Accelerating Simulated Quantum Annealing with GPU and Tensor Cores

No Author Given

No Institute Given

Abstract. Inspired by quantum annealing, simulated quantum annealing (SQA) mimics quantum tunneling effects on classical computers to perform annealing through a path-integral Monte Carlo simulation, which increases the potential to find the global optima faster than traditional annealing algorithms for large-size combinatorial optimization problems while today's quantum annealing systems are of a limited number of qubits. As previous studies have accelerated SQA with Graphics Processing Unit (GPU) and specialized hardware such as Field Programmable Gate Array (FPGA), we propose an innovative parallelizing strategy called hierarchical update to vastly improve the efficiency of parallel computing, which is capable of accelerating state-of-the-art SQA implementations further by 7X-47.2X based on our case studies. Furthermore, we develop a tensorizing scheme to leverage the Tensor Cores on modern GPUs to deliver up to 1.83X of additional speedup. Overall, our work solves fully-connected Ising models faster than any previous SQA work. Our solution outperforms existing GPU-based solutions by 86.6X and FPGA-based solutions by 14X.

Keywords: High performance computing · GPU acceleration · Tensor Cores · simulated quantum annealing · optimization problems.

1 Introduction

Quantum annealing (QA) is a metaheuristic method to solve combinatorial optimization problems [1], such as portfolio optimization [2], protein folding [3], and traveling salesman problem [4], which are often NP-hard problems. These combinatorial optimization problems can be converted into Ising models [5] and solved with QA and traditional simulated annealing (SA) [6]. Unlike SA, QA could potentially solve large problems as it owns the property not subject to the combinatorial explosion. While SA uses thermal excitations, QA uses quantum tunneling [7] to tunnel through the tall but narrow barriers, which may help escape from local minima and find the global best solution for the combinatorial optimization problems. However, QA systems such as D-Wave are very expensive today with limited qubits to model the problems.

Simulated quantum annealing (SQA) [8] is inspired by QA and tries to mimic the quantum tunneling effect on classical computers through a path-integral Monte Carlo simulation. By mimicking the quantum tunneling effect, SQA has

the potential to find the global minima quicker than traditional SA. It is also possible to solve large-size combinatorial optimization problems with SQA, while state-of-the-art QA systems are of limited qubits. To further accelerate SA and SQA, several studies have employed hardware accelerators, such as Application Specific Integrated Circuits (ASICs) [9] [10], Field Programmable Gate Array (FPGA) [11] [12] [13], as well as Graphics Processing Unit (GPU) [14] [15] [16].

Some annealing platforms require an Ising model to be mapped restrictively onto special topologies, as their processors have limited connectivity. For example, D-Wave demands Chimera graph [17] or Pegasus graph [18], which requires an additional transformation procedure called minor embedding [19] and may affect the solution quality with additional computation overhead. In this work, we utilize GPU memory to support a fully-connected Ising model to improve solution quality and avoid the overhead of embedding. Meanwhile, high-performance GPUs are widely available today to provide the most cost-effective solutions.

To perform SQA on a GPU, a fully-connected Ising model is transformed into a matrix form, so that the computing cores on the GPU can operate on the elements of the matrices in parallel. Among the aforementioned accelerated solutions, GPU provides a cost-effective, scalable approach, not only because SQA can compute with thousands of cores on a GPU chip, but it can also run with thousands of GPU chips in a modern supercomputer [20].

In this work, we further exploit the Tensor Cores [21] that are available on modern GPU chips to give 7X-86X speed over the previous work [22] for solving fully-connected Ising models, which is the most efficient SQA implementation as far as we know. We have developed several tensorizing and parallelizing schemes to utilize the Tensor Cores and increase the parallelism for the GPU code. With our optimization, running SQA on GPU with Tensor Cores is even faster than FPGA-accelerated SQA [12]. Moreover, since Tensor Cores have been actively developed to support neural network applications, we expect that the proposed method continues to benefit significantly from this trend.

The remainder of this paper is organized as follows. Sec. 2 explains the Ising Model, describes the algorithm and implementations of SQA and introduces the Tensor Cores. Sec. 3 proposes a new parallelization scheme to improve the utilization of GPU cores and further optimizes the code with the Tensor Cores. Sec. 4 compares our work to previous works to reveal the performance advantage of the proposed methods. Finally, we conclude this paper and point out some potential research directions in the future in Sec. 5.

2 Background and Previous Works

This section first mentions the basics of Ising models for solving combinatorial optimization problems. Then, we explain how simulated quantum annealing (SQA) solves on Ising models, and survey the previously proposed hardware methods for accelerating SQA. Finally, we introduce the Tensor Cores.

2.1 Ising Model

An Ising model is one of the simplest systems that exhibits a phase transition of the thermodynamic system [23][24][25][26]. In the Ising model, the total energy of the system for a lattice with N spins is given as:

$$H(\sigma) = -\sum_{i<j} J_{i,j}\sigma_i\sigma_j - \sum_i h_i\sigma_i \tag{1}$$

where binary spin $\sigma_i \in \{1, -1\}$, $J_{i,j}$ indicates the strength of the coupled interaction between σ_i and σ_j, and h_i is the external magnetic field. Besides, the sign of the coupling term $J_{i,j}$ tells us whether the σ_j prefers to align or to anti-align with the σ_i. The size of the external magnetic field h_i represents how strong its field is, revealing how much higher the energy of the σ_i is than the others. And the sign of h_i tells whether it's spin up or spin down that's preferred. Moreover, h_i also represents the external field trying to align all the spins in one direction.

Solving Ising models is NP-complete [5], which means that the solution of many NP-hard problems, e.g. the traveling salesman problem, can be obtained by solving Ising models. Hence, many works have focused on solving Ising models to establish the solution of combinatorial optimization problems [2][3][4].

2.2 Simulated Quantum Annealing (SQA)

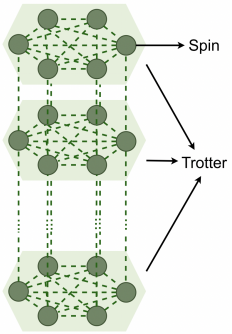


Fig. 1: Transverse-field Ising model

Stoquastic Hamiltonian is applied to various simulation algorithms based on the classical Markov chain. And these algorithms are collectively referred to as Quantum Monte Carlo (QMC) methods [28] [29]. Different QMC methods can be applied to the QA Hamiltonian equation [30] [31]. The version based on the path-integral [29] representation of the thermal state is considered in this paper. Proved by Suzuki, Masuo [8], it is known that the partition function of the m-dimensional quantum Ising model in the traversed field is mapped to the

partition function of the classical Ising model with (m+1)-dimension as shown in Fig. 1, and expressed by [14]:

$$H(\sigma) = -\sum_{k=1}^{M}(\sum_{i<j}\frac{J_{i,j}}{M}\sigma_{i,k}\sigma_{j,k} + \sum_{i}\frac{h_i}{M}\sigma_{i,k} + J^{\dagger}\sum_{i}\sigma_{i,k}\sigma_{i,k+1}) \tag{2}$$

where $J^{\dagger} = T/2 \cdot \ln\coth(\Gamma(t)/(MT))$ indicates the strength of couplings between replicas, T is a controlled temperature, M is the number of replicas, and $\Gamma(t)$ is the controlled transverse field. By reducing the temperature T and transverse field $\Gamma(t)$, the system is expected to have the lowest energy. As the temperature T becomes close to zero at the end of the annealing process, the strength of the coupling $J^{\dagger}$ will approach infinity and each replica will have the same spin configuration with high probability. When M is large enough, the behavior of SQA resembles that of QA. However, it doesn't mean that the larger M has a higher probability to achieve the ground state energy [7].

For an Ising model formulated by Eq. (1), the local-field energy of each spin on the fully-connected graph is the cross dot between all the coupling coefficients and the corresponding spins on the same trotter, as shown in Eq. (3):

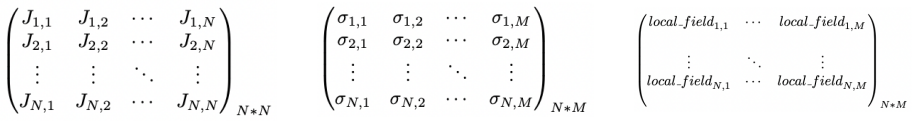
$$\text{local\_field}_{i,m} = \sum_{j=1}^{N} J_{i,j}\sigma_{j,m} \tag{3}$$

where $\sigma_{j,m}$ is the j-th spin in the m-th trotter. The couplings, spins, and local-field energy are mapped to the following three matrix forms:

- **Coupling matrix:** As shown in Fig. 2a, the corresponding coupling coefficient between σ_i and σ_j is placed in the location in the coupling matrix with column i and row j. Since the interaction relations between σ_i and σ_j are equal, the coupling matrix is symmetric. Hence, the matrix size is $N * N$.
- **Spin matrix:** As displayed in Fig. 2b, the spins on the same trotter are placed in the same column of the spin matrix. That is, for example, if there is an i-th spin on the j-th trotter, then the spin is notated as $\sigma_{i,j}$, and it is placed on the location where row $= i$ and column $= j$. So, with spin number $= N$ and trotter number $= M$, the matrix size will be $N * M$.
- **Local-field energy matrix:** Exhibited in Fig. 2c, the matrix elements of the local-field energy matrix are in the same order as the spin matrix. Hence, the local-field energy $\text{local\_field}_{i,j}$ of $\sigma_{i,j}$ is positioned in the place where row $= i$ and column $= j$. Also, the size of the matrix is equal to $N * M$.

Algorithm 1 shows the SQA algorithm. The algorithm first sets up the Ising model and the initial condition by (1) loading the coupling data derived by the Ising problem into the coupling matrix (J) in Line 1, (2) initializing the elements in the spin matrix (σ) randomly with either -1 or 1 in Line 2, and then (3) constructing the local-field matrix based on Eq. (3) in Lines 3-11.

The construction of the local-field matrix is basically done with matrix multiplication, the entry $\text{local\_field}_{i,j}$ of the local-field matrix is derived by multiplying



(a) Coupling matrix (b) Spin matrix (c) Local-field matrix

Fig. 2: The matrices form satisfy the input constraints of Tensor Cores

Algorithm 1: SQA Algorithm

```
1  Coupling Matrix ← Couplings Data
2  Randomly initialize σ ∈ {−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} * σ_{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 |  |  |  ΔH = σ_{i,m}(local_field_{i,m} − J†(σ_{i,m+1}+σ_{i,m−1}))
16 |  |  |  // Judge_to_Flip
17 |  |  |  if e^{−ΔH/T} >random() then
18 |  |  |  |  σ_{i,m}=−σ_{i,m}
19 |  |  |  |  // Update local-field energy
20 |  |  |  |  for j = 1 to N do
21 |  |  |  |  |  local_field_{j,m}+= 2 * J_{i,j} * σ_{i,m}
22 |  |  |  |  end
23 |  |  |  end
24 |  |  end
25 |  end
26 end
```

term-by-term the entries of the i-th row of coupling matrix and the j-th column of the spin matrix, and summing these N products as shown in Eq. (3).

Line 12 to Line 26 of Algorithm 1 is the main algorithm of SQA. The temperature of the system will decrease during each MC-STEP in Line 11. Besides, a larger MC-STEP usually produces a better result closer to the global opti-

mum. The loop in Line 13 and Line 14 executed all spins on all trotters of the Ising model of SQA. If making a flip of the queried spin will cause a decrease in the system energy, then the spin will be flipped. After one spin-flip happens, the local-field energies of the spins on the same trotter will be updated from Line 20 to Line 22. Suppose the k-th spin on m-th trotter is flipped, $\text{local\_field}_{i,m}, i \in \{1, ..., N\}$ should be updated as follows:

$$\begin{aligned} \text{local\_field}'_{i,m} &= J_{i,1}\sigma_{1,m} + ... + J_{i,k}\sigma'_{k,m} + ... + J_{i,N}\sigma_{N,m} \\ &= \text{local\_field}_{i,m} + J_{i,k}\sigma'_{k,m} - J_{i,k}\sigma_{k,m} \\ &= \text{local\_field}_{i,m} + J_{i,k}\sigma'_{k,m} - J_{i,k}(-\sigma'_{k,m}) \\ &= \text{local\_field}_{i,m} + 2 * J_{i,j}\sigma'_{k,m} \\ &= \text{local\_field}_{i,m} + J_{i,j}(2 * \sigma'_{k,m}) \end{aligned} \tag{4}$$

where the $\sigma'_{k,m}$ is the new state of the flipped spin, and the $\text{local-field}'_{i,m}$ is the new local-field energy of the corresponding spin.

Many previous attempts at using specialized hardware to accelerate the annealing processing of SQA were proposed. References [11] [12] [13] [14] provide FPGA implementation on SQA. Among these attempts, [14] focuses on solving king-graph Ising models with limited connectivity, while the others solve fully-connected Ising models. Since the spins without connections can be updated simultaneously, different topologies of SQA may affect the time to sweep all the spins. However, not-fully-connected SQA requires an additional minor embedding process [19], which may deeply affect not only the spins required to do the annealing process but also the solution quality [32] due to additional constraints to fit the target problems into the desired topology.

Several works [15] [16][22] make use of GPUs to accelerate the SQA algorithm. While [15] [16] target sparse-connectivity Ising models, [22] uses GPUs to solve fully-connected Ising models. Compared to using FPGAs as an accelerator, GPUs provide an easier programming interface and have plenty of computational resources. Besides, GPU can accommodate large-scale Ising models.

In [22], the authors proposed a GPU-based SQA accelerator for fully-connected Ising models. They used parallel reduction to perform spatial parallel processing of local-field computation and used concurrent kernel execution to implement temporal parallel processing for multiple-spin-flips.

2.3 Tensor Cores

NVIDIA introduced *Tensor cores* [21] as a specialized hardware accelerator for performing matrix operations in deep learning applications. The Tensor Cores introduced in the Volta GPU architecture can perform matrix-multiply-and-accumulate on 4x4 matrices per clock cycle, as shown in Fig. 3, for low-precision inputs represented in FP16 or INT8. In the latest Ampere GPU architecture, the Tensor Core supports more low-precision data representations such as TF32 and BFLOAT16 as well as sparse matrix operations [33].

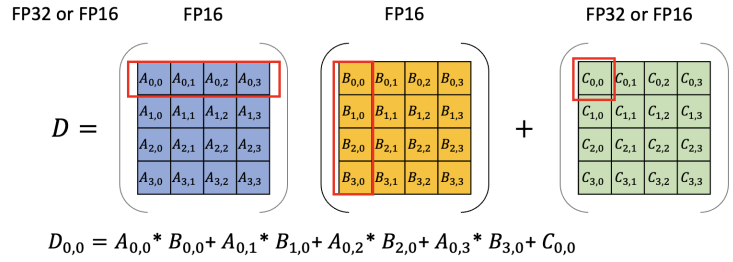


Fig. 3: In the Volta GPU, a tensor core performs a matrix-multiply-and-accumulate on 4x4 matrices per clock cycle.

We use the NVIDIA cuBLAS [34] library to program Tensor Cores for *General Matrix to Matrix Multiplication* (GEMM). The cuBLAS library calls the CUDA Warp Matrix Multiply and Accumulation (WMMA) API [39] [40] to perform 16x16 matrix-multiply-and-accumulate using a warp (32 threads). The cuBLAS library dynamically decides the number of WMMA warps to spawn based on the matrix size and the GPU resources. The annealing process of SQA involves lots of multiply-and-accumulate operations during the update of local-field energies, which can be transformed to run on the Tensor Cores. To our knowledge, this work is the first published work to accelerate annealing algorithms with Tensor Cores.

3 Methodology

In the previous work, [22], while the loop which does Judge_to_Flip (Lines 14-24 in Algorithm 1) can be parallelized, it does not offer a high degree of parallelism as the loop is bounded by the number of trotters (M), nor can it utilize the Tensor Cores. When M is small, we have to find a way to increase the degree of parallelism to fully utilize the GPU. However, since the input is a fully-connected Ising model, any two spins can be connected so that the computation for the spins on the same trotter cannot be further parallelized within this loop. Thus, we propose a *hierarchical update* strategy and modify the algorithm, as shown in Algorithm 2, to increase the number of GPU threads and let each thread forward computation results to the affected threads to remove the data dependency as soon as possible, which is discussed in Sec 3.1. Matrix operations can be accelerated directly with cuBLAS [34], as mentioned in Sec. 2.3. However, while the SQA algorithm appears to manipulate matrices, it is still necessary to modify the algorithm in order to solve fully-connected Ising problems effectively with the Tensor Cores. Thus, we need to convert the operations of SQA into matrix forms and use cuBLAS, which is discussed in Sec. 3.2.

3.1 Hierarchical Update

In a fully-connected Ising problem, if the flipping takes place for one spin, the local-fields for all the spins must be updated to calculate ΔH in Lines 20-22 of Algorithm 1, which prohibits it from flipping multiple spins before updating the entire local-field energies. Since the cost of updating the entire local-field energies is high, we propose to perform *hierarchical update* in Algorithm 2, which has the loop indexed by k (Lines 20-22) calculate the local-field energy in advance only for the block i to keep ΔH updated, so the judgment $(e^{\frac{-\Delta H}{T}} > random())$ in Line 18 could proceed for the later iterations without updating the entire local-field energies. However, this strategy introduces additional overhead, which increases with blk_sz, so it is important to adjust the blk_sz for best performance.

Performance-wise, the execution time of Algorithm 2 is dominated by the Monte Carlo simulation performed in Lines 12-36. We apply *blocking* to the loop indexed by i (Line 13) to reduce the number of CUDA kernel launches and increase the data locality as well as the opportunity for tensorizing. Within the loop, there are two program regions: *Judge_to_Flip* (Lines 15-26) and *Update_Local-Field_Energy* (Lines 28-34).

In **Judge_to_Flip**, the spins on each trotter (m) have to be judged sequentially to decide whether they should be flipped, while all the trotters can be processed in parallel. In our implementation, the iterations in the loop indexed by m are parallelized to run across the streaming multiprocessors (SMs) in the GPU, but the loop indexed by j has to be executed sequentially. The judgment $(e^{\frac{-\Delta H}{T}} > random())$ in Line 18 decides to flip a spin if the energy reduction caused by the flipping is larger than the random number.

There are $blk\_sz * M$ spins to be queried in each sub-task. blk_sz is called block size. Due to the dependency of the upper and lower trotters to the proposed spin, the spins in the same worldline cannot be updated simultaneously from different parities. We propose to update all the spins with two rounds as shown in Fig. 4, where the spins framed in the same color can be queried simultaneously. For example, assume $blk\_sz = 128$, the spins numbered from (k+1) to (k+64) on the odd trotters and the spins numbered from (k+65) to (k+128) on the even trotters will be queried to flip sequentially in the orange block. After that, the spins numbered from (k+65) to (k+128) on the odd trotters and the spins numbered from (k+1) to (k+64) on the even trotters will be queried to flip sequentially in the blue block. Within two rounds, all the spins in the sub-task will be judged completely.

The loop indexed by (m), highlighted in red in Line 15 of Algorithm 2, is executed by multiple streaming processors (SMs). That is, each trotter is handled by one SM. The loop indexed by (k), highlighted in blue (Line 20), updates the local-field energies with multiple threads on a SM. If the spin is flipped, the spin state will be updated and the related local-field energies will also be updated to fully utilize the resources of the GPU in Lines 20-22. After each thread has updated its corresponding local-field energies into local memory, the values are used to determine whether the next spin will be flipped when the program revisits Line 17. The call to `__syncthreads()` is needed to enforce that the threads on

Algorithm 2: Proposed Algorithm

```
1  Coupling Matrix ← Coupling Data
2  Randomly initialize σ ∈ {−1, 1}
3  local_field ← 0
4  // Construct_local_field_Energy, using Tensor Cores
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} * σ_{j,m}
9  |  |  end
10 |  end
11 end
12 for t = 1 to MC-STEP do
13 |  for i = 1; i ≤ N / blk_sz; i++ do
14 |  |  // Judge_to_Flip (ii = i * blk_sz)
15 |  |  for m = 1; m ≤ M; m++ do
16 |  |  |  for j = 1; j ≤ blk_sz; j++ do
17 |  |  |  |  ΔH = σ_{ii+j,m} (local_field_{ii+j} − J†(σ_{ii+j,m+1}+σ_{ii+j,m−1}))
18 |  |  |  |  if e^{−ΔH/T} >random() then
19 |  |  |  |  |  σ_{ii+j,m} = −σ_{ii+j,m}
20 |  |  |  |  |  for k = 1; k ≤ blk_sz; k++ do
21 |  |  |  |  |  |  local_field_{ii+k} += 2 * J_{ii+j,i+k} * σ_{ii+j,m}
22 |  |  |  |  |  end
23 |  |  |  |  end
24 |  |  |  |  __syncthreads()
25 |  |  |  end
26 |  |  end
27 |  |  // Update local_field Energy, using Tensor Cores
28 |  |  for m = 1; m ≤ M; m++ do
29 |  |  |  for j = 1; j ≤ N; j++ do
30 |  |  |  |  for k = 1; k ≤ blk_sz; k++ do
31 |  |  |  |  |  local_field_{j,m} += 2 * J_{j,k} * σ_{k,m}
32 |  |  |  |  end
33 |  |  |  end
34 |  |  end
35 |  end
36 end
```

a SM are synchronized. With sufficient GPU resources, the implementation pre-calculates the related local-fields with k = 1-*blk_sz* when updating the related spins located in the assigned block. Note that the random numbers used in Line 18 are pre-calculated with log(*random*()) in the CPU and transferred to the GPU for every spin in each step. Since the time to generate and transfer random numbers on the CPU is overlapped with the computation on the GPU, it reduces the workload for the GPU without increasing the time for each annealing step.

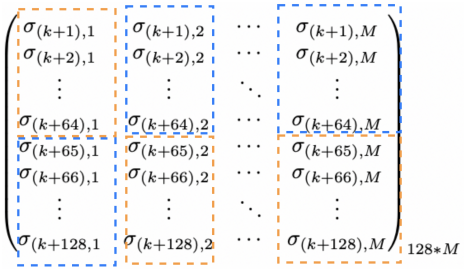


Fig. 4: Update order of spins

Finally, **Update_local-field_energy** can be expressed by matrix multiplication of the coupling matrix and the resulting matrix of judged spin plus the previous local-field energy matrix. This region can be performed well by the GPU with the cuBLAS library. It can be further accelerated with Tensor Cores, as discussed in the following subsection.

3.2 Utilizing the Tensor Cores

To utilize the Tensor Cores, we re-implement the update of local-field energy (Lines 28-34) in Algorithm 2 by calling `cublasGemmEx` in the cuBLAS library as shown in Listing 1.1, where M is the number of trotters, N is the number of spins and K is set to blk_sz. The value of *alpha* (Line 4) is set to $1.0f$, which is the scalar used for multiplication, and the value of *beta* (Line 7) is also set to $1.0f$, which is the scalar utilized for accumulation. In Lines 5-6, *couplings_matrix* and *spin_matrix* are both declared as FP16 with the `CUDA_R_16F` data type for best performance. If needed, these matrices can be declared as other data types supported by Tensor Cores, such as FP32, TF32, BF16, etc. The $local\_field$ matrix is declared FP32 with `CUDA_R_32`, which provides the needed range to accumulate the multiplication results of the *couplings* and *spin matrices*.

```
1  K = blk_sz
2  cublasErrCheck(cublasGemmEx(cublasHandle,
3      CUBLAS_OP_N, CUBLAS_OP_N,
4      N, M, K,
5      &alpha,
6      couplings, CUDA_R_16F, N,
7      judged_spin, CUDA_R_16F, K,
8      &beta,
9      local_field, CUDA_R_32, N,
10     CUBLAS_COMPUTE_32F,
11     CUBLAS_GEMM_DEFAULT_TENSOR_OP));
```

Listing 1.1: Updating local-field energy

As shown in Fig. 5, compared to FP32, the number of bits of FP16 is reduced in half. The benefits of using FP16 include reduced usage for GPU device memory, less time to transfer data from the GPU device memory to the GPU cores,

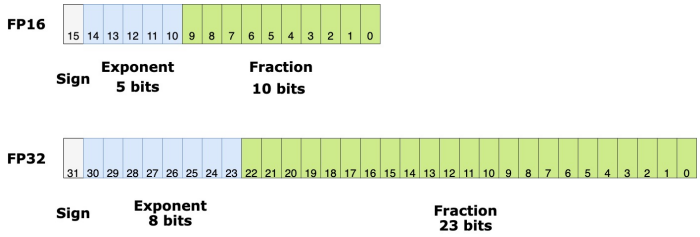


Fig. 5: The format of half precision (FP16) and single precision (FP32)

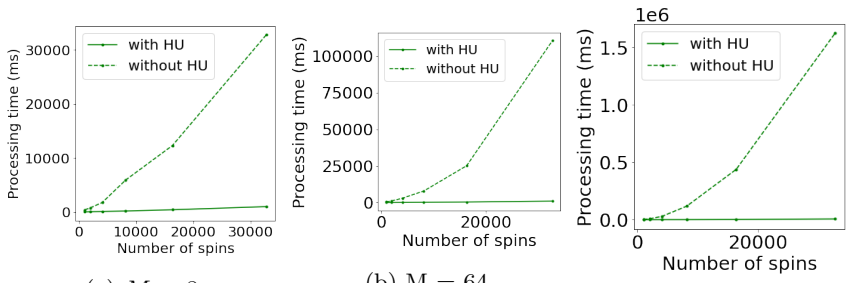
and a higher peak FLOPS rate. However, the maximum integer range of consecutive without precision loss of FP16 is $[-1024, 1024]$, which limits the range of the coupling values. Since the *couplings_matrix* and *spin_matrix* can be represented well with FP16 in our case studies, and the intermediate results of matrix multiplication are stored without loss in the $local\_field$ matrix with FP32, the precision of the proposed system should not be affected by this optimization.

4 Performance Evaluation

We evaluate the proposed method by measuring its performance in solving the MAX-CUT problem [36], which basically aims to split a graph into two mutually exclusive subgraphs with maximum number of inter-subgraphs edges. The MAX-CUT problem is a representative combinatorial problem that can be solved by SA and SQA. Several previous works have chosen it for performance evaluation and reported the results [22] [37] [38]. In these experiments, as discussed in Eq. (2), the temperature T the magnetic field $\Gamma(t)$ are gradually reduced, as we schedule $\frac{1}{T} = t * (1 - \frac{1}{8})/(\text{MC-STEP})$, and $\Gamma(t) = G_0 * (1 - t/\text{MC-STEP})$ during the annealing process, where G_0 is initially set to 8. We measure the time required to complete each annealing step and the number of inter-subgraph edges reported by the annealers at the end. Sec. 4.1 evaluates the benefits of the proposed hierarchical update strategy. Sec. 4.2 compares the processing time of previous work and our work. Sec. 4.3 discusses the choice of blk_sz, which affects the task granularity and impacts the performance significantly. To further break down the execution time, we examine our proposed method in accelerating the update of local-field energy by using the Tensor Cores in Sec. 4.4. Finally, we examine the quality of solutions provided by the proposed method in Sec. 4.5

4.1 Benefits of Hierarchical Update

The hierarchical update strategy adopted in our work calculates the local-field energy of the spins in the block so that judgment can proceed continuously in the later iterations without updating whole local-field energies. We perform 100 annealing steps with and without the hierarchical update strategy on an Intel(R) Core(TM) i9-9900KF CPU running at 3.60GHz with Ubuntu 18.04.5 LTS OS



(a) $M = 8$ (b) M = 64 (c) $M = 512$

Fig. 6: The processing time for 100 annealing steps

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

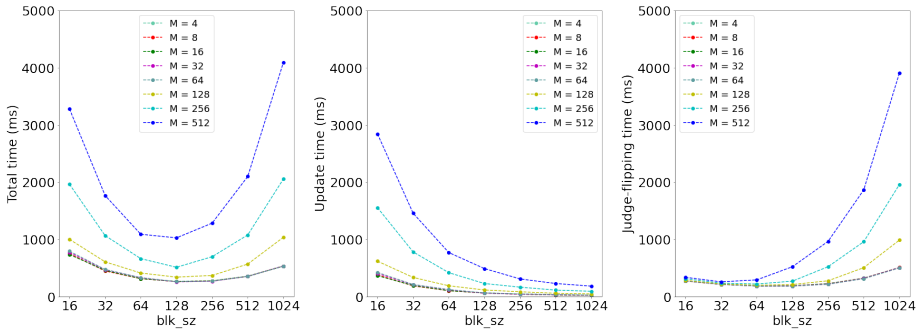
Table 1: Latency of per annealing step (ms). Tohoku_GPU and Tohoku_FPGA are from [22] and [12] respectively, while HU w/o TC is the setting using hierarchical update without Tensor Cores and HU w/ TC with Tensor Cores.

and a GeForce RTX 3080 GPU with CUDA 11.3 library to compare the total processing time for various numbers of trotters (M), as shown in Fig. 6. It is clear that the proposed hierarchical update strategy effectively accelerates the annealing process, especially when the number of spins (N) is large.

4.2 Per-Step Annealing Time

In this subsection, we compare the proposed method against the state-of-the-art GPU-based and FPGA-based quantum annealing simulators published by Tohoku University in [22] [12]. [1]. For a fair comparison, we perform the experiments

[1] We acknowledge the authors for sending us the experiment results.



(a) Total-annealing time (b) Update_Local-field time (c) Judge_to_Flip time

Fig. 7: Processing of the update function, as $N = 8192$

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. The setup in [22] 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.

Table 1 shows the processing time cost per MC-STEP for each solution as N increases from 4096 to 65536 and M increases from 4 to 512. The table also calculates the relative performance using Tohoku's GPU implementation running on a Quadro V100 GPU (Tohoku_GPU) as the baseline. The baseline implementation can solve up to 32768 spins due to the constraint of GPU local memory. The FPGA solution proposed by Tohoku (Tohoku_FPGA), running on an Intel Arria 10 FPGA chip outperforms the baseline by 2-3.8X, but the FPGA solution could not solve large problems due to the memory limitation, and thus some of the data are missing in the table. Notice that the per-step annealing time grows linearly with M and N^2 in theory, but it may not be the case when M or N is too small to fully utilize the GPU or the FPGA. Also notice that when $(N,M)=(4096,512)$, Tohoku_GPU takes an unusually long time to execute, so we treat this data point as an anomaly and exclude it from the following discussions.

Our proposed hierarchical update strategy enables our GPU solution to outperform both solutions provided by Tohoku. Even without using the Tensor Cores, our solution (HU w/o TC) is 7X to 48X faster than Tohoku_GPU[2]. With the Tensor Cores, our solution (HU w/ TC) provides 7X to 86X speedup over the baseline. When N or M increases, the proposed solutions provide greater acceleration over the baseline as the degree of parallelism becomes higher and it appears that our solution utilizes the parallelism more efficiently than Tohoku's GPU solution does. The Tensor Cores version is especially effective for solving large problems. For example, when N=32768, it provides an extra 1.8X speedup

[2] The speedup is up to 120X if $(N,M)=(4096,512)$ is included.

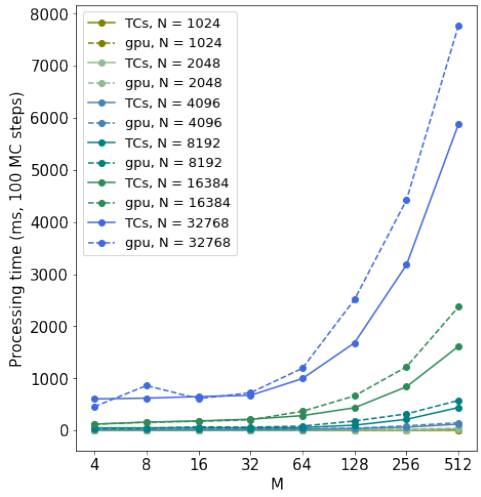


Fig. 8: Tensor Cores impact on Update_local-field_Energy

over the non-TC version. Notice that our solution does not exhibit the anomaly for (N,M)=(4096,512). As far as the problem size is concerned, our GPU implementation can solve Ising problems up to 65536 spins if the coupling matrix is represented with FP16 with 32GB GPU memory.

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

Table 2: The Gset benchmarks and their best known solutions

4.3 The Choice of *blk_sz*

As mentioned in Sec. 3.1, the choice of *blk_sz* impacts the effectiveness of the performance with the hierarchical update, and the aforementioned experimental

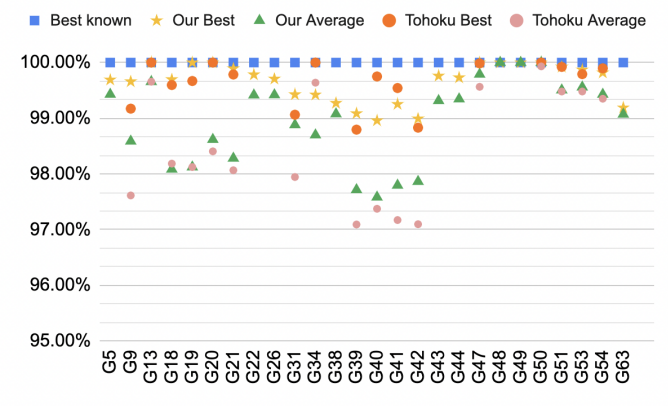


Fig. 9: The solution quality of our work, relative to the previous works

results were done by setting blk_sz to 128, which was chosen after a series of experiments. To evaluate the impact of blk_sz, we profile the annealing time under various blk_sz. Fig. 7a shows the total annealing time under different blk_sz and M, which basically decreases when blk_sz increases from 16 to 128 and then increases when blk_sz increases beyond 128, so it is obvious that the performance is best when blk_sz is set to 128. To further investigate the impact of blk_sz, we break down the total annealing time into two dominating parts: Update_Local-field_Energy and Judge_to_Flip. As shown in Fig. 7b, the time required to update the local-field energies decreases as blk_sz increases, as the larger blk_sz enables more efficient utilization of the Tensor Cores to perform the matrix multiplication. On the other hand, Fig. 7c shows that the time required by Judge_to_Flip decreases initially, but increases when blk_sz becomes large.

The time required by Judge_to_Flip decreases initially when blk_sz increases from 16 to 128 because the larger block size reduces the number of launches of the Judge_to_Flip kernel to the GPU and utilizes more GPU cores to perform the computation. However, when blk_sz keeps increasing, the larger block size requires each Judge_to_Flip kernel to examine more spins and causes additional overhead, which dominates the execution time when blk_sz increases beyond 128. The choice of blk_sz depends mainly on the system architecture, i.e. the GPU and the GPU memory, but is independent of N and M. Thus, we set blk_sz to 128 to produce most of the experimental results in this paper.

4.4 The Impact of Tensor Cores

In order to utilize the Tensor Cores, the update of local-field energy is done in a mixed-precision fashion, where the data types of the two matrices (coupling and spin) for matrix multiplication are half precision (FP16) while the local-field energy matrix is single precision (FP32). We run the experiments with N from 1024 to 32768 and M from 1 to 50 and record the time to process. The

blk_sz is set to 128 (the selection of blk_sz is discussed in subsection 4.3) in the experiment and the annealing step is set to 100. Under different N and M, we ran each experiment 10 times and averaged all the outcomes as the results.

As shown in Fig. 8, the solid-line and dashed-line represent the processing time of Update_Local-field_Energy with and without using Tensor Cores respectively, and the experiments with the same M are presented in the same color. The results show that the processing time grows around 4x as N is doubled when $M = 512$ in both cases, as its computational complexity, is $O(N^2)$ according to Algorithm 1. Since the GPU resources are effectively utilized, when the size of M is smaller than 512, the required time increases less than 4x as N grows twice. Moreover, in those cases of $M \geq 32$, the programs accelerated by Tensor Cores are faster than those which only use the GPU.

4.5 The Quality of Solution

The proposed hierarchical update strategy may have slight effects on the results, as it alters the order of data accesses and computations, so we examine the quality of the solution by comparing our results to previous works to reveal how close the solutions were delivered by our work. The benchmarks are listed in Table 2 with the best known results reported by [22]. As shown in Fig. 9, our best results are within 99.66% of the best known solutions listed in Table 2, while our average results are within 98.97% of the best known results, exhibiting good stability while solving the problems much faster than the previous methods.

5 Conclusion

Our experimental results show that the proposed hierarchical update strategy effectively accelerates SQA on modern GPUs and outperforms the previous works by far due to efficient parallelization and tensorization. We show that Tensor Cores can be used to further speed up the proposed solutions, especially for problems with a large number of spins or many trotters. In particular, increasing the number of trotters allows SQA to impose more of the quantum tunneling effect and can lead to higher solution quality for certain Ising problems, although it is not guaranteed as the algorithm is heuristic. Our experiments also show that the proposed algorithm delivers stable solutions that are very close to the best Known results from previous work. The memory capacity of today's GPUs has limited the problem size which can be solved on one GPU. As shown in Fig. 8, the Tensor Cores accelerate more effectively with large M at the cost of increased memory usage. This limitation could be relaxed with more GPU memory capacity in the future or multiple GPUs. Even though this paper mainly focuses on GPU implementations, the proposed hierarchical update strategy is a general approach that works to accelerate SQA and SA[3]. It can be extended to make use of multiple CPUs/GPUs/accelerators to scale the performance, which also enlarges the solvable problems with larger distributed memories.

[3] SA can be treated as a special case of SQA with M=1.

References

1. Kadowaki, T., Nishimori, H. (1998). Quantum annealing in the transverse Ising model. Physical Review E, 58(5), 5355.
2. Venturelli, D., Kondratyev, A. (2019). Reverse quantum annealing approach to portfolio optimization problems. Quantum Machine Intelligence, 1(1), 17-30.
3. Perdomo-Ortiz, A., Dickson, N., Drew-Brook, M., Rose, G., Aspuru-Guzik, A. (2012). Finding low-energy conformations of lattice protein models by quantum annealing. Scientific reports, 2(1), 1-7.
4. Papalitsas, C., Andronikos, T., Giannakis, K., Theocharopoulou, G., Fanarioti, S. (2019). A QUBO model for the traveling salesman problem with time windows. Algorithms, 12(11), 224.
5. Cipra, B. A. (2000). The Ising model is NP-complete. SIAM News, 33(6), 1-3.
6. Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., Teller, E. (1953). Equation of state calculations by fast computing machines. The journal of chemical physics, 21(6), 1087-1092.
7. Heim, B., Ronnow, T. F., Isakov, S. V., Troyer, M. (2015). Quantum versus classical annealing of Ising spin glasses. Science, 348(6231), 215-217.
8. Suzuki, M. (1976). Relationship between d-dimensional quantal spin systems and (d+ 1)-dimensional ising systems: Equivalence, critical exponents and systematic approximants of the partition function and spin correlations. Progress of theoretical physics, 56(5), 1454-1469.
9. Abdel-Aty, A. H., Khedr, A. N., Saddeek, Y. B., Youssef, A. A. (2018). Thermal entanglement in quantum annealing processor. International Journal of Quantum Information, 16(01), 1850006.
10. Yamaoka, M., Yoshimura, C., Hayashi, M., Okuyama, T., Aoki, H., Mizuno, H. (2015). A 20k-spin Ising chip to solve combinatorial optimization problems with CMOS annealing. IEEE Journal of Solid-State Circuits, 51(1), 303-309.
11. Waidyasooriya, H. M., Hariyama, M., Miyama, M. J., Ohzeki, M. (2019). OpenCL-based design of an FPGA accelerator for quantum annealing simulation. The Journal of Supercomputing, 75(8), 5019-5039.
12. Waidyasooriya, H., Hariyama, M. (2019). Highly-parallel FPGA accelerator for simulated quantum annealing. IEEE Transactions on Emerging Topics in Computing.
13. Liu, C. Y., Waidyasooriya, H. M., Hariyama, M. (2019, November). Data-transfer-bottleneck-less architecture for FPGA-based quantum annealing simulation. In 2019 Seventh International Symposium on Computing and Networking (CANDAR) (pp. 164-170). IEEE.
14. Okuyama, T., Hayashi, M., Yamaoka, M. (2017, November). An ising computer based on simulated quantum annealing by path integral monte carlo method. In 2017 IEEE international conference on rebooting computing (ICRC) (pp. 1-6). IEEE.
15. Weigel, M. (2012). Performance potential for simulating spin models on GPU. Journal of Computational Physics, 231(8), 3064-3082.
16. Cook, C., Zhao, H., Sato, T., Hiromoto, M., Tan, S. X. D. (2018). GPU based parallel Ising computing for combinatorial optimization problems in VLSI physical design. arXiv preprint arXiv:1807.10750.
17. Dwave, `https://www.dwavesys.com/`
18. Dattani, N., Szalay, S., Chancellor, N. (2019). Pegasus: The second connectivity graph for large-scale quantum annealing hardware. arXiv preprint arXiv:1901.07636.

19. Choi, V. (2008). Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. Quantum Information Processing, 7(5), 193-209.
20. JUQCS-G, https://www.fz-juelich.de/portal/DE/Home/home_node.html
21. Markidis, S., Der Chien, S. W., Laure, E., Peng, I. B., Vetter, J. S. (2018, May). Nvidia tensor core programmability, performance & precision. In 2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW) (pp. 522-531). IEEE.
22. Waidyasooriya, H. M., Hariyama, M. (2020). A GPU-based quantum annealing simulator for fully-connected ising models utilizing spatial and temporal parallelism. IEEE Access, 8, 67929-67939.
23. Heim, B., Rønnow, T. F., Isakov, S. V., Troyer, M. (2015). Quantum versus classical annealing of Ising spin glasses. Science, 348(6231), 215-217.
24. Isakov, S. V., Zintchenko, I. N., Rønnow, T. F., Troyer, M. (2015). Optimised simulated annealing for Ising spin glasses. Computer Physics Communications, 192, 265-271.
25. Steinberg, A. P., Kosowsky, M., Fraden, S. (2013). Simulations: The Ising Model.
26. Gould, H., Tobochnik, J. (2010). Statistical and Thermal Physics. Princeton University Press.
27. Kirkpatrick, S., Gelatt Jr, C. D., Vecchi, M. P. (1983). Optimization by simulated annealing. science, 220(4598), 671-680.
28. Suzuki, M., Miyashita, S., Kuroda, A. (1977). Monte Carlo simulation of quantum spin systems. I. Progress of Theoretical Physics, 58(5), 1377-1387.
29. Bravyi, S. (2014). Monte Carlo simulation of stoquastic Hamiltonians. arXiv preprint arXiv:1402.2295.
30. Crosson, E., Harrow, A. W. (2016, October). Simulated quantum annealing can be exponentially faster than classical simulated annealing. In 2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS) (pp. 714-723). IEEE.
31. Bravyi, S., Divincenzo, D. P., Oliveira, R. I., Terhal, B. M. (2006). The complexity of stoquastic local Hamiltonian problems. arXiv preprint quant-ph/0606140.
32. Bernal, D. E., Booth, K. E., Dridi, R., Alghassi, H., Tayur, S., Venturelli, D. (2020, September). Integer programming techniques for minor-embedding in quantum annealers. In International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research (pp. 112-129). Springer, Cham.
33. NVIDIA ampere, https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf
34. cublas, https://docs.nvidia.com/cuda/cublas/index.html
35. cutlass, https://github.com/NVIDIA/cutlass
36. Gset, https://web.stanford.edu/~yyye/yyye/Gset/
37. Benlic, U., Hao, J. K. (2013). Breakout local search for the max-cutproblem. Engineering Applications of Artificial Intelligence, 26(3), 1162-1173.
38. Goto, H., Endo, K., Suzuki, M., Sakai, Y., Kanao, T., Hamakawa, Y., ..., Tatsumura, K. (2021). High-performance combinatorial optimization based on classical mechanics. Science Advances, 7(6), eabe7953.
39. CUDA-9, https://developer.nvidia.com/blog/programming-tensor-cores-cuda-9/
40. WMMA, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#wmma