IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 7 February 2024, accepted 12 March 2024, date of publication 21 March 2024, date of current version 28 March 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3380479

APPLIED RESEARCH

A GPU-Based Ising Machine With a Multi-Spin-Flip Capability for Constrained Combinatorial Optimization

SATORU JIMBO[1], (Graduate Student Member, IEEE), TATSUHIKO SHIRAI[2], (Member, IEEE), NOZOMU TOGAWA[2], (Member, IEEE), MASATO MOTOMURA[1], (Fellow, IEEE), AND KAZUSHI KAWAMURA[1], (Member, IEEE)

[1]Tokyo Institute of Technology, Yokohama 226-8502, Japan
[2]Department of Computer Science and Communications Engineering, Waseda University, Tokyo 169-8555, Japan

Corresponding author: Satoru Jimbo (jimbo.satoru@artic.iir.titech.ac.jp)

This work was supported in part by JST CREST JPMJCR18K3, in part by JST CREST JPMJCR19K4, and in part by JSPS KAKENHI under Grant JP23H05489.

ABSTRACT Ising machines are domain-specific computers that solve combinatorial optimization problems (COPs). They utilize an Ising model to represent a COP and search for the optimal spin configuration of the Ising model to solve the COP. Most Ising machines are based on simulated annealing (SA) and update the spin configuration according to single-spin-flip Markov Chain Monte Carlo methods. However, supporting a multi-spin flip is important to enhance the search performance of SA-based Ising machines when constrained COPs are solved. In this paper, we extend the merge method, which was introduced to provide a multi-spin-flip capability for SA-based Ising machines, and formulate it as a series of matrix multiplications that can be executed on a graphics processing unit (GPU) efficiently. We then construct a GPU-based Ising machine that implements the GPU-oriented merge method together with an extended SA algorithm. We finally demonstrate its superiority over the state-of-the-art GPU-based Ising machine for quadratic knapsack problems.

INDEX TERMS Constrained combinatorial optimization, graphics processing unit (GPU), Ising machine, Ising model, multi-spin flip, simulated annealing (SA).

I. INTRODUCTION

Ising machines [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12], [13], [14] have attracted attention in various industry fields as efficient solvers for combinatorial optimization problems (COPs). They utilize an Ising model to represent COPs [15], [16], [17]. The Ising model has a system energy called Hamiltonian as a function of binary variables called spins; each spin takes either $+1$ or -1. Solving a COP corresponds to finding the ground state (i.e., the optimal spin configuration that minimizes the system energy).

The associate editor coordinating the review of this manuscript and approving it for publication was Shih-Wei Lin.

Various Ising machines [3], [4], [5], [6], [7], [8], [9], [10], [11] have been developed so far. Most of them are based on simulated annealing (SA) that updates the spin configuration according to single-spin-flip Markov Chain Monte Carlo (MCMC) methods; i.e, a next candidate spin configuration is generated by a single-spin-flip operation that changes a spin value from $+1$ to -1 or from -1 to $+1$. SA-based Ising machines implement the single-spin-flip operations on application specific integrated circuit (ASIC) [3], [9] or graphics processing unit (GPU) [11] to accelerate the ground-state search.

However, the single-spin-flip operations are not suitable for solving constrained COPs such as knapsack problem,

traveling salesman problem, etc. When a constrained COP is converted into an Ising model, the Hamiltonian is constructed such that the feasible solutions (FSs), i.e., the solutions satisfying the constraints, have lower energy than the infeasible solutions (IFSs). Since a multi-spin flip is always necessary to make transitions among FSs, passing IFSs in the search process is inevitable as long as single-spin-flip MCMC methods are adopted. Nevertheless, making a transition to an IFS causes significant energy increase, and hence is difficult to be accepted in the latter half of annealing. Therefore, enabling direct transitions among FSs without passing IFSs should be a mandatory requirement to enhance the search performance of SA-based Ising machines.

Some studies have introduced a multi-spin-flip operation into Ising-machine hardware; such Ising machines are classified into two types. The first type [10] updates a specified set of spins simultaneously and induces a transition within the subspace of FSs. However, introducing this mechanism into Ising-machine hardware makes it specialized for some COPs, and hence limits the applicability. The second type [4], [6] updates the spin configuration according to multi-spin-flip MCMC methods. However, this type of Ising machines is difficult to induce a direct transition between two FSs since the methods decide to flip each spin according to the energy difference caused by its single flip. For these reasons, it is challenging and important to develop an Ising machine with a multi-spin-flip capability that meets the three requirements below:

(i) The Ising machine does not require any operations specialized for limited COPs.
(ii) The Ising machine decides to flip multiple spins according to the energy difference caused by the multi-spin flip.
(iii) The Ising machine is efficiently realized on parallel computing platforms such as ASIC or GPU.

In recent years, the merge method [18] was proposed to provide a multi-spin-flip capability for SA-based Ising machines. It can engineer a multi-spin flip using single-spin-flip MCMC methods by temporarily generating Hamiltonian H' that is deformed from the original Hamiltonian H. For example, when a single spin i in H' describes multiple spins 1, 2, and 3 in H, a single-spin flip of spin i is equivalent to the three-spin flip; i.e., their spin values σ_1, σ_2, and σ_3 are simultaneously changed to $-\sigma_1$, $-\sigma_2$, and $-\sigma_3$, respectively. Note that the merge method meets requirements (i) and (ii) since H' keeps the Ising-model format and the single-spin flip in H' is evaluated by the energy difference of the corresponding multi-spin flip in H. However, requirement (iii) is not met since it sequentially constructs H', and thus is unsuitable for the implementation onto parallel computing platforms. Furthermore, the previous work implemented this method on a CPU outside an Ising machine, which incurs a large computation cost for data transfer between the CPU and the Ising machine. For truly evaluating the benefit of the merge method, integrating it with Ising-machine hardware is necessary.

This paper proposes a GPU-based Ising machine with a GPU-oriented merge method that meets all the requirements (i), (ii), and (iii). GPU is a powerful computing platform to realize Ising machines [6], [11], [12], [13]. The main contributions of this paper are summarized as follows:

- We propose a GPU-oriented merge method. Its main body is a series of matrix multiplications, which can be executed on a GPU efficiently. In the proposed merge method, we newly introduce a merging matrix that gives the relation between H and H'.
- We extend the proposed merge method for replica parallel execution. The merging matrix is decomposed into two matrices: one is shared among all replicas and the other is a diagonal matrix (i.e., unnecessary to store the off-diagonal elements). As a result, we can save the memory to store multiple merging matrices.
- We develop a GPU-based Ising machine that integrates the proposed merge method into an extended SA algorithm. The experimental results for quadratic knapsack problems (QKPs) demonstrate its superiority over the state-of-the-art GPU-based Ising machine.

The rest of this paper is organized as follows. In Section II, we review the merge method. In Section III, we first show an obstacle to implement the merge method on a GPU and then propose a GPU-oriented merge method. The proposed merge method is further extended in Section IV for replica parallel execution on a GPU. In Section V, we show the experimental results for QKPs and verify the effectiveness of our GPU-based Ising machine with the proposed merge method. This paper is finally wrapped up in Section VI.

II. BACKGROUND KNOWLEDGE

This section introduces SA-based Ising machines utilized for solving COPs. Then, we review the merge method to improve the search performance of SA-based Ising machines.

A. SA-BASED ISING MACHINE

When a COP is solved on an Ising machine, the Ising model provides a unified way to represent the problem. The Hamiltonian H for an N-spin Ising model is defined as a function of the spin configuration $\boldsymbol{\sigma} = \{\sigma_i | \sigma_i \in \{+1, -1\}\}_{i=1}^{N}$:

$$H(\boldsymbol{\sigma}) = -\frac{1}{2}\sum_{i\neq j} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i, \tag{1}$$

where $J_{ij} \in \mathbb{R}$ and $h_i \in \mathbb{R}$ are the interaction between spins i and j and the bias on spin i, respectively. The interactions are symmetric (i.e., $J_{ij} = J_{ji}$), and the diagonal elements are set to 0 (i.e., $J_{ii} = 0$). A set of the interactions $\boldsymbol{J} \in \mathbb{R}^{N\times N}$ and the biases $\boldsymbol{h} \in \mathbb{R}^N$ represents a COP. The spin configuration $\boldsymbol{\sigma}$ that makes the system energy (i.e., the value of H) minimum corresponds to the optimal solution of the COP. Efficient Ising-model formulations of COPs have been discussed in [15] and [16].

SA-based Ising machines [3], [9], [11] iteratively update the spin configuration according to single-spin-flip MCMC methods such that the next configuration is generated by flipping one spin. In the search process, the spin-flip probability is controlled with a hyperparameter T (> 0) called pseudo temperature; the value of T is initially set to be large for aggressively flipping spins and is gradually decreased for finally fixing the spin configuration. The finite temperature helps the spin configuration escape from local optimal solutions. However, a multi-spin flip capability is required to efficiently solve constrained COPs on SA-based Ising machines, as explained in Section I.

B. MERGE METHOD

1) OVERVIEW

The merge method [18] engineers a multi-spin flip on SA-based Ising machines by deforming the Hamiltonian H. In this method, one or more spins are merged into another spin to describe multiple spins using a single spin. When spin i is merged into spin j, a flip of spin i is synchronized with a flip of spin j, resulting in the two-spin flip. Here, spin i is called a merged spin, and spin j is called the destination of spin i; a spin not merged into another spin is called an unmerged spin. The deformed Hamiltonian H' keeps the Ising-model format (i.e., (1)) and has different interactions $\boldsymbol{J}'$ and biases $\boldsymbol{h}'$ from the original Hamiltonian H (see the specific form of $\boldsymbol{J}'$ and $\boldsymbol{h}'$ in (3) and (4)):

$$H'(\boldsymbol{\sigma}') = -\frac{1}{2}\sum_{i\neq j} J'_{ij}\sigma_i\sigma_j - \sum_i h'_i\sigma_i, \tag{2}$$

where $\boldsymbol{\sigma}'$ is a subset of $\boldsymbol{\sigma}$ and consists of unmerged spins only. Utilizing the mechanism where a single-spin flip in H' is equivalent to a multi-spin flip in H, the merge method potentially enables direct transitions among FSs.

Fig. 1 shows the annealing computation flow with the merge method. In the main loop on an Ising machine, each spin is sequentially and iteratively updated according to the flip probability calculated from the local field and the pseudo temperature. The merge method integrates merge process and demerge process into this computation flow as a sub-loop. Here, the number of iterations of the sub-loop is denoted by merge interval I_{M}. First, in the merge process, a deformed Hamiltonian H' is generated from H and $\boldsymbol{\sigma}$. Then, spins in $\boldsymbol{\sigma}'$ are updated on the Ising machine using the deformed Hamiltonian H'. Finally, in the demerge process, the complete spin configuration $\boldsymbol{\sigma}$ is recovered from the output spin configuration $\boldsymbol{\sigma}'$.

The merge process generates two vectors $\boldsymbol{m} = \{m_i\}_{i=1}^N$ and $\boldsymbol{s} = \{s_i\}_{i=1}^N$ to construct the deformed Hamiltonian H'. m_i takes a value from the set of integers $\{1, 2, \cdots, N\}$ so that $m_{m_i} = m_i$ is satisfied. Spin i is an unmerged spin if $m_i = i$ and is a merged spin if $m_i = j$ ($j \neq i$); in the latter case, spin j is the destination of spin i. s_i takes a value of $+1$ or -1 and is given by $s_i = \sigma_i\sigma_{m_i}$. When spin i is merged into spin j, s_i takes $+1$ if $\sigma_i = \sigma_j$, otherwise s_i takes -1. On the other hand,

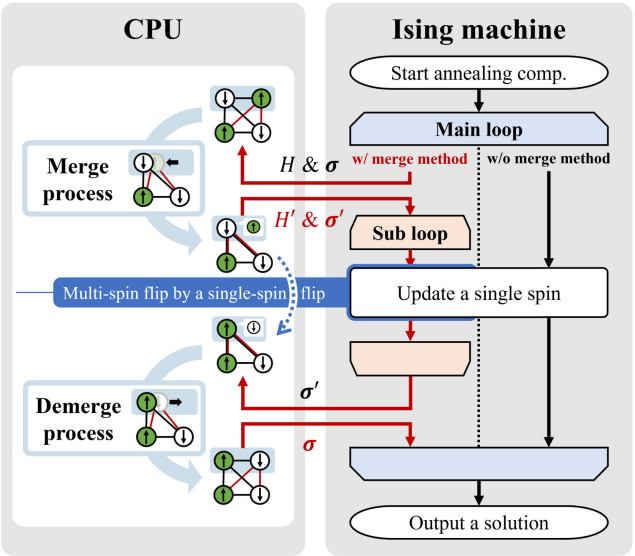


FIGURE 1. Annealing computation flow w/ and w/o the merge method [18]. Up arrow in open circle and down arrow in filled circle indicate the spin values of +1 and −1, respectively.

Algorithm 1 Merge Process

Input: $\boldsymbol{\sigma}, \boldsymbol{J}, \boldsymbol{h}, P_{\mathrm{merge}}$
Output: $\boldsymbol{m}, \boldsymbol{s}, \boldsymbol{\sigma}', \boldsymbol{J}', \boldsymbol{h}'$

$\boldsymbol{\sigma}' \leftarrow \boldsymbol{\sigma}, \boldsymbol{J}' \leftarrow \boldsymbol{J}$, and $\boldsymbol{h}' \leftarrow \boldsymbol{h}$
$m_i \leftarrow i$ and $s_i \leftarrow 1$ for $1 \le i \le N$
for $i = 1$ to N **do**
  Generate a random number rand $\in [0, 1)$
  if rand $< P_{\mathrm{merge}}$ **then**
    Randomly select an unmerged spin j ($j \neq i$) for $1 \le j \le N$
    $m_i \leftarrow j$ and $s_i \leftarrow \sigma_i\sigma_j$
    $\boldsymbol{\sigma}' \leftarrow \boldsymbol{\sigma}'\backslash\{\sigma_i\}$
    $\boldsymbol{J}' \leftarrow$ update_interactions($\boldsymbol{J}'$, i, j, s_i)
    $\boldsymbol{h}' \leftarrow$ update_biases($\boldsymbol{h}'$, i, j, s_i)
  end if
end for
for $i = 1$ to N **do**
  while $m_{m_i} \neq m_i$ **do**
    $s_i \leftarrow s_{m_i}s_i$
    $m_i \leftarrow m_{m_i}$
  end while
end for

when spin i is an unmerged spin, s_i always takes $+1$. Then, the interactions and the biases of H' are respectively given as:

$$J'_{ij} = \begin{cases} \displaystyle\sum_{k=1}^{N}\sum_{l=1}^{N} s_k s_l J_{kl}\delta_{m_k,i}\delta_{m_l,j} : & i \neq j \\ 0 : & i = j, \end{cases} \tag{3}$$

$$h'_i = \sum_{k=1}^{N} s_k h_k \delta_{m_k,i}, \tag{4}$$

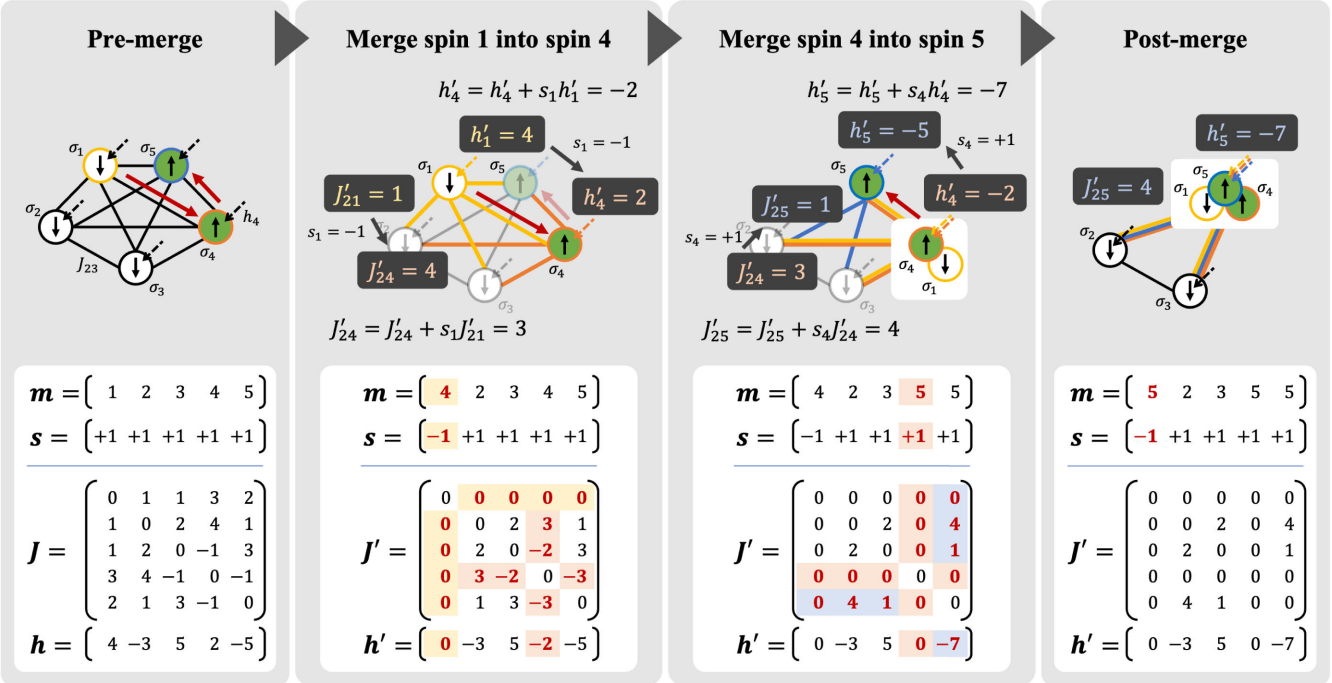


FIGURE 2. A demonstration of the merge process using a 5-spin Ising model with a spin configuration $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\} = \{-1, -1, -1, +1, +1\}$. In this example, spin 1 is merged into spin 4, and then spin 4 is merged into spin 5.

where $\delta_{i,j}$ denotes the Kronecker delta. The demerge process recovers $\boldsymbol{\sigma}$ by $\sigma_i = s_i \sigma_{m_i}$ for $1 \leq i \leq N$.

2) ALGORITHM OF MERGE PROCESS

Algorithm 1 shows a merge process to assign m_i and s_i from the current spin configuration $\boldsymbol{\sigma}$. It sequentially attempts to merge spin i into a randomly selected unmerged spin j with probability P_{merge} (lines 3–12). Here, P_{merge} is a hyperparameter called merge probability to control the number of merged spins. The interactions connected to spin j and the bias on spin j are updated when spin i is merged into spin j (lines 9–10). After completing the sequential merge attempt for all spins, s_i and m_i are updated for merged spins to satisfy $m_{m_i} = m_i$ (lines 13–18).

Algorithm 2 shows the update functions of $\boldsymbol{J}'$ and $\boldsymbol{h}'$ in **Algorithm 1**. For a given merged spin i and its destination spin j, the interactions connected to spin j and the bias on spin j are updated in lines 4–5 and 12, respectively. Then, the interactions connected to spin i and the bias on spin i are set to 0 in lines 7 and 13, respectively, since they do not exist in the deformed Hamiltonian H'.

Fig. 2 demonstrates **Algorithm 1** when a 5-spin Ising model with $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\} = \{-1, -1, -1, +1, +1\}$ is given. In this example, spin 1 is merged into spin 4, and then spin 4 is merged into spin 5. When spin 1 is merged into spin 4, we obtain $m_1 = 4$ and $s_1 = \sigma_1\sigma_4 = -1$. After that, the interactions connected to spin 4 are updated using those connected to spin 1, and the bias on spin 4 is updated using that on spin 1. Spin 4 is merged into spin 5 in the same way;

Algorithm 2 Functions to Update $\boldsymbol{J}'$ and $\boldsymbol{h}'$

```
1:  function update_interactions(J', i, j, s_i)
2:      for k = 1 to N do
3:          if k ≠ j then
4:              J'_kj ← J'_kj + s_i J'_ki
5:              J'_jk ← J'_jk + s_i J'_ik
6:          end if
7:          J'_ki ← 0 and J'_ik ← 0
8:      end for
9:      return J'
10: end function
11: function update_biases(h', i, j, s_i)
12:     h'_j ← h'_j + s_i h'_i
13:     h'_i ← 0
14:     return h'
15: end function
```

we obtain $m_4 = 5$ and $s_4 = \sigma_4\sigma_5 = +1$. Finally, s_1 and m_1 are updated to $s_1 = -1$ and $m_1 = 5$ to satisfy $m_{m_i} = m_i$.

We also demonstrate using Fig. 2 that a single-spin flip in H' is equivalent to a multi-spin flip in H. In this example, the original Hamiltonian H is a function of $\{\sigma_1, \sigma_2, \sigma_3, \sigma_4, \sigma_5\}$, and the spin configuration is $\{-1, -1, -1, +1, +1\}$ (see the leftmost panel). On the other hand, the deformed Hamiltonian H' is a function of $\{\sigma_2, \sigma_3, \sigma_5\}$, and the spin configuration is $\{-1, -1, +1\}$ (see the rightmost panel). When spin 5 in H' is flipped (i.e., the spin value changes from $+1$ to -1), the system energy changes from $H'(\{-1, -1, +1\}) = 12$ to

$H'(\{-1,-1,-1\}) = -12$; thus, the energy decreases by -24 as a result of the single-spin flip. In the original Hamiltonian, the energy changes from $H(\{-1,-1,-1,+1,+1\}) = 18$ to $H(\{+1,-1,-1,-1,-1\}) = -6$ when spins 1, 4, and 5 are flipped simultaneously; i.e., the single-spin flip in H' decreases the same energy as the three-spin flip in H. In this way, the merge method captures the energy difference caused by a multi-spin flip on SA-based Ising machines.

III. GPU-ORIENTED MERGE METHOD

The previous work [18] has demonstrated that the merge method improves the search performance of SA-based Ising machines when constrained COPs are solved. However, it is necessary to integrate the merge method with Ising-machine hardware for truly evaluating its benefit. In this section, we first discuss the obstacle to implement the merge method onto parallel computing platforms. Then, we propose an extended merge method that can be executed on a GPU efficiently.

A. PROBLEM AND MOTIVATION

We need to integrate the merge method with an Ising machine so that its processing time can be comparable to the annealing computation time. Ising machines accelerate the annealing computation by exploring parallel operations. However, the merge method is difficult to accelerate on parallel computing platforms since the merge process (i.e., **Algorithm 1**) is a sequential process; interactions updated in a single merge attempt are dependent on the subsequent attempt, as shown in Fig. 2.

In **Algorithm 1**, assigning m_i and s_i can be separated from updating interactions and biases; i.e., the algorithm can be rewritten as follows:

1) Assign m_i and s_i for all spins using the spin configuration $\boldsymbol{\sigma}$ and merge probability P_{merge}.
2) Generate the interactions $\boldsymbol{J}'$ and the biases $\boldsymbol{h}'$ from the original Hamiltonian (i.e., $\boldsymbol{J}$ and $\boldsymbol{h}$) using $\boldsymbol{m}$ and $\boldsymbol{s}$ obtained in step 1.

It should be noted that we can execute step 1 in parallel for all spins by first arranging spins into two groups, merged spins and unmerged spins, and then assigning the destination of each merged spin.

However, it is challenging to eliminate the sequential process in step 2. Modern GPUs process matrix multiplications quite efficiently since they have mixed-precision matrix multiplication units called Tensor Cores. Therefore, in the next section, we formulate step 2 as a series of matrix multiplications to execute it on a GPU efficiently.

B. FORMULATION

This section newly presents a theorem to generate a deformed Hamiltonian (i.e., $\boldsymbol{J}'$ and $\boldsymbol{h}'$) with matrix multiplications.

Theorem 1: Set a merging matrix $\boldsymbol{M}$ by (5) such that the k-th column keeps m_k and s_k by storing the value s_k at the m_k-th row and padding 0 for the remaining rows.

$$\boldsymbol{M} = \{M_{ik}\}^{N\times N},\ M_{ik} = \begin{cases} s_k : & m_k = i \\ 0 : & Otherwise. \end{cases} \tag{5}$$

Then, $J'_{ij} = (\boldsymbol{MJM}^{\mathrm{T}})_{ij}$ for $i \neq j$ and $h'_i = (\boldsymbol{hM}^{\mathrm{T}})_i$.

Proof 1: The interaction J'_{ij} between two unmerged spins i and j after completing the merge process is first explored. Let X_i be the set of spins merged into spin i; i.e., a spin $k \in X_i$ satisfies $m_k = i$. As illustrated in Fig. 3, interactions merged into J_{ij} are classified into three groups: (a) j and $k \in X_i$, (b) i and $l \in X_j$, and (c) $k \in X_i$ and $l \in X_j$. On the basis of this grouping, the interaction J'_{ij} in (3) is calculated as follows:

$$J'_{ij} = J_{ij} + \sum_{k\in X_i} s_k J_{kj} + \sum_{l\in X_j} s_l J_{il} + \sum_{k\in X_i}\sum_{l\in X_j} s_k s_l J_{kl}, \tag{6}$$

where the second, third, and fourth terms in the right-hand side correspond to groups (a), (b), and (c), respectively. Also, the bias h'_i after completing the merge process (i.e., (4)) is calculated as follows:

$$h'_i = h_i + \sum_{k\in X_i} s_k h_k. \tag{7}$$

Then, $\tilde{\boldsymbol{J}}' := \boldsymbol{MJM}^{\mathrm{T}}$ and $\tilde{\boldsymbol{h}}' := \boldsymbol{hM}^{\mathrm{T}}$ must satisfy the following conditions:

(i) $\tilde{\boldsymbol{J}}'$ is a symmetric matrix.
(ii) $\tilde{J}'_{ij} = J'_{ij}$ in (6) holds for unmerged spins i and j, and $\tilde{J}'_{kj} = 0$, $\tilde{J}'_{il} = 0$, and $\tilde{J}'_{kl} = 0$ hold for merged spins $k \in X_i$ and $l \in X_j$.
(iii) $\tilde{h}'_i = h'_i$ in (7) holds for an unmerged spin i, and $\tilde{h}'_k = 0$ holds for a merged spin $k \in X_i$.

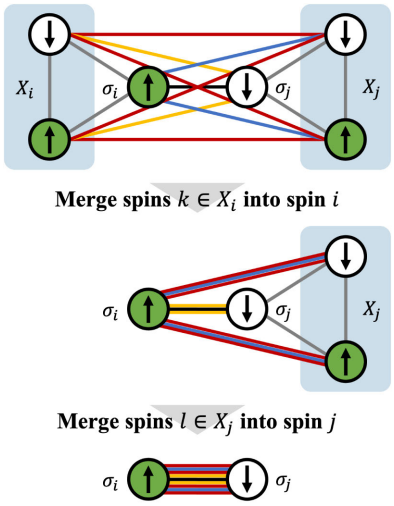


FIGURE 3. Change in the interaction J'_{ij} between two unmerged spins i and j through the merge process. Interactions finally merged into J_{ij} are classified into three groups: (a) $\{J_{kj}|k \in X_i\}$ (yellow edges), (b) $\{J_{il}|l \in X_j\}$ (blue edges), and (c) $\{J_{kl}|k \in X_i, l \in X_j\}$ (red edges). The interactions $\{J_{kk'}|k,k' \in (X_i \cup \{i\}), k \neq k'\}$ and $\{J_{ll'}|l,l' \in (X_j \cup \{j\}), l \neq l'\}$ (gray edges) are left out in the merge process.

As for condition (i), an off-diagonal element $\tilde{J}'_{ij}$ is transformed into $\tilde{J}'_{ji}$ as follows:

$$\begin{aligned}\tilde{J}'_{ij} &= (\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}})_{ij} = \sum_{b=1}^{N}\left(\sum_{a=1}^{N} M_{ia}J_{ab}\right)M_{jb} \\ &= \sum_{a=1}^{N}\left(\sum_{b=1}^{N} M_{jb}J_{ba}\right)M_{ia} = (\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}})_{ji} = \tilde{J}'_{ji}. \end{aligned} \tag{8}$$

The above transformation uses the symmetric property of the original interactions (i.e., $J_{ab} = J_{ba}$).

As for conditions (ii) and (iii), it is necessary to consider each row of the merging matrix $\boldsymbol{M}$. The i-th row has at least one non-zero element and is given by:

$$M_{ik} = \begin{cases} s_k : & k \in X_i \\ 1 : & k = i \\ 0 : & \text{Otherwise.} \end{cases} \tag{9}$$

Similarly, the j-th row is given by:

$$M_{jl} = \begin{cases} s_l : & l \in X_j \\ 1 : & l = j \\ 0 : & \text{Otherwise.} \end{cases} \tag{10}$$

From (9) and (10), $\tilde{J}'_{ij} = (\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}})_{ij}$ and $\tilde{h}'_i = (\boldsymbol{h}\boldsymbol{M}^{\mathrm{T}})_i$ are respectively transformed as follows:

$$\begin{aligned}\tilde{J}'_{ij} &= \sum_{b=1}^{N}\left(\sum_{a=1}^{N} M_{ia}J_{ab}\right)M_{jb} \\ &= \sum_{b=1}^{N}\left(J_{ib} + \sum_{k\in X_i} s_k J_{kb}\right)M_{jb} \\ &= \left(J_{ij} + \sum_{k\in X_i} s_k J_{kj}\right) \\ &\quad + \sum_{l\in X_j}\left(J_{il} + \sum_{k\in X_i} s_k J_{kl}\right)s_l \\ &= J_{ij} + \sum_{k\in X_i} s_k J_{kj} + \sum_{l\in X_j} s_l J_{il} \\ &\quad + \sum_{k\in X_i}\sum_{l\in X_j} s_k s_l J_{kl}, \end{aligned} \tag{11}$$

$$\tilde{h}'_i = \sum_{a=1}^{N} h_a M_{ia} = h_i + \sum_{k\in X_i} s_k h_k. \tag{12}$$

On the other hand, all the elements in the k and l-th rows are 0 since $m_k = i\ (\neq k)$ and $m_l = j\ (\neq l)$ indicate that no spins are merged into spins $k \in X_i$ and $l \in X_j$.

$$\tilde{J}'_{kj} = \sum_{b=1}^{N}\left(\sum_{a=1}^{N} \underline{M_{ka}}J_{ab}\right)M_{jb}, \tag{13}$$

$$\tilde{J}'_{il} = \sum_{b=1}^{N}\left(\sum_{a=1}^{N} M_{ia}J_{ab}\right)\underline{M_{lb}}, \tag{14}$$

$$\tilde{J}'_{kl} = \sum_{b=1}^{N}\left(\sum_{a=1}^{N} \underline{M_{ka}}J_{ab}\right)\underline{M_{lb}}, \tag{15}$$

$$\tilde{h}'_k = \sum_{a=1}^{N} \underline{M_{ka}}h_a. \tag{16}$$

Equations from (13) to (16) show that $\tilde{J}'_{kj} = 0$, $\tilde{J}'_{il} = 0$, $\tilde{J}'_{kl} = 0$, and $\tilde{h}'_k = 0$ hold since all the underlined values are 0. Thus, conditions (ii) and (iii) are satisfied.

Since all the conditions are satisfied, $J'_{ij} = (\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}})_{ij}$ for $i \neq j$ and $h'_i = (\boldsymbol{h}\boldsymbol{M}^{\mathrm{T}})_i$. □

Theorem 1 enables generating the interactions $\boldsymbol{J}'$ and the biases $\boldsymbol{h}'$ of the deformed Hamiltonian with matrix multiplications only. In other words, we can efficiently execute step 2 of the two-stage merge process on Tensor Cores.

Now, we revisit the example shown in Fig. 2 and confirm that $\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}}$ and $\boldsymbol{h}\boldsymbol{M}^{\mathrm{T}}$ give $\boldsymbol{J}'$ and $\boldsymbol{h}'$, respectively. Since spins 1 and 4 are merged into spin 5 in this example, $\boldsymbol{m}$ and $\boldsymbol{s}$ are given by:

$$\begin{aligned}\boldsymbol{m} &= \begin{bmatrix}5 & 2 & 3 & 5 & 5\end{bmatrix}, \\ \boldsymbol{s} &= \begin{bmatrix}-1 & +1 & +1 & +1 & +1\end{bmatrix}.\end{aligned}$$

Thus, the merging matrix $\boldsymbol{M}$ is constructed as follows:

$$\boldsymbol{M} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & +1 & 0 & 0 & 0 \\ 0 & 0 & +1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 \end{bmatrix}.$$

Note that, as mentioned earlier, the row vectors for unmerged spins (i.e., 2, 3, and 5) have one or more non-zero elements. Using the merging matrix $\boldsymbol{M}$, we obtain

$$\begin{aligned}\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}} &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 4 \\ 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 1 & 0 & -12 \end{bmatrix}, \\ \boldsymbol{h}\boldsymbol{M}^{\mathrm{T}} &= \begin{bmatrix}0 & -3 & 5 & 0 & -7\end{bmatrix}.\end{aligned}$$

The interaction matrix $\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}}$ is the same as $\boldsymbol{J}'$ in Fig. 2 except for a diagonal element (i.e., $(\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}})_{55}$). Since the deformed Hamiltonian H' only depends on the off-diagonal elements of $\boldsymbol{J}'$ (see (2)), $\boldsymbol{M}\boldsymbol{J}\boldsymbol{M}^{\mathrm{T}}$ is equivalent to $\boldsymbol{J}'$. Also, the bias vector $\boldsymbol{h}\boldsymbol{M}^{\mathrm{T}}$ is the same as $\boldsymbol{h}'$ in Fig. 2.

Moreover, a minor modification to **Theorem 1** can correct the diagonal elements to zeros as follows:

Corollary 2:

$$\boldsymbol{J}' = \boldsymbol{M}\boldsymbol{J}^{*}\boldsymbol{M}^{\mathrm{T}}, \tag{17}$$

where

$$\boldsymbol{J}^{*} = \{J^{*}_{ij}\}^{N\times N},\ J^{*}_{ij} = \begin{cases} 0 : & m_i = m_j \\ J_{ij} : & \textit{Otherwise.} \end{cases} \tag{18}$$

Proof 2: Equation (11) shows that non-zero values of a diagonal element J'_{ii} arise due to the two types of interactions; one is those between spins i and $k \in X_i$, and the other is those between two different spins in X_i. Note that $m_k = i$ is only satisfied for spins $k = i$ or $k \in X_i$. Thus, we can make $J'_{ii} = 0$ by preparing a masked interaction matrix $\boldsymbol{J}^*$ instead of $\boldsymbol{J}$. More explicitly,

$$\begin{aligned} J'_{ii} &= (\boldsymbol{M}\boldsymbol{J}^*\boldsymbol{M}^{\mathrm{T}})_{ii} \\ &= \sum_{b=1}^{N}\left(\sum_{a=1}^{N} M_{ia}J^*_{ab}\right)M_{ib} \\ &= 2\sum_{k\in X_i} s_k J^*_{ki} + \sum_{k\in X_i}\sum_{k'\in X_i} s_k s_{k'} J^*_{kk'} \\ &= 0. \end{aligned} \tag{19}$$

As demonstrated in Fig. 3, the above two types of interactions are left out in the merge process. This means that the masked interactions in (18) do not affect the off-diagonal elements of $\boldsymbol{J}'$ calculated by (11), and thus $\boldsymbol{J}' = \boldsymbol{M}\boldsymbol{J}^*\boldsymbol{M}^{\mathrm{T}}$. □

Also, the demerge process, i.e., $\sigma_i = s_i\sigma_{m_i}$ $(1 \le i \le N)$, is formulated using the merging matrix $\boldsymbol{M}$ as follows:

Corollary 3: Set a spin configuration $\tilde{\boldsymbol{\sigma}}$ by:

$$\tilde{\boldsymbol{\sigma}} = \{\tilde{\sigma}_i\}^N,\ \tilde{\sigma}_i = \begin{cases} \sigma_i : & m_i = i \\ X : & Otherwise, \end{cases} \tag{20}$$

where X shows a don't care value. Then, the spin configuration $\boldsymbol{\sigma}$ is recovered by $\tilde{\boldsymbol{\sigma}}\boldsymbol{M}$, i.e., $\boldsymbol{\sigma} = \tilde{\boldsymbol{\sigma}}\boldsymbol{M}$

Proof 3: The i-th element of $\tilde{\boldsymbol{\sigma}}\boldsymbol{M}$ is given by:

$$(\tilde{\boldsymbol{\sigma}}\boldsymbol{M})_i = \sum_{a=1}^{N} \tilde{\sigma}_a M_{ai}. \tag{21}$$

From the definition of $\boldsymbol{M}$ in (5), the i-th column of $\boldsymbol{M}$ has only one non-zero element s_i at the m_i-th row; i.e., $M_{ai} = s_i$ if $a = m_i$, otherwise $M_{ai} = 0$. Then,

$$(\tilde{\boldsymbol{\sigma}}\boldsymbol{M})_i = s_i\tilde{\sigma}_{m_i}. \tag{22}$$

Moreover, $\tilde{\sigma}_{m_i} = \sigma_{m_i}$ holds because $m_{m_i} = m_i$. Thus, $(\tilde{\boldsymbol{\sigma}}\boldsymbol{M})_i = s_i\sigma_{m_i} = \sigma_i$. □

IV. GPU-BASED ISING MACHINE WITH MERGE METHOD

This section constructs a GPU-based Ising machine with the merge method proposed in Section III. When an annealing algorithm is implemented on a GPU, preparing replicas is an effective way to enhance the execution efficiency. However, integrating the proposed merge method with the replica parallel execution results in large memory consumption. In this section, we first discuss the memory issue, and then propose a decomposition method for saving the memory.

A. MEMORY ISSUE

When a GPU-based Ising machine is developed, preparing multiple replicas that search for multiple solutions in parallel is a major strategy to fully utilize computing resources on a GPU [12], [13]. This replica parallel execution reduces the required time per a single solution. Let R be the number of replicas. The spin vector $\boldsymbol{\sigma}$ is extended to an $R \times N$ matrix represented by:

$$\boldsymbol{\sigma}^* = \begin{bmatrix} \boldsymbol{\sigma}^{(1)} \\ \boldsymbol{\sigma}^{(2)} \\ \vdots \\ \boldsymbol{\sigma}^{(R)} \end{bmatrix},$$

where $\boldsymbol{\sigma}^{(r)}$ $(r = 1, 2, \ldots, R)$ shows the spin configuration of the r-th replica.

In this work, we focus on parallel-trial SA (PSA) [9], one of the SA-based annealing algorithms. It inspects all spins in parallel, and then selects one spin from spin-flip candidates. In the replica parallel execution of PSA, the calculation of local fields on $R \times N$ spins at every Monte Carlo (MC) step is the most computationally expensive part. The local fields $\hat{\boldsymbol{h}}^{(r)}$ of the r-th replica are given by:

$$\hat{\boldsymbol{h}}^{(r)} = \boldsymbol{\sigma}^{(r)}\boldsymbol{J} + \boldsymbol{h}. \tag{23}$$

When the proposed merge method is integrated into PSA with multiple replicas, an individual merging matrix $\boldsymbol{M}^{(r)}$ is required for every replica; this is because the merging matrix is constructed using the current spin configuration. In other words, $O(RN^2)$ memory space is required to store merging matrices. As a result, the number of replicas is strictly limited due to the memory capacity of a GPU, making it difficult to enhance the execution efficiency. Therefore, we should take measures to reduce memory consumption.

B. MERGING MATRIX DECOMPOSITION

One approach for reducing memory consumption is to share some information among replicas. If replicas share the combination of merged spins and their destinations, we can make the vector $\boldsymbol{m}$ common in all replicas. However, even in this case, the vector $\boldsymbol{s}$ must be kept in each replica individually, preventing the merging matrix $\boldsymbol{M}$ from being shared among replicas. In order to resolve this situation, we propose a new theorem to decompose the merging matrix $\boldsymbol{M}$:

Theorem 4: Set two $N \times N$ matrices $\boldsymbol{L}$ and $\boldsymbol{D}$ by (24) and (25), respectively.

$$\boldsymbol{L} = \{L_{ik}\}^{N\times N},\ L_{ik} = \begin{cases} 1 : & m_k = i \\ 0 : & Otherwise, \end{cases} \tag{24}$$

$$\boldsymbol{D} = \mathrm{diag}(\boldsymbol{s}). \tag{25}$$

Then, $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{D}$.

Proof 4: From (5) and (24), $M_{ik} = L_{ik}s_k$ holds. Then,

$$(\boldsymbol{L}\boldsymbol{D})_{ik} = \sum_{a=1}^{N} L_{ia}D_{ak} = L_{ik}D_{kk} = L_{ik}s_k. \tag{26}$$

Thus, $M_{ik} = (\boldsymbol{L}\boldsymbol{D})_{ik}$. □

Theorem 4 separates $\boldsymbol{m}$ and $\boldsymbol{s}$ by $\boldsymbol{L}$ and $\boldsymbol{D}$, respectively. As a result, the matrix $\boldsymbol{L}$ can be shared among all replicas by

introducing the assumption that the combination of merged spins and their destinations is shared among replicas. Here, spin i is a merged spin when $L_{ii} = 0$, whereas an unmerged spin when $L_{ii} = 1$. Although the matrix $\boldsymbol{D}$ remains required for individual replicas, we need to store the diagonal elements only. Therefore, the memory space is reduced to $O(N^2) + O(RN)$ by this matrix decomposition.

C. MERGE PSA

In this section, we propose an algorithm, Merge PSA, where the merge method is integrated into PSA with multiple replicas. After that, we explain how to implement this algorithm on a GPU.

1) ALGORITHM

In Merge PSA, the local-field calculation of the r-th replica is extended to:

$$\hat{\boldsymbol{h}}^{(r)} = \tilde{\boldsymbol{\sigma}}^{(r)}\boldsymbol{J}'^{(r)} + \boldsymbol{h}'^{(r)}, \tag{27}$$

where $\boldsymbol{J}'^{(r)}$ and $\boldsymbol{h}'^{(r)}$ are the deformed interaction matrix and bias vector of the r-th replica, respectively, and $\tilde{\boldsymbol{\sigma}}^{(r)}$ is a spin configuration given by (20). Here, it is necessary to use (17) to obtain $\boldsymbol{J}'^{(r)}$ since the diagonal elements of $\boldsymbol{J}'^{(r)}$ affect the values in $\hat{\boldsymbol{h}}^{(r)}$. This means that we need to calculate the masked interaction matrix $\boldsymbol{J}^*$ given by (18) in the merge process. Note that $\boldsymbol{J}^*$ can be calculated as follows:

Corollary 5:

$$\boldsymbol{J}^* = \boldsymbol{J} - \boldsymbol{J} \odot (\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}), \tag{28}$$

where $\odot$ is Hadamard product.

Proof 5: An element $(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L})_{ij}$ gives the inner product of the i-th and the j-th column vectors in $\boldsymbol{L}$. It becomes 1 only when $m_i = m_j$, otherwise it becomes 0:

$$(\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L})_{ij} = \sum_{a=1}^{N} L_{ai}L_{aj} = \begin{cases} 1: & m_i = m_j \\ 0: & \text{Otherwise} \end{cases} \tag{29}$$

since the i-th (j-th) column of $\boldsymbol{L}$ has only one element with value 1 at the m_i-th (m_j-th) row. Comparing (29) with (18) implies that $\boldsymbol{J}^* = \boldsymbol{J} - \boldsymbol{J} \odot (\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L})$. □

As a result, (27) is rewritten as follows:

$$\begin{aligned} \hat{\boldsymbol{h}}^{(r)} &= \tilde{\boldsymbol{\sigma}}^{(r)}\boldsymbol{M}^{(r)}\boldsymbol{J}^*\left(\boldsymbol{M}^{(r)}\right)^{\mathrm{T}} + \boldsymbol{h}\left(\boldsymbol{M}^{(r)}\right)^{\mathrm{T}} \\ &= \boldsymbol{\sigma}^{(r)}\boldsymbol{J}^*\left(\boldsymbol{L}\boldsymbol{D}^{(r)}\right)^{\mathrm{T}} + \boldsymbol{h}\left(\boldsymbol{L}\boldsymbol{D}^{(r)}\right)^{\mathrm{T}} \\ &= \boldsymbol{\sigma}^{(r)}\boldsymbol{J}^*\boldsymbol{D}^{(r)}\boldsymbol{L}^{\mathrm{T}} + \boldsymbol{h}\boldsymbol{D}^{(r)}\boldsymbol{L}^{\mathrm{T}}, \end{aligned} \tag{30}$$

where we have used $\boldsymbol{\sigma}^{(r)} = \tilde{\boldsymbol{\sigma}}^{(r)}\boldsymbol{M}^{(r)}$ in **Corollary 3**.

Algorithm 3 shows Merge PSA; the two-stage merge process in Section III-A and the demerge process are introduced into PSA. The merge process first generates $\boldsymbol{L}$ and $\boldsymbol{D}^{(r)}$ in the first stage (line 3). Then, in the second stage, the merge process generates $\boldsymbol{J}'^{(r)}$ and $\boldsymbol{h}'^{(r)}$ to calculate the local fields in (30), which is explicitly given as follows:

Algorithm 3 Merge PSA for a Single Replica (r)

Input: $\boldsymbol{\sigma}^{(r)}, \boldsymbol{J}, \boldsymbol{h}, P_{\mathrm{merge}}, S, I_{\mathrm{M}}, T(s)$
Output: $\boldsymbol{\sigma}^{(r)}$

1: **for** $s = 1$ to S **do**
2: **if** $s \bmod I_{\mathrm{M}} = 1$ **then**
3: $(\boldsymbol{L}, \boldsymbol{D}^{(r)}) \leftarrow \mathrm{generate}(\boldsymbol{\sigma}^{(r)},\ P_{\mathrm{merge}})$
4: $\boldsymbol{h}'^{(r)} \leftarrow \boldsymbol{h}\boldsymbol{D}^{(r)}\boldsymbol{L}^{\mathrm{T}}$
5: $\boldsymbol{J}^* \leftarrow \boldsymbol{J} - \boldsymbol{J} \odot (\boldsymbol{L}^{\mathrm{T}}\boldsymbol{L})$
6: **end if**
7: $\hat{\boldsymbol{h}}^{(r)} \leftarrow \boldsymbol{\sigma}^{(r)}\boldsymbol{J}^*\boldsymbol{D}^{(r)}\boldsymbol{L}^{\mathrm{T}} + \boldsymbol{h}'^{(r)}$
8: **for** $i = 1$ to N **do**
9: $p_i \leftarrow \dfrac{1}{1 + \exp\left(2\hat{h}_i^{(r)}\sigma_i^{(r)}/T(s)\right)}$
10: Generate a random number $\mathsf{rand} \in [0, 1)$
11: $f_i \leftarrow (p_i > \mathsf{rand})\ \&\ (L_{ii} = 1)$
12: **end for**
13: **if** $\exists i,\ f_i = \mathsf{True}$ **then**
14: Randomly select a spin j with $f_j = \mathsf{True}$
15: **for** $k = 1$ to N **do**
16: $\sigma_k^{(r)} \leftarrow (-2 \times L_{jk} + 1) \times \sigma_k^{(r)}$
17: **end for**
18: **end if**
19: **end for**

- The second term of (30) is calculated in advance (line 4) since R bias vectors (i.e., $\boldsymbol{h}'^{(r)}$ for $1 \le r \le R$) consume only $O(RN)$ memory space.
- The masked interaction matrix $\boldsymbol{J}^*$ is calculated in advance (line 5). $\boldsymbol{J}^*$ is shared among all replicas, and thus consumes only $O(N^2)$ memory space.

The demerge process is executed with a single-spin-flip operation in PSA (lines 13–18); in lines 15–17, all spins with $L_{jk} = 1$ (i.e., $m_k = j$) are flipped, meaning that merged spins $k \in X_j$ are flipped simultaneously with an unmerged spin j. Note that we need to remove merged spins (i.e., spins k with $L_{kk} = 0$) from spin-flip candidates in PSA, as in line 11.

Algorithm 4 shows the function to generate $\boldsymbol{L}$ and $\boldsymbol{D}^{(r)}$ in **Algorithm 3**. This algorithm arranges all N spins into two groups, merged spins and unmerged spins, in lines 3–8. L_{ii} is set to 1 for an unmerged spin i (line 6), whereas it remains 0 if spin i is selected as a merged spin. For an unmerged spin i, $D_{ii}^{(r)} = s_i^{(r)} = +1$. Then, the destination of each merged spin is assigned in lines 9–14. For a merged spin k (i.e., spin k with $L_{kk} = 0$), an unmerged spin i (i.e., spin i with $L_{ii} = 1$) is randomly selected (line 11) and is set as the destination by $L_{ik} = 1$ and $D_{kk}^{(r)} = s_k^{(r)} = \sigma_k^{(r)}\sigma_i^{(r)}$ (line 12).

2) GPU IMPLEMENTATION

Algorithm 3 is implemented on a GPU to execute R replicas in parallel. This implementation stores the matrix $\boldsymbol{D}^{(r)}$ in the form of:

$$\boldsymbol{D}^* = \{D_{ri}^*\}^{R\times N},\ D_{ri}^* = D_{ii}^{(r)}. \tag{31}$$

Algorithm 4 A Function to Generate $\boldsymbol{L}$ and $\boldsymbol{D}$

```
1: function generate(σ^(r), P_merge)
2:     L_ik ← 0 and D_ik^(r) ← 0 for 1 ≤ i, k ≤ N
3:     for i = 1 to N do
4:         Generate a random number rand ∈ [0, 1)
5:         if rand ≥ P_merge then
6:             L_ii ← 1, D_ii^(r) ← +1
7:         end if
8:     end for
9:     for k = 1 to N do
10:        if L_kk = 0 then
11:            Randomly select a spin i with L_ii = 1
12:            L_ik ← 1, D_kk^(r) ← σ_k^(r) σ_i^(r)
13:        end if
14:    end for
15:    return L, D^(r)
16: end function
```

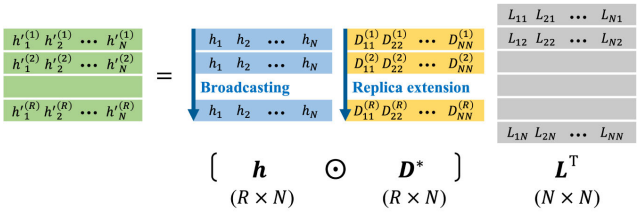


FIGURE 4. Matrix operations in (32).

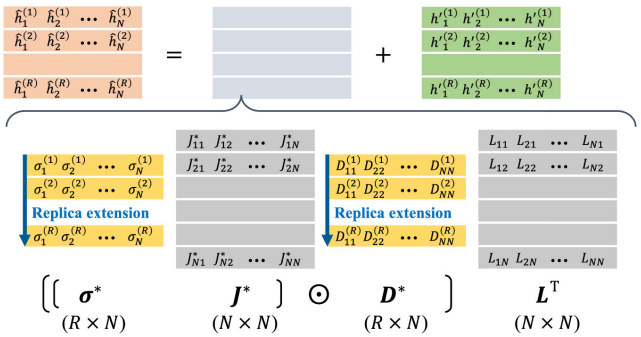


FIGURE 5. Matrix operations in (33).

Then, as functions of $\boldsymbol{\sigma}^*$ and $\boldsymbol{D}^*$, $\boldsymbol{h}'^{(r)}$ in line 4 and $\hat{\boldsymbol{h}}^{(r)}$ in line 7 are calculated across all replicas as follows:

$$h_i'^{(r)} = \left(\left(\boldsymbol{h} \odot \boldsymbol{D}^*\right)\boldsymbol{L}^{\mathrm{T}}\right)_{ri}, \tag{32}$$

$$\hat{h}_i^{(r)} = \left(\left(\left(\boldsymbol{\sigma}^*\boldsymbol{J}^*\right) \odot \boldsymbol{D}^*\right)\boldsymbol{L}^{\mathrm{T}}\right)_{ri} + h_i'^{(r)}. \tag{33}$$

Figs. 4 and 5 illustrate the matrix operations in (32) and (33), respectively. In (32), the bias vector $\boldsymbol{h}$ is broadcast to the $R \times N$ matrix, and then is multiplied by the matrix $\boldsymbol{D}^*$ element-wise; this element-wise multiplication corresponds to $\boldsymbol{h}\boldsymbol{D}^{(r)}$ for $1 \le r \le R$. After that, the resulting matrix is multiplied by the matrix $\boldsymbol{L}^{\mathrm{T}}$. In (33), the spin matrix $\boldsymbol{\sigma}^*$ is multiplied by the matrix $\boldsymbol{J}^*$, and then is multiplied by the matrix $\boldsymbol{D}^*$ element-wise to obtain $\boldsymbol{\sigma}^{(r)}\boldsymbol{J}^*\boldsymbol{D}^{(r)}$ for $1 \le r \le R$. Finally, the resulting matrix is multiplied by the matrix $\boldsymbol{L}^{\mathrm{T}}$, and then the deformed bias matrix given by (32) is added to it.

The matrix operations in (32) and (33) are classified into two types; one is matrix multiplication between a $R \times N$ matrix and a $N \times N$ matrix, and the other is element-wise multiplication or addition between two $R \times N$ matrices. The former type has $O(RN^2)$ computational complexity, being processed on Tensor Cores. On the other hand, the latter type is executed on general-purpose parallel computing units, and its computational complexity is $O(RN)$. Furthermore, line 3 and lines 8–12 in **Algorithm 3** have $O(RN)$ computational complexity, and thus are classified into the latter type.

TABLE 1. Specification of eight QKP instances [19] used in this paper.

Instance	# of items K	Capacity c	The rate of non-zero values in $\{p_{ij} \mid 1 \le i < j \le K\}$
r_200_25_1	200	4027	0.25
r_200_50_1	200	3547	0.50
r_200_75_1	200	2866	0.75
r_200_100_1	200	4785	1.0
r_300_25_1	300	376	0.25
r_300_50_1	300	3550	0.50
r_300_50_7	300	328	0.50
r_300_50_10	300	7040	0.50

V. EXPERIMENTAL RESULTS

In order to evaluate the performance of Merge PSA, we developed GPU-implemented Merge PSA (GiMP) and GPU-implemented PSA (GiP), and solved several QKP instances provided in [19]. In this evaluation, we used CUDA® and cuBLAS on NVIDIA® TITAN RTX™ as a GPU, AMD EPYC™7542 as a CPU, Ubuntu 18.04 LTS as an operating system. Moreover, we solved the same QKP instances using a state-of-the-art GPU-based Ising machine [11] to compare the results with GiMP; in this section, this GPU-based Ising machine is labeled as "Ising machine" for simplicity.

A. SETUP

1) ISING-MODEL FORMULATION OF QKP

QKP is one of the COPs, which packs items into a knapsack with a capacity c and maximizes the total profit P brought by the packed items. In the K items given in this problem, item i has weight w_i and profit p_{ii}. Also, unlike the basic knapsack problem, QKP gives profit p_{ij} $(i \ne j)$ that is added to the total profit P when items i and j are both packed into the knapsack. In this experiment, we selected eight instances in a QKP benchmark suite [19] considering the distribution of three factors (the number of items K, capacity c, and the rate of non-zero values in $\{p_{ij} \mid 1 \le i < j \le K\}$) that characterize QKP; their specification is summarized in Table 1.

A QKP is formulated as a quadratic unconstrained binary optimization (QUBO) problem [18]. In this formulation, K binary variables $\boldsymbol{x} = \{x_i \mid x_i \in \{1, 0\}\}_{i=1}^{K}$ are used, where x_i

shows whether item i is packed into the knapsack ("1") or not ("0"). Solving the QKP is finding a configuration $\boldsymbol{x}$ that maximizes the total profit P given by:

$$P = \sum_{i=1}^{K} \sum_{j=i}^{K} p_{ij} x_i x_j, \tag{34}$$

while satisfying the capacity constraint given by:

$$W = \sum_{i=1}^{K} w_i x_i \leq c. \tag{35}$$

Based on the objective function (34) and the constraint (35), the energy function H_{QUBO} is defined by:

$$H_{\mathrm{QUBO}} = -P + A \left\{ W - \left(c - \sum_{d=1}^{w_{\max}} y_d \right) \right\}^2, \tag{36}$$

where A is a constraint coefficient and $w_{\max}$ is the maximum weight of all K items. Equation (36) introduces auxiliary binary variables $\{y_d\}_{d=1}^{w_{\max}}$ to represent the inequality (35) in the energy function. As a result, $K + w_{\max}$ binary variables are required to formulate a QKP as a QUBO problem. Note that (36) is transformed into (1) when a QKP is solved on GiMP, GiP, or Ising machine.

2) EXPERIMENTAL CONDITIONS

In order to compare GiMP, GiP, and Ising machine, we evaluate their performance from the perspective of solution quality and execution time. Since they search for solutions according to stochastic processes, the performance evaluation is based on k ($\gg 1$) solutions and time to obtain the k solutions as follows:

For k solutions obtained in t_{total} seconds, FSs that satisfy the capacity constraint (35) are extracted; here, k_{FS} denotes the number of extracted solutions. Then, the performance is evaluated using the effective time t' and the average residual energy $\overline{\Delta H}$ in (37) and (38), respectively:

$$t' = \frac{t_{\mathrm{total}}}{k_{\mathrm{FS}}} \ [s], \tag{37}$$

$$\overline{\Delta H} = \frac{\sum_{r=1}^{k_{\mathrm{FS}}} (H^{(r)} - H_{\mathrm{opt}})}{k_{\mathrm{FS}}}, \tag{38}$$

where $H^{(r)}$ is equivalent to the total profit P derived from the r-th replica, and H_{opt} is the optimal total profit recorded in [19]. By utilizing (37) and (38), we can study the correlation between the time to obtain a FS and its average quality. Now, how to obtain k solutions by GiMP/GiP and Ising machine is explained in the following:

GiMP and GiP can specify the number of replicas R and the number of MC steps S. Then, we obtained k solutions by selecting the top-k ($k \leq R$). In this experiment, we varied S as shown in Table 2, whereas we fixed R to 128. Referring to the previous work [20], [21], we used the pseudo temperature scheduling given by:

$$T(s) = T_{\mathrm{init}} \times \left(\frac{T_{\mathrm{fin}}}{T_{\mathrm{init}}} \right)^{\frac{s-1}{S-1}}, \tag{39}$$

TABLE 2. Parameter settings used in GiMP and GiP.

Notation	Description	Value
S	# of MC steps	$10^4, 10^5, 10^6$
R	# of replicas	128
T_{init}	Pseudo temperature at the first MC step	$0.01N \times \max\{\lvert J_{ij}\rvert\}$
T_{fin}	Pseudo temperature at the last MC step	$0.1 \times \min_{J_{ij}\neq 0}\{\lvert J_{ij}\rvert\}$

TABLE 3. Merge interval I_{M} used in GiMP.

# of MC steps S	Merge interval I_{M}
10^4	5, 10, 100
10^5	5, 10, 50, 100
10^6	5, 10, 50, 100, 500, 1000

and set T_{init} and T_{fin} as shown in Table 2. In GiMP, we also need to specify the merge probability P_{merge} and the merge interval I_{M}. We varied I_{M} for each S as in Table 3, whereas we set $P_{\mathrm{merge}} = 0.7$. For every QKP instance and every parameter set, we conducted the following procedure:

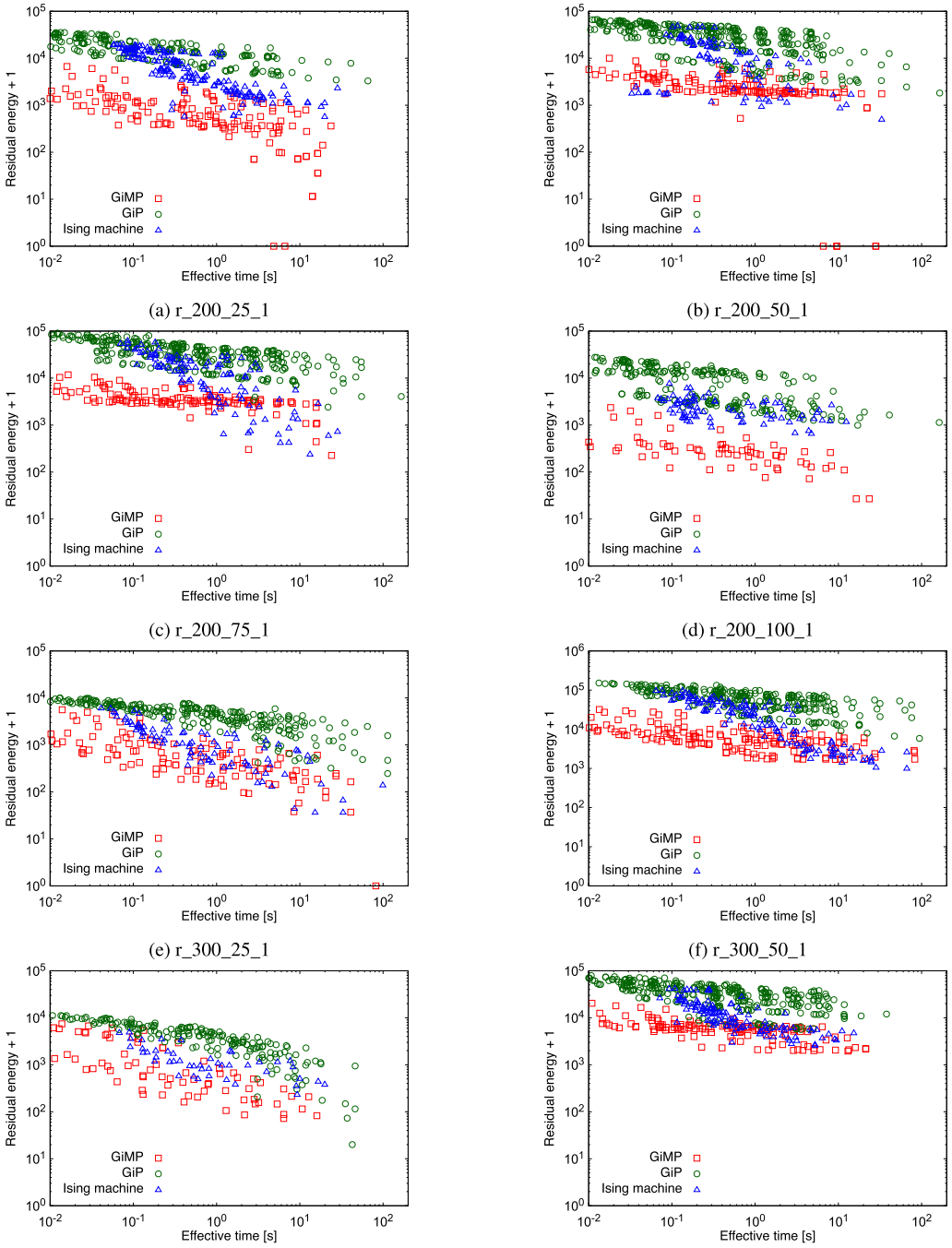
1) Initialize the constraint coefficient A in (36) to 5.
2) Run GiMP (or GiP) and obtain $R = 128$ solutions.
3) Extract FSs and calculate the FS rate.
4) Increase A by 5 and return to step 2 if the FS rate is below 0.9; otherwise, exit.

As for Ising machine, on the other hand, we only specify the approximate annealing time t_{app} and the maximum number of solutions $R_{\max}$ output from it. In this experiment, we assumed three settings for t_{app}: 0.1, 1, and 10 seconds. Note that we can also run Ising machine without specifying $R_{\max}$. Given a QKP instance and a setting of t_{app}, we conducted the following procedure to obtain k solutions:

1) Initialize the constraint coefficient A in (36) to 5.
2) Run Ising machine 20 times without specifying $R_{\max}$ and estimate the number of solutions R_{est} expected to obtain in a single run. When we have obtained R_{total} solutions in this step, R_{est} is calculated by $\lfloor R_{\mathrm{total}}/20 \rfloor$.
3) Calculate the FS rate based on R_{total} solutions.
4) Specify $R_{\max}$ to 1.
5) Run Ising machine 20 times with specifying $R_{\max}$ and obtain k ($\approx 20 \times R_{\max}$) solutions.
6) Increase $R_{\max}$ by Δ_R and repeat step 5 as long as $R_{\max} < R_{\mathrm{est}}$ is satisfied. $\Delta_R = 1$ if $R_{\max} \leq 10$; otherwise, $\Delta_R = 5$.
7) Increase A by 5 and return to step 2 if the FS rate is below 0.9; otherwise, exit.

B. RESULTS

Experimental results when eight QKP instances are solved by GiMP (□), GiP (○), and Ising machine (△) are shown in Fig. 6. The horizontal and the vertical axes of each figure are the effective time t' calculated by (37) and the average residual energy $\overline{\Delta H}$ calculated by (38), respectively. Small t' means that a large number of FSs are obtained in a short time,



(a) r_200_25_1 (b) r_200_50_1

(c) r_200_75_1 (d) r_200_100_1

(e) r_300_25_1 (f) r_300_50_1

(g) r_300_50_7 (h) r_300_50_10

FIGURE 6. Experimental results when eight QKP instances [19] are solved by GiMP (□), GiP (○), and Ising machine (△).

whereas small $\overline{\Delta H}$ indicates that many high-quality solutions are obtained. We expect that increasing the effective time improves the average residual energy. The results in Fig. 6 demonstrate that GiMP, GiP, and Ising machine follow this trend regardless of the three factors in QKP. Also, GiMP outperforms the others in all the instances. In particular, the superiority is observed in the region where the effective time is small; i.e., GiMP can find relatively high-quality FSs quickly. Furthermore, GiMP finds the optimal solutions for some problem instances (see Figs. 6a, 6b, and 6c).

In Fig. 7, the dependence of the merge interval I_{M} on the performance of GiMP is evaluated. As mentioned earlier, we should evaluate the performance based on the effective time and the average residual energy. We solved the instance r_300_50_1 by GiMP while varying $I_{\mathrm{M}} \in \{5, 10, 100\}$. It is apparent that using a small I_{M} tends to increase the execution

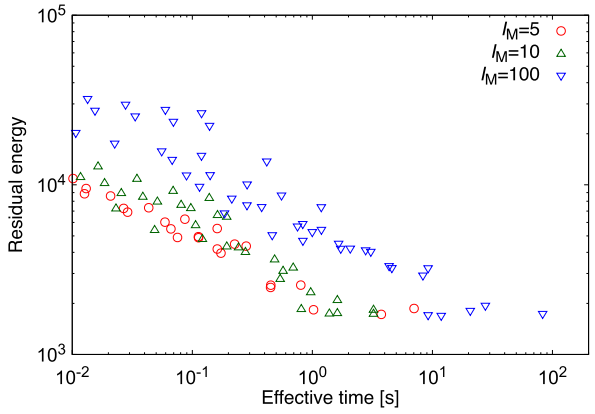


FIGURE 7. Relationship between the effective time and the average residual energy while varying the merge interval I_M of GiMP. The QKP instance used in this evaluation is r_300_50_1.

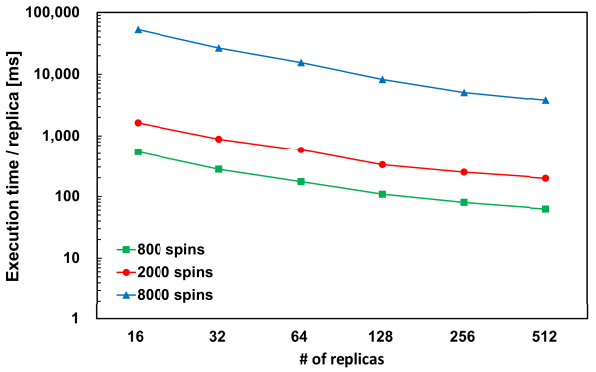


FIGURE 8. Relationship between the number of replicas and the execution time per replica. In this experiment, we ran GiMP with the number of MC steps $S = 10^5$ for Ising models with 800, 2000, and 8000 spins.

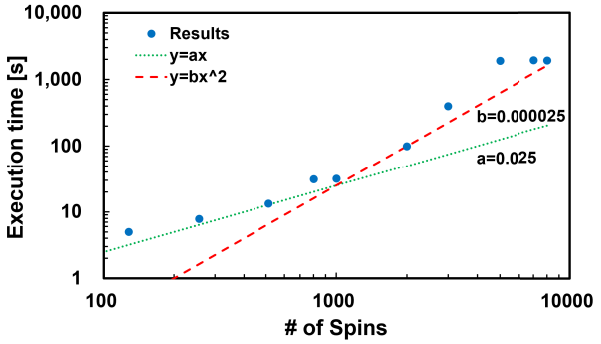


FIGURE 9. Relationship between the number of spins and the execution time.

time because the frequency of the merge process increases. However, the results of $I_M = 5$ and 10 show a similar trend with respect to the performance. This means that GiMP with $I_M = 5$ obtains high-quality FSs with fewer MC steps than that with $I_M = 10$. Also, the result of $I_M = 100$ is inferior to the others. Since the merge fixes the combination of merged spins and their destinations, we need to use a relatively small I_M and try various combinations.

C. DISCUSSION

In this section, we discuss the performance differences between GiMP and the others.

First, we compare the results of GiMP and GiP in Fig. 6. We can find that the merge method has a significant effect on the performance. The Ising-model formulation of QKP implies that GiP tends to avoid packing more items than the knapsack capacity since the energy is always increased by this operation. On the other hand, GiMP can keep searching for solutions without violating the capacity constraint. This is because GiMP can flip multiple spins to pack and unpack several items simultaneously.

Next, we compare the results of GiMP and Ising machine in Fig. 6. We can observe large differences in the region where the effective time is small. This means that GiMP can find relatively high-quality FSs even in a short time. As the effective time increases, the solution quality is improved both in GiMP and Ising machine, making the differences small.

D. SCALABILITY OF GIMP

In this section, we study the impact of the number of spins N and the number of replicas R on the execution time of GiMP.

First, we evaluate the dependence of the execution time per replica on the number of replicas R at each N. We prepared three Ising models with $N = 800$, $N = 2000$, and $N = 8000$, and run GiMP while varying R in the range of 2^4 to 2^9. We set the number of MC steps S and the merge interval I_M to 10^5 and 10, respectively. Fig. 8 shows the result. In all three models with 800 spins (□), 2000 spins (○), and 8000 spins (Δ), the required time per a single solution decreases as the number of replicas R increases. More specifically, the execution efficiency is improved about 10 times by increasing R from 2^4 to 2^9. When solving a constrained COP, we usually need to obtain a lot of solutions beyond the required number since we must remove infeasible solutions. The scalability of GiMP offers a great advantage in such a situation.

Next, we evaluate the dependence of the execution time on the number of spins N. We added seven Ising models with $N = 128$, 256, 512, 1024, 3000, 5000, 7000 and ran GiMP for ten Ising models. We assumed $R = 512$, $S = 10^5$, and $I_M = 10$ in this evaluation. The result is shown in Fig. 9 with the blue dots (●). In the range of $N < 1000$, the execution time varies proportionally to N (see the dotted line in Fig. 9), whereas it follows a proportion to the square of N for $N > 1000$ (see the dashed line in Fig. 9). We can explain why the scaling changes beyond a border (i.e., beyond $N = 1000$ in this case) as follows. As discussed in Section IV-C2, GiMP has $O(RN)$ and $O(RN^2)$ computation processes. For small N, the $O(RN)$ process dominantly contributes the execution time. However, the weight of the $O(RN^2)$ process grows as

N increases. Note that the border depends on both R and the specification of GPU used to implement GiMP.

VI. CONCLUSION

The recently-proposed merge method is an important technique to enhance the search performance of SA-based Ising machines. However, its sequential process was an obstacle to integrate it into existing Ising-machine hardware. In order to overcome this obstacle, this paper newly proposed the GPU-oriented merge method by formulating the merge method as a series of matrix multiplications using the merging matrix. Then, we developed Merge PSA by integrating the proposed merge method into replica parallel execution of PSA. Finally, Merge PSA was implemented on a GPU and evaluated for QKP instances. Compared to the state-of-the-art GPU-based Ising machine, our proposed Ising machine demonstrated the superiority because high-quality FSs could be obtained in a shorter time. Moreover, we confirmed that the replica parallel execution effectively reduces the time per a single solution.

In this paper, we constructed the merging matrix such that merged spins and their destinations are selected at random. On the other hand, the previous work [18] has demonstrated that putting restrictions on the selection further enhances the search performance for some types of constrained COPs. In the future, we will introduce such restrictions into our GPU-oriented merge method by modifying the generation process of the matrix $\boldsymbol{L}$, and verify our method for a wider range of constrained COPs.

REFERENCES

[1] M. W. Johnson, M. H. S. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk, and E. M. Chapple, "Quantum annealing with manufactured spins," *Nature*, vol. 473, pp. 194–198, May 2011.

[2] C. McGeoch and P. Farré, "Advantage processor overview," D-Wave Syst., Burnaby, BC, Canada, Tech. Rep., 14-1058A-A, 2022.

[3] K. Kawamura, J. Yu, D. Okonogi, S. Jimbo, G. Inoue, A. Hyodo, A. L. García-Arias, K. Ando, B. H. Fukushima-Kimura, R. Yasudo, T. V. Chu, and M. Motomura, "Amorphica: 4-replica 512 fully connected spin 336 MHz metamorphic annealer with programmable optimization strategy and compressed-spin-transfer multi-chip extension," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2023, pp. 42–43.

[4] K. Yamamoto, K. Kawamura, K. Ando, N. Mertig, T. Takemoto, M. Yamaoka, H. Teramoto, A. Sakai, S. Takamaeda-Yamazaki, and M. Motomura, "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 J. Solid-State Circuits*, vol. 56, no. 1, pp. 165–178, Jan. 2021.

[5] T. Takemoto, K. Yamamoto, C. Yoshimura, M. Hayashi, M. Tada, H. Saito, M. Mashimo, and M. Yamaoka, "A 144Kb annealing system composed of 9×16Kb annealing processor chips with scalable chip-to-chip connections for large-scale combinatorial optimization problems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2021, pp. 64–65.

[6] T. Okuyama, T. Sonobe, K.-I. Kawarabayashi, and M. Yamaoka, "Binary optimization by momentum annealing," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 100, no. 1, pp. 1–9, Jul. 2019.

[7] 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 J. Solid-State Circuits*, vol. 51, no. 1, pp. 303–309, Jan. 2016.

[8] Y. Su, T. T. Kim, and B. Kim, "FlexSpin: A scalable CMOS Ising machine with 256 flexible spin processing elements for solving complex combinatorial optimization problems," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, vol. 65, Feb. 2022, pp. 1–3.

[9] 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 Phys.*, vol. 7, p. 48, Apr. 2019.

[10] M. Bagherbeik, P. Ashtari, S. F. Mousavi, K. Kanda, H. Tamura, and A. Sheikholeslami, "A permutational Boltzmann machine with parallel tempering for solving combinatorial optimization problems," in *Proc. Int. Conf. Parallel Problem Solving Nature*, 2020, pp. 317–331.

[11] *Fixstars Amplify AE*. Accessed: Apr. 1, 2023. [Online]. Available: https://amplify.fixstars.com/en/engine

[12] H. Goto, K. Endo, M. Suzuki, Y. Sakai, T. Kanao, Y. Hamakawa, R. Hidaka, M. Yamasaki, and K. Tatsumura, "High-performance combinatorial optimization based on classical mechanics," *Sci. Adv.*, vol. 7, no. 6, pp. 1–9, Feb. 2021.

[13] H. Goto, K. Tatsumura, and A. R. Dixon, "Combinatorial optimization by simulating adiabatic bifurcations in nonlinear Hamiltonian systems," *Sci. Adv.*, vol. 5, no. 4, pp. 1–8, Apr. 2019.

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

[15] A. Lucas, "Ising formulations of many NP problems," *Frontiers Phys.*, vol. 2, no. 5, pp. 1–15, 2014.

[16] F. Glover, G. Kochenberger, and Y. Du, "A tutorial on formulating and using QUBO models," 2018, *arXiv:1811.11538*.

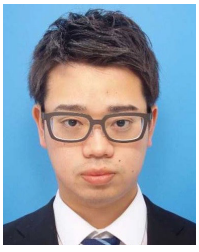
[17] T. Shirai and N. Togawa, "Spin-variable reduction method for handling linear equality constraints in Ising machines," *IEEE Trans. Comput.*, vol. 72, no. 8, pp. 2151–2164, Aug. 2023.

[18] T. Shirai and N. Togawa, "Multi-spin-flip engineering in an Ising machine," *IEEE Trans. Comput.*, vol. 72, no. 3, pp. 759–771, Mar. 2023.

[19] A. Billionnet and E. Soutif. (2005). *(QKP) Instances*. [Online]. Available: https://cedric.cnam.fr/~soutif/QKP/

[20] K. Tamura, T. Shirai, H. Katsura, S. Tanaka, and N. Togawa, "Performance comparison of typical binary-integer encodings in an Ising machine," *IEEE Access*, vol. 9, pp. 81032–81039, 2021.

[21] S. Jimbo, D. Okonogi, K. Ando, T. V. Chu, J. Yu, M. Motomura, and K. Kawamura, "A hybrid integer encoding method for obtaining high-quality solutions of quadratic knapsack problems on solid-state annealers," *IEICE Trans. Inf. Syst.*, vol. 105, no. 12, pp. 2019–2031, 2022.



SATORU JIMBO (Graduate Student Member, IEEE) received the B.E. degree in mechanics from Tohoku University, in 2021, and the M.E. degree in information and communication engineering from Tokyo Institute of Technology, in 2023, where he is currently pursuing the Ph.D. degree. His research interest includes Ising computing.

TATSUHIKO SHIRAI (Member, IEEE) received the B.Sci., M.Sci., and Dr.Sci. degrees from The University of Tokyo, in 2011, 2013, and 2016, respectively. Currently, he is an Assistant Professor with the Department of Computer Science and Communications Engineering, Waseda University. His research interests include quantum dynamics, statistical mechanics, and computational science. He is a member of JPS.

NOZOMU TOGAWA (Member, IEEE) received the B.Eng., M.Eng., and Dr.Eng. degrees from Waseda University, in 1992, 1994, and 1997, respectively, all in electrical engineering. Currently, he is a Professor with the Department of Computer Science and Communications Engineering, Waseda University. His research interests include VLSI design, graph theory, and computational geometry. He is a member of IEICE and IPSJ.

MASATO MOTOMURA (Fellow, IEEE) received B.S., M.S., and Dr.Eng. from Kyoto University, Kyoto, Japan, in 1985, 1987, and 1996, respectively. In 1987, he joined NEC Central Research Laboratories, Kawasaki, Japan, working on various hardware architectures, including approximate text search engines, multi-threaded on-chip parallel processors, computing-in-memory chips, and reconfigurable systems. From 2001 to 2008, he was with NEC Electronics, Kawasaki, Japan, where he led the research and business development of the dynamically reconfigurable processor (DRP) he invented. He was also a Visiting Researcher at the MIT Laboratory for Computer Science, Cambridge, USA, from 1991 to 1992 and a Group Manager of architecture-circuits interdisciplinary research in NEC central laboratory from 2008 to 2011, respectively. In 2011, he changed his position from industry to academia and became a Professor at Hokkaido University, Sapporo, Japan, to cultivate solid-state circuit research activities with younger generations. Later he became a Professor at Tokyo Institute of Technology (TokyoTech), Yokohama, Japan, in 2019, where he established and has been leading the artificially intelligent computing (ArtIC) research unit. Since 2011, he has been actively working on reconfigurable and parallel architectures for deep neural networks, machine learning, annealing machines, and general intelligent/domain-specific computing. His group has published "AI chip" papers almost every year at ISSCC and symposium on VLSI since 2017. He is a member of IEICE, IPSJ, JSAI, and EAJ. He received the IEEE JSSC Annual Best Paper Award in 1992, the IPSJ Annual Best Paper Award in 1999, and the IEICE Achievement Award in 2011, respectively. He was also awarded Ichimura Academic Award and Yamasaki Award in 2022 for his leadership in developing and productizing DRP technology (a series of DRP-based microcontroller products are now produced by Renesas Electronics), as well as the accumulation of AI-chip achievements in recent years. He is a 2022 IEEE Fellow (SSCS) for contributions to memory-logic integration of reconfigurable chip architecture.

KAZUSHI KAWAMURA (Member, IEEE) received the B.Eng., M.Eng., and Dr.Eng. degrees from Waseda University, in 2012, 2013, and 2016, respectively, all in computer science. From 2018 to 2019, he was an Assistant Professor with the Department of Communications and Computer Engineering, Waseda University. Currently, he is a Specially Appointed Assistant Professor with the Institute of Innovative Research, Tokyo Institute of Technology. His research interests include Ising computing and LSI design methodologies. He is a member of IEICE and IPSJ.