

ISC High Performance 2022: Accelerating Simulated Quantum Annealing with GPU and Tensor Cores

Presenter: Yi-Hua Chung

Advisor: Shih-Hao Hung

Yi-Hua Chung, Cheng-Jhih Shih, Shih-Hao Hung

/

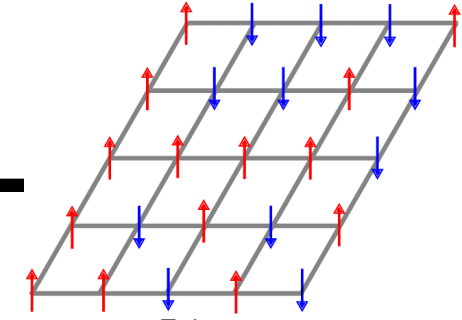
Department of Computer Science & Information Engineering, National Taiwan University

Yi-Hua Chung, ro9944072@csie.ntu.edu.tw

- Introduction
 - Ising Model
 - Simulated Quantum Annealing (SQA)
 - Acceleration Strategy Adopted by Tohoku University (SOTA)
- Methodology
 - Hierarchical Update (HU)
 - Utilizing the Tensor Cores
- Performance Evaluation
 - Five Experiments
- Conclusion

3

Ising Model

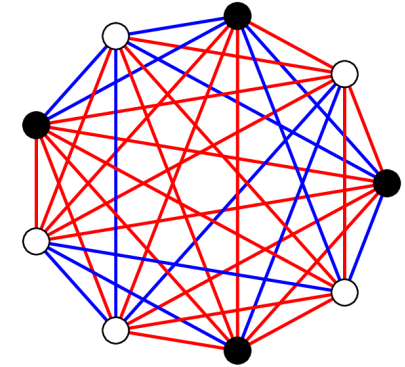
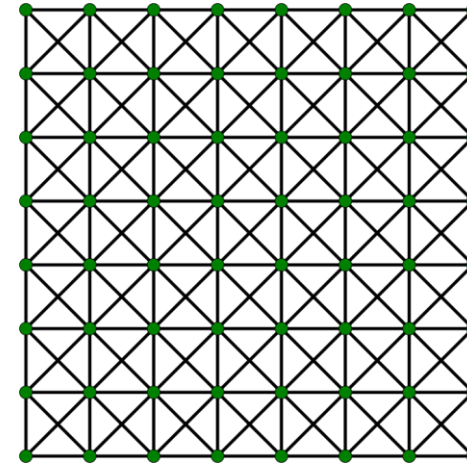
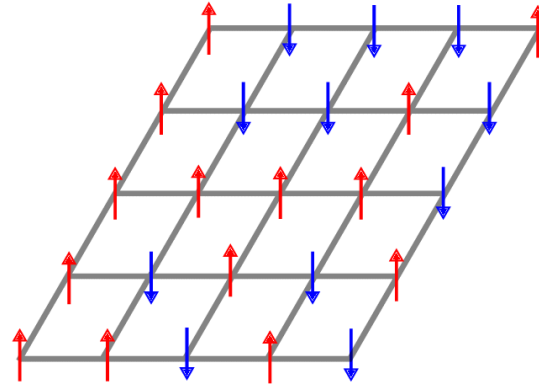
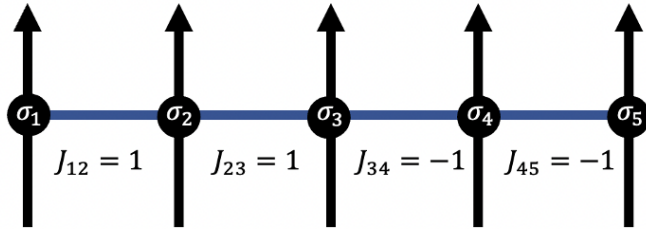


- Definition of Ising model
 - The Ising model is a mathematical model that doesn't correspond to an actual physical system. It's a huge lattice with N spins (σ_i), where each spin can be in one of two states ($+1, -1$)
 - The interaction terms between the connected spins are called **couplings** (J_{ij}). Besides, we can formulate the Hamiltonian of Ising model as:

$$H(\sigma) = -J \sum_{\langle ij \rangle} \sigma_i \sigma_j - h \sum_j \sigma_j.$$

- When solving the Ising problem, our goal is to find a set of spin arrangements that minimize the total energy of the system
 - **It is a combinatorial problem (Ising model: NP-Complete)**
- By reducing NPC to Ising model, we can solve some NP-hard problems by solving the Ising model
- Generally, the Ising problem is currently solved in a heuristic. One of the heuristic algorithms, “**Simulated Quantum Annealing**,” is an algorithm for solving combinatorial optimization problems

$$H(\sigma) = -J \sum_{\langle ij \rangle} \sigma_i \sigma_j - h \sum_j \sigma_j.$$

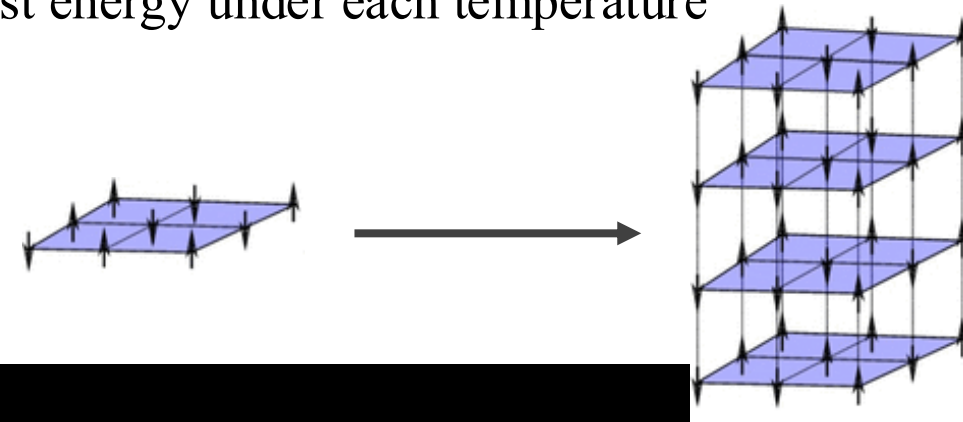


- There are many different Ising models. Different Ising models have different degrees of connectivity
- The higher spin connectivity for an Ising model, the more difficult for it to be accelerated
- In our work, we accelerate solving the **fully-connected Ising model** by adopting **simulated quantum annealing (SQA)** algorithm

5

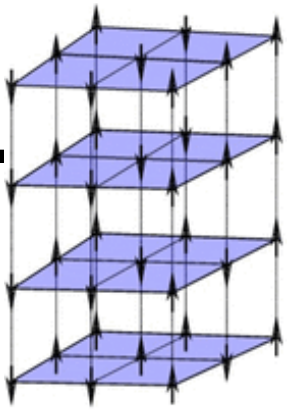
Simulated Quantum Annealing (SQA)

- Simulate quantum annealing (SQA) is a quantum-inspired algorithm
 - To simulate the quantum phenomenon, SQA duplicates original Ising model into M copies. This multi-trotter Ising model is called a transverse field Ising model (TFIM)
 - Different from the original Ising model, each spin has the connections with the spin in the same position located at the previous and the next trotter of TFIM
- A path integral quantum Monte Carlo method
 - Given an initial temperature $T(0)$ and magnetic field $\Gamma(0)$, and random generate a set of spin configuration initially
 - As time increases (MC-step increases), the value of $T(t)$ will gradually decrease. We make the spins flip and try to find a spin configuration with the lowest energy under each temperature (reach the equilibrium with the environment)



Spin Energy of TFIM

$$H(\sigma) = -J \sum_{\langle ij \rangle} \sigma_i \sigma_j - h \sum_j \sigma_j.$$



- Hamiltonian for the Fully-connected TFIM ^{Ref-1, Ref-2} :

$$H_{TFIM} = -\sum_{m=1}^M \left(\sum_{(i,j)} \frac{J_{ij}}{M} \sigma_{i,m} \sigma_{j,m} + \sum_{(i)} \frac{h_i}{M} \sigma_{i,m} - J^{\dagger} \sum_{(i)} \sigma_{i,m} (\sigma_{i,m-1} + \sigma_{i,m+1}) \right),$$

$$\text{for } J^{\dagger} = -\frac{T}{2} * \ln \left(\frac{\coth(\Gamma(t))}{MT} \right) \in \text{const (under a fixed MC-step)}$$

- By equation above, we know the spin energy for $\sigma_{i,m}$, the i -th spin on m -th trotter is:

$$\text{spin_energy}_{i,m} = -\left(\sum_{j=1}^N \frac{J_{ij}}{M} \sigma_{i,m} \sigma_{j,m} + \frac{h_i}{M} \sigma_{i,m} - J^{\dagger} \sigma_{i,m} (\sigma_{i,m+1} + \sigma_{i,m-1}) \right)$$

- Let $J^{\dagger'} = J^{\dagger} / M$:

$$\text{spin_energy}_{i,m} = -\sigma_{i,m} \left(\sum_{j=1}^N J_{ij} \sigma_{j,m} \right) - \sigma_{i,m} h_i + J^{\dagger'} \sigma_{i,m} (\sigma_{i,m+1} + \sigma_{i,m-1})$$

- Most of the problem can be solved by setting $h_i = 0$

$$\text{spin_energy}_{i,m} = -\sigma_{i,m} \left(\sum_{j=1}^N J_{ij} \sigma_{j,m} \right) + J^{\dagger'} \sigma_{i,m} (\sigma_{i,m+1} + \sigma_{i,m-1})$$

- We can determine the delta energy caused by a spin-flip by knowing the spin energy for each spin

Algorithm 1: SQA Algorithm

```
1 Coupling Matrix ← Couplings Data
2 Randomly initialize  $\sigma \in \{-1, 1\}$ 
3 local_field ← 0
4 // Construct local-field energy
5 for  $m = 1$  to  $M$  do
6   for  $i = 1$  to  $N$  do
7     for  $j = 1$  to  $N$  do
8       | local_field $_{i,m}$  +=  $J_{i,j} * \sigma_{j,m}$ 
9     end
10  end
11 end
```

```
12 for  $t = 1$  to MC-STEP do
13   for  $i = 1$  to  $N$  do
14     for  $m = 1$  to  $M$  do
15        $\Delta H = \sigma_{i,m}(\text{local\_field}_{i,m} - J^\dagger(\sigma_{i,m+1} + \sigma_{i,m-1}))$ 
16       // Judge_to_Flip
17       if  $e^{\frac{-\Delta H}{T}} > \text{random}()$  then
18          $\sigma_{i,m} = -\sigma_{i,m}$ 
19       // Update local-field energy
20       for  $j = 1$  to  $N$  do
21         | local_field $_{j,m}$  +=  $2 * J_{i,j} * \sigma_{i,m}$ 
22       end
23     end
24   end
25 end
26 end
```

Initialization

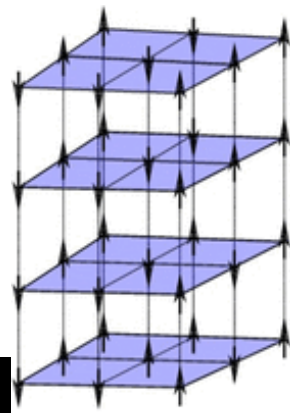
- Initialization

1. Random generate N spin states for M trotters
2. Prepare the local-field for all the spins

- From the previous page

$$\text{spin_energy}_{i,m} = -\sigma_{i,m} \left(\sum_{j=1}^N J_{ij} \sigma_{j,m} \right) + J^{\dagger'} \sigma_{i,m} (\sigma_{i,m+1} + \sigma_{i,m-1})$$

local-field



Implementation of SQA Algorithm (1)

Algorithm 1: SQA Algorithm

```
1 Coupling Matrix  $\leftarrow$  Couplings Data
2 Randomly initialize  $\sigma \in \{-1, 1\}$ 
3 local_field  $\leftarrow 0$ 
4 // Construct local-field energy
5 for  $m = 1$  to  $M$  do
6   for  $i = 1$  to  $N$  do
7     for  $j = 1$  to  $N$  do
```

$$spin_energy_{i,m} = -\sigma_{i,m} \left(\sum_{j=1}^N J_{ij} \sigma_{j,m} \right) + J^{\dagger} \sigma_{i,m} (\sigma_{i,m+1} + \sigma_{i,m-1})$$

```
12 for  $t = 1$  to MC-STEP do
13   for  $i = 1$  to  $N$  do
14     for  $m = 1$  to  $M$  do
15        $\Delta H = \sigma_{i,m} (\text{local\_field}_{i,m} - J^{\dagger} (\sigma_{i,m+1} + \sigma_{i,m-1}))$ 
16       // Judge_to_Flip
17       if  $e^{\frac{-\Delta H}{T}} > \text{random}()$  then
18          $\sigma_{i,m} = -\sigma_{i,m}$ 
19         // Update local-field energy
20         for  $j = 1$  to  $N$  do
21           | local_field $_{j,m} += 2 * J_{i,j} * \sigma_{i,m}$ 
22         end
23       end
24     end
25   end
26 end
```

Implementation of SQA Algorithm (1)

8

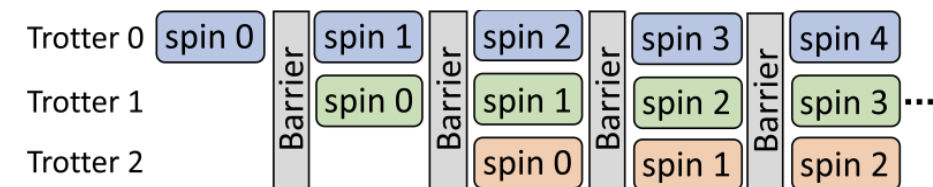
- We can only flip a spin in a time
 - Since all the spins on the same trotter are connected, we can only flip a spin in a time
 - Besides, if a spin flip, all the local-field of the spins on the same trotter need to be update

Update local-field

- $local_field_{k,m} = \sum_{j=1 \neq i}^N J_{k,j} \sigma_{j,m} + J_{k,i} \sigma_{i,m}$
- As k -th spin flip on m -th trotter for i -th spin-flip:
 - $local_field_{k,m}' = \sum_{j=1 \neq i}^N J_{k,j} \sigma_{j,m} + J_{k,i} \sigma_{i,m}'$
 - $local_field_{k,m}' - local_field_{k,m}$
 $= (\sum_{j=1 \neq i}^N J_{k,j} \sigma_{j,m} + J_{k,i} \sigma_{i,m}') - (\sum_{j=1 \neq i}^N J_{k,j} \sigma_{j,m} + J_{k,i} \sigma_{i,m})$
 - $= J_{k,i} \sigma_{i,m}' - J_{k,i} \sigma_{i,m}$
 - $= J_{k,i} \sigma_{i,m}' - J_{k,i} (-\sigma_{i,m}') = 2 * J_{k,i} \sigma_{i,m}'$

For $e^{\frac{-\Delta H}{T}}$: (IEEE international conference on rebooting computing (ICRC) "An ising computer based on simulated quantum annealing by path integral monte carlo method")

- Acceleration are needed, state of the art (SOTA, both conducted by Tohoku University)
 - **GPU-based:** A GPU-based quantum annealing simulator for fully-connected Ising models utilizing spatial and temporal parallelism (2020, IEEE Access)
 - **FPGA-based:** Highly-parallel FPGA accelerator for simulated quantum annealing (2019, IEEE Transactions)
- Parallel the calculation of updating local-field by using accelerators (GPU or FPGA)
- Ideas proposed by Tohoku University
 - Parallel processing of spins belonging to three different trotters

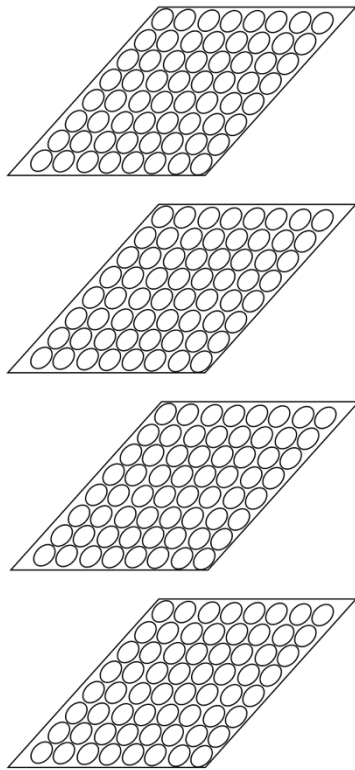


- **Hierarchical Update** - Re-forming the SQA algorithm to implement the algorithm more parallelly
- **Utilizing the Tensor Cores** - Not only **GPU** is used, **Tensor Cores (TCs)** provided by NVIDIA are also used to accelerate the algorithm in our work

11

Hierarchical Update (HU)

- Different from the original SQA, which needs to update N local field after a spin-flip, HU will **only update the local-field of some spins** but all the spins on the same trotter



$$N = 64 (8*8), M = 4$$

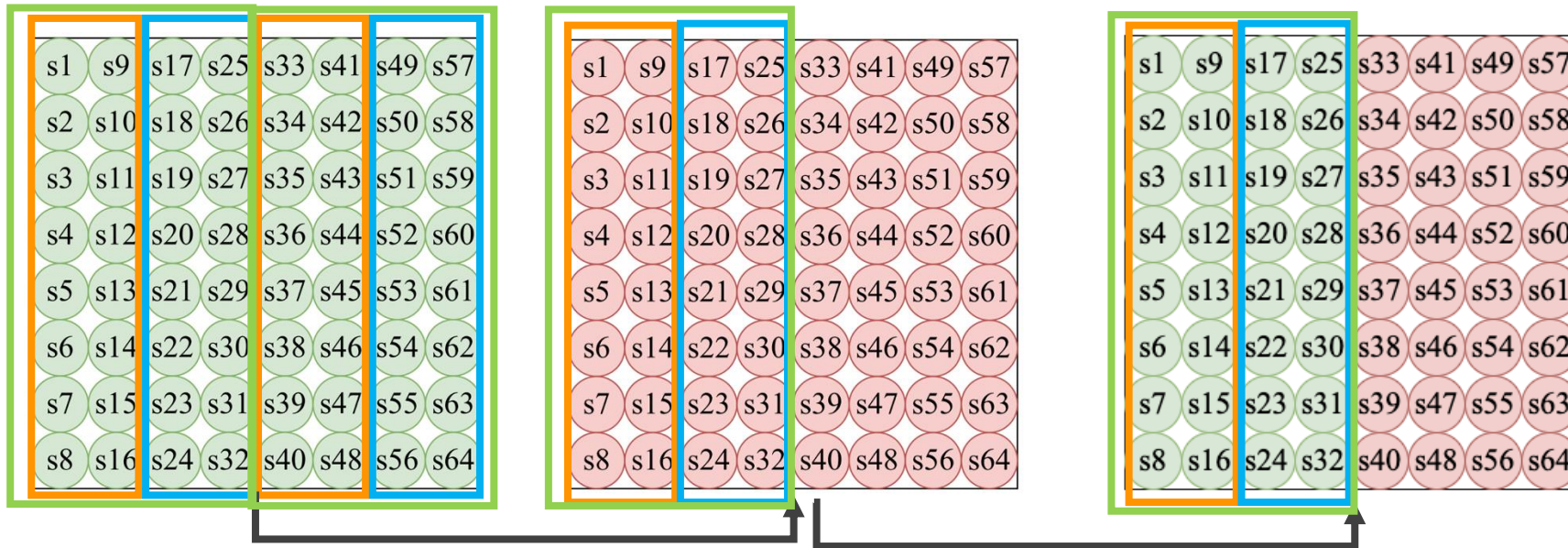
s1	s9	s17	s25	s33	s41	s49	s57
s2	s10	s18	s26	s34	s42	s50	s58
s3	s11	s19	s27	s35	s43	s51	s59
s4	s12	s20	s28	s36	s44	s52	s60
s5	s13	s21	s29	s37	s45	s53	s61
s6	s14	s22	s30	s38	s46	s54	s62
s7	s15	s23	s31	s39	s47	s55	s63
s8	s16	s24	s32	s40	s48	s56	s64

Spins' energies are correct

s1	s9	s17	s25	s33	s41	s49	s57
s2	s10	s18	s26	s34	s42	s50	s58
s3	s11	s19	s27	s35	s43	s51	s59
s4	s12	s20	s28	s36	s44	s52	s60
s5	s13	s21	s29	s37	s45	s53	s61
s6	s14	s22	s30	s38	s46	s54	s62
s7	s15	s23	s31	s39	s47	s55	s63
s8	s16	s24	s32	s40	s48	s56	s64

Spins' energies are not correct

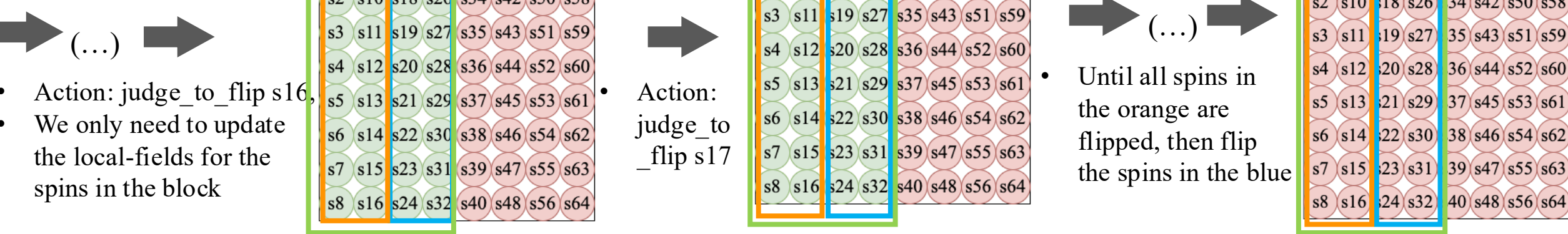
- block_size (blk_sz) is set
 - Then we split the N spins into $\frac{N}{(blk_sz)}$ blocks (Green), and each block is separated into two small blocks (orange and blue) assume $blk_sz = 32$



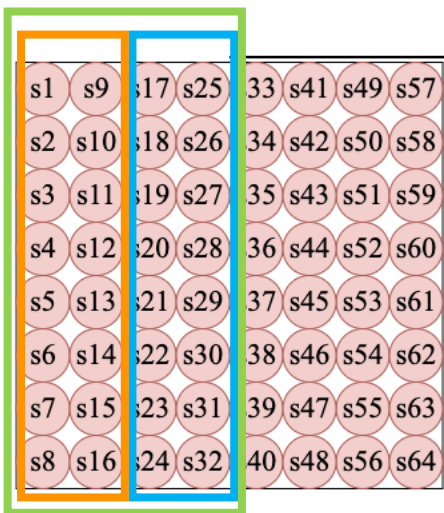
- Action: judge_to_flip s1,
If s1 flip, then all the local-field of the spins on the same trotter are wrong
- We update all the local-field of the spins in the same block (by adding $2 * J_{k,i} \sigma_{i,m'}$)



- Action: judge_to_flip s1,
If s1 flip, then the local-field of the spins on the same trotter are wrong
- We update the local-field of the spins in the same block (by adding $2 * J_{k,i} \sigma_{i,m'}$)
- Action: judge_to_flip s2,
• If s2 flip, update the local-field for the spins in the block

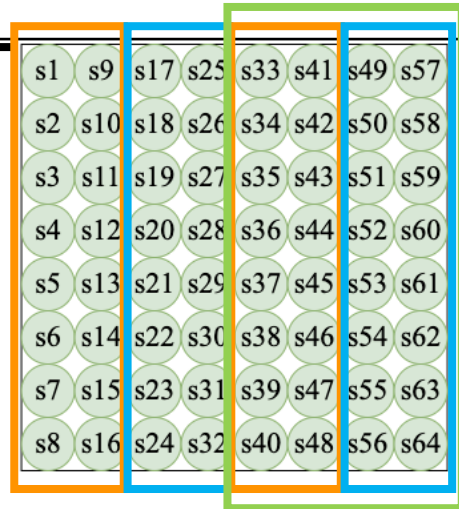


- Action: judge_to_flip s16,
• We only need to update the local-fields for the spins in the block
- Action: judge_to_flip s17
- Until all spins in the orange are flipped, then flip the spins in the blue

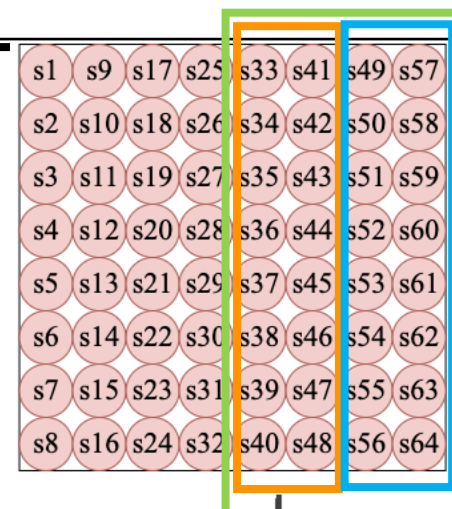


- After all the spins are judged to flip in the block, then all the local-field of spins are incorrect

Then we cost $O(N)$ to update all the local-fields for N spins (the same as the original method)

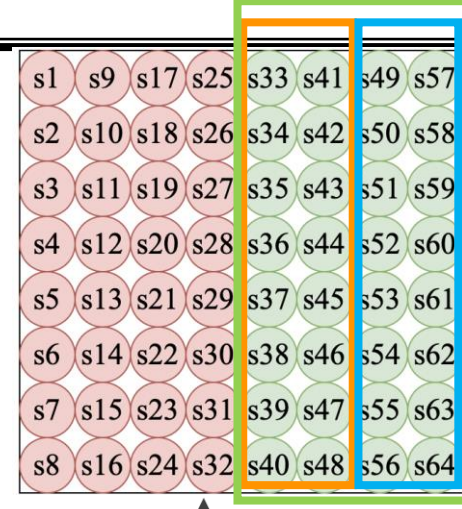


- And heading to flip the spin in the next block



- Action: judge_to_flip s_{33} ,
If s_{33} flip, the local-field of the spins on the same trotter are wrong

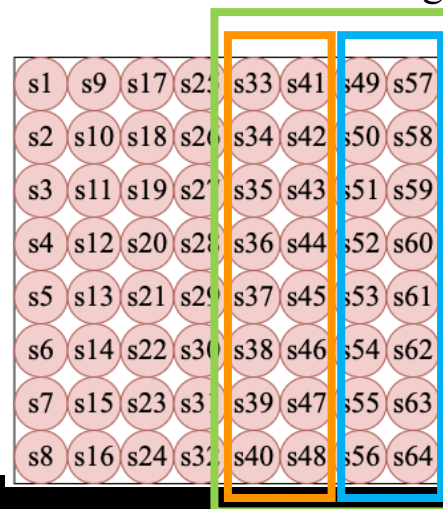
- We update the local-field of the spins in the same block (by adding $2 * J_{k,i} \sigma_{i,m'}$)



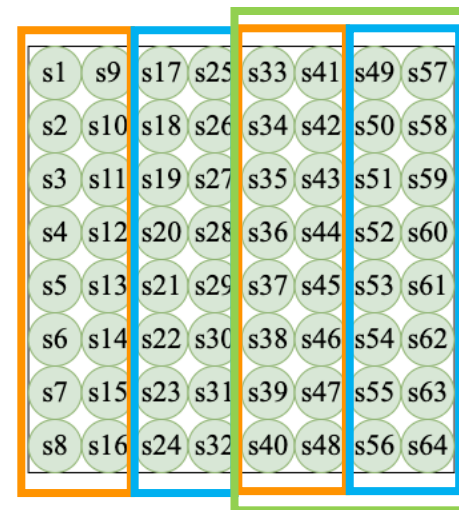
➡ (...) ➡

- judge_to_flip all the spins in the orange small block, then heading to flip all the spins in blue

➡ (...) ➡

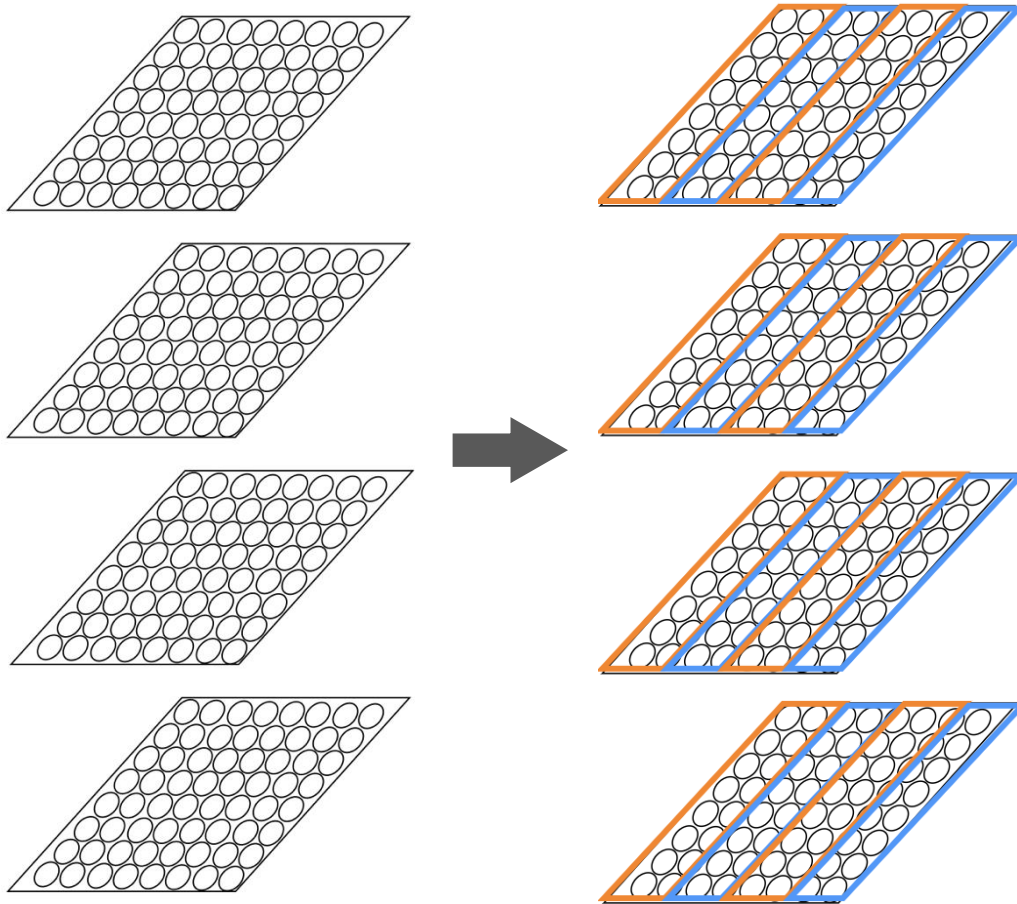


- Then we cost $O(N)$ to update all the local-fields for N spins (the same as the original method)

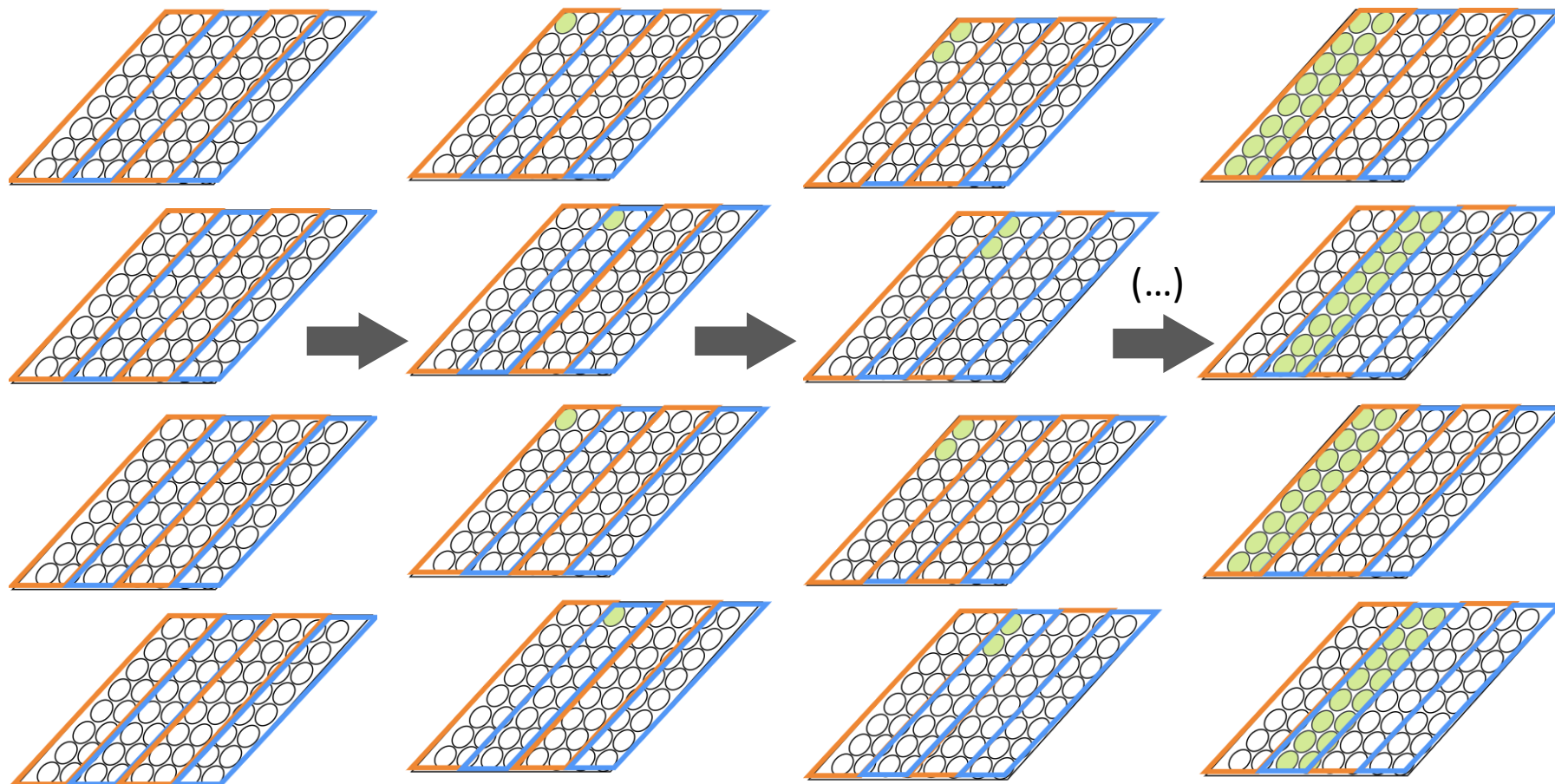


Flip Different Block on the Different Trotter

- By flipping different small block of each trotter to maintain the parallelism on M-dimension

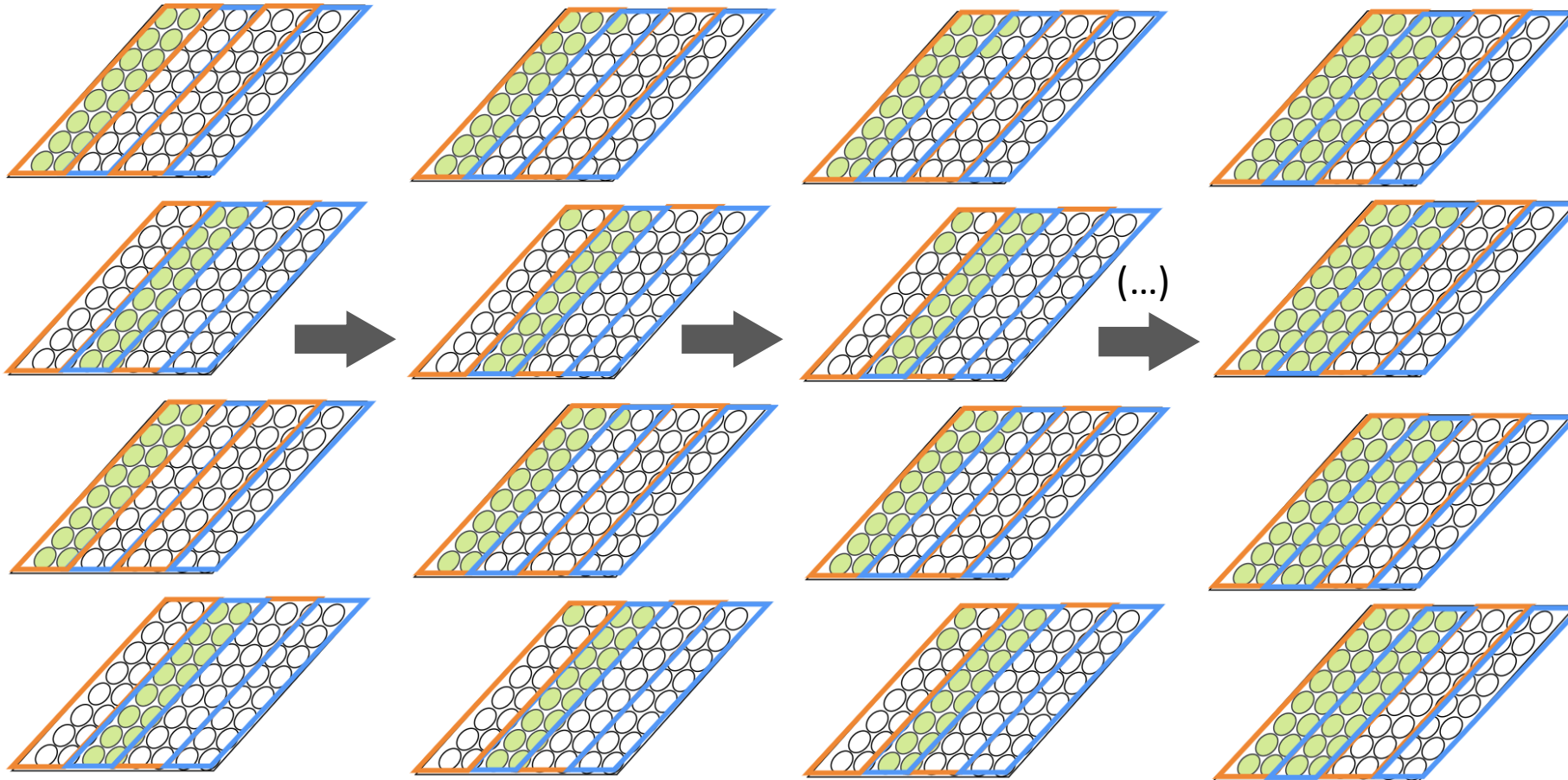


- 1-th round
 - Odd trotter → flip the first orange block
 - Even trotter → flip the first blue block
 - Odd trotter → flip the first blue block
 - Even trotter → flip the first orange block
 - __syncthreads();
- Update all the $M*N$ local-field
- 2-th round
 - Odd trotter → flip the first orange block
 - Even trotter → flip the first blue block
 - Odd trotter → flip the first blue block
 - Even trotter → flip the first orange block
 - __syncthreads();
- Update all the $M*N$ local-field



- 1-th round

- **Odd trotter** → flip the first orange block
- **Even trotter** → flip the first blue block



- 1-th round

- Odd trotter → flip the first orange block
- Even trotter → flip the first blue block
- **Odd trotter → flip the first orange block**
- **Even trotter → flip the first blue block**
- `__syncthreads();`

Update all the $M*N$ local-field

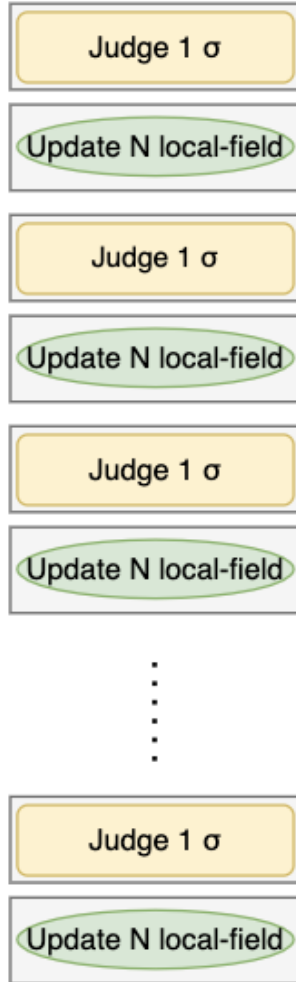
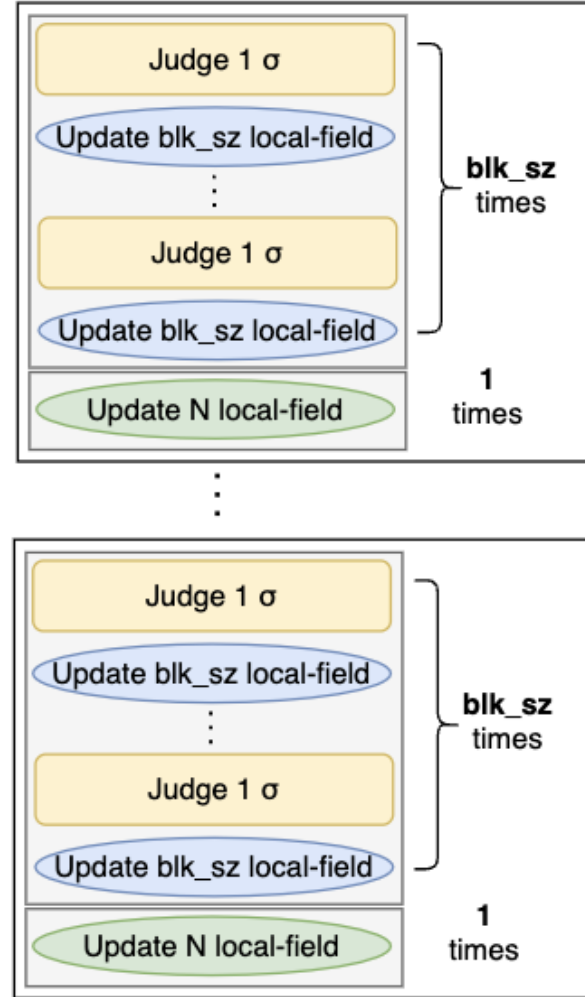
18

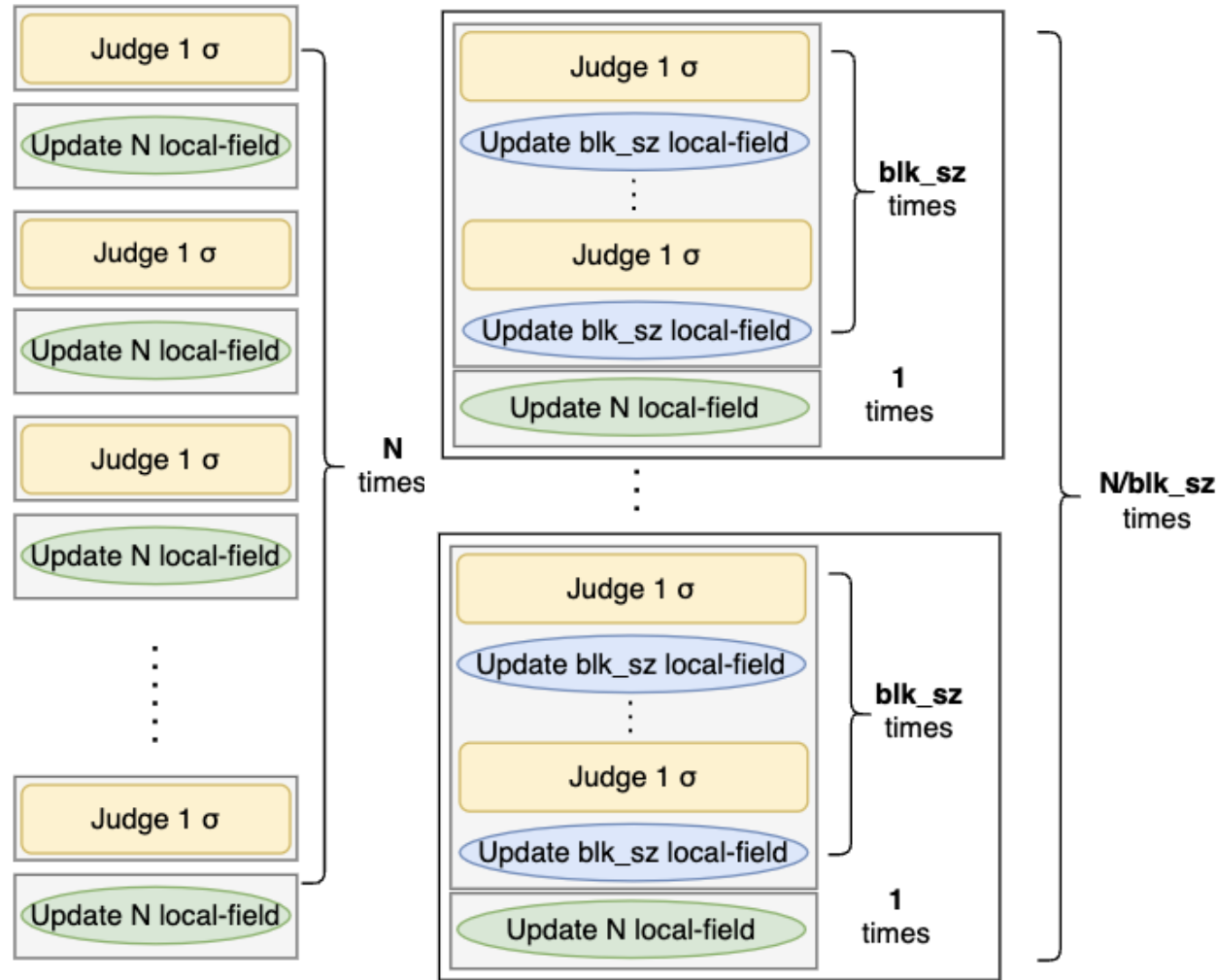
Updating Local-Field by Using Tensor Cores

- After judge_to_flip blk_sz spins on M trotters, we need to update $M*N$ local field
- During update, we only need to conduct the matrix multiplication between coupling matrix and spin matrix for $N * blk_sz * M$ to update all the local-field of $M*N$ spins

$$\begin{array}{ccc}
 \begin{array}{c} \text{\textit{blk_sz}} \\ N \end{array} \begin{pmatrix} J_{1,1} & J_{1,2} & \cdots & J_{1,N} \\ J_{2,1} & J_{2,2} & \cdots & J_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ J_{N,1} & J_{N,2} & \cdots & J_{N,N} \end{pmatrix}_{N*N} & \begin{array}{c} M \\ \text{\textit{blk_sz}} \end{array} \begin{pmatrix} \sigma_{1,1} & \sigma_{1,2} & \cdots & \sigma_{1,M} \\ \sigma_{2,1} & \sigma_{2,2} & \cdots & \sigma_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{N,1} & \sigma_{N,2} & \cdots & \sigma_{N,M} \end{pmatrix}_{N*M} & \begin{array}{c} M \\ N \end{array} \begin{pmatrix} local_field_{1,1} & \cdots & local_field_{1,M} \\ \vdots & \ddots & \vdots \\ local_field_{N,1} & \cdots & local_field_{N,M} \end{pmatrix}_{N*M} \\
 \text{(a) Coupling matrix} & \text{(b) Spin matrix} & \text{(c) Local-field matrix}
 \end{array}$$

: kernel launch

Judge 1 σ N times, sequential partUpdate N local-field N times, parallel partFor one spin: $2 * J_{k,i} * \sigma_{i,m}'$ For N spins, N times:Multiplication: $(2N) * (N)$ Addition: $(1N) * (N)$  N times N/blk_sz timesJudge 1 σ N times, sequential partUpdate blk_sz local-field $blk_sz * (N/blk_sz)$ times, parallel partFor one spin: $2 * J_{k,i} * \sigma_{i,m}'$ For blk_sz spins, N times:Multiplication: $(2 * blk_sz) * (N)$ Addition: $(1 * blk_sz) * (N)$ Update N local-field (N/blk_sz) times, parallel partFor one spin: $2 * J_{k,i} * \sigma_{i,m}'$ For N spins, N/blk_sz times:Multiplication: $(2N) * (N/blk_sz)$ Addition: $(1N) * (N/blk_sz)$



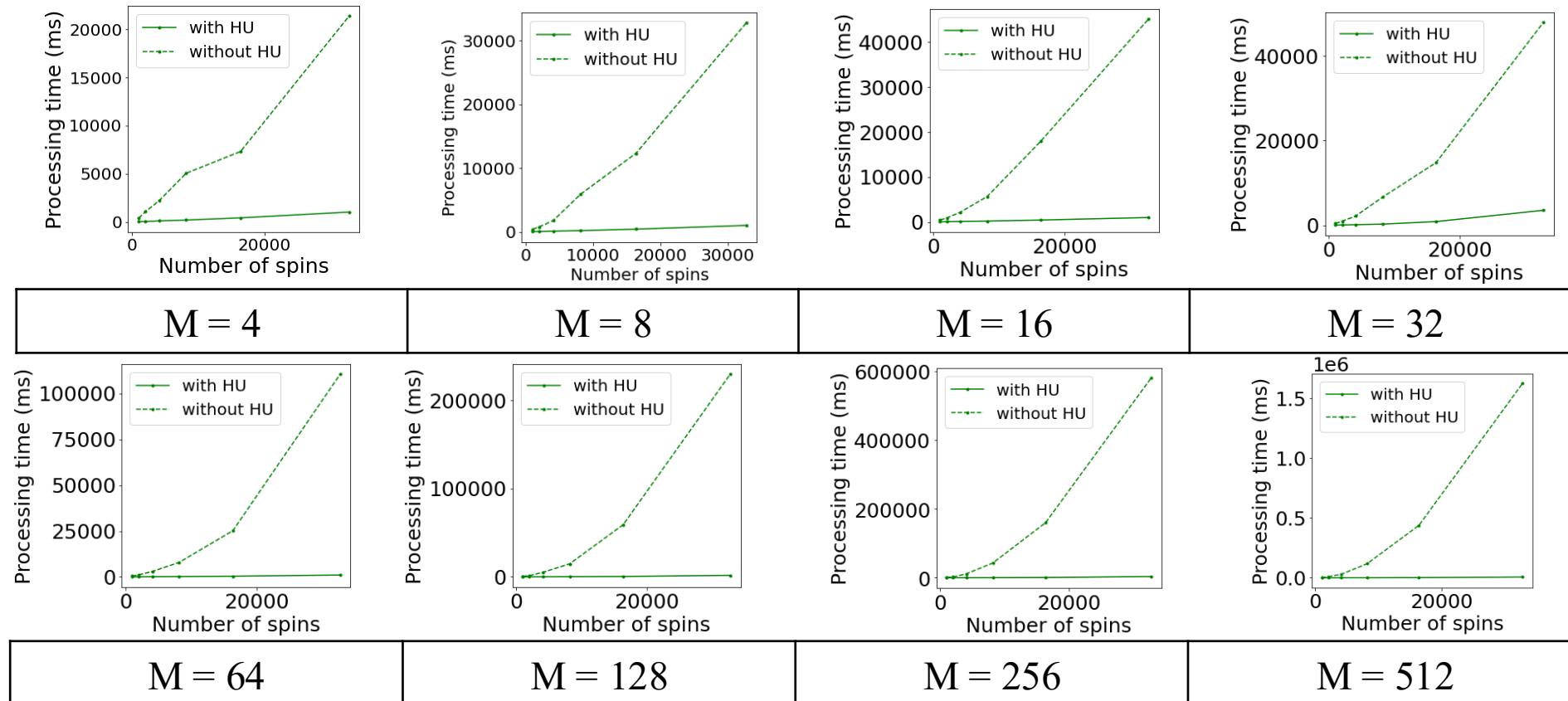
- With adopting the HU, the times to launch the GPU kernel is less than the method without using HU
→ **Less launch overhead**
- Since blk_sz is small, computing data can be placed in the memory of local processor, but the global memory of the GPU
→ **Faster memory access**
- More acceleration can be done by using TCs to calculate the green part
→ **Tensor Cores acceleration**
- We still maintain the parallelism in the M-dimension

- Exp -1. Benefits of Hierarchical Update
- Exp -2. Per-Step Annealing Time
 - Compared with Tohoku University (both GPU and FPGA Version)
- Exp -3. The Choice of *blk_sz*
- Exp -4. The Impact of Tensor Cores
- Exp -5. The Quality of Solution
 - MAX-CUT problem, Gset served by Stanford University

- Environment for exp -1, exp -3, exp -4, and exp -5
 - An Intel(R) Core(TM) i9-9900KF CPU running at 3.60GHz with Ubuntu 18.04.5 LTS OS and a GeForce RTX 3080 GPU with CUDA 11.3 library
- Environment for exp -2
 - For a fair comparison, we run the experiment 2 on a setup that consists of an Intel(R) Xeon(R) Gold 6154 CPU chip running at 3.00GHz with CentOS Linux release 7.8.2003(Core) and a **Tesla V100-SXM2** GPU with the CUDA 11.3 library
 - Tohoku University uses **Quadro GV100 GPU**, which adopts the same Volta architecture as the Tesla V100 and performs single-precision FP operations at 14.8 TFLOPS/s, while the V100-SXM2 delivers 15.7 TFLOPS/s.
 - Our setup has a 6% advantage over Tohoku's

Benefits of Hierarchical Update

- 100 annealing steps with and without the hierarchical update strategy
- It is clear that the proposed HU strategy effectively accelerates the annealing process, especially when the number of spins (N) is large

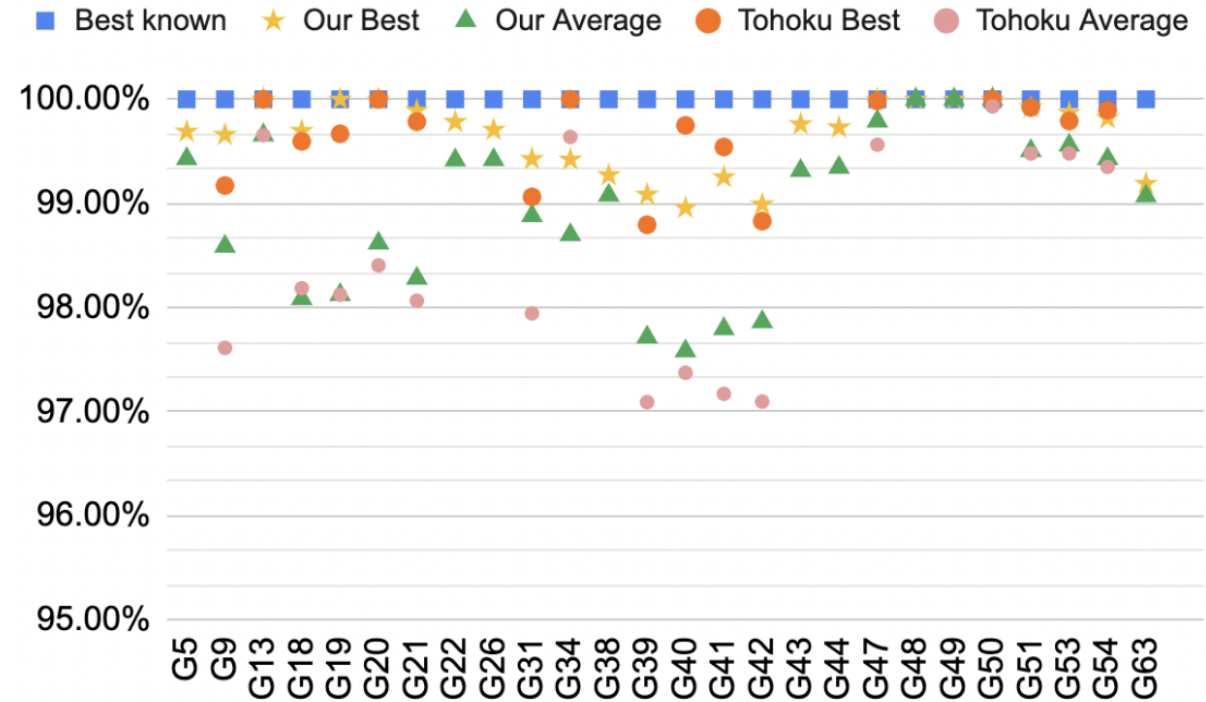


Method	N	M=4	M=16	M=64	M=256	M=512
Tohoku_GPU	4096	12.0 (1X)	15.0 (1X)	32.0 (1X)	215.0 (1X)	826.0 (1X)
Tohoku_FPGA		4.61 (2.6X)	5.11 (2.9X)	N/A	N/A	N/A
HU w/o TC		1.7 (6.9X)	1.8 (8.3X)	2.1 (15.1X)	4.4 (48.6X)	6.9 (119.5X)
HU w/ TC		1.7 (6.9X)	1.7 (8.8X)	1.8 (17.7X)	3.3 (65.2X)	5.3 (156.0X)
Tohoku_GPU	8192	35.0 (1X)	44.0 (1X)	75.0 (1X)	269.0 (1X)	898.0 (1X)
Tohoku_FPGA		17.45 (2.0X)	18.7 (2.4X)	N/A	N/A	N/A
HU w/o TC		3.4 (10.3X)	3.9 (11.2X)	4.7 (15.9X)	11.3 (23.9X)	18.6 (48.4X)
HU w/ TC		3.6 (9.8X)	3.5 (12.4X)	3.7 (20.1X)	8.1 (33.3X)	13.5 (66.7X)
Tohoku_GPU	16384	151.0 (1X)	169.0 (1X)	239.0 (1X)	1009.0 (1X)	2130.0 (1X)
Tohoku_FPGA		65.77 (2.3X)	70.33 (2.4X)	N/A	N/A	N/A
HU w/o TC		7.5 (20.3X)	9.3 (18.1X)	13.2 (18.1X)	32.1 (31.4X)	56.8 (37.5X)
HU w/ TC		7.4 (20.3X)	7.7 (22.0X)	8.9 (26.9X)	20.9 (48.4X)	36.6 (58.1X)
Tohoku_GPU	32768	1003.0 (1X)	1069.0 (1X)	1301.0 (1X)	4754.0 (1X)	8732.0 (1X)
Tohoku_FPGA		264.93 (3.8X)	N/A	N/A	N/A	N/A
HU w/o TC		19.7 (50.8X)	27.5 (38.9X)	37.1 (35.1X)	104.0 (45.7X)	184.8 (47.2X)
HU w/ TC		19.3 (52.0X)	20.6 (51.8X)	24.9 (52.2X)	59.8 (79.5X)	100.9 (86.6X)
Tohoku_GPU	65536	N/A	N/A	N/A	N/A	N/A
Tohoku_FPGA		N/A	N/A	N/A	N/A	N/A
HU w/o TC		54.1	82.8	129.6	376.6	683.1
HU w/ TC		54.2	62.0	81.9	203.2	352.8

- Unit: (ms)
- Our solution (HU w/o TC) is 7X to 48X faster than Tohoku GPU
- With the Tensor Cores, our solution (HU w/ TC) provides 7X to 86X speedup over the baseline
- As M grows, the acceleration from TCs becomes larger

- Problem: Gset, MAXCUT, Stanford University
- Annealing steps: 1000 steps (the same as Tohoku)
- By adjusting $T(0)$ and the decreasing value of the temperature per MC-step to find better solution quality

Gset	N	#Edges	Best Known	Gset	N	#Edges	Best Known
G5	800	19176	11631	G41	2000	11785	2405
G9	800	19176	2054	G42	2000	11779	2481
G13	800	1600	582	G43	1000	9990	6660
G18	800	4694	992	G44	1000	9990	6650
G19	800	4661	906	G47	1000	9990	6657
G20	800	4672	941	G48	3000	6000	6000
G21	800	4667	931	G49	3000	6000	6000
G26	2000	19990	13328	G50	3000	6000	5880
G31	2000	19990	3309	G51	1000	5909	3848
G34	2000	4000	1384	G53	1000	5914	3850
G38	2000	11779	7688	G54	1000	5916	3852
G39	2000	11778	2408	G63	7000	41459	27045



- Accelerating SQA algorithm to solve the Ising problems
- What we have done in this work
 - **Hierarchical Update** - Re-forming the SQA algorithm to implement the algorithm more parallelly
 - **Utilizing the Tensor Cores** - Not only **GPU** is used, **Tensor Cores (TCs)** provided by NVIDIA are also used to accelerate the algorithm in our work
- Original state of the art (SOTA) at that time
 - Both SOTAs accelerated by different accelerators are done by Tohoku University
 - **GPU-based:** A GPU-based quantum annealing simulator for fully-connected ising models utilizing spatial and temporal parallelism (2020, IEEE Access)
 - **FPGA-based:** Highly-parallel FPGA accelerator for simulated quantum annealing (2019, IEEE Transactions)
- Achievement
 - Faster than SOTA GPU-based solution: 1.7x - 86.6x
 - Faster than SOTA FPGA-based solution: 2.65x - 13.68x
 - Maintaining the solution quality no less than 97.5% (Gset (MAX-CUT) provided by Stanford)

Thank you for your time!
/
Any Questions/Ideas/Thoughts/Feedback?

Yi-Hua Chung
Contact me: r09944072@csie.ntu.edu.tw
