**CODAR: A Contextual Duration-Aware Qubit Mapping for Various NISQ Devices**

Haowei Deng, Yu Zhang∗ and Quanxi Li  
School of Computer Science and Technology, University of Science and Technology of China, Hefei, China  
Email: jackdhw@mail.ustc.edu.cn, yuzhang@ustc.edu.cn∗, crazylxq@mail.ustc.edu.cn

**Abstract—** Quantum computing devices in the NISQ era share common features and challenges like limited connectivity between qubits. Since two-qubit gates are allowed on limited qubit pairs, quantum compilers must transform original quantum programs to fit the hardware constraints. Previous works on qubit mapping assume different gates have the same execution duration, which limits them to explore the parallelism from the program. To address this drawback, we propose a Multi-architecture Adaptive Quantum Abstract Machine (maQAM) and a COntext-sensitive and Duration-Aware Remapping algorithm (CODAR). The CODAR remapper is aware of gate duration difference and program context, enabling it to extract more parallelism from programs and speed up the quantum programs by 1.23 in simulation on average in different architectures and maintain the fidelity of circuits when running on OriginQ quantum noisy simulator.

I. INTRODUCTION

Quantum Computing (QC) has attracted huge attention in recent a decade due to its ability to exponentially accelerate some important algorithms [22]. Both QC algorithm designers and programmers work at a very high level, and know little about (future) NISQ devices that (will) execute quantum programs. There exists a gap, however, between NISQ devices and the hardware requirements (e.g., size and reliability) of QC algorithms. To bridge the gap, QC requires abstraction layer and toolchain to translate and optimize quantum programs [8]. QC compilers typically translate high-level QC code into (optimized) circuit-level assembly code in multiple stages.

In order to use NISQ hardware, quantum circuit programs have to be compiled to the target device, which includes mapping logical qubits to physical ones of the device. The mapping step, which we focus on in this paper, faces a tough challenge because further physical constraints have to be considered. In fact, 2-qubit gates can only be applied to certain physical qubit pairs. A common method to solve this problem is to insert additional SWAP operations in order to “move” the logical qubits to positions where they can interact with each other. This qubit mapping problem has been proved to be a NP-Complete problem [26].

Previous solutions to this problem can be classified into two types. One of them is to formulate the problem into an equivalent mathematical problem and apply a solver [6, 7, 23, 29, 30, 32] and the other type is to use heuristic search to obtain approximate results [4, 15, 16, 18, 33, 35]. The former suffers from a very long runtime and can only apply to small-size cases. The latter is better in runtime, especially when the circuit is large scale. All these algorithms assume that different gates have the same execution duration.

On NISQ hardware, however, different gates have different durations (see Table [4]). Ignoring the gate duration difference may cause these algorithms to find the shortest depth but not the shortest execution time. The real execution time of the circuit is associated with the weighted depth, in which different gates have different duration weights. Considering gate duration difference will help the compiler make better use of the parallelism of quantum circuits and generate circuits with shorter execution time.

In this paper, we focus on solving the qubit mapping problem by heuristic search with the consideration of gate duration difference and program context to explore more program’s parallelism. To address the challenges of qubit mapping problem and adapt to different quantum technologies, we first give several examples to explain our motivation, then propose a Multi-architecture Adaptive Quantum Abstract Machine (maQAM) for studying the qubit mapping problem. The maQAM is modeled as a coupling graph with limited qubit connectivity and configurable durations of different kinds of quantum gates. Based on the maQAM, we further propose two mechanisms that enable CONtext-sensitive and Duration-Aware Remapping algorithm (CODAR) to solve the qubit mapping problem with the awareness of gate duration difference and program context.

The main contributions of this paper are as follows:

- We summarize the features of different QC technologies including their available gates, operation fidelity and execution duration, and establish the quantum architecture abstraction with configurable parameters – maQAM.
- We propose a SWAP-based heuristic algorithm for qubit remapping, CODAR, which considers the gate duration difference and program context and can further speed up the quantum program.
- We evaluate CODAR with various benchmarks on several latest hardware models such as Google’s 54-qubit Sycamore processor [3]. Experimental results show that CODAR speeds up quantum programs by 1.212~1.258 at the average in comparison with the best-known algorithm, and maintains the fidelity of circuits when running on OriginQ quantum noisy simulator [24].

II. PROBLEM ANALYSIS

A. Recent Work on Qubit Mapping

There are a lot of research on the qubit mapping problem. Here we focus on analyzing some valuable solutions in recent two years [4, 18, 21, 26, 28, 32, 35]. All of them are proposed for some IBM QX architectures, and none of them consider the gate duration difference.

a) Solutions only considering qubit coupling: [26, 32] provide solutions for 5-qubit IBM QX architectures with directed coupling. Siraichi et al. [26] propose an optimal algorithm based on dynamic programming, which only fits for small circuits; then they propose a heuristic one which is fast but oversimplified with results worse than IBM’s solution. Wille et al. [32] present a solution with a minimal number of additional SWAP and H operations, in which qubit mapping problem is formulated as a symbolic optimization problem with high complexity. They utilize powerful reasoning engines to solve the computationally task.

b) Solutions using heuristic search: [18, 35] use heuristic search to provide good solutions in acceptable time for large scale circuits. Zulehner et al. [35] divide the two-qubit gates into independent layers, then use A∗ search plus heuristic cost function to determine compliant mappings for each layer. Li et al. [18] propose a SWAP-based bidirectional heuristic search algorithm – SABRE, which can produce comparable results with exponential speedup against previous solutions such as [35].
TABLE I: Parameter information of several quantum computing devices.

|                | Ion Trap |                | IBM Q5 | IBM Q16 | IBM Q20 | Neutral Atom |
|----------------|----------|----------------|--------|---------|----------|--------------|
|                | Ion Q5   | Ion Q11        | X, Y, Z, H, S, T | CNOT (CX) | CNOT (CX) |
| Available 1-qubit gate | R_x | XX |                  |         |         |
| Available 2-qubit gate |                 |                |                  |         |         |
| Fidelity        | 99.1(5)% | 99.5%          | 99.7%         | ~99.8%  | ~99.56%  | 99.995%      |
| 1-qubit gate    | 97(1)%   | 97.5%(95.1%,98.9%) | 96.5%       | ~96%    | ~96%     | 97%          |
| 1-qubit readout| (0):99.7(1)%| (1):<99.1(1)%| 99.3%      | ~93%    | ~91.2%   | 98.6%        |
| Time            | 20µs s   | 25µs s         | 130 ns      | ~80%    | ~80%     | ≥130 ns      |
| Depolarization  | ~∞       | ~∞             | ~250-450 ns | ~60µs s | ~70µs s  | ~10µs s      |
| Spin dephasing  | ~0.5s    | ~0.5s          | ~250-450 ns | ~60µs s | ~70µs s  | ~10µs s      |

b) Solutions further considering error rates: [4, 21, 29]

provide another type of perspective for solving the qubit mapping problem. They consider the variation in the error rates of different qubits and connections to generate directly executable circuits that improve reliability rather than minimize circuit depth and number of gates. Based on the error rate data from real IBM Q16 and Q20 respectively, [21, 29] use a SMT solver to schedule gate operations to qubits with lower error probabilities. Ash-Saki et al. propose two approaches, Sub-graph Search and Greedy approach, to optimize gate-errors [4]. Circuits generated by them may suffer from long execution time due to no consideration of the minimal circuit depth.

c) What we consider in the qubit mapping: We want to produce solutions for the qubit mapping problem with speedup against previous works and maintain the fidelity as well. Besides the coupling map, we have all the detailed information about the system context and the gate duration difference, which affect the design of qubit mapping. Considering these factors will help to find remapping solution with approximate optimal execution close to reality.

B. Motivating Examples

We use several examples written in OpenQASM [10] to explain our motivation for considering program context and gate duration difference in the qubit remapping process. The two examples based on the coupling map of four physical qubits Q_0~Q_3 and the assumed gate durations defined in Fig. 1(a) and (b). We directly map the logical qubits q[0]~q[3] initially to physical qubits Q_0~Q_3 for easier explanation.

Fig. 1: An example reflecting the impact of program context on SWAP-based transformations: SWAP q[3], q[1] is selected in (d) to avoid using q[2] operated by the previous T gate, accordingly increasing parallelism.

a) Impact of program context: Consider the OpenQASM code fragment shown in Fig1(b). Since qubits Q_0 and Q_3 are non-adjacent, the instruction “CX q[0],q[3]” at line 2 cannot be applied. To solve the problem, SWAP operation is required before performing the CX operation. In this case, there are four candidate SWAP pairs, i.e., (Q_0, Q_1), (Q_0, Q_2), (Q_2, Q_3), and (Q_3, Q_2). If the program context, i.e., the predecessor instruction “T q[2]”, is not considered, there are no differences among the four candidates when selecting. However, SWAP operation on pair (Q_2, Q_3) or (Q_3, Q_2) conflicts with the context instruction “T q[2]” due to operating the same Q_2, and has to be executed serially after T operation as shown in Fig1(c). SWAP on pair (Q_2, Q_1) or (Q_3, Q_1) does not conflict with “T q[2]” and can be executed in parallel as shown in Fig1(d). With the awareness of the context information, SWAP operations that improve parallelism can be sifted out.

b) Impact of gate duration difference: We use a 4-qubit QFT (Quantum Fourier Transform) circuit to explain the limitation of ignoring the duration of quantum gates. Fig2(b) lists the fragment of a 4-qubit QFT OpenQASM program, which is generated by ScaffCC compiler [2]. Similar to the first example, SWAP operation is required before performing the CX operation, and there are also the same four candidate SWAP pairs. Instructions “T q[2]” and “CX q[0],q[2]” can be executed in parallel and we assume both of them start at cycle 0. If the difference of gate durations is ignored, the two gates “T q[2]” and “CX q[0],q[2]” are assumed to finish at the same time t and the four candidate SWAP operations have to start after t. However, if the duration of CX is twice as much as that of T, we find that “T q[2]” will finish at cycle 1 while “CX q[0],q[2]” at cycle 2. As a result, SWAP between q[3] and q[1] can start at cycle 1 as shown in Fig2(d), while other three candidate SWAP operations have to start at cycle 2 since one of operands Q_2 or Q_3 is occupied as shown in Fig2(c). Fig2(d) has better parallelism, which can be deduced by the awareness of different quantum gate durations.

Fig. 2: A 4-qubit QFT example reflecting the impact of gate duration difference; “SWAP q[3],q[1]” is the best candidate since it can start immediately after “T q[1]” while “CX q[0],q[2]” has not finished yet, increasing the parallelism of the circuit.
III. QUANTUM ARCHITECTURE ABSTRACTION

Since the qubit mapping problem is affected by the constraints of underlying QC devices, which based on various and evolving quantum technologies, it is essential to design quantum mapping algorithms that are compatible with different quantum technologies.

A. Characteristics of NISQ devices

Table[I] lists parameter information of some QC devices based on ion trap, superconducting and neutral atomic technologies, respectively. From the table, we see that two-qubit gate executes slower (at least 2××) and has lower fidelity than single-qubit gate on both superconducting and ion trap platforms; the ion trap system is about 1000× slower than the superconducting system, but can execute more gates before decoherence. The directly implementable elementary gates in ion trap system are single-qubit gate $R_{\alpha}$ (rotations by an angle $\theta$ about any axis $\alpha$) and two-qubit gate $XX$. Specifically, $\text{CNOT}$ gate can be implemented by a one-$XX$ and four-$R_{\alpha}$ gates. Neutral atoms are similar in magnitude to trapped ions. However, the two-qubit gate applied to neutral atoms may not perform slower than a single-qubit gate, but the fidelity is much worse.

B. Definition of the maQAM

In view of the above, we consider the qubit connectivity of various NISQ devices, and take each gate duration as a multiple of quantum clock cycle $\tau_a$, which can be analogized to the classic clock cycle. We then introduce a Multi-architecture Adaptive Quantum Abstract Machine (maQAM) which consists of static and dynamic structures, denoted as $A = (A_s, A_d)$. Table[I] shows the definitions for maQAM, where $A_s = (Q_H, G, \mathcal{M}, \tau, \mathcal{D})$, and $A_d = (\pi, \mathcal{C})$. We assume the device can provide enough physical qubits ($\text{denote the number as } N$) for the program's execution ($\text{denote the number of logical qubits in the program as } n$), i.e., $N \geq n$.

For a QC device, we abstract its qubit layout as a graph $\mathcal{M}$ where qubits are vertices and there are edges between qubit pairs where a two-qubit gate is allowed to apply on them. We introduce the Gate Duration Map $\tau$ into $A_s$, which maps each kind of quantum gate to its duration, depending on the information from quantum architecture. We assume the same kind of quantum gates have the same duration and fidelity. We also introduce the shortest distance map $\mathcal{D}$ between each pair of physical qubits for quick selection of exchangeable qubits in our CODAR scheduling algorithm.

IV. DESIGN OF THE QUBIT REMAPPING ALGORITHM

In this section, we discuss our design for CODAR: context-sensitive and Duration-Aware Remapping algorithm (CODAR). We introduce two fundamental mechanisms that enable CODAR context-sensitivity and duration awareness, i.e., qubit lock and commutativity detection.

A. Qubit Lock

CODAR is based on a reasonable assumption: Two or more gates cannot be applied to the same qubit at the same time. If a gate occupies a qubit, we call the qubit busy, and other gates can no longer be applied to the qubit. As the example shown in Fig. 1 when inserting $\text{SWAP}$ for a specific two-qubit gate $CX q[0], q[3]$, the neighbour qubit $q[2]$ of the target qubits may be occupied by the contextual gate which has started in an earlier time. Using the occupied qubits to route the two-qubit gate will reduce the parallelism of the program because the routing process has to wait until the occupied qubits become free.

To make CODAR aware of the qubit occupation by the past contextual gate, we introduce a qubit lock $t_{\text{end}}$ for each physical qubit in $Q$. When start applying a quantum gate on a physical qubit in $Q$ whose duration is $\tau_g$ at time $t$ to a physical qubit in $Q$, CODAR will update this qubit’s $t_{\text{end}}$ as $t + \tau_g$, which means the qubit is occupied before $t + \tau_g$. A qubit is free only when its lock $t_{\text{end}} \leq$ current time, which means all the past gates applied on this qubit are finished. When trying to find a routing path for a specific two-qubit gate, by comparing $t_{\text{end}}$ of each qubit with the current time, CODAR can be aware of which qubits are occupied by the past contextual gate. Fig. 3 shows an example. Gates can only be applied to the physical qubits in a free state. We call the gates whose associate physical qubits are all free as lock-free gates.

![Fig. 3: Qubit lock $t_{\text{end}}$ of qubit q is 2 means q is busy until time 2.](image)

Qubit lock also makes CODAR aware of the gate duration difference. Different gate kinds have different duration and CODAR updates the operated qubit’s lock $t_{\text{end}}$ with different values. As a result, qubits occupied by gates with shorter duration will be set smaller $t_{\text{end}}$ and become free earlier. Thus CODAR can use those earlier free qubits to route two-qubit gates and improve the parallelism of the program. As the example shown in Fig. 2(d), suppose the program starts at time 0 and $\tau_1 = \tau_2 = 2$. Then $t_{\text{end}}$ of $Q_1$ is set to 1 while $t_{\text{end}}$ of $Q_0$ and $Q_2$ are set to 2. $Q_1$ becomes free at time 1 while $Q_2$ is still busy. CODAR can use $Q_1$ to route for the third gate and need not wait for the freedom of $Q_2$.

B. Commutativity Detection

Qubit lock brings CODAR awareness of the past contextual gate. On the other hand, considering gate commutation relation can expose more future contextual gate for CODAR to decide routing path.

| Notation | Definition |
|----------|------------|
| $Q_H$ | The set of physical qubits, $|Q_H| = N$; $\forall Q \in Q_H$, $Q.t_{\text{end}}$ is the qubit lock described in Section IV-A |
| $G$ | The set of elementary quantum operations and $\text{SWAP}$; $|G| = M$ |
| $\mathcal{M}$ | The coupling graph of a quantum device, $\mathcal{M} = Q_H \times Q_H$ |
| $\mathcal{P}$ | Map from quantum operations to their durations |
| $\mathcal{D}$ | Map from physical qubit pairs to their shortest path lengths on the $\mathcal{M}$, if there is no path between $Q_i$ and $Q_j$, then $\mathcal{D}(Q_i, Q_j) = \text{INT}_\text{MAX}$ |
| $\pi: Q_P \rightarrow Q_H$ | Mapping from logical qubits to physical qubits |
| $\mathcal{CF}(G)$ | Commutative Front gate set of a gate sequence $G$, defined in Definition[I] |
| $\text{auxiliary function}$ | $q_{g\in G}(g)$, the name of a given gate $g$ |
| $\text{Variables}$ | $q(1,2,\ldots,N)$ Physical qubits, $Q_i \in Q_H$, $1 \leq i \leq N$ |
| $q(1,2,\ldots,M)$ Logical qubits, $q_i \in Q_P$, $1 \leq i \leq M$ |
| $g(1,2,\ldots)$ Physical quantum operations, $g_i \in G$, $1 \leq i \leq M$ |
| $d$ | Quantum operations in the circuit program |
| $t$ | A sequence of quantum operations, $t = [t_1,t_2,\ldots,t_k]$ if $k = |t|$, and the length of $J$ is written as $|J|$ |

TABLE II: Definition of Quantum Abstract Machine.
The commutation relation between two-qubit gates \( g_a, g_b \) that share qubits with each other can be resolved by checking the relevant unitary operators \( AB = BA \). Gates applied to disjoint qubits are obviously commutative with each other.

All the CF gates in sequence \( I \) are denoted as \( I_{CF} \). CF gates can be moved to the head of \( I \), which means they can be executed instantly from logical perspective. Compared to the methods that ignore the commutativity between quantum gates, choosing CF gates as logically-executable gates can expose more contextual gates for the heuristic search to determine better remapping solutions.

For example, suppose a sequence \( I \) contains two gates: \( C_X q_1,q_3 \) and \( C_X q_2,q_3 \) in order. The second gate shares \( q_3 \) with the first and might not be regarded as logically executable due to qubit dependence. However, because the second commutes with the first one, both of the gates are CF gates in \( I \) and instantly executable in fact. Commutativity detection will expose both \( C_X \)s for the heuristic search which improve the contextual look-ahead ability of CODAR.

### C. Algorithms for CODAR Remapping

Now we discuss how CODAR transforms the quantum circuit to fit the hardware limitation. Given the coupling graph \( M = (Q, E_M) \), initial mapping \( \pi \) and gate duration map \( \tau \), our algorithm takes an original gate sequence \( I \) as input, and generates an executable gate sequence \( E \). The overview of the algorithm is shown in Fig.4. The algorithm starts with the current time and each qubit lock initialized as 0. There are three steps in each iteration.

a) **Step 1:** At the start of each cycle, the algorithm first gets all CF gates in the input sequence \( I \) denoted as \( I_{CF} \). The algorithm will terminate if there is no gate in \( I_{CF} \).

b) **Step 2:** In this step, the algorithm will select all directly executable gates in \( I_{CF} \). A gate \( g \) is directly executable only when it satisfies two conditions below.

- \( g \) is a lock-free gate.
- If \( g \) is a two-qubit gate like \( CNOT \), its two target qubits are connected in the coupling graph.

Then the algorithm will apply those executable gates by moving them from the input sequence \( I \) to the output sequence \( E \) and update the qubit lock for each gate in the way described in Section IV-C.

c) **Step 3:** For the remaining \( CNOTs \) in \( I_{CF} \), due to the connectivity limitation, the algorithm needs to decide which \( CNOTs \) should be inserted into the output sequence. The algorithm first searches all lock-free edges associated with the qubits of each two-qubit gate \( g \) in \( I_{CF} \) to avoid huge overhead cost by global searching. For example, suppose \( g = C_X q_a,q_1 \), \( Q_k = \pi(q_1) \) is the corresponding physical qubit of \( q_1 \) and we locate all of \( Q_k \)'s adjacent qubits as \( Q_k1, Q_k2, ..., Q_{ki} \). Since the number of physical qubits may be greater than that of logical qubits, not every physical qubit has a corresponding logical qubit, thus the \( SWAP \) operation can only be applied to physical qubits. If the \( SWAP \) between \( Q_ka \) and \( Q_ki \) \((1 \leq i \leq 4)\) is lock free, then \( SWAP \) \( Q_ka, Q_ki \) will be regarded as a candidate. We repeat this process for all gates associated with gate \( g \) to get the candidate \( SWAP \) set denoted as \( C_{swap} \).

Next, the algorithm repeatedly select the best \( SWAP \) in \( C_{swap} \) and insert it into \( E \) until no positive-priority \( SWAP \) remains in \( C_{swap} \). The priority of a \( SWAP \) denotes the benefit it can bring to the circuit and we discuss how to calculate the priority for each \( SWAP \) in next subsection. In each selection iteration, the algorithm first calculates the priority for each \( g_{swap} \) in \( C_{swap} \). Then the algorithm inserts the gate with the highest priority in \( C_{swap} \) into \( E \) and updates the qubit locks to remove the no longer lock-free \( SWAPs \) from \( C_{swap} \).

An example shown in Fig.4 illustrates the remapping process. Suppose a \( CNOT \) between qubits \( q_1 \) and \( q_5 \) need be applied at cycle 2 in a 3 x 3 grid model. For an easy explanation, we suggest in this cycle all qubit locks are 0 except the lock of qubit \( q_3 \) is 3, which means that \( q_3 \) is busy. As shown in Fig.4(a), the algorithm inspects each free edge associated with \( q_1 \) and \( q_6 \) and calculates their priority. Because the lock of \( q_3 \) is larger than the current time, the edge between \( q_3 \) and \( q_6 \) is not free. Finally, all the candidate \( SWAPs \) are colorized and suppose the red one has the highest priority. Then as shown in Fig.4(b), the algorithm applies the red one and updates the lock of qubit \( q_1 \) and \( q_2 \) (suppose the duration of \( SWAP \) is 6 cycle). This update causes the edge between \( q_1 \) and \( q_4 \) is no longer free, and it is removed from \( C_{swap} \). The algorithm repeats the process and applies the \( SWAP \) between \( q_5 \) and \( q_6 \). In the end, no gate exists in \( C_{swap} \), which leads to the termination of this cycle.

### D. Design of Heuristic Cost Function

The heuristic cost function for \( SWAP \) \( g_{swap} \) is to measure the benefits that gate \( g \) can bring to solve the mapping problem. When comparing the priority between two \( SWAPs \), \( H_{basic} \) is compared first and \( H_{line} \) is compared only when two gates have the same \( H_{basic} \).

a) **Basic Priority:** Suppose a two-qubit gate \( g \) applied to logical qubits \( q_1 \) and \( q_2 \), \( \pi(q_1), \pi(q_2) \) denotes the distance of two qubits in the coupling graph \( M \). When the distance becomes 1, the gate fits the hardware limitation of the device. \( L(\pi(q_1), \pi(q_2)) \) calculates the distance of \( g \in I_{CF} \) based on the mapping \( \pi \). To evaluate a candidate \( SWAP \), we first temporarily use that \( SWAP \) to update \( \pi \) and get \( \pi_{new} \). Then we calculate how much \( \pi_{new} \) can reduce (or increase) the distance of all \( g \in I_{CF} \) compared to the original \( \pi \). Equation (1) shows the basic priority heuristic function \( H_{basic} \).

\[
H_{basic} = \sum_{g \in I_{CF}} L(\pi(g)) - L(\pi_{new}(g)) \tag{1}
\]

If the basic heuristic function of a specific \( SWAP \leq 0 \), this \( SWAP \) won’t shorten the total distance and cannot bring any benefit to the circuit. If we find no \( SWAP \) has a positive basic heuristic, there is no best \( SWAP \) gate. But sometimes it happens that neither \( SWAP \) gate nor other executable gates could be launched while all qubits are free. Such situations are called “deadlock” by us, and at this time we should just choose a \( SWAP \) with the highest priority to launch, even if its \( H_{basic} \) may not be positive.
Fig. 6: The circuit is going to apply CX on q1 and q6 while q5 will be locked for a long time. The SWAP between \{q1,q2\} and between \{q1,q4\} have the same $H_{basic}$. But routing q1 through q4 will be blocked by q5 while routing through q2 can get better parallelism.

Fig. 7: Example of heuristic search for the high parallelism SWAP. The number near the qubits denotes the qubit lock $t_{end}$.

b) Fine Priority: In many cases, there are several candidate SWAPs with the same basic priority, so we design fine priority as shown in Equation 2 which applies to 2D lattice model.

$$H_{fine} = -|VD(\pi_{new}, g) - HD(\pi_{new}, g)|$$

Functions HD and VD stand for the horizontal and vertical distance on the lattice between the two qubits of $g$. The main idea of fine priority is: suppose a two-qubit gate $g$ with specific distance $D = HD + VD$, there may be $C_D = \frac{VD}{HD}$, possibly the shortest routing way for $g$, which increases as $VD - HD$ decreases. Choose a suitable SWAP gate to make $VD - HD$ smaller may let the remapping algorithm perform better in the future.

Retaining more possible routing way can improve the parallelism. Fig. 6 shows a situation: $H_{basic}$ indicates the algorithm to choose SWAP that can let the CX gate has closer vertical and horizontal distance which can avoid waiting for $q_5$ to be free.

E. Example

Now we use an example shown in Fig. 7 to explain our algorithm. Suppose there is a 6-qubit device and we are given a gate sequence $I$ that contains a CX on $\{q_0,q_2\}$, a T on $\{q_1\}$ and a CX on $\{q_0,q_3\}$. The number near the qubit node represents the value of its $t_{end}$. All the three gates are CF gates. Due to the coupling limitation, CX on $\{q_0,q_3\}$ is not directly executable and SWAP is needed. The algorithm simulates the execution timeline and starts at cycle 0.

At cycle 0, the first gate "CX $q_0,q_2$" and the second gate "T $q_1$" are directly executable so both of them will be launched and qubits \{$q_0,q_1,q_2$\}'s $t_{end}$ locks are updated with the gate duration (T=1 cycle, CX=2 cycle). Each of \{$q_0,q_1,q_2$\} has bigger $t_{end}$ than current time and thus they are locked. Therefore the SWAP between \{$q_1,q_3$\} and \{$q_2,q_3$\} are blocked. SWAP between \{$q_3,q_5$\} with $H_{basic} < 0$ (which means the SWAP wont shorten the total distance of CF gates according to our heuristic cost function) moves q3 away from q0 and will not be inserted. As a result, no SWAP will be inserted in cycle 0 and the mapping $\pi$ stays unchanged.

At cycle 1, qubit q1 becomes free while q2 stays busy. The SWAP between \{$q_1,q_3$\} becomes free while the SWAP between \{$q_3,q_4$\} is still blocked. Therefore the algorithm knows that the SWAP between \{$q_1,q_3$\} can start earlier than SWAP between \{$q_3,q_4$\} and choose SWAP $q_3,q_1$ to solve the remapping problem. After launching "SWAP $q_1,q_3$", qubit locks of \{$q_1,q_3$\} are also updated by the sum of its start time (cycle 1) and the duration of SWAP (6 cycle) as 7.

V. EXPERIMENTAL EVALUATION

In this section, we evaluate CODAR with benchmarks based on the latest reported hardware models.

a) Comparison with Previous Algorithms: Several recent algorithms proposed by IBM [13], Siraichi et al. [20], Zulehner et al. [35] and Li et al. [13] try to find solutions of the qubit mapping problem with small circuit depth. Among them, Li’s SABRE [13] beats the other three in the performance of benchmarks, thus it is used for comparison in this paper.

b) Hardware Configuration: We test our algorithm on several latest reported architectures, including IBM Q20 Tokyo [18], IBM Q16 Melbourne [1], 6 x 6 grid model proposed by Enfield’s GitHub and Google Q54 Sycamore [3]. The gate duration difference configuration is based on experimental data of symmetric superconducting technology shown in Table I where two-qubit gate duration is generally twice as much as that of the single-qubit gate.

c) Benchmarks: To evaluate our algorithm, we totally collect 71 benchmarks which are selected from the previous work, including: 1) programs from IBM Qiskit [9]’s Github and RevLib [31]; 2) several quantum algorithms compiled from ScaffCC [2] and Quipper [13]; 3) benchmarks used in the best-known algorithm SABRE [13]. The size of the benchmarks ranges from using 3 qubits up to using 36 qubits and about 30,000 gates. For the IBM Q16, Q20 and 6 x 6 architectures, 68 benchmarks out of the 71 benchmarks except 36-qubit programs are tested. While all 71 benchmarks are tested on Google Q54 Sycamore.

Fig. 8: Speedup ratio of all 71 benchmarks compared between CODAR and SABRE in four architectures. The benchmarks are listed from left to right in the ascending order of the number of qubits used.

A. Circuit Execution Speedup

We collect the weighted circuit depth of the circuits produced by CODAR and SABRE for the 71 benchmarks. Initial mapping has been proved to be significant for the qubit mapping problem, and for a fair comparison, we use the same method as SABRE to create the initial mapping for the benchmarks. We use the depth of circuits produced by SABRE compared with the one of CODAR to show the ability of our algorithm to speed up the quantum program. As shown in Fig. 8.
the average speedup ratio of CODAR on four architecture models, IBM Q16 Melbourne, Enfield 6×6, IBM Q20 Tokyo and Google Q54 are respectively 1.212, 1.241, 1.214 and 1.258.

B. Fidelity Maintenance

Fidelity has been proved to be significant for the quantum computer in NISQ era. CODAR focuses on exploring the parallelism of the program to reduce the execution time and ignores the number of SWAPs inserted into the program. Compared to SABRE, CODAR may insert more SWAPs, which may bring more noise to the program. However, less execution time will improve the fidelity of the circuit on the contrary. To show CODAR’s ability to maintain the fidelity, we use a distributed noisy quantum virtual machine made by Origin Quantum [24], which is based on Qubit Dephasing and Damping model [22] to simulate the fidelity of 7 famous quantum algorithms. The result (Fig.9) indicates that CODAR can speed up the circuits and maintain the fidelity of the circuits at the same time.

![Fidelity of the circuit produced by CODAR and SABRE for several quantum algorithms.](image)

**Fig. 9:** Fidelity of the circuit produced by CODAR and SABRE for several quantum algorithms. When the noise mainly caused by qubit dephasing, CODAR performs better than SABRE and the fidelity of several circuits produced by CODAR reach nearly 1. When the noise mainly caused by qubit damping, CODAR performs about the same with SABRE.

VI. CONCLUSION

In NISQ era, most quantum programs are not directly executable because two-qubit gates can be applied between arbitrary two logical qubits while it can only be implemented between two adjacent physical qubits due to hardware constraints. To solve this problem, in this paper we propose CODAR that can transform the original circuit and insert necessary SWAP operations making the circuit comply with the hardware constraints. With the design of qubit lock and commutativity detection, CODAR is aware of the program context and the gate duration difference, which help CODAR remapper find the remapping with good parallelism and reduce QC’s weighted depth. Experimental results show that CODAR remapper can speed up quantum programs by 1.212~1.258 in different quantum architectures compared with the best known algorithm on average and maintain the fidelity of the benchmarks when running on OriginQ quantum noisy simulator.

ACKNOWLEDGEMENT

This work was partly supported by the grants of Anhui Initiative in Quantum Information Technologies (No. AHY150100), the National Natural Science Foundation of China (No. 61772487) and Anhui Provincial Major Teaching and Research Project (No. 2017jxmx0005). The authors would like to thank Prof. Peng Xu et al. from Wuhan Institute of Physics and Mathematics for their enthusiastic answers to the consultation on neutral atomic quantum technologies; and also thank Origin Quantum Computing Technology Co., Ltd. (OriginQ) for providing quantum noisy simulation platform for testing.
[29] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology*, 3(2):025004, 2018.

[30] Davide Venturelli, Minh Do, Eleanor G Rieffel, and Jeremy Frank. Temporal planning for compilation of quantum approximate optimization circuits. In 26th *IJCAI*, pages 4440–4446, 2017.

[31] R. Wille, D. Groe, L. Teuber, G. W. Dueck, and R. Drechsler. RevLib: An online resource for reversible functions and reversible circuits. In 38th *ISMVL*, pages 220–225, May 2008.

[32] Robert Wille, Lukas Burgholzer, and Alwin Zulehner. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In 56th *DAC*, page 142. ACM, 2019.

[33] Robert Wille, Oliver Kesznocz, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for nearest neighbor optimization of 1D and 2D quantum circuits. In 21st *ASP-DAC*, pages 292–297. IEEE, 2016.

[34] K. Wright, K. M. Beck, S. Debnath, J. M. Amini, Y. Nam, N. Grzesiak, J. S. Chen, N. C. Pisenti, et al. Benchmarking an 11-qubit quantum computer. *arXiv e-prints*, page arXiv:1903.08181, Mar 2019.

[35] A. Zulehner, A. Paler, and R. Wille. An efficient methodology for mapping quantum circuits to the IBM QX architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, July 2019.