A New Parallel Algorithm for Monte Carlo Simulations of 2D Ising, Potts and XY Models

Tian-Shen He[1,*] and Yan Chen[1,2,†]

[1] *Department of Physics, State Key Laboratory of Surface Physics and Laboratory of Advanced Materials, Fudan University, Shanghai 200433, China*
[2] *Collaborative Innovation Center of Advanced Microstructures, Nanjing 210093, China*

(Dated: 29 May 2021)

The Swendsen–Wang (SW) multi-cluster spin flip algorithm is closely related to connected component labeling and analysis (CCL & CCA) algorithms in computer vision. Recently, a new parallel algorithm: Hardware Accelerated 4-connected CCL & CCA (HA4) was proposed. It utilizes Graphical Processing Units (GPUs) and the Common Unified Device Architecture (CUDA) to conduct CCL & CCA efficiently, and surpasses all previous CCL & CCA algorithms. However, the HA4 identifies only and all adjacent set pixels to be in one component, whereas in the SW algorithm bonds are generated between neighbor spin pairs with a probability. We present a new algorithm that combines the HA4 with the SW, with application to Monte Carlo simulations of 2D Ising and Potts models. The algorithm naturally absorbs periodic boundary conditions (PBC) into consideration. In simulating a 32×32 square lattice Ising model with a NVIDIA RTX 2060 graphic card, compared with a currently prevalent CUDA C++ package based on Komura equivalence algorithm, ours is approximately 2.3 times faster.

CONTENTS

I. INTRODUCTION

In computational physics, Monte Carlo methods form the largest and most important class of numerical methods used for solving statistical physics problems [1]. The Metropolis algorithm [2] with a single spin flip has been a success as a standard method of simulation of many-particle systems. However, as we approach the critical temperature, the single-spin-flip algorithm often suffers from the problem of slow dynamics or the critical slowing down; that is, the combination of large critical fluctuations and long correlation time makes the errors on measured quantities grow enormously. To overcome this difficulty, multi-cluster [3] and single-cluster [4] spin flip algorithms based on the Fortuin and Kasteleyn [5] representation have been proposed. By flipping similarly oriented spins in their entirety all at a time, the undesirable effects mentioned above are greatly reduced.

The advance in computer technology promotes the development of computational physics. Parallel computation utilizing graphics processing units (GPUs) is extensively applied to reduce processing times in scientific computations, computer science and finance [6]. Using the common unified device architecture (CUDA) released by NVIDIA [7], it is now easy to implement parallel algorithms on GPUs using standard C or C++ language with CUDA specific extension. Furthernore, pseudo-random number generators (PRNGs) have developed enormously [8], and it's no longer troublesome to generate random numbers in Monte Carlo procedures on GPUs.

Among all cluster flip algorithms, the Swendsen–Wang [3](SW) multi-cluster spin flip algorithm is considered to be the best candidate for parallel transformation [9][10], as it is intrinsically a variant of connected component labeling and analysis (CCL & CCA), which is a central algorithm in computer vision. Parallel algorithms for CCL & CCA have undergone many updates [11], the most recent ones being the Komura equivalence [12][13], the Cabaret distanceless label propagation[14], the Playne equivalence[15], and the Hardware Accelerated 4-connected CCL & CCA (HA4) [16][17]. The HA4 is so far the fastest, but its unique structure makes it impratical to be directly em-

* e-mail: jerryhts@gmail.com
† e-mail: yanchen99@fudan.edu.cn

ployed in the SW.

In this paper, we propose a new parallel multi-cluster flip algorithm for a single GPU that combines the SW with the HA4. In Section II, we briefly describe the SW multi-cluster spin flip algorithm for the Ising and Potts model and its correspondence with CCL & CCA. In Section III we explain the HA4 algorithm in computer vision. In Section IV, we describe the new cluster flip algorithm on a single GPU. In Section V, we compare the performance of the new cluster flip algorithm with a currently prevalent CUDA package based on Komura equivalence. Finally, we present a summary and discussion in Section VI.

Algorithm 1 Modified distance operators for 32-bit bitmasks

```
__device__ function m_start_distance (pixels, tx)
    return __clz(∼ (pixels << (32 − tx)))
end function
__device__ function m_end_distance (pixels, tx)
    a = __ffs(∼ (pixels >> tx))
    if a then
        return a
    else
        return 32 − tx
    end if
end function
```

II. SWENDSEN–WANG CLUSTER ALGORITHM, CONNECTED COMPONENT LABELING AND ANALYSIS

The Hamiltonian of Ising and Potts models is given by

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \delta_{S_i,S_j}, \quad S_i = 1, 2, \ldots, q \tag{1}$$

and for $q = 2$ this corresponds to the Ising model. Here, J is the coupling constant and S_i is the Potts spin on the lattice site i. The summation is taken over the nearest neighbor pairs $\langle i, j \rangle$. Periodic boundary conditions (PBC) are employed.

The Hamiltonian of the classical XY model, a continuous spin model, is given by

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} \boldsymbol{s}_i \cdot \boldsymbol{s}_j = -J \sum_{\langle i,j \rangle} \cos\left(\theta_i - \theta_j\right) \tag{2}$$

where $\boldsymbol{s}_i$ is a planar unit vector with two real components, $(\cos\theta_i, \sin\theta_i)$. For actual implementation, the value of θ_i is discretized as $2\pi p_i/q$ with $p_i = 1, 2, \ldots, q$. This discretized model is referred to as the q -state clock model. When q tends to infinity, the clock model becomes the classical XY model. To make a cluster flip of vector spins, we can use the idea of embedded cluster introduced by Wolff [4], and project vector spins to form Ising spin clusters.

Assign each nearest neighbor pairs $\langle i, j \rangle$ a bond variable n_{ij}, and one Monte Carlo step (MCS) of the spin-update process of the SW cluster algorithm can be formulated as three following steps [18][19]:

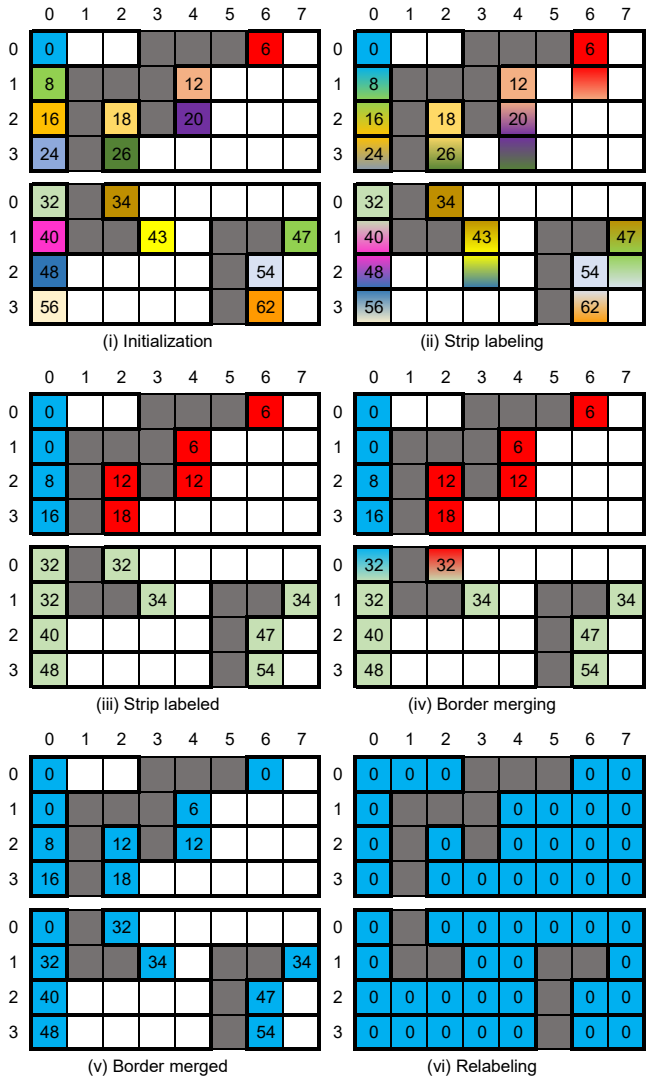

FIG. 1. Example of the HA4 algorithm on an 8×8 image divided into two strips of height 4. (Adapted from Fig.4 in Ref.[16]) Colored and white squares represent pixels with data 1, and grey ones with data 0. Number written on a square denotes its label. In (i), each segment start is initialized with its linear address. In (ii), local equivalences are resolved for each strip. In (iv), we merge the equivalence trees of the two strips. Finally, in (vi), each segment start finds the root of its tree and shares it with the other threads of the segment for relabeling.

1. Set $n_{ij} = 0$ if $S_i \neq S_j$, or assign values $n_{ij} = 1$ and 0 with probability p and $1 - p$, respectively, if $S_i = S_j$, where $p = 1 - e^{-\beta J}$, $\beta = 1/kT$ the inverse reduced temperature, k the Boltzmann constant and T the temperature,

2. Identify clusters of spins that are connected by "active" bonds ($n_{ij} = 1$),

3. Draw a random value $(1, 2, \ldots, q)$ with equal probability $1/q$ (including one-site clusters), and assign the value to all spins in a cluster.

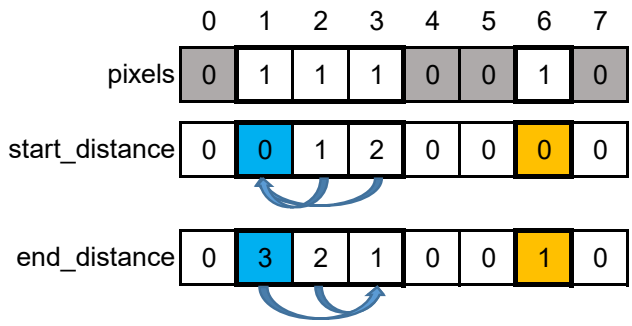

FIG. 2. Example of the original distance operators in the HA4 on a 8-bit bitmask. (Adapted from Fig.2 in Ref.[16])

The second step: identifying clusters of spins connected by active bonds, corresponds directly to CCL & CCA. The CCL problem is an algorithmic application of graph theory that considers a graph of nodes and assigns a unique label for each subset of connected vertices. Many applications consider datasets with regular structures instead of arbitrary graphs. The connections between nodes are often defined by a function of the node data rather than from a set of edges. These features match perfectly with regular lattices and bonds.

Concerning a whole imaging project, CCL algorithms are usually followed by post-processing which computes some features such as the number of pixels in a component, which natually fit in the calculation of total magnetization. Full labeling is required for human visualization, which is homologous with spin value assignment. The post-processing of CCL is called Connected Component Analysis (CCA).

III. HARDWARE ACCELERATED 4-CONNECTED CONNECTED COMPONENT ANALYSIS ALGORITHM

We will not go through all the CUDA terminologies (kernel, device, thread, warp, block, grid, atomic operations, bank conflicts) and the technical details of the HA4, please refer to Ref.[16] for a more comprehensive explanation of the algorithm and its pesudocode. In general, the HA4 algorithm consists of 4 steps: initialization, strip labeling, border merging and relabeling, as illustrated in Fig.1. We use a similar structure with modifications in our algorithm. It is worth mentioning that the HA4 deals with 4-connected binary image; that is, the data of every pixel is either zero or one, and connectivity is defined between only and all pairs of ones that are horizontal or vertical closest neighbors. In addition, the width of the image is required to be a multiple of 32. We name the following variables:

- $BLOCK\_W, BLOCK\_H$: The dimensions of a block (We choose them to be 32 and 4, repectively, as suggested in Ref.[16] to optimize performance.)
- bx, by: The block indices in the grid
- tx, ty: The thread indices in a block
- x, y: The column/row indices in the image

The image is devided into horizontal strips of rows, and in each row, the HA4 identifies consecutive ones (refered to as a segment) and labels each segment start with its linear address: $adr = y \times width + x$. Notice that x, y, adr all start from zero. Subsequently, the HA4 vertically check for segments that can be merged by means of label equivalence algorithm proposed in Ref.[12], first within strips then between strips, and it is also done only at segment starts. Eventually, each segment start delegates the task of finding the equivalence tree's root, which is the final label, and propagates it to the rest of the segment. Component features are calculated as by-products within the algorithm.

Essentially, the HA4 derives its efficiency from strip devision and CUDA warp-level primitives [21]. The former minimizes the number of computationally expensive atomic operations, which is the major drawback of previous parallel CCL algorithms (so-called State-of-the-art algorithms), and the latter enables rapid data transportation within a warp, which avoids unnecessary memory accesses and potential bank conflicts.

IV. A NEW PARALLEL MULTI-CLUSTER FLIP ALGORITHM

In this section, we explain why the HA4 can not be direcly employed in the SW algorithm, and demonstrate the new parallel multi-cluster flip algorithm.

IV.1. Limitations of the HA4

Owing to the fact that the HA4 recognizes all and only "1"s, one can attribute "1" neither to spins with a certain spin number nor to spins to which bonds are connected. The former practice neglects bonds between spins of other spin numbers, and both practices ignore the probability of consecutive "1"s belonging to different clusters due to inactive bonds. Expressly, the local and stochastic properties of bond formation make it arduous to propose a general criterion regarding whether a spin belongs to a cluster or not, which is unfortunately the premise of the HA4.

Our new algorithm is capable of overcoming this difficulty.

IV.2. Modified distance operators

The HA4 defines two distance operators: `start_distance` and `end_distance` (see Fig.2), which serve to obtain the distance of a pixel to the start or end of its segment via bitwise operations. We propose

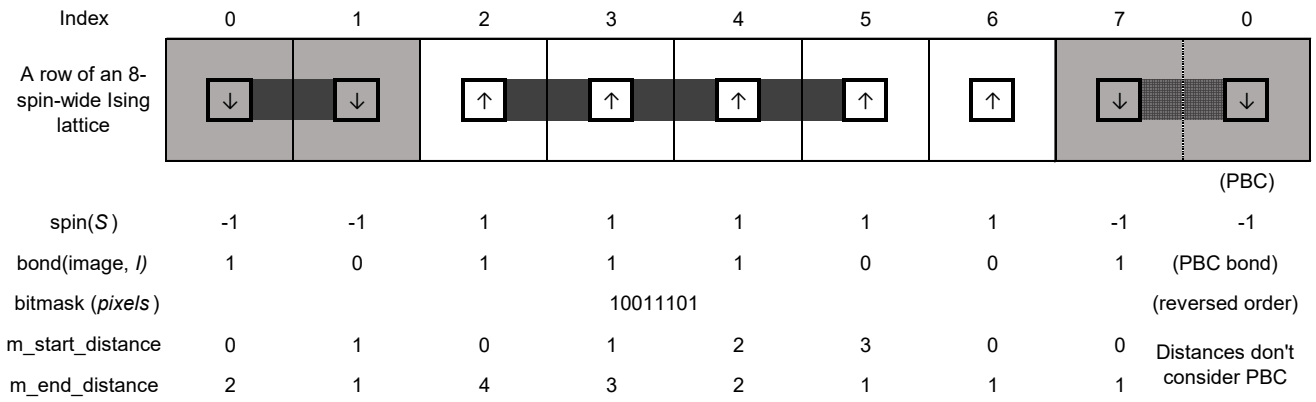

FIG. 3. Example of modified distance operators on a 8-bit bitmask derived from a row of a 8-spin-wide Ising lattice. (Style adapted from Fig.1 in Ref.[20]) Notice that the leftmost spin with index 0 ends up in the lowest digit of the bitmask, resulting in a reverse in order.

letting the activation status of horizontal bonds, rather than spin numbers, play the role of image data, and modify these distance operators correspondingly (see Algo.1). The `__clz`(*Count Leading Zeros*) intrinsic returns the number of consecutive zeros starting from the most significant bit and going down inside a 32-bit register. The `__ffs`(*Find First Set*) intrinsic returns the position of the first bit set to one, starting from the least significant bit and going up inside a 32-bit register. The "∼" stands for bit inversion. The "<<" and the ">>" are bit-shift operators. They move each digit in a number's binary representation left and right, repectively.

We define the size of lattice to be $n \times n$, and allocate two-dimensional arrays $I_{y,x}$ and $L_{y,x}$ (x, y both start from zero) to denote respectively the image data and label of the pixel (in our case, spin) in the y^{th} row and x^{th} column, in accordance with Ref.[16]. For each spin position (y, x), $I_{y,x}$ stores its bond with its right-hand side neighbor, except for the rightmost spins, where the PBC bonds with the leftmost spins are stored instead (see Fig.3). We individually judge whether a bond is active or not by comparing a newly generated random number, which distributes uniformly between zero and one, with the probability $p = 1-e^{-\beta J}$. By doing so, inactive bonds directly seperate different horizontal segments. Besides, distance operators are altered, identifying the first "0" following a segment of "1"s to be part of that segment. In order to distinguish the concept of "segment" (successive "1"s) in the bitmask from connected spins, we allude to the latter with "a fragment". Just like the HA4, we utilize the `__ballot_sync` instruction to obtain a 32-bit bitmask where the i^{th} bit is "1" if the i^{th} bond is active, and act distance operators upon bitmasks to avoid bank conflicts.

The new distance operators possess useful properties. For the start of a fragment `start_distance` is always equal to zero, and `end_distance` is always equal to the number of spins in the fragment. Moreover, unlike the HA4 which ignores isolated "0" pixels, isolated spins are treated as one-spin fragments.

IV.3. New merging strategy

Similar to the HA4, the lattice is separated into horizontal strips of rows, and each one is attributed to a block. In order to tackle lattice with arbitary size, we use the grid-stride loop [22] to design flexible kernels. This pattern also brings the benefit of thread use which amortizes the cost of shared data initialization along with threads creation and destruction.

In the initialization process we label every fragment start with its linear address, including one-spin fragments. Thenceforth, we vertically seek for fragments between which bonds are connected so that they can be merged. This process is only done at fragment starts. We introduce another variable: V, which stands for vertical bonds, and it is also a $n \times n$ array. However, in contrast to horizontal bonds, we temporarily assume that $p = 1$, which guarantees that all vertical bonds are active between spins with the same spin numbers. For each spin position (y, x), $V_{y,x}$ stores its bond with its upward side neighbor, except for spins in the uppermost row, where the PBC bonds with the bottommost spins are stored instead (see Fig.4). V is precalculated in the initialization process. When trying to merge two vertically adjacent fragments, we look in the horizontal overlap for the horizontal index where at least one fragment is at its fragment start, and check the bond at this horizontal index between these two fragments. If the bond turns out active, it actually signifies that all spins in these two fragments share the same spin number, and vice versa. We now make allowances for $p = 1-e^{-\beta J}$. We define *overlap* to be the number of spins in the horizontally intersecting region, and it can be easily deduced that the probability for that merger to happen is $p_{merge} = 1 - (1-p)^{overlap}$, because it only requires at least one of the "overlapping" bonds to be active. Fortunately, *overlap* can also be derived from distance operators.

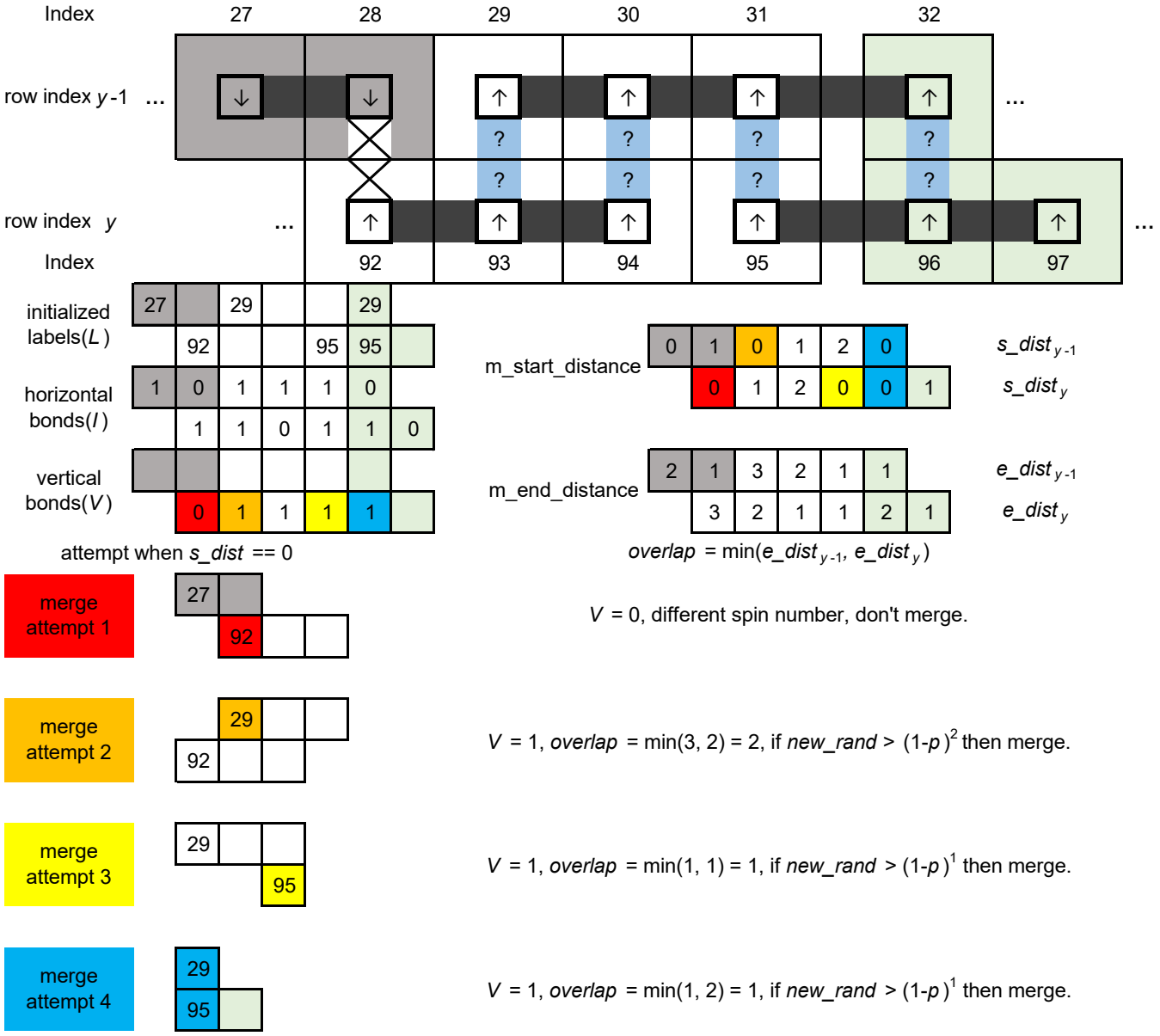

FIG. 4. Example of the new merging strategy on part of a 64×64 Ising lattice. The separation between spins is due to $BLOCK\_W = 32$.

One needs to be cautious at the edge of blocks. In the HA4, the strip labeling kernel passes two variables: $distance_{y-1}$ and $distance_y$ (see Fig.3 of Ref.[16]). As the grid-stride loop moves the block towards the right-hand side, these two variables pass the proper value of `start_distance` to the leftmost pixel of the new block. They help the new block to locate the true segment starts in the previous block, and label them in the new block correctly. If $distance_{y-1}$ and $distance_y$ of two rows are both greater than zero, it reveals that these two segments have already been merged, and therefore they don't need to be merged again. In our kernel, we keep these two variables for labeling, but we maintain the `start_distance` of the leftmost pixels in a block to be zero, so that *distance* doesn't interfere the calculation of *overlap* and merger probability. After strip labeling is done, border merging between strips are performed with similar fashion to fragment merging.

An example of the new merging strategy is elucidated in Fig.4. Thanks to this strategy, the CCL procedure is now mathematically equivalent to the multi-spin cluster finding in the SW, but tailored to the parallel implementation in the style of the HA4.

IV.4. PBC on arbitary size lattices

For horizontal PBC, one simply merges the two fragments containing the leftmost and the rightmost spin if there's an active bond connecting the two spins. Vertical PBC is included in the border merging process, where the strip containing the uppermost row seeks to merge with the bottommost strip.

The HA4 requires image width to be a multiple of 32 in order for warp primitives to work correctly. Thanks to PBC, we no longer impose such restrictions on the size of the lattice. For lattices with sizes that are not multiples of 32, we simply find the smallest multiples of 32 greater than the sizes, and execute the algorithm on the larger lattice. One only need to make sure that the extra spins are not connected to the ones that are actually being considered, and adjust the PBC instructions correspondingly. It can be deduced that such expansion of the lattice doesn't brings extra computational time, as no extra mergers are executed.

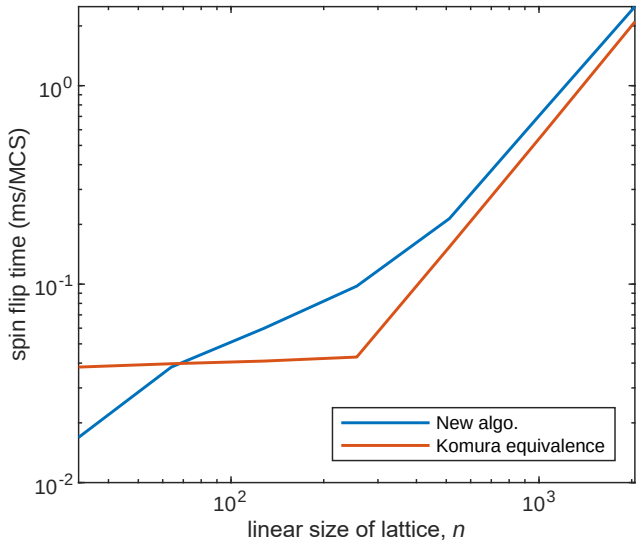

FIG. 5. Spin flip time comparison between the package in [13] and ours. The spin flip time on 32×32 lattices, where the grid-stride loop iterates for only once, best reflects the speed-up of our algorithm.

IV.5. Spin flip as relabeling and iterative CCA

In the last step of the HA4, the final label of pixels are propagated from segment starts via the `__shfl_sync` instruction. We substitute the label being propagated with the new spin number of that label. Thus the entire CCL process has completed and it equates with one MCS of the SW.

As is previously mentioned, the calculation of physical quantity (magnetization, energy, etc.) is relevant to CCA of clusters. However, as many iterations are required to accomplish the SW, we don't perform the CCA simutaneously with CCL. Instead we collect the information of magnetization and energy at the beginning of the next MCS, alongside with the bond initialization process.

V. EXPERIMENTAL EVALUATION

We wrote a CUDA C++ package and compare it to a currently prevalent CUDA C++ package based on Komura equivalence algorithm [13]. We use square lattice Ising models as test materials and the test environment is a NVIDIA RTX 2060 graphic card (see Fig.5). The performance on Potts and q-state XY models are basically identical as the algorithm treat these three models with the same fashion.

It is well worth mentioning that the package in [13] uses fixed block numbers, which cuts the computational time compared to grid-stride loops, but limits the number of spins to be less than the maximum grid dimension times the maximum block dimension. On the contrary, our package gives up some computational speed in exchange for the capability to process arbitarily large lattices. When the linear size of the lattice is no greater than the size of a warp, which is 32, the grid-stride loop doesn't take effect, and thereby the speed-up of our algorithm is clearly revealed.

VI. SUMMARY AND OUTLOOK

We presented a new algorithm that integrates the SW with the HA4 to boost parallel Monte Carlo simulation of 2D Ising and Potts model. Compared with a currently prevalent CUDA C++ package based on Komura equivalence, ours is dramatically faster in small lattices, and is able to perform simulations on arbitarily large lattices.

Our algorithm can be easily extended to triangular or hexagonal lattices, or square lattices with frustration, whereas the extension to 3D cases might be rather nontrivial [23].

AUTHOR CONTRIBUTIONS

Y. C. supervised the project. T.-S. H. conceived the project, designed the algorithm, wrote the CUDA C++ package, conducted the performance tests and wrote the paper.

CODE AVAILABILITY

The codes that support the plots within this paper and other findings of this study are publicly available from the corresponding author on reasonable request.

[1] M. Newman, G. Barkema, and I. Barkema, *Monte Carlo Methods in Statistical Physics* (Clarendon Press, 1999).

[2] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, Equation of state calculations by fast computing machines, J. Chem. Phys. **21**, 1087 (1953).

[3] R. H. Swendsen and J.-S. Wang, Nonuniversal critical dynamics in monte carlo simulations, Phys. Rev. Lett. **58**, 86 (1987).

[4] Wolff and Ulli, Collective monte carlo updating for spin systems, Phys. Rev. Lett. **62**, 361 (1989).

[5] C. M. Fortuin and P. W. Kasteleyn, On the random-cluster model: I. introduction and relation to other models, Physica **57**, 536 (1972).

[6] Y. Cai and S. See, *GPU Computing and Applications*, EBL-Schweitzer (Springer Singapore, 2014).

[7] S. Cook, *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*, 1st ed. (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2012).

[8] V. Kindratenko, *Numerical Computations with GPUs* (Springer International Publishing, 2014).

[9] Y. Komura and Y. Okabe, Gpu-based swendsen–wang multi-cluster algorithm for the simulation of two-dimensional classical spin systems, Comput. Phys. Commun. **183**, 1155 (2012).

[10] Y. Komura and Y. Okabe, Cuda programs for the gpu computing of the swendsen–wang multi-cluster spin flip algorithm: 2d and 3d ising, potts, and xy models, Comput. Phys. Commun. **185**, 1038 (2014).

[11] L. He, X. Ren, Q. Gao, X. Zhao, B. Yao, and Y. Chao, The connected-component labeling problem: A review of state-of-the-art algorithms, Pattern Recognition **70**, 25 (2017).

[12] Y. Komura, Gpu-based cluster-labeling algorithm without the use of conventional iteration: Application to the swendsen–wang multi-cluster spin flip algorithm, Comput. Phys. Commun. **194**, 54 (2015).

[13] Y. Komura and Y. Okabe, Improved cuda programs for gpu computing of swendsen–wang multi-cluster spin flip algorithm: 2d and 3d ising, potts, and xy models, Comput. Phys. Commun. **200**, 400 (2016).

[14] L. Cabaret, L. Lacassagne, and D. Etiemble, Distanceless label propagation: An efficient direct connected component labeling algorithm for gpus, in *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)* (2017) pp. 1–6.

[15] D. P. Playne and K. Hawick, A new algorithm for parallel connected-component labelling on gpus, IEEE Transactions on Parallel and Distributed Systems **29**, 1217 (2018).

[16] A. Hennequin, L. Lacassagne, L. Cabaret, and Q. Meunier, A new direct connected component labeling and analysis algorithms for gpus, in *2018 Conference on Design and Architectures for Signal and Image Processing (DASIP)* (2018) pp. 76–81.

[17] F. Lemaitre, A. Hennequin, and L. Lacassagne, How to speed connected component labeling up with simd rle algorithms, in *Proceedings of the 2020 Sixth Workshop on Programming Models for SIMD/Vector Processing*, WPMVP'20 (Association for Computing Machinery, New York, NY, USA, 2020).

[18] W. Janke, Monte carlo simulations of spin systems, in *Computational Physics* (Springer, 1996) pp. 10–43.

[19] D. P. Landau and K. Binder, *A Guide to Monte Carlo Simulations in Statistical Physics*, 2nd ed. (Cambridge University Press, 2005).

[20] F. Wende and T. Steinke, Swendsen-wang multi-cluster algorithm for the 2d/3d ising model on xeon phi and gpu, in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (2013) pp. 1–12.

[21] V. G. Yuan Lin, Using cuda warp-level primitives (2018).

[22] M. Harris, Cuda pro tip: Write flexible kernels with grid-stride loops (2013).

[23] F. Lemaitre, A. Hennequin, and L. Lacassagne, Taming voting algorithms on gpus for an efficient connected component analysis algorithm, in *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2021) pp. 7903–7907.