. https://doi.org/10.1109/TETC.2019.2957177
15. Liu C, Waidyasooriya HM, Hariyama M (2019) Data-transfer-bottleneck-less architecture for fpga-based quantum annealing simulation. In: 2019 Seventh International Symposium on Computing and Networking (CANDAR), pp 164–170
16. Suzuki M (1976) Relationship between d-dimensional quantal spin systems and (d+1)-dimensional ising systems: Equivalence, critical exponents and systematic approximants of the partition function and spin correlations. Prog Theor Phys 56(5):1454–1469
17. Stinchcombe R B (1973) Ising model in a transverse field. i. basic theory. J Phys C: Solid State Phys 6(15):2459–2483
18. Fujitsu (2019) Fujitsu Digital Annealer. https://www.fujitsu.com/global/services/business-services/digital-annealer/what-is-digital-annealer/index.html,
19. Yamaoka M, Yoshimura C, Hayashi M, Okuyama T, Aoki H, Mizuno H (2016) A 20k-spin ising chip to solve combinatorial optimization problems with cmos annealing. IEEE J Solid-State Circuits 51(1):303–309
20. Weigel M (2012) Performance potential for simulating spin models on gpu. J Comput Phys 231(8):3064–3082

21. Cook C, Zhao H, Sato T, Hiromoto M, Tan SX-D (2018) Gpu based parallel ising computing for combinatorial optimization problems in vlsi physical design. arXiv preprint arXiv:1807.10750,
22. Waidyasooriya HM, Hariyama M (2020) A gpu-based quantum annealing simulator for fully-connected ising models utilizing spatial and temporal parallelism. IEEE Access 8:67929–67939
23. Okuyama T, Hayashi M, Yamaoka M (2017) An ising computer based on simulated quantum annealing by path integral monte carlo method. In: 2017 IEEE International Conference on Rebooting Computing (ICRC), pp 1–6
24. Okuyama T, Hayashi M, Yamaoka M (Nov 2017) An ising computer based on simulated quantum annealing by path integral monte carlo method. In: 2017 IEEE International Conference on Rebooting Computing (ICRC), pp 1–6
25. Waidyasooriya HM, Araki Y, Hariyama M (2018) Accelerator architecture for simulated quantum annealing based on resource-utilization-aware scheduling and its implementation using opencl. In: 2018 International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS), pp 335–340
26. Waidyasooriya HM, Hariyama M, Miyama MJ, Ohzeki M (2019) OpenCL-based design of an FPGA accelerator for quantum annealing simulation. J Supercomput 75(8):5019–5039
27. Lucas Andrew (2014) Ising formulations of many np problems. Front Phys 2:5
28. Terasic (2018) Terasic DE10-Pro. https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=248&No=1144,
29. Intel (2018) Intel Stratix 10 FPGAs. https://www.intel.com/content/www/us/en/products/programmable/fpga/stratix-10.html,