QuCloud+: A Holistic Qubit Mapping Scheme for Single/Multi-programming on 2D/3D NISQ Quantum Computers

LEI LIU, Beihang University, China
XINGLEI DOU, Beihang University, China

Qubit mapping for NISQ superconducting quantum computers is essential to fidelity and resource utilization. The existing qubit mapping schemes meet challenges, e.g., crosstalk, SWAP overheads, diverse device topologies, etc., leading to qubit resource underutilization and low fidelity in computing results. This article introduces QuCloud+, a new qubit mapping scheme that tackles these challenges. QuCloud+ has several new designs. (1) QuCloud+ supports single/multi-programming quantum computing on quantum chips with 2D/3D topology. (2) QuCloud+ partitions physical qubits for concurrent quantum programs with the crosstalk-aware community detection technique and further allocates qubits according to qubit degree, improving fidelity, and resource utilization. (3) QuCloud+ includes an X-SWAP mechanism that avoids SWAPs with high crosstalk errors and enables inter-program SWAPs to reduce the SWAP overheads. (4) QuCloud+ schedules concurrent quantum programs to be mapped and executed based on estimated fidelity for the best practice. Experimental results show that, compared with the existing typical multi-programming study [12], QuCloud+ achieves up to 9.03% higher fidelity and saves on the required SWAPs during mapping, reducing the number of CNOT gates inserted by 40.92%. Compared with a recent study [30] that enables post-mapping gate optimizations to further reduce gates, QuCloud+ reduces the post-mapping circuit depth by 21.91% while using a similar number of gates.

CCS Concepts: • Computer systems organization → Quantum computing

Additional Key Words and Phrases: Quantum computing, multi-programming, qubit mapping, fidelity, Quantum OS

ACM Reference format:
Lei Liu and Xinglei Dou. 2024. QuCloud+: A Holistic Qubit Mapping Scheme for Single/Multi-programming on 2D/3D NISQ Quantum Computers. ACM Trans. Arch. Code Optim. 21, 1, Article 9 (January 2024), 27 pages. https://doi.org/10.1145/3631525

This work is supported by the Key-Area R&D Program of Guangdong (grant no. 2021B0101310002) and NSFC (grant no. 62072432).

This is an extension of a conference paper. The additional contributions of this article over the previously published work of Liu and Dou at HPCA-2021 [31] include the following. (1) We update the qubit mapping mechanism that works on both 2D and 3D devices. (2) We devise new mechanisms that make QuCloud crosstalk-aware, mitigating crosstalk errors in practice. (3) We design a new scheduler that can select appropriate concurrent quantum programs. (4) We conduct new experiments on the latest quantum computing devices with diverse configurations.

Authors’ address: L. Liu (Corresponding author) and X. Dou, Beihang University, No. 37 Xueyuan Road Zhongguancun, Haidian District, Beijing, China, 100191; e-mails: lei.liu@zoho.com, liulei2010@buaa.edu.cn.

This work is licensed under a Creative Commons Attribution International 4.0 License.

© 2024 Copyright held by the owner/author(s).
1544-3566/2024/01-ART9
https://doi.org/10.1145/3631525


1 INTRODUCTION

Quantum computers are gradually changing people’s lives. They exhibit significant potential in various critical applications, such as machine learning [8], database search [19], and chemistry simulation [23]. Many organizations are actively studying quantum computer systems. At present, quantum devices with hundreds of quantum bits (qubits) have been delivered by IBM [25, 56]. Modern quantum computers belong to the Noisy Intermediate-Scale Quantum (NISQ) category [42] – the qubits and the links between them have variational reliability and are vulnerable, making quantum computers susceptible to errors. Several competing qubit technologies are available for the physical implementation of quantum devices, e.g., trapped ion qubits [13], superconducting qubits [26], silicon qubits [33], and photonic qubits [54]. Among these technologies, using superconducting qubits can be promising [1, 12, 14, 28, 31].

Quantum programs need to be mapped onto quantum devices for execution. However, superconducting quantum devices do not have sufficient pair-wise qubit connections to satisfy all two-qubit gates in quantum circuits. Qubit mapping tackles this problem by using SWAP operations when the circuits are mapped. Thus, one qubit can be swapped to the neighbor of another qubit with a connection that two-qubit gates can be conducted. In addition, gate optimizations can reduce the number of gates and circuit depth. Some qubit mapping mechanisms map only one quantum program on a specific chip for higher fidelity [28, 36, 50]. Recent studies [12, 14, 31, 43] introduce multi-programming for higher throughput and resource utilization of quantum computers. Enabling multi-programming can effectively improve the utilization of robust qubits on quantum computers and speed up variational quantum algorithms (VQAs) [43]. However, it also brings new challenges. One problem is that the activity of one program can negatively affect the reliability of co-located programs because of (i) a limited number of robust qubits, (ii) crosstalk noise caused by unexpected interactions or in-corrected control of qubits [4, 37], and (iii) long qubit SWAP paths. Previous studies [12, 31] on multi-programming show that running two quantum programs on a specific quantum chip incurs a non-negligible reduction in fidelity.

The existing qubit mapping policies have several shortcomings when handling multi-programming cases. (1) The existing approaches often divide a large on-chip area with robust qubits into small regions so that the programs with more qubits cannot be satisfied. (2) When mapping multiple quantum programs, SWAP operations during the mapping transition for each quantum program lead to an unpredictable impact on fidelity and reliability. (3) Crosstalk errors [4, 37] degrade fidelity. (4) Scheduling concurrent quantum programs on a specific quantum chip in the cloud environment may lead to fidelity degradation and resource under-utilization [12, 31]. To this end, we propose QuCloud+ based on our prior studies [14, 31] to improve the throughput and resource utilization of NISQ computers in the cloud environment while reducing the negative impacts of multi-programming on fidelity. QuCloud+ has several key features. (1) QuCloud+ supports both single-program mapping and multi-program mapping and can easily switch across the two modes on demand. (2) QuCloud+ is crosstalk-aware. It mitigates crosstalk errors by orchestrating the qubit allocations and SWAP operations (Sections 4.2 and 4.3.2). (3) QuCloud+ partitions the physical qubits for concurrent quantum programs leveraging community detection technique [39], avoiding the waste caused by the topology-unaware algorithms. (4) QuCloud+ can enable inter-program SWAPs when quantum programs are allocated to neighboring qubits, reducing SWAP overheads. (5) QuCloud+ performs well on the quantum chips with 2D and 3D topology.

Compared with the existing typical multi-programming study [12, 31], QuCloud+ is a holistic qubit mapping approach, which works for both single and multi-programming on 2D and 3D NISQ chips. The experimental results show that QuCloud+ improves the fidelity by 9.03% and
6.84% on the real IBM Nairobi and simulated IBMQ Toronto, respectively; it also saves SWAPs required during mapping in 4-program multi-programming cases on IBMQ Toronto, reducing the number of CNOT gates inserted by 40.92%. Furthermore, when compared with a recent heuristic-based approach [30] that leverages post-mapping gate optimizations to further reduce the number of CNOT gates after the mapping process, QuCloud+ still improves the fidelity by 8.15% and 7.55% on IBM Nairobi and IBMQ Toronto, respectively, and reduces the post-mapping circuit depth by 21.92% in 4-program multi-programming cases while using a similar number of gates.

To sum up, we make the following contributions. (1) We design a new qubit mapping scheme, including a new approach (CDAP) that generates initial mapping for concurrent quantum programs and a mapping transition mechanism (X-SWAP) that enables inter-program SWAP when quantum programs are mapped adjacently, reducing SWAP overheads. (2) We further enhance our design based on the newly proposed quantum program profiling approach, and show that QuCloud+ has advantages on 2D quantum chips as well as the emerging 3D quantum chips. (3) We design crosstalk-aware initial mapping generation and crosstalk-aware mapping transition, mitigating the crosstalk problem on quantum devices. (4) We design the QuCloud+ scheduler that selects appropriate quantum programs for multi-programming.

2 BACKGROUND

2.1 Quantum Computing Mechanics
Quantum computing can solve conventionally hard problems [8, 19, 23, 42]. The foundation of quantum computing lies on qubits. \( |0\rangle \) and \( |1\rangle \) are two basis states of a qubit. The state of a qubit can be a linear combination of \( |0\rangle \) and \( |1\rangle \) as \( |\psi\rangle = \alpha |0\rangle + \beta |1\rangle \), where \( |\alpha|^2 + |\beta|^2 = 1 \). The state of a specific qubit can be manipulated using single-qubit gates, e.g., \( H \), \( X \), \( Y \), \( Z \), etc. When the qubit is measured, the measurement result gives either 0 with the probability of \( |\alpha|^2 \), or 1 with the probability of \( |\beta|^2 \). Two or more qubits can be entangled with two-qubit gates, e.g., Control-NOT (CNOT) gates. A CNOT gate flips the state of the target qubit when the control qubit is in the state \( |1\rangle \). Likewise, the state of a two-qubit system is represented by \( |\psi\rangle = a_{00} |00\rangle + a_{01} |01\rangle + a_{10} |10\rangle + a_{11} |11\rangle \). Any quantum gate involving three or more qubits can be decomposed with single- and two-qubit gates [5].

2.2 Quantum Computers
There are several competing technologies for the implementation of quantum computers [13, 26, 33, 54]. IBM develops superconducting quantum chips with Josephson-junction-based transmon qubits [26] and microwave-tunable two-qubit gates [44]. The physical qubits only have connections to neighbors. Figure 1 shows the topology of IBMQ Toronto.
2.3 Errors on Quantum Computers
NISQ computers have to face reliability challenges. As the physical qubits are fragile and susceptible to interference, the following errors may occur when quantum programs run on quantum computers. (i) Decoherence errors caused by short qubit state retention time [1]. (ii) Operational errors caused by error-prone quantum gates [1]. (iii) Readout errors caused by measurement operations [1]. (iv) Crosstalk errors caused by unexpected interactions between quantum gates or imprecise quantum control when quantum gates are executed simultaneously [45]. In practice, operational error rates, readout error rates, and the coherence time of a quantum chip are all reported in its calibration data [1]. The error rates vary over different qubits, links, and days. Crosstalk errors can be measured using Simultaneous Randomized Benchmarking (SRB) [16]. In practice, quantum programs should be mapped to qubits with lower error rates.

2.4 Quantum Programs and Quantum Circuits
A quantum circuit is the low-level representation of a high-level quantum program. A quantum circuit is composed of a series of quantum gates. For example, Figure 2(a) shows the circuit of the decomposed Toffoli gate, where each horizontal line represents a program qubit and each block and vertical line represents a single- and two-qubit gate, respectively. Figure 2(b) shows the Directed Acyclic Graph (DAG) of the circuit. The depth of the circuit is equal to the length of the DAG’s critical path. A quantum gate is logically executable when it has no unexecuted predecessors in the DAG. A CNOT operation cannot be executed unless the two program qubits involved are mapped physically adjacent. A SWAP operation can exchange the physical mapping between two program qubits [28]. Generally, it takes two steps for a heuristic-based policy to map a quantum program: (1) Initial mapping generation. The policy maps the program qubits onto physical qubits. (2) Mapping transition. The policy meets all two-qubit constraints by inserting SWAPs to the circuit so that every two-qubit gate in the program can be executed. After these two steps, the circuit can be further optimized by using commutation rules [22] or gate cancellation [32], etc. In this work, we focus on enhancing initial mapping generation and mapping transition.

3 MOTIVATIONS
3.1 New Architecture, Paradigm, and System Stack

3.1.1 Quantum Chip with 3D Topology. Superconducting quantum chips are often with 2D topology architectures. Their physical qubits are organized in a lattice layout on a 2D plane [14, 29, 31]. Connections only exist between neighboring physical qubits. SWAPs are required when two program qubits involved in a quantum gate are not mapped on neighboring physical qubits with interconnection. Some studies [9, 15, 20, 52] show that quantum computing benefits from the quantum chip topology with a higher dimension, e.g., a 3D quantum chip whose qubits are organized in a 3D lattice layout. A quantum chip with a higher dimension has a denser qubit topological structure, higher physical qubit degrees, and more direct interconnections between its physical qubits. These features reduce the additional SWAPs required during the mapping transition. Moreover, new advances in quantum chip manufacturing, e.g., qubit memory technology [38], qubit integration and packaging technology [35], and quantum chip architecture design [29], also provide possibilities for the devices with a denser structure and a higher dimension. The new trends motivate us to have new quantum chip topologies, e.g., 3D topology, and new mapping/allocation schemes for quantum devices with high-degree physical qubits. Even though near-term quantum devices do not have high-degree physical qubits and dense interconnections, they still have physical qubits with varying degrees, and 3D topology can be a new trend. The new mapping schemes can reduce SWAP overheads by leveraging the physical qubits with the highest degree on existing quantum devices.
3.1.2 Multi-programming on Quantum Computers. Enabling multi-programming on a specific quantum device (chip) is essential since we only have limited robust quantum resources in the NISQ era [12, 31, 42]. Studying multi-programming on current NISQ computers is meaningful for the following reasons. (1) It should be even more useful as qubit counts grow quickly on a specific device [3] and the global demands for quantum experiments increase [12, 31]; even the early step devices have various noise sources and inadequacy of current quantum chips to host a single large circuit. (2) Multi-programming can improve the qubit resource utilization and throughput for the expensive quantum computer, which is at least as important as multi-programming on classical computers. (3) It can speed up some quantum algorithms, such as VQAs [43] and Grover’s search [41]. For quantum computing service providers, multi-programming becomes more useful than ever as the number of qubits increases and researchers worldwide want to use quantum computers. However, multi-programming on quantum computers is not a free lunch. Several studies [12, 14, 31, 43] provide solutions to optimize throughput and fidelity in multi-programming scenarios, making multi-programming practical. This article proposes new approaches to address qubit mapping and crosstalk among concurrent quantum programs.

3.1.3 A New OS for Quantum Computers. It is the right time for the quantum computer to have a new OS – QuOS. QuOS is the manager of both hardware and software resources in the quantum computer. It is responsible for managing and configuring the qubits with varying hardware implementations, scheduling quantum programs, and providing the best mapping strategies for higher fidelity. Our work is among the first step studies [10, 21, 31] that discuss the prototype of OS for quantum computers. The quantum computer has different design principles from the classical Von Neumann architecture. The critical OS components, e.g., ISA, scheduling, process management, etc., are incompatible with the quantum architecture. Therefore, the OS’s design principles and implementation strategies for quantum computers are different from traditional ones. We are developing OS and runtime stack technologies for quantum computing.

3.2 Key Challenges – What Should We Do?

3.2.1 Noise on Quantum Computers - Fidelity. Quantum computers in the NISQ era are susceptible to errors. The state of a specific qubit can only keep in a short time (e.g., 30-100 μs) [1]. In reality, the error rates are variational across all qubits and links on NISQ computers. To map a specific quantum program, previous noise-aware mapping techniques for a single quantum program [36, 50] employ greedy or heuristic approaches to discover the mapping policies that use the most reliable qubits and links.

Many prior studies [31, 36, 50, 55] assume that errors on quantum computers are local and independent. However, crosstalk errors violate either the locality or the independence of quantum operations (or both) [45]. Crosstalk commonly exists on NISQ systems. Previous studies demonstrate that crosstalk errors mainly occur among CNOT pairs [37]. When two CNOT gates are executed simultaneously, the error rates of both CNOT gates are amplified. In this work, we use the error amplification ratio to indicate the effect of crosstalk, defined as \( r_{ij} = e_{ij}/e_i \); \( e_i \) denotes the error rate of gate i when it is executed independently; \( e_{ij} \) denotes the error rate of gate i when it is executed simultaneously with gate j.

3.2.2 Qubit Mapping Can Be More Efficient. Enabling multi-programming on a specific quantum chip brings challenges for existing qubit mapping technologies. (1) The available qubits are subject to being divided into smaller-scale segments. Some are suitable to being involved during mapping, but some are not due to their high error rates and weak links. Hence, some programs might have weak qubits when mapping multiple quantum programs, leading to unreliable results for concurrent quantum programs. (2) Although it is possible to combine multiple quantum
Table 1. Qualitative Comparison of Quantum Program Mapping Approaches

| Studies   | Approaches       | Optimization Objectives | Contributions                                                                 | Needs Improvements                      |
|-----------|------------------|--------------------------|--------------------------------------------------------------------------------|------------------------------------------|
| SABRE [28]| Heuristic        | Number of gates          | Simplified heuristic search space reduces algorithm complexity; reserve traversal for high-quality initial mapping | Unawareness of gate errors               |
| NASSC [30]| Heuristic        | Number of gates          | Leveraging post-mapping gate optimizations to reduce the number of CNOT gates after the mapping process | Unawareness of gate errors in initial mapping generation |
| FRP [12]  | Heuristic        | Fidelity                 | Multi-programming for higher throughput; fair qubit partitioning for fidelity; delayed instruction scheduling for lower decoherence errors | Unaware of topology in initial mapping   |
| QuCloud [31]| Heuristic     | Fidelity / number of gates | Community detection assisted qubits partitioning for fidelity; inter-program SWAPs for fewer additional gates | Unawareness of crosstalk                |
| Palloq [40]| Heuristic        | Fidelity                 | Placing idle qubits between concurrent circuits for crosstalk mitigation       | Under-utilization of robust qubits due to physical buffers |
| SATMAP [34]| Optimal         | Number of gates          | Maximum satisfiability (MAXSAT) based solver; reducing the size of MAXSAT constraints by exploiting the structure of quantum circuits | Unawareness of gate errors               |
| BIP [17]  | Optimal          | Fidelity / circuit depth | Binary integer program (BIP) based solver; merging CNOTs with adjacent SWAPs for fewer additional gates | Reducing crosstalk conflicts with the optimization of errors and depth |
| OLSQ-GA [49]| Optimal       | Number of gates          | SWAP absorption for fewer additional gates, i.e., incorporating SWAPs into other gates without additional cost | Unawareness of gate errors               |
| Work in [6]| Optimal         | Circuit depth            | Integer Linear Programming (ILP) formulation for the mapping problem             | Unawareness of gate errors               |
| Qmap [27] | Optimal + Heuristic | Number of gates / circuit latency | Considering actual gate duration to reduce circuit latency; MOVE operations for fewer additional gates | Unawareness of gate errors               |
| MUQUT [7] | Optimal + Heuristic | Number of gates / circuit depth / fidelity | Integer Linear Programming (ILP) formulation for the mapping problem; sliding the initial mapping over the chip for higher fidelity | Incompatible with non-grid architecture |

programs into one circuit and then map it using the mapping policies dedicated to single quantum programs, the following problems may occur. (i) Reliable resources (robust qubits) on a specific quantum chip are limited, and fairness is not guaranteed for allocating reliable qubits among co-located quantum programs. (ii) The number of concurrent quantum programs cannot be adjusted on-the-fly. For example, when a significant fidelity reduction happens for multi-programming, the parallel mode cannot be reverted to separate execution mode. (iii) Some optimization opportunities for multi-programming are missed. For example, the SWAP overheads can be reduced by leveraging inter-program SWAPs. If a program occupies any qubits on the shortest SWAP path for any other co-located program, the SWAP process has to suffer higher overheads, i.e., involving more SWAP operations across more qubits. More operations lead to more errors. Therefore, the overall fidelity is negatively affected. The previous multi-programming technique [12] supports co-locating two quantum programs. Some studies propose solutions for mapping a single quantum program [28, 50]. They mainly rely on heuristic policies or greedy algorithms. Yet, multi-programming raises more challenges, inspiring us to devise a new qubit mapping mechanism.

3.3 Related Work

The qubit mapping problem has been proved to be NP-Complete [46, 47]. In Table 1, we summarize some typical recent studies on quantum program mapping and compare them qualitatively. Generally, these studies employ either heuristic approaches or optimal approaches [48] for quantum program mapping. Heuristic approaches are flexible but do not guarantee optimal mapping results. Optimal approaches typically formulate the mapping problem as an equivalent mathematical opti-
mization problem and then apply a solver to solve it. They yield optimal solutions but have higher complexity. In terms of heuristic approaches, SABRE [28] reduces the heuristic search space and speeds up the mapping process. NASSC [30] reduces the number of gates in the post-mapping gate optimization process. Regarding multi-programming, several recent studies [12, 31, 40] are proposed. QuCloud [31] and FRP [12] devise new mapping algorithms, aiming to optimize fidelity in multi-programming. QuCloud is the first work that proposes the inter-program swap approach. Palloq [40] mitigates crosstalk errors in multi-programming cases. For the studies that employ optimal approaches, Qmap [27], OSLQ-GA [49], and BIP [17] leverage techniques such as MOVE operations or SWAP absorption to reduce the number of gates after mapping. SATMAP [34] reduces the problem size based on the quantum circuit structure. The work in [6] splits the circuits into blocks, enabling the trade-off between runtime and circuit depth by adjusting the block size. Some studies [7, 27] use heuristic and optimal approaches in different steps of the mapping process. More details are in Table 1. This article proposes a new crosstalk-aware heuristic mapping scheme for both single-programming and multi-programming on both 2D and 3D quantum chip architectures.

4 THE ART OF OUR DESIGN

Overview. We design a new qubit mapping mechanism —QuCloud+. The workflow of QuCloud+ is illustrated in Figure 3. First, the QuCloud+ scheduler selects quantum programs for multi-programming. Then, the quantum circuit profiler identifies the CNOT patterns of the quantum programs and provides the profiling information for mapping process reference. Next, QuCloud+ enables CDAP to generate the initial mapping for concurrent circuits and enables X-SWAP for mapping transition. Finally, the mapped quantum programs are forwarded to NISQ devices to execute.

In addition to working on the 2D quantum chips, QuCloud+ is also designed to work on emerging 3D quantum devices. We simulate a 3D quantum device with 27 physical qubits shown in Figure 4. QuCloud+ makes no difference whether the algorithm is deployed to 2D or 3D quantum chips. QuCloud+ can leverage the quantum chip’s high-degree qubits and denser connections to reduce SWAP overheads. Compared with the 2D IBMQ Toronto, qubits in the 3D quantum chip have a higher degree, and the connections between the qubits are denser. 2D quantum chips that have 4-qubit all-to-all connections (4-qubit buses [29]), e.g., IBMQ50 [24, 31] in Section 5.2.2, also have high-degree qubits that QuCloud+ can leverage to reduce SWAP overheads.

4.1 Quantum Program Profiling

We observe that in a specific quantum circuit, some qubits are consecutively involved in several CNOTs. If a qubit is used in a long sequence of consecutive CNOTs, the qubits it interacts with should be mapped as its neighbors to reduce additional SWAPs. This section shows the opportunity for optimizing qubit mapping by leveraging this observation.

We have a quantum program profiling approach to reveal the CNOT patterns. 1-qubit gates are not considered here as they can be performed without SWAPs. We show an example circuit
of qft_4’s 2-qubit gates in Figure 5(a); the profiling outputs the involvement lists in Figure 5(b). Each program qubit has an involvement list containing several tuples. Each tuple has two values, showing when and how long the program qubit is involved in consecutive CNOTs. We show these tuples in Figure 5(a), and further use dark gray to highlight the stages where qubits are involved in consecutive CNOTs. For instance, the stage highlighted with a red border has a tuple of <1, 6>, indicating that the program qubit $q_1$ is involved in 6 consecutive CNOTs, starting from the gate set with index 1. The profiling algorithm used to extract the involvement lists is detailed in Algorithm 1. The time complexity of the algorithm is $O(N)$. $N$ denotes the number of CNOT gates in the quantum programs. In addition, the profiling outputs the coupling strength matrix in Figure 5(c). The number in each cell denotes the number of interactions (i.e., CNOT gates) between two qubits.

These profiling results guide initial mapping generation and mapping transition. For initial mapping generation, the profiling results for the entire circuit are not helpful and can even be misleading in practice. Using the profiling results for some CNOTs in the front of the circuit is sufficient. This is because the mapping of program qubits changes during the mapping transition. We use the profiling results for a subcircuit with a minimum number of CNOTs that include all program qubits. For mapping transition, we enhance the look-ahead ability of the heuristic cost function with the involvement list. Instead of selecting a fixed number of CNOTs for the extended set in topological order for the heuristic cost function, this study selects CNOTs based on the involvement list. Refer to Sections 4.2.3 and 4.3.2 for more details about how the profiling results are used. The profiling results are only used during the mapping process. Therefore, in the following gate optimizations, even though there are changes in the profiling results due to gate reordering or cancellation [22, 32], the changes will not affect the mapping process.

### 4.2 The Design of a New Qubit Mapping Scheme

The initial mapping is critical for a specific quantum program. An excellent initial mapping reduces the SWAP overheads and fully utilizes the robust qubits and links on the quantum chips. Notably, to prevent high crosstalk errors, CNOT gates that cause high crosstalk errors should be avoided in the initial mapping for a specific quantum program and multi-programming cases. We have the following observations for the quantum chips and multi-programming. (1) The robust qubits and links on a specific quantum chip are limited. (2) Some qubits on the quantum chip have more connections to their surroundings, e.g., as shown in Figure 1, $Q_1$ has links to the three adjacent physical qubits, whereas $Q_6$ only has a link to one qubit. (3) Quantum programs have many intra-program qubit interactions and rare inter-program communications. Thus, the qubits in a specific program should be closely allocated (mapped); the allocations for qubits belonging to different quantum programs should avoid mutual interferences, fairly leveraging robust resources. (4) Enabling CNOTs with high crosstalk error possibility should be avoided.
ALGORITHM 1: Involvement List Generation

Input: Quantum programs
Output: Involvement lists (IL)

1. Initialize involvement gates (IG) as an empty list;
2. Initialize IL as a map, map each program qubit to an empty list;
3. for each quantum program do
   4. Gate_Set_Index ← 1;
   5. for each gate set of the program (subcircuit if IL is used for initial mapping generation) do
      6. Initialize Gates_To_Remove as an empty list;
      7. for each gate in IG do
         8. if any program qubit of this gate is involved in any CNOT in this gate set then
            9. Append the gate to Gates_To_Remove;
         end
      end
      10. for each gate in the gate set do
          11. for each program qubit of the gate do
              12. Get the list in IL corresponding to the program qubit;
              13. if the program qubit is involved in any gate in IG then
                  14. Add 1 to the second item of the last tuple in the list;
              else
                  15. Append <Gate_Set_Index,1> to the list;
              end
          end
      end
   16. for each gate in Gates_To_Remove do
      17. Remove the gate from IG;
   end
   18. for each gate in the gate set do
      19. Append the gate to IG;
   end
   20. Gate_Set_Index ← Gate_Set_Index + 1;
end

In this article, we propose a new technology – Community Detection Assisted Partitioning (CDAP) – to construct a hierarchy tree for searching the robust qubits tightly connected for initial allocation. Figure 6 shows how CDAP works. The figure shows IBM Q London’s architecture and calibration data obtained from IBMQ API [1]. The value in a node represents the readout error rate (in %) of the qubit, and the value on a link means the error rate (in %) of the CNOT operation. CDAP creates a hierarchy tree according to the coupling graph and calibration data. In the hierarchy tree, a leaf node denotes a specific physical qubit; an internal node represents the union of its sub-nodes. The values in nodes are the index of the physical qubits. CDAP then partitions by iterating the hierarchy tree from bottom to top to find a set of physical qubits for each quantum program. Finally, each quantum program is allocated to its corresponding physical qubits. We show the details below.
4.2.1 Quantum Chip Profiling. The hierarchy tree is a profile of a quantum chip, which helps to locate reliable qubit resources on the quantum chip and helps to avoid high crosstalk CNOT pairs. We construct the hierarchy tree based on the FN community detection algorithm [39]. The algorithm clusters the physical qubits on a specific quantum chip into communities. Qubits in a community have reliable and close interconnections. Couplings in a community have low crosstalk errors. By contrast, the links between communities have relatively low reliability.

In the algorithm, each community corresponds to a node in the hierarchy tree and is a candidate set of physical qubits for allocating qubits. Each tree node has a set of indices for the physical qubits in the corresponding community. When the algorithm starts, for each physical qubit, it creates a leaf node and adds it to the list of tree nodes, i.e., each physical qubit forms an individual community. Then, the algorithm keeps merging two nodes that can maximize the reward function $F$ until there is only one community containing all qubits. In each merge process, the algorithm takes two nodes $A$ and $B$ from the list that are with the largest value of $F(A, B)$, creates a new node and sets $A$ and $B$ as the new node’s left subtree and right subtree, and finally appends the new node to the list of tree nodes. The reward function $F$ is defined as the benefit of merging two communities:

$$F = Q_{\text{merged}} - Q_{\text{origin}} + \omega EVX,$$

where $Q$ is the modularity of a partition (i.e., $Q = \sum_i (e_{ii} - a_i^2)$ [39], in which $e_{ii}$ is the fraction of within-group edges in group $i$, and $a_i$ is the fraction of all edges associated with vertices in group $i$). A higher value of $Q$ indicates a better partition. $Q_{\text{origin}}$ and $Q_{\text{merged}}$ denotes the modularity of the original partition and the new partition after merging the two communities, respectively. $E$ denotes the average reliability (i.e., $1$ minus the error rate) of CNOTs on the inter-group edges of the two communities. $V$ denotes the average reliability of readout operations on the qubits of the two communities. $X$ denotes the average conditional reliability of CNOTs on the within-group edges that have a crosstalk CNOT in the other community. CDAP considers physical topology, CNOT and crosstalk error rates when performing partitioning using the reward function $F$. $\omega$ is a weight parameter. For a specific quantum chip, we can change the value of $\omega$ for adjusting the weight of physical topology and the error rates. If $\omega = 0$, CDAP conducts partitioning completely according to physical topology. The weight of error rates rises as $\omega$ increases. If $\omega$ keeps increasing, the weight of the error rates will exceed the weight of the physical topology, resulting in the degradation of CDAP to a greedy algorithm mainly based on error rates. Details on how the value of $\omega$ is selected are in Section 4.2.4.

The hierarchy tree has several features: (1) Every node in the hierarchy tree is a candidate set of physical qubits for initial allocation. (2) The physical qubits in a node (i.e., a community) are tightly interconnected. (3) The qubits with a low readout error rate and robust links are preferentially merged. Thus, the more robust the qubit set is, the higher the node depth. Whether the hierarchy tree is balanced doesn’t impact the qubit mapping result, as the most reliable set of physical qubits can always be selected for a specific program. The hierarchy tree helps to locate the robust resources on quantum computers, providing better initial mapping for quantum programs.

Further, we explain why the hierarchy tree helps to select the initial allocation with the example shown in Figure 6. (i) $Q_0$ and $Q_1$ are merged first because the link between them has the lowest error rate. (ii) Then, $Q_2$ instead of $Q_3$ is merged into the community $\{0, 1\}$, though the link $Q_1-Q_3$ has a lower CNOT error rate than $Q_1-Q_2$. This is because the algorithm tends to merge more interconnected nodes into one community, avoiding the waste of robust physical qubits. Likewise, $Q_3$ and $Q_4$ are merged. (iii) Finally, all qubits are merged as the root of the hierarchy tree. The algorithm avoids wasting robust resources caused by the topology-unaware greedy algorithm and improves resource utilization.
ALGORITHM 2: Qubit Partition

\textbf{Input}: Hierarchy tree, Quantum programs
\textbf{Output}: Partition

1 \textbf{if} there is only one quantum program to be mapped \textbf{then}
2 \hspace{1em} Add the root of the Hierarchy tree to Partition and return Partition;
3 \textbf{end}

4 Sort quantum programs in descending order of CNOT density;
5 \textbf{for} each quantum program \textbf{do}
6 \hspace{1em} Initialize candidate nodes (C) as an empty set;
7 \hspace{1em} \textbf{for} each leaf node in the hierarchy tree \textbf{do}
8 \hspace{2em} Search a node that has enough number of on-chip qubits for allocating the program from the leaf to its parent nodes;
9 \hspace{2em} Add the node to C;
10 \hspace{1em} \textbf{end}
11 \hspace{1em} \textbf{if} C \textbf{is empty} \textbf{then} \# The current qubit state cannot meet the requirements
12 \hspace{2em} Fail and revert to separate execution;
13 \hspace{1em} \textbf{end}
14 \hspace{1em} Construct a tuple for each node in C for sorting (detailed in Section 4.2.2);
15 \hspace{1em} Add the first node in sorted C to Partition;
16 \hspace{1em} Remove qubits in the node from all other nodes in the hierarchy tree;
17 \hspace{1em} \textbf{if} the node’s sibling node is isolated \textbf{then} \# The sibling node has no path to other nodes in the hierarchy tree
18 \hspace{2em} Remove qubits in the sibling node from its parent nodes;
19 \hspace{2em} Remove the link from the sibling node to its parent;
20 \hspace{1em} \textbf{end}
21 \textbf{end}
22 Return Partition.

For constructing the hierarchy tree using the FN algorithm [39], the time complexity is $O(N^4)$. Here, $N$ is the number of physical qubits on the quantum chip. Doing so does not impede practice because the calibration data stays mostly the same (e.g., IBM calibrates the devices once a day [1]). The hierarchy tree only needs to be constructed once during each calibration cycle. The hierarchy tree can be reused without incurring additional computing overheads.

4.2.2 Qubit Partitioning for Concurrent Programs. For multi-programming cases, Algorithm 2 partitions the qubits according to the hierarchy tree. The algorithm prioritizes the programs with a higher value of the CNOT density, which is defined as (the number of CNOT instructions) / (the number of program qubits). A higher value of CNOT density indicates that more CNOTs are executed on a specific number of physical qubits. Thus, quantum programs with higher CNOT density are more susceptible to CNOT gate errors. For each program, the algorithm searches the hierarchy tree from bottom to top to find the candidate qubit sets available.

Selecting the qubit set for a specific quantum program is based on the following principles. (1) Candidates with dense coupling can reduce SWAPs. (2) Candidates with fewer physical qubits can save resources. (3) Candidates with more reliable qubits can improve fidelity. (4) Candidates with fewer crosstalk-prone CNOT pairs lead to fewer barriers and lower decoherence errors. We construct a tuple for each candidate for sorting. The tuple has four items. For the first item, we sample at most 500 possible mappings for the quantum program on this qubit set candidate. Then, we have the average of the shortest path lengths between any two program qubits for each possible mapping; the smallest average value is used as the first item in the tuple. The second is the number of physical qubits in the candidate. The third is the average error rate of the candidate. For the fourth item, we get all possible CNOT pairs and find the crosstalk-prone ones (CNOT pairs with an error amplification ratio greater than 2 are crosstalk-prone ones). Then, the number of crosstalk-prone CNOT pairs is used as the fourth item in the tuple. Finally, we sort the candidates
ALGORITHM 3: Qubit Allocation

**Input:** Quantum program, Involvement lists, Coupling strength matrix  
**Output:** Layout

1. Get the subgraph containing all physical qubits assigned to the quantum program;
2. Get the degree of each physical qubit in the subgraph;
3. Get the degree of each program qubit according to the coupling strength matrix;
4. Initialize the list of pending program qubits (P);
5. Sort P based on the first tuple in each program qubit’s involvement list;
6. Initialize candidate qubits (C) as a list containing all physical qubits whose degree is greater than or equal to the degree of the first pending program qubit in P;
7. if $C$ is not empty then
   8. Select the physical qubit with the lowest average CNOT error rate in C;
   else
   9. Select the physical qubit with the lowest value of $1/(\text{degree} \times \text{average CNOT reliability})$ in the subgraph;
10. end
11. Map the first pending program qubit in P to the selected physical qubit;
12. Remove the first pending program qubit from P;
13. for each pending program qubit in P do
   14. Get neighbors of mapped physical qubits;
   15. Select the neighbor that provides the most reliable SWAP paths between this program qubit and all other program qubits that have been mapped;
   16. Map the pending program qubit to the selected neighbor;
   17. Remove the pending program qubit from P;
18. end

in ascending order based on the tuple by comparing the low-index values. If low-index values are equal, high-index values are compared. The first one in the sorted candidates is selected for mapping the quantum program.

In Algorithm 2, searching for a candidate node from each leaf node takes $O(N)$ time, where $N$ denotes the number of leaf nodes (physical qubits). It takes $N$ steps to traverse from the leaf to the root in the worst case, where each internal node has at least one leaf node among its children. All leaf nodes are traversed in the algorithm. This leads to a time complexity of $O(N^2)$ for searching all candidate nodes from the hierarchy tree. The time complexity for iterating the candidate nodes is $O(1)$, as the number of candidate nodes is insignificant. The time complexity for constructing the tuple for each candidate is $O(N^2)$.

### 4.2.3 Qubit Allocation Leveraging High-Degree Qubits

SWAP operations are required when two program qubits are not mapped adjacently. Allocating the program qubit with more interactive neighbors to a physical qubit with a higher degree is effective in reducing the SWAP overheads, especially for future quantum chips with a denser structure and higher dimension (e.g., 3D quantum chips). We devise a new qubit mapping strategy that utilizes high-degree physical qubits to enhance initial mapping. Details are in Algorithm 3. First, we have the degree of physical qubits and program qubits. The degree of a physical qubit is the number of its connections to adjacent physical qubits. The degree of a program qubit is the number of program qubits that interact with it. Next, all qubits in the quantum program are added to a list, waiting for allocating/mapping. We sort these pending program qubits in ascending order of the first tuple’s first value in each qubit’s involvement list. Then, we select a high-degree physical qubit to map the first pending program qubit. Details are in lines 6 to 13 in Algorithm 3. Finally, we select the most appropriate neighbor of mapped physical qubits for each of the pending program qubits. We map the program qubit onto the physical qubit that provides the most reliable SWAP paths between mapped program qubits and this program qubit.
The average number of redundant qubits in the hierarchy tree varies with $\omega$. A gray dot with darker color represents that more cases are overlapping in this result. The knee solution refers to the value of $\omega$, which makes the change of redundant qubits slow down with the increase of $\omega$. We use the knee solution because it can reduce the redundant qubits as much as possible without making the community partitioning depend too much on error rate.

The time complexity of qubit allocation is $O(N^2)$. $N$ is the number of program qubits. The time complexity is calculated as the multiplication of the number of program qubits ($O(N)$), the number of mapped physical qubits’ neighbors ($O(1)$), the time complexity to calculate the reliability score for each neighbor ($O(N)$).

### 4.2.4 Discussion

Our design merges the reliable qubits with robust links and lower read-out error rates into a specific community in each iteration. Unreliable qubits would be added into the community at last. When performing qubits allocation, CDAP searches the hierarchy tree from bottom to top to find candidates for partition. Unreliable qubits are less likely to be selected, thereby improving the overall fidelity. Using CDAP might lead to a case in which the number of allocated qubits for a quantum program exceeds the number of qubits the program needs. For example, suppose that a 4-qubit quantum program is mapped on the quantum chip in Figure 6. In that case, the only available community is the root of the hierarchy tree, i.e., $\{0,1,2,3,4\}$, leaving one qubit unmapped/unused, i.e., the redundant qubit. To avoid waste, we label these redundant qubits to be used when a quantum program is being allocated to an adjacent community.

We define the maximum redundant qubits for a specific node in the hierarchy tree, which refers to the maximum possible number of unused qubits when a quantum program is allocated to the community. The number of maximum redundant qubits of a node is node.n_qubits - (1 + max(node.left.n_qubits, node.right.n_qubits)). We observe that the increase of $\omega$ in the reward function leads to the degradation of the hierarchy tree, i.e., in each merge process when constructing a hierarchy tree, only one leaf node containing one qubit is added to the new community. In this case, the number of maximum redundant qubits of the new community is 0. Thus, the increase of $\omega$ leads to a reduction in average redundant qubits. As illustrated in Figure 7, we vary the $\omega$ from 0 to 2.5 for 20 different calibration data of IBMQ Toronto and record the average number of the redundant qubits in the hierarchy tree. We take the knee solution, in which the value of $\omega$ is 0.4. In this case, CDAP considers both physical topology and the error rate.

### 4.3 The Design of the Mapping Transition Scheme —X-SWAP

Multi-programming brings new challenges for mapping transition. In this article, we design the X-SWAP, which includes inter- and intra-program SWAPs. It provides the ideal SWAP solution by considering the SWAP possibilities across all qubits. Inter-program SWAPs are more likely to
Fig. 9. (a) $P_1$ and $P_2$ are mapped on a quantum chip with 9 qubits. The next gate to be solved is CNOT, which involves $q_1$ and $q_5$. (b) X-SWAP scheme takes shortcuts to satisfy the constraint of CNOT $q_1$, $q_5$.

occur when the concurrent quantum programs are mapped to adjacent physical qubits. The cost of inter-program SWAPs can be less than that in cases in which only intra-program SWAPs are used. The next section shows the details.

4.3.1 The Advantages of Inter-program SWAP. We find two main advantages of inter-program SWAP. First, an inter-program SWAP can replace several intra-program SWAPs. Figure 8 shows a case in which two quantum programs are mapped on a specific quantum chip. Figure 8(a) shows two quantum programs ($P_1$ and $P_2$) and (b) illustrates how they are mapped on a quantum chip with 6 physical qubits. According to the initial mapping, for $P_1$, the $g_1$ and $g_2$ can be executed directly without any SWAP operations. However, the $g_3$ cannot be executed directly unless a SWAP operation between $q_2$ and $q_3$ is executed first. The same thing happens for $P_2$. A SWAP operation between $q_4$ and $q_6$ should be involved before $g_6$ can be executed. To sum up, two intra-program SWAPs are needed for this case. By contrast, if the two programs could be mapped together, the inter-program SWAP operation could be enabled. Figure 8(c) shows that the new policy only needs one inter-program SWAP $\{q_1, q_5\}$, achieving the same goal but having lower overheads.

Second, inter-program SWAPs take shortcuts. For instance, Figure 9(a) shows two quantum programs mapped on a quantum chip with nine qubits. $q_1$ and $q_5$ are not mapped physically adjacent; SWAPs are required to satisfy their constraint to make CNOT $q_1$, $q_5$ executable. As illustrated in Figure 9(b), an inter-program SWAP, i.e., $\{q_1, q_9\}$, takes only one SWAP operation to move $q_1$ and $q_5$ adjacent. By contrast, to achieve the same goal, the previous intra-program scheme has to introduce three SWAPs, i.e., $\{q_1, q_2\}, \{q_1, q_3\}, \{q_1, q_4\}$. Briefly, enabling inter-program SWAPs could result in fewer SWAPs in the cases in which multiple quantum programs are mapped as neighbors on a specific quantum chip, therefore reducing the SWAP overheads and benefiting the overall fidelity.

4.3.2 X-SWAP. Instead of generating a schedule for each quantum program separately and merging them, we are the first to design an approach (X-SWAP) that produces the global scheduling solution for all programs simultaneously and enables inter-program SWAP operations. The design details are as follows.

Heuristic search space. To show our design, we have a circuit for a quantum program $P$ in Figure 10. The CNOT gates in the circuit can be divided into 4 gate sets. A specific gate set includes CNOTs that can be executed in parallel. The first gate set is the Front Layer (denoted as F) of $P$, which denotes the set of all gates without unexecuted predecessors in the DAG of $P$. Critical Gates (CG) denote the set of CNOT gates in F with successors on the second gate set. For example, in F, $g_1$ has a successor $g_3$ on the second gate set, but $g_2$ has no successors. Thus, $g_1$ is a critical gate; $g_2$ is not. If the critical gate $g_1$ is executed and removed from the DAG, the data dependency
of g₃ is resolved and the front layer F will be updated. By contrast, handling g₂ first doesn’t help to update F.

We build the mapped circuit by creating an empty circuit, then adding SWAPs and executable quantum gates to the circuit. For each program Pᵢ, if there are quantum gates that can be executed directly in Fᵢ from DAGᵢ, we remove them from DAGᵢ and add them to the mapped circuit. When there are no quantum gates that can be executed directly, SWAPs are needed for mapping transition. Among all of the CNOTs that cannot be executed directly, the data dependency of critical gates needs to be handled first for updating the front layer and reducing the post-mapping circuit depth. Thus, we only search the SWAPs associated with qubits in critical CNOT gates. Figure 11 shows an example, in which g₁ and g₃ are critical gates illustrated in DAGs. All SWAPs associated with the critical gates are SWAP candidates. They are highlighted on the coupling graph of the quantum chip. The best SWAP among candidates is selected according to the heuristic cost function (details refer to the following). We add the SWAP to the mapped circuit, which updates the mapping accordingly. Some CNOTs become executable when their constraints are eliminated using SWAPs. This repeats until the constraints of all CNOTs in the DAG are satisfied.

**Design of the heuristic cost function.** The mapping policy should keep qubits belonging to a specific program close to each other. Otherwise, high SWAP overheads will occur between two qubits mapped far away from each other when a CNOT operation is required. Nearest Neighbor Cost (NNC) is the length of the shortest path between two program qubits mapped on a quantum chip. NNC-based heuristic function (H) is used in SABRE [28] to choose the best SWAP among the SWAP candidates. We also use it as a component in our approach. It represents the sum of the cost in the front layers and the cost in the extended sets [28]. The extended set contains a fixed number of closest successors of the gates in the front layer. Each set’s cost is calculated as the averaged NNC of all CNOT gates in the set. The NNC-based heuristic function H ensures that the algorithm terminates. By employing H, X-SWAP can select SWAPs with low heuristic function values that do not increase the NNC value. This means that after using these SWAPs, the mappings for program qubits involved in the same CNOT will be closer and eventually become adjacent, making the CNOT executable.

We optimize H by selecting gates for the extended set based on the *involvement list*. An extended set is introduced in SABRE [28] for look-ahead ability. However, the look-ahead ability is limited because the size of the extended set is fixed, and the gates in the extended set are added in a topological order. Thus, not all gates that contribute to a proper SWAP can be added to the extended set. Some gates added to the extended set can even be misleading, resulting in SWAPs that would incur higher SWAP overheads. To tackle this problem, we select gates for the extended set according to the *involvement list*. When all gates in the front layer are not executable, we initialize the extended set as empty. We first get the gate set index of each gate in the front layer. For each program qubit, we get the last tuple from its *involvement list* that satisfies the constraint — the tuple’s first item is less equal than the gate set index of each gate in the front layer. A tuple means that a program qubit is involved in consecutive CNOTs. For each CNOT in the set of successive CNOTs, if it is neither applied to the mapped circuit nor in the front layer, it is added to the extended set.

We also prioritize inter-program SWAPs on the shortest SWAP path. Given the coupling graph of a quantum chip and the qubit allocations, we define the term *distance matrix D*, in which each cell denotes the the shortest path length between two physical qubits on the quantum chip. For each program Pᵢ, we define Dᵢ as the shortest path matrix for qubits that have not been occupied by other programs, including unused physical qubits and the physical qubits on which Pᵢ is mapped. In essence, D represents the shortest path matrix to perform mapping transition for concurrent quantum programs with inter-program SWAPs enabled; Dᵢ represents the shortest path matrix to perform mapping transition for Pᵢ without using inter-program SWAPs. For a two-qubit gate g,


ALGORITHM 4: X-SWAP Mechanism

```
Input: Quantum chip coupling graph, Quantum programs, Initial mapping
Output: Final Schedule (FS)
1. Generate a DAG for each program;
2. Generate a Front Layer for each DAG;
3. while not all gates’ constraints are satisfied do
   4. if gates that can be executed directly exist then
      5. Append executable quantum gates to FS;
      6. Remove executable quantum gates from the DAG and update the Front Layer;
   7. else
      8. for each Front Layer do
         9. Append CNOTs in the Front Layer that have subsequent CNOTs on the second layer to Critical Gates;
      10. Add SWAPs associated with the qubits in Critical Gates to SWAP candidates;
      11. Find a SWAP with the lowest crosstalk error from the SWAP candidates with the minimum value of score(SWAP);
      12. Append the SWAP to FS and update mapping;
   13. end
   14. end
   15. end
16. Return FS.
```

we denote the two program qubits involved in g as g.q₁ and g.q₂. We define the physical qubit on which a program qubit q is mapped as σ(q). The shortest path between two qubits involved in a two-qubit gate minus 1 is the minimum number of SWAPs required to satisfy their constraint.

In our design, if D₁[σ(g.q₁)][σ(g.q₂)] is greater than D[σ(g.q₁)][σ(g.q₂)] for a two-qubit gate g in a specific quantum program P₁, it means that inter-program SWAPs outperform intra-program SWAPs when satisfying the constraint of g. In such cases, the X-SWAP scheme should enable inter-program SWAPs to reduce the quantum programs’ mapping transition cost. For example, in Figure 9(b), as it takes either 1 inter-program SWAP or 3 intra-program SWAPs to satisfy the constraint of CNOT q₁, q₅, it delivers D[σ(q₁)][σ(q₅)] = 2 and D₁[σ(q₁)][σ(q₅)] = 4. In terms of inserting SWAPs to satisfy g’s constraint, we define the number of SWAPs saved by X-SWAP scheme as follows:

\[
gain(g) = D[σ(g.q₁)][σ(g.q₂)] - D₁[σ(g.q₁)][σ(g.q₂)],
\]

and we define the heuristic cost function as follows:

\[
\text{score}(\text{SWAP}) = H(\text{SWAP}) + \sum_{F_i \in F} \frac{1}{|F_i|} \sum_{g \in F_i} \text{gain}(g)I(\text{SWAP}, g).
\]

As the sizes of different front layers vary, the gain is normalized to their sizes accordingly. The shortest SWAP path for satisfying g’s constraint involves several qubits. When the program qubits involved in the current SWAP operation are on the shortest SWAP path, I(\text{SWAP}, g) = 1. Otherwise, I(\text{SWAP}, g) = 0. This indicates that only the SWAPs on the shortest SWAP path are prioritized. The SWAP with the minimum value of score is the best among the candidates. When there are more than one SWAP with the minimum score, we select the SWAP with the lowest crosstalk error to mitigate crosstalk.

Our approach has the same space overhead as SABRE [28] because they both have the same heuristic search space size, i.e., O(N), where N is the number of physical qubits. The O(N) heuristic search space size is superior over an exhaustive search solution [55] that has O(\exp(N)) search space size. Regarding time complexity, moving two program qubits adjacent by inserting SWAPs has the same time complexity as SABRE, i.e., O(N^{2.5}). It is calculated as the multiplication of the
ALGORITHM 5: The QuCloud+ Scheduler

**Input:** List of incoming jobs (IJ), Hierarchy tree

1. **while not all jobs in IJ are scheduled** do
2.    Initialize `cur_job_set` as a list having the first job in IJ;
3.    Initialize candidates as a list containing the following N jobs that can be co-located with the first job;
4.    Sort candidates in descending order of the fitness with the first job;
5.    **for each candidate in candidates** do
6.        Append candidate to `cur_job_set`;
7.        **for each job in cur_job_set** do
8.            Calculate job’s sep_EPST;
9.            Calculate job’s co_EPST;
10.           Calculate job’s EPST_violation as 1 - (co_EPST/ sep_EPST);
11.           **if EPST_violation > ε then**
12.               Remove candidate from `cur_job_set`;
13.               Break;
14.           **end**
15.    **end**
16.    **if cur_job_set’s length = M then**
17.        Break;
18.    **end**
19.    Algorithm 4 is called to map programs in `cur_job_set`;
20.    Submit `cur_job_set` for execution;
21.    Remove all programs in `cur_job_set` from IJ.
22.  **end**

maximum number of SWAP candidates (O(N)), the time to calculate the heuristic cost function for a SWAP candidate (O(N)), and the maximum steps required to move two program qubits together (O(√N)).

4.3.3 Discussion on Security and Privacy Issues. Multi-programming and inter-program SWAPs may raise security and privacy concerns in some cases. The execution of the pre-SWAP programs affects the qubit state and errors; therefore, the state and errors may get transferred to the post-SWAP program. This is harmful to the security of quantum applications. To this end, a solution is to enable multi-programming on demand according to the specific security requirements. For quantum applications with high-security demands, such as quantum key distribution [53] and quantum communication [18], multi-programming should be disabled. For quantum applications that prioritize efficiency over security, such as quantum machine learning [8] and quantum simulation [23], multi-programming can be enabled to enhance the throughput and resource utilization of quantum computers.

4.4 The Design of the QuCloud+ Scheduler

Randomly selected quantum programs for multi-programming workloads often lead to qubit resource underutilization or fidelity degradation. We design the QuCloud+ scheduler to have appropriate combinations of concurrent quantum programs that promise high fidelity and high throughput simultaneously. Algorithm 5 shows its core scheduling logic. The first step is to generate co-location candidates. The scheduler selects candidates that can be co-located on the quantum chip with the first incoming job $P_0$. The total number of qubits in the selected workloads cannot exceed that of the quantum chip. The selected candidates are sorted in descending order of the fitness with $P_0$. Quantum programs with fewer qubits require fewer resources and help to protect fidelity when robust resources are limited. Moreover, quantum programs with similar circuit depths are unlikely to cause serious decoherence errors in multi-programming cases. As the deeper quantum circuit
requires a longer execution time, other co-located circuits may suffer decoherence in waiting for the deeper circuit execution completion. Therefore, the QuCloud+ scheduler selects quantum programs with similar depths for co-location. The fitness between $P_0$ and another quantum program $P_i$ is calculated as follows:

$$\text{fitness} = \frac{1}{(P_i.n\_qubits \times S(|P_i.depth - P_0.depth|))},$$

(4)

where $S(x)$ denotes the Sigmoid function, i.e., $S(x) = \frac{1}{1 + e^{-x}}$. The function can map $x$ in the interval $[0, +\infty]$ to the interval $[0.5, 1]$. The greater the difference between the depths of $P_0$ and $P_i$, the closer the value of $S(x)$ is to 1. By contrast, the smaller the difference is, the closer the value of $S(x)$ is to 0.5, i.e., the fitness is doubled. A higher value of fitness indicates that the programs can be more appropriate for co-location.

The second step is threshold-based workload selection. The Estimated Probability of a Successful Trial (EPST) is proposed to estimate the fidelity of executing a circuit on a specific quantum chip. The EPST is derived by calculating the probability of error-free execution for all gates and readout operations. It is defined as the multiplication of the reliability of each gate and readout operation:

$$\text{EPST} = r_{2q}^{\text{[CNOTs]}} r_{1q}^{\text{[1qgates]}} r_{ro}^{\text{[qubits]}},$$

(5)

in which $r_{2q}$, $r_{1q}$, and $r_{ro}$ denote the average reliability of CNOTs, the average reliability of 1-qubit gates, and the average reliability of readout operations on the allocated physical qubits, respectively. $\text{[CNOTs]}$, $\text{[1qgates]}$, and $\text{[qubits]}$ denote the number of CNOT gates, the number of single-qubit gates, and the number of the quantum program’s qubits, respectively. A higher EPST indicates that the quantum program is mapped to a set of physical qubits with more robust resources and that a higher PST may be obtained during the execution in practice.

Separate EPST (Sep_EPST) is the maximum EPST that a program can achieve. To obtain the Sep_EPST, Algorithm 2 is called to allocate a set of physical qubits for every single quantum program. Co-located EPST (Co_EPST) represents the EPST when multiple quantum programs are co-located on a quantum chip. To obtain the Co_EPST, Algorithm 2 is called to generate a partition for all programs in the workload. EPST violation is calculated according to Sep_EPST and Co_EPST of each quantum program. If the EPST violation of all quantum programs in the workload is less than the threshold $\varepsilon$, the workload can be co-located on the chip. Experimental results in Section 5.2.4 show that $\varepsilon=0.15$ yields higher throughput and fidelity than randomly selected two-programmed combination cases. Otherwise, they cannot be co-located. QuCloud+ scheduler supports to co-locate more than two quantum programs on a quantum computer. We set the maximum number of co-located programs to $M$ (3 by default) to avoid severe performance degradation. To ensure efficiency and fairness, the size of the candidates is set to $N$ (10 by default in practice). More details are in Algorithm 5.

5 EVALUATIONS

5.1 Methodology

5.1.1 Metrics. The following metrics are used in evaluations.

**Probability of a Successful Trial (PST).** The PST [12, 50] is used to evaluate the fidelity of the quantum program execution. The PST is defined as the fraction of trials that produce a correct result. A higher PST value indicates a higher level of fidelity. To get the PST, we map and run each workload on the target quantum chip for 8192 trials (same as [12]).

**Number of post-mapping CNOT gates.** We use the number of post-mapping CNOT gates [28, 30] to evaluate the policy’s ability to reduce the SWAP overheads when mapping multiple quantum programs.
Table 2. Quantum Programs Used for Evaluations

| Type       | ID  | Name       | Num. of qubits | Num. of CNOTs | Type       | ID  | Name       | Num. of qubits | Num. of CNOTs |
|------------|-----|------------|----------------|---------------|------------|-----|------------|----------------|---------------|
| tiny-sized | 1   | bv_n3      | 3              | 2             | 15         | 4gt4-v0_72 | 6              | 113            |
|            | 2   | bv_n4      | 4              | 3             | 16         | af_276    | 6              | 336            |
|            | 3   | perex_3    | 3              | 7             | 17         | alu-bdd_288 | 7              | 38             |
|            | 4   | toffoli_3  | 3              | 6             | 18         | ext2_227  | 7              | 275            |
|            | 5   | fredkin_3  | 3              | 8             | 19         | ham7_104  | 7              | 149            |
|            | 6   | xor5_254   | 6              | 5             | 20         | C17_204   | 7              | 205            |
| small-sized| 7   | 3_17_13    | 3              | 17            | 21         | bv_n10    | 10             | 9              |
|            | 8   | 4mod5-v1_22| 5              | 11            | 22         | ising_model_10 | 10             | 90             |
|            | 9   | mod5nils_65| 5              | 16            | 23         | qft_10    | 10             | 90             |
|            | 10  | alu-v0_27  | 5              | 17            | 24         | syst6-v0_111 | 10             | 98             |
|            | 11  | decod24-v2_43 | 4           | 22            | 25         | sym9_146  | 12             | 148            |
|            | 12  | 4gt13_92   | 5              | 30            | 26         | rd53_311  | 13             | 124            |
| large-sized| 13  | aj-e11_165 | 5              | 69            | 27         | qft_16    | 16             | 240            |
|            | 14  | alu-v2_31  | 5              | 198           | 28         | cnt3-5_180 | 16             | 215            |

Post-mapping circuit depth. The quantum program’s post-mapping circuit depth [30] is used to evaluate the policy’s capability for reducing the decoherence error.

Trial Reduction Factor (TRF). The TRF [12] is used to evaluate the improvement of the throughput brought by multi-programming policies. The TRF is defined as the ratio of trials needed when programs are executed separately to the trials needed when multi-programming is enabled.

5.1.2 Evaluation Platforms. Due to regional access restrictions of the IBM Qiskit runtime service [1], paid access to quantum chips with up to 27 qubits is only available in some regions. Our access is limited to quantum chips with a maximum of 7 physical qubits. Thus, we can evaluate QuCloud+ on IBM Nairobi [1], a recent quantum chip open to the public with 7 physical qubits. Additionally, we simulate IBMQ Toronto [1] (27 qubits), IBMQ50 [24, 31] (50 qubits) and a 3D quantum chip (Figure 4, 27 qubits) using Qiskit’s QASM simulator [2] for evaluating QuCloud+ on quantum chips with a larger scale. The real IBM Nairobi and simulated IBMQ Toronto are involved in single-programming and multi-programming experiments in fidelity evaluations, respectively. IBMQ50 is used for evaluating SWAP overheads for multi-programming workloads. We also demonstrate that QuCloud+ can save SWAPs on the 3D quantum chip. In all evaluations, the calibration data for the devices or simulators were consistent across the experiments. We employ the approach in [4] to simulate the conditional errors attributed to crosstalk. The simulator uses realistic qubit topology and calibration data obtained from IBM for IBMQ Toronto [1], and randomly generates the crosstalk amplification ratios. For IBMQ50 and the 3D quantum chip, the calibration data are generated using a uniform random model.

5.1.3 Benchmarks. We employ the quantum programs (in Table 2) used in previous studies [11, 28, 51, 55]. The tiny/small-sized programs have around five qubits and tens of CNOT gates; the large-sized ones have about ten qubits and hundreds of CNOT gates. These quantum programs include implementation of quantum gates (IDs 3-5), Bernstein-Vazirani algorithm (IDs 1, 2, 21), quantum Fourier transform (IDs 23, 27), encoding functions (IDs 11, 19), arithmetic functions (IDs 8, 9, 12, 15, 26), etc. They can be found in Revlib [51] and QASMBench [11].

5.1.4 Competing approaches. Baseline. The multi-programming baseline [12] devises algorithms for mapping concurrent quantum programs, aiming to reduce the impact of multi-programming on fidelity.

SABRE. SABRE [28] is a heuristic approach that reduces SWAP overheads without considering noise. In multi-programming scenarios, we merge multiple concurrent circuits into a single circuit and map them using SABRE.
Table 3. PST (in %) and SWAP Overheads for Single-Programming Workloads on IBM Nairobi

| ID  | SABRE | Baseline | QuCloud | CDAP-only | X-SWAP-only | QuCloud+ | NASSC (opt.) | QuCloud+ (opt.) |
|-----|-------|----------|---------|-----------|-------------|----------|--------------|----------------|
|     | PST   | addi     | PST     | addi      | PST         | addi     | PST          | addi           | PST                  | addi     |
| 1   | 80.04 | 3        | 82.78   | 3         | 89.39       | 3        | 90.14        | 3              | 86.68                | 3        |
| 2   | 67.68 | 3        | 61.07   | 3         | 79.28       | 0        | 77.65        | 0              | 60.67                | 3        |
| 3   | 86.15 | 3        | 81.64   | 6         | 81.70       | 12       | 88.82        | 3              | 86.60                | 3        |
| 4   | 82.56 | 3        | 85.46   | 3         | 89.72       | 3        | 79.79        | 3              | 89.49                | 3        |
| 5   | 87.07 | 3        | 82.53   | 3         | 89.00       | 3        | 85.69        | 3              | 84.80                | 3        |
| 6   | 53.63 | 6        | 66.72   | 18        | 77.03       | 6        | 78.08        | 6              | 73.67                | 9        |
| 7   | 47.78 | 21       | 41.39   | 21        | 52.73       | 21       | 42.60        | 21             | 49.74                | 18       |
| 8   | 37.29 | 18       | 31.74   | 24        | 47.57       | 21       | 35.52        | 18             | 33.44                | 12       |
| 9   | 49.07 | 27       | 61.24   | 30        | 52.14       | 30       | 61.40        | 27             | 53.45                | 21       |
| 10  | 22.67 | 27       | 33.17   | 33        | 32.75       | 33       | 29.57        | 27             | 30.55                | 24       |
| 11  | 13.62 | 36       | 20.40   | 39        | 17.82       | 33       | 40.53        | 33             | 25.52                | 33       |
| 12  | 8.75  | 45       | 18.88   | 51        | 23.78       | 54       | 34.67        | 51             | 14.45                | 42       |
| avg | 53.03 | 16.25    | 55.59   | 19.50     | 61.06       | 18.25    | 62.87        | 16.25          | 56.61                | 14.50    |

W/o post-mapping gate optimizations, QuCloud+ achieves 11.59%, 9.03%, and 3.56% higher PST compared with SABRE, Baseline, and QuCloud, on average, respectively. With post-mapping gate optimizations, QuCloud+ (opt) achieves 8.15% higher PST compared with NASSC (opt), on average.

**NASSC.** NASSC [30] reduces the number of gates via post-mapping gate optimizations, such as gate cancellation [32] or commutation [22]. We use the original version of NASSC without noise awareness.

**QuCloud.** QuCloud is designed for mapping concurrent quantum programs [31]. Crosstalk and the profiling results of the quantum programs (Section 4.1) are not considered in QuCloud.

We show the breakdown of our approach, i.e., CDAP-only and X-SWAP-only, separately. We also show the effectiveness of our approach that enables both CDAP and X-SWAP at the same time. CDAP-only employs the same mapping transition approach in SABRE. The X-SWAP-only strategy uses the identical initial mapping strategy as SABRE.

### 5.2 Experimental Results

#### 5.2.1 Evaluations on Fidelity

For fidelity evaluation, we employ tiny-sized and small-sized quantum programs in Table 2. We conduct single-programming experiments on the real IBM Nairobi and multi-programming experiments on the simulated IBMQ Toronto, as detailed in Section 5.1.2.

**Single-programming experiments on real IBM Nairobi.** Table 3 shows the experimental results for single-programming cases. The PST and addi columns denote the PST and the number of additional CNOT gates (i.e., the difference between the mapped circuit and the original circuit after inserting SWAPs), respectively. The post-mapping gate optimizations are not involved in prior studies [12, 28, 46, 47]. We do not use post-mapping gate optimizations for all competing approaches except for NASSC [30]. This is because NASSC is the only work considering the post-mapping gate optimizations during the mapping process, reducing the number of gates in the optimized quantum circuits. We first evaluate QuCloud+, QuCloud, SABRE, Baseline, CDAP-only, and X-SWAP-only without post-mapping gate optimizations. The average PST values of QuCloud+, QuCloud, SABRE, Baseline, CDAP-only and X-SWAP-only are 64.62%, 61.06%, 53.03%, 55.59%, 62.87%, and 56.61%, respectively. The fidelity of QuCloud+ outperforms SABRE, Baseline, and QuCloud by 11.59%, 9.03%, and 3.56%, on average, respectively. QuCloud+ achieves the highest fidelity, primarily due to its consideration of topology and error rates simultaneously during the initial mapping process. QuCloud+ also saves additional CNOT gates compared with SABRE, Baseline, and QuCloud, primarily due to the enhanced look-ahead ability from X-SWAP.
We also compare QuCloud+ with NASSC, which uses post-mapping gate optimizations. The post-mapping gate optimizations detect subcircuits composed of consecutively single-/two-qubit gates and replace them with equivalent subcircuits that use fewer gates. We also make QuCloud+ have post-mapping gate optimizations. These two approaches are denoted as QuCloud+ (opt) and NASSC (opt) in Table 3. The average PST values of QuCloud+ (opt) and NASSC (opt) are 66.48% and 58.33%, respectively. The average number of additional CNOT gates of QuCloud+ (opt) and NASSC (opt) are 11.75 and 7.58, respectively. Although NASSC (opt) effectively reduces gates by leveraging post-mapping gate optimizations, it exhibits a lower average PST value than QuCloud+ (opt). For instance, when NASSC (opt) maps the quantum program with ID 9, i.e., mod5mils_65, even NASSC (opt) saves 8 additional CNOT gates and the fidelity drops by 12.52% compared with QuCloud+ (opt). This is because NASSC (opt) employs the same initial mapping generation approach as SABRE, which is unaware of noise. As a result, an initial mapping with high error rates is generated, leading to fidelity degradation. By contrast, our approach often has better initial mapping results.

**Multi-programming Experiments on Simulated IBMQ Toronto.** We evaluate PST for two-programmed workloads performed on simulated IBMQ Toronto (27 qubits) in Table 4. The two quantum programs in a workload have similar depths to avoid potential decoherence errors. In experiments without post-mapping gate optimizations, the average PST values of QuCloud+, QuCloud, SABRE, Baseline, CDAP-only, and X-SWAP-only are 52.94%, 52.45%, 42.38%, 46.10%, 51.32%, and 43.19%, respectively. For fidelity, QuCloud+ outperforms SABRE, Baseline, and QuCloud by 10.56%, 6.84%, and 0.49%, on average, respectively. With post-mapping gate optimizations, QuCloud+ (opt) achieves 7.56% higher PST compared with NASSC (opt), on average.

The benefit of our approach mainly comes from CDAP. CDAP improves the fidelity by providing a high-quality initial mapping with reliable qubits and links. For example, as shown in Table 4, in the combination of quantum programs with IDs 1, 1 (bv_n3 and bv_n3), CDAP-only improves the fidelity significantly by 15.74% over the Baseline by providing a better initial mapping. The underlying reasons behind the advantage include the following. (1) Gates performed on a reliable set of physical qubits have lower error rates. (2) A better initial mapping reduces additional gates required during the mapping transition. (3) QuCloud+ can achieve fairness for multi-programming workloads by allocating robust resources to error-prone quantum programs. This is achieved by giving priority to the qubit partitioning of quantum programs with higher CNOT density (see Table 4. PST (in %) and SWAP Overheads for Multi-programming Workloads on IBMQ Toronto

| ID | QuCloud+ | SABRE | Baseline | Opt-Baseline | CDAP-only | X-SWAP-only | NASSC (opt) | QuCloud+ (opt) | NASSC (opt) | PST1 | PST2 | PST2 | PST1 | PST2 | PST2 | PST1 | PST2 | PST2 | PST1 | PST2 | PST2 | PST1 | PST2 | PST2 |
|----|----------|-------|----------|-------------|-----------|-------------|-------------|---------------|-------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 1  | 35.79    | 67.24 | 32.41    | 68.27       | 69.51    | 69.64       | 29.78        | 38.47          | 30.34        | 79.85| 67.47| 58.94| 63.55| 59.47| 60.05| 89.38| 89.38| 65.97| 68.75| 52.99| 54.46| 1     | 3     |
| 2  | 33.83    | 66.35 | 33.05    | 67.43       | 68.38    | 68.10       | 29.62        | 38.53          | 30.14        | 78.74| 67.46| 59.45| 63.00| 58.90| 60.22| 89.14| 89.18| 66.37| 69.17| 53.01| 54.62| 1     | 3     |
| 3  | 31.75    | 65.42 | 31.96    | 66.35       | 67.25    | 67.28       | 29.22        | 38.23          | 30.87        | 78.41| 66.35| 59.45| 62.88| 58.79| 60.14| 88.90| 88.54| 65.97| 69.17| 53.01| 54.62| 1     | 3     |
| 4  | 30.61    | 64.52 | 31.45    | 65.30       | 66.05    | 66.08       | 28.82        | 38.00          | 30.50        | 77.98| 65.96| 59.45| 62.88| 58.79| 60.14| 88.50| 88.15| 64.37| 68.93| 53.01| 54.62| 1     | 3     |
| 5  | 32.17    | 66.67 | 31.96    | 66.35       | 67.25    | 67.28       | 29.22        | 38.23          | 30.87        | 78.41| 66.35| 59.45| 62.88| 58.79| 60.14| 88.90| 88.54| 65.97| 69.17| 53.01| 54.62| 1     | 3     |
| 6  | 31.75    | 65.42 | 31.96    | 66.35       | 67.25    | 67.28       | 29.22        | 38.23          | 30.87        | 78.41| 66.35| 59.45| 62.88| 58.79| 60.14| 88.90| 88.54| 65.97| 69.17| 53.01| 54.62| 1     | 3     |
| 7  | 30.61    | 64.52 | 31.45    | 65.30       | 66.05    | 66.08       | 28.82        | 38.00          | 30.50        | 77.98| 65.96| 59.45| 62.88| 58.79| 60.14| 88.50| 88.15| 64.37| 68.93| 53.01| 54.62| 1     | 3     |
| 8  | 32.17    | 66.67 | 31.96    | 66.35       | 67.25    | 67.28       | 29.22        | 38.23          | 30.87        | 78.41| 66.35| 59.45| 62.88| 58.79| 60.14| 88.90| 88.54| 65.97| 69.17| 53.01| 54.62| 1     | 3     |
| 9  | 31.75    | 65.42 | 31.96    | 66.35       | 67.25    | 67.28       | 29.22        | 38.23          | 30.87        | 78.41| 66.35| 59.45| 62.88| 58.79| 60.14| 88.90| 88.54| 65.97| 69.17| 53.01| 54.62| 1     | 3     |
| 10 | 30.61    | 64.52 | 31.45    | 65.30       | 66.05    | 66.08       | 28.82        | 38.00          | 30.50        | 77.98| 65.96| 59.45| 62.88| 58.79| 60.14| 88.50| 88.15| 64.37| 68.93| 53.01| 54.62| 1     | 3     |

W/o post-mapping gate optimizations, QuCloud+ achieves 10.56%, 6.84%, and 0.49% higher PST compared with SABRE, Baseline, and QuCloud, on average, respectively. With post-mapping gate optimizations, QuCloud+ (opt) achieves 7.56% higher PST compared with NASSC (opt), on average.
For experiments without post-mapping gate optimizations, QuCloud+ reduces the number of additional CNOT gates by 37.81%, 40.92%, and 31.94% compared with SABRE, Baseline, and QuCloud, on average, respectively. For experiments with post-mapping gate optimizations, QuCloud+ (opt) achieves a similar number of additional gate operations as NASSC (opt) while reducing the circuit depth by 21.92%.

Section 4.2.2, as higher CNOT density indicates increased susceptibility to CNOT errors. CDAP proves to be a valuable technique in enhancing fidelity in various scenarios. CDAP-only improves fidelity by 7.28% for single-programming experiments on the real IBM Nairobi and by 5.22% for multi-programming experiments on simulated IBMQ Toronto compared with the Baseline.

X-SWAP-only exhibits only 3.58% and 0.81% higher fidelity than SABRE in single-programming cases (Table 3) and multi-programming cases (Table 4), respectively. X-SWAP-only doesn’t exhibit significant advantages on fidelity improvement for the following reasons. (1) Few SWAPs are needed when mapping tiny/small-sized quantum programs; the number of gates saved by X-SWAP is small. Thus, the advantages of X-SWAP are not reflected in fidelity. (2) Inter-program SWAPs are unlikely to be enabled when the allocation of the quantum programs are not adjacent. However, X-SWAP performs better in an enlarged SWAP search space when larger-sized quantum programs are co-located on a quantum chip with more qubits.

Both CDAP and X-SWAP in QuCloud+ are crosstalk aware. CDAP considers crosstalk when constructing the hierarchy tree (see Section 4.2.1) and partitioning qubits (see Section 4.2.2). During the mapping transition, X-SWAP mitigates crosstalk by selecting the SWAP with the lowest crosstalk error among SWAPs with the minimum heuristic cost. Generally, compared with QuCloud, QuCloud+ improves PST by 3.56% on the real quantum device IBM Nairobi and improves PST by 0.49% on the simulated IBMQ Toronto.

### Table 5. SWAP Overhead Comparison of 4-Programmed Workloads on IBMQ50

| Original circuits | SABRE | Baseline | QuCloud | CDAP-only | X-SWAP-only | QuCloud+ | NASSC (opt) | QuCloud+ (opt) |
|-------------------|-------|----------|---------|-----------|-------------|----------|------------|--------------|
|                   | depth | addi | t     | depth | addi | t     | depth | addi | t     | depth | addi | t     | depth | addi | t     | depth | addi | t     | avg  |
| 13,14,15,16     | 825  | 760  | 477   | 804  | 648  | 415   | 701  | 424  | 319   | 794  | 376  | 210   | 781  | 396  | 219   | 673  |
| 17,18,19,20     | 677  | 630  | 315   | 654  | 604  | 275   | 578  | 480  | 379   | 563  | 426  | 278   | 552  | 396  | 278   | 523  |
| 15,16,22,23     | 720  | 674  | 350   | 698  | 639  | 310   | 591  | 522  | 341   | 587  | 502  | 307   | 602  | 445  | 292   | 542  |

For experiments without post-mapping gate optimizations, QuCloud+ reduces the number of additional CNOT gates by 37.81%, 40.92%, and 31.94% compared with SABRE, Baseline, and QuCloud, on average, respectively. For experiments with post-mapping gate optimizations, QuCloud+ (opt) achieves a similar number of additional gate operations as NASSC (opt) while reducing the circuit depth by 21.92%.

5.2.2 Evaluations on SWAP Overheads. We evaluate SWAP overheads using two metrics: (1) the depth of the post-mapping quantum circuit and (2) the difference between the number of CNOT gates before and after mapping, i.e., the number of additional CNOT gates. The SWAP operations need to be transformed into CNOT gates for execution. Therefore, the number of additional CNOT gates can be used to measure the SWAP overheads for different approaches.

We map 4-programmed workloads on IBMQ50. The workloads are randomly selected, aiming to cover as many orthogonal combinations as possible. Experimental results are in Table 5. The left CXs column shows the number of CNOT gates in the workload before mapping. The depth, addi, and t columns denote the post-mapping circuit depth, the number of additional CNOT gates, and the time consumed for mapping the circuits, respectively. We first evaluate the competing approaches without post-mapping gate optimizations. By inserting fewer SWAPs, QuCloud+ effectively saves 37.81%, 40.92%, and 31.94% additional CNOT gates and reduces 15.24%, 19.49%, and 33.96% post-mapping circuit depth compared with SABRE, Baseline, and QuCloud, on average, respectively. CDAP in QuCloud+ saves SWAP overheads in the following ways: (1) CDAP allocates a tightly inter-connected set of physical qubits to each quantum program (see...
Section 4.2.2), reducing the length of SWAP paths. (2) CDAP allocates program qubits with more interactive neighbors to high-degree physical qubits, reducing SWAP costs. In Table 5, for an example combination of quantum programs with IDs 18, 20, 21, and 23, CDAP-only leverages the high-degree qubits on IBMQ50 and generates a high-quality initial mapping for concurrent circuits. In this example, CDAP-only effectively reduces the number of additional CNOT gates by 42.53% compared with SABRE and by 23.49% compared with Baseline. In terms of X-SWAP in QuCloud+, it includes two features for reducing the SWAP overheads. (1) X-SWAP uses inter-program SWAPs, taking shortcuts to minimize the number of SWAPs used. (2) X-SWAP selects gates for the extended set based on the “involvement list”, enhancing the look-ahead ability of the heuristic cost function. Thus, X-SWAP in QuCloud+ effectively reduces the number of SWAPs required. In Table 5, the X-SWAP-only approach uses the same initial mapping generation technique as SABRE. By employing inter-program SWAPs and an enhanced heuristic function with improved lookahead ability, X-SWAP-only effectively reduces the number of SWAPs inserted during mapping transition, reducing 6.90% additional CNOT gates compared with SABRE.

We further evaluate QuCloud+ against NASSC with post-mapping gate optimizations enabled. We denote them as QuCloud+ (opt) and NASSC (opt) in Table 5. Compared with NASSC (opt), QuCloud+ uses a similar number of additional CNOT gates while achieving a reduction in circuit depth by 21.92%. Although NASSC (opt) can reduce the number of additional gates through post-mapping gate optimizations, it faces challenges in handling multi-programming scenarios. NASSC (opt) employs the reverse traversal initial mapping generation technique in SABRE. Thus, program qubits with interactions can be mapped far from each other due to the reverse traversal starting from a randomly generated initial mapping. For instance, in Table 5, in the combination of quantum programs with IDs 13, 14, 15, and 16, NASSC (opt) incurs 65.69% more additional gate operations compared with QuCloud+ (opt).

Additionally, QuCloud+ exhibits lower time overhead for mapping the quantum programs. Table 5 shows the time consumed to map quantum programs using QuCloud+ and SABRE, respectively. In QuCloud+, the hierarchy tree is constructed only once, as all experiments' calibration data is identical. Each time when mapping quantum programs using QuCloud+, the pre-built hierarchy tree stored as binary is read, saving time on quantum chip profiling. On average, QuCloud+ takes 1.91 s to map the 4-program workloads on IBMQ50, while SABRE takes 4.13 s. QuCloud+ is scalable. It reduces the SWAP overheads for 4-program workloads on IBMQ50 (Table 5). It also can be used on a larger quantum chip with more qubits for the following reasons. (1) The community detection approach in CDAP has proven effective for large networks. (2) X-SWAP reduces SWAP overheads when quantum programs are mapped adjacently. They both work well regardless of the scale of a specific quantum chip.

5.2.3 Evaluations on the 3D quantum chip. We evaluate large-sized single-programming cases on the 27-qubit 3D quantum chip (Figure 4). We show that QuCloud+ can save SWAPs on 3D quantum chip architectures. The results are in Table 6. Generally, QuCloud+ has the fewest additional gates and achieves the minimum post-mapping circuit depth. On average, it reduces the number of additional CNOT gates by 9.10%, 14.37%, and 11.48% compared with SABRE, Baseline, and QuCloud, respectively. The core reason behind these results is that QuCloud+ can effectively utilize the high-degree qubits in the 3D quantum chip by leveraging the profiling information to reduce SWAP overheads. CDAP in QuCloud+ can map program qubits with more interactive neighbors to high-degree physical qubits, enabling adjacent physical mappings for program qubits requiring CNOT gates and reducing the SWAP overheads. As for X-SWAP, only CNOTs that contribute to the look-ahead ability of the heuristic cost function are added to the extended set during the mapping transition, reducing the number of inserted SWAPs. With QuCloud+, it makes no
Table 6. SWAP Overheads Comparison for Single-Programming Workloads on the 3D Quantum Chip

| ID  | name               | CXs | SABRE depth | SABRE addi | Baseline depth | Baseline addi | QuCloud depth | QuCloud addi | QuCloud+ depth | QuCloud+ addi |
|-----|--------------------|-----|-------------|------------|----------------|---------------|---------------|--------------|---------------|---------------|
| 13  | aj-e11_165         | 69  | 205         | 99         | 226            | 117           | 211           | 90           | 202           | 93            |
| 14  | alu-v2_31          | 198 | 625         | 336        | 626            | 330           | 607           | 273          | 576           | 258          |
| 15  | 4gt4-v0_72         | 113 | 348         | 180        | 357            | 168           | 349           | 171          | 350           | 165          |
| 16  | sf_276             | 336 | 1068        | 528        | 1060           | 549           | 1044          | 474          | 1051          | 465          |
| 17  | alu-bdd_288        | 38  | 124         | 63         | 111            | 60            | 114           | 51           | 113           | 57           |
| 18  | ex2_227            | 275 | 885         | 453        | 873            | 435           | 844           | 417          | 842           | 387          |
| 19  | ham7_104           | 149 | 480         | 234        | 456            | 228           | 454           | 204          | 434           | 213          |
| 20  | C17_204            | 205 | 662         | 324        | 642            | 321           | 629           | 333          | 586           | 297          |
| 21  | bv_n10             | 9   | 26          | 9          | 33             | 21            | 29            | 12           | 22            | 18           |
| 22  | ising_model_10     | 90  | 93          | 0          | 137            | 45            | 145           | 51           | 125           | 15           |
| 23  | qft_10             | 90  | 162         | 66         | 169            | 81            | 152           | 129          | 184           | 66           |
| 24  | sys6-v0_111        | 98  | 217         | 153        | 215            | 189           | 210           | 183          | 211           | 153          |
| 25  | sym9_146           | 148 | 332         | 237        | 398            | 249           | 353           | 240          | 392           | 234          |
| 26  | rd53_311           | 124 | 319         | 219        | 356            | 237           | 327           | 231          | 339           | 204          |
| 27  | qft_16             | 240 | 342         | 207        | 389            | 237           | 328           | 327          | 381           | 210          |
| 28  | cnt3-5_180         | 215 | 522         | 354        | 544            | 408           | 577           | 369          | 516           | 312          |
| avg |                   |     | 400.63      | 216.38     | 412.00         | 229.69        | 398.31        | 222.19       | 395.25        | 196.69       |

QuCloud+ reduces the number of additional CNOT gates by 9.10%, 14.37%, and 11.48% compared with SABRE, Baseline, and QuCloud, on average, respectively.

Fig. 12. Performance of the QuCloud+ scheduler. PST and TRF stand for fidelity and throughput, respectively. Higher is better for both. The increase of \( \epsilon \) leads to higher throughput but may cause fidelity reduction.

...difference whether the core algorithm is used on 2D or 3D chips. It leverages high-degree qubits to reduce SWAP overheads. Both 3D quantum chips (e.g., Figure 4) and 2D quantum chips with 4-qubit all-to-all connections (e.g., IBMQ50 in Section 5.2.2) have high-degree qubits. IBMQ50 has even more high-degree qubits than the 3D quantum chip. For example, IBMQ50 has 24 qubits with a degree of 6, while the 3D quantum only has one qubit with a degree of 6.

5.2.4 Evaluations on the QuCloud+ Scheduler. We build three task lists to evaluate the QuCloud+ scheduler. Each list contains 100 quantum programs randomly selected from Table 2. We use the QuCloud+ scheduler to schedule the workloads with the estimated fidelity (EPST) violation threshold \( \epsilon \) ranging from 0.05 to 0.20. Then, the workloads are executed on the IBMQ Toronto QASM simulator. The PST and TRF averaged across the 3 task lists are shown in Figure 12. Figure 12 also shows the performance in separate execution cases and the randomly selected two-programmed combination cases.
Separate execution supports the best average PST of 55.25% with a TRF of 1 (no parallelism). Randomly selected two-programmed combination cases have an average PST of 45.23%, and the TRF is 2 (i.e., two programs in parallel all the time). By contrast, the QuCloud+ scheduler reaches the highest TRF of 2.94 when $\varepsilon$ is 0.15 (i.e., only multi-programming cases leading to less than 15% estimated fidelity reduction could be scheduled). In this case, the average fidelity of the workloads is 49.81%, which is only 5.44% worse than the separate cases but still 4.58% higher than the randomly selected two-programmed combination cases. The TRF is 2.94, indicating that the throughput is improved by 194% compared with separate execution cases. The experimental results show that the QuCloud+ scheduler provides better solutions to balance the quantum computers’ throughput and fidelity.

In addition, the QuCloud+ scheduler can avoid potential decoherence errors by executing the quantum programs separately if there are no similar-depth quantum programs in the task list. Suppose that there is a case in which the value of $\varepsilon$ is 0.15 and there are only 4mod5-v1_22 and alu-v2_31 in the task list (with circuit depths of 12 and 255, respectively). The QuCloud+ scheduler executes them separately instead of using multi-programming.

6 CONCLUSION
Quantum computers are attracting increasing attention. QuCloud+ presents a new qubit mapping policy for multi-programming quantum computing, improving quantum computers’ fidelity and resource utilization. Experimental results show that QuCloud+ outperforms the state-of-the-art multi-programming strategy. As multi-programming is becoming increasingly important in cloud environments, we hope that our studies will benefit future researchers.

ACKNOWLEDGMENTS
We thank reviewers AE and EIC for their invaluable comments. We thank Prof. Mingsheng Ying, Prof. Wei Zhao, and Academician Guojie Li for their valuable suggestions and attention. X. Dou is a student member in Sys-Inventor Lab.

REFERENCES
[1] IBM Quantum Experience. Accessed: April 2, 2020. https://quantum-computing.ibm.com/
[2] Qiskit: An open-source framework for quantum computing. Accessed: April 2, 2020. https://www.qiskit.org/
[3] IBM unveiled 433-qubit quantum chip, 1,121-qubit Condor processor planned for 2023. Accessed: March 2, 2023. https://www.technology.org/2022/11/10/ibm-unveiled-433-qubit-quantum-chip-1121-qubit-condor-processor-planned-for-2023/
[4] A. Ash-Saki, M. Alam, and S. Ghosh. 2020. Experimental characterization, modeling, and analysis of crosstalk in a quantum computer. In Quantum Computing.
[5] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter. 1995. Elementary gates for quantum computation. In Physical Review A.
[6] D. Bhattacharjee and A. Chattopadhyay. Depth-optimal quantum circuit placement for arbitrary topologies. In arXiv 1703.08540.
[7] D. Bhattacharjee, A. A. Saki, M. Alam, A. Chattopadhyay, and S. Ghosh. 2019. MUQUT: Multi-constraint quantum circuit mapping on NISQ computers. In IEEE/ACM International Conference on Computer-Aided Design (ICCAD’19).
[8] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd. 2017. Quantum machine learning. In Nature.
[9] H. Bombin. 2015. Gauge color codes: optimal transversal gates and gauge fixing in topological stabilizer codes. In New Journal of Physics.
[10] H. Corrigan-Gibbs, D. J. Wu, and D. Boneh. 2017. Quantum operating systems. In Hot Topics in Operating Systems (HotOS’17).
[11] QASMBench Benchmark Suite - A low-level OpenQASM benchmark suite for NISQ evaluation and simulation. Accessed: August 23, 2023. https://github.com/pml/QASMBench/tree/master
[12] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi. 2019. A case for multi-programming quantum computers. In IEEE/ACM International Symposium on Microarchitecture (Micro’19).

ACM Transactions on Architecture and Code Optimization, Vol. 21, No. 1, Article 9. Publication date: January 2024.
[41] G. Park, K. Zhang, K. Yu, and V. Korepin. 2022. Quantum multi-programming for Grover’s search. In Quantum Information Processing.

[42] J. Preskill. 2018. Quantum computing in the NISQ era and beyond. In Quantum.

[43] S. Resch, A. Gutierrez, J. S. Huh, S. Bharadwaj, Y. Eckert, G. Loh, M. Oskin, and S. Tannu. Accelerating variational quantum algorithms using circuit concurrency. In arXiv 2109.01714.

[44] C. Rigetti and M. Devoret. 2010. Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies. In Physical Review B.

[45] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout. 2020. Detecting crosstalk errors in quantum information processors. In Quantum.

[46] M. Siraichi, V. F. D. Santos, S. Collange, and F. M. Q. Pereira. 2018. Qubit allocation. In International Symposium on Code Generation and Optimization (CGO’18).

[47] B. Tan and J. Cong. 2020. Optimal layout synthesis for quantum computing. In IEEE/ACM International Conference On Computer Aided Design (ICCAD’20).

[48] B. Tan and J. Cong. 2020. Optimality study of existing quantum computing layout synthesis tools. In IEEE Transactions on Computers.

[49] B. Tan and J. Cong. 2021. Optimal qubit mapping with simultaneous gate absorption. In IEEE/ACM International Conference On Computer Aided Design (ICCAD’21).

[50] S. S. Tannu and M. K. Qureshi. 2019. Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers. In ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS’19).

[51] RevLib - An Online Resource for Reversible Benchmarks. Accessed: August 23, 2023. https://www.revlib.org/

[52] B. Wu, X. He, S. Yang, L. Shou, G. Tian, J. Zhang, and X. Sun. Optimization of CNOT circuits under topological constraints. In arXiv 1910.14478.

[53] W. Zhang, T. van Leent, K. Redeker, R. Garthoff, R. Schwonnek, F. Fertig, S. Eppelt, W. Rosenfeld, V. Scarani, C. C. Lim, and H. Weinfurter. 2022. A device-independent quantum key distribution system for distant users. In Nature.

[54] H.-S. Zhong et al.. 2020. Quantum computational advantage using photons. In Science.

[55] A. Zulehner, A. Paler, and R. Wille. 2018. Efficient mapping of quantum circuits to the IBM QX architectures. In IEEE Design, Automation & Test in Europe Conference & Exhibition (DATE’18).

[56] IBM Unveils Breakthrough 127-Qubit Quantum Processor. Accessed: August 23, 2023. https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor

Received 8 December 2022; revised 1 September 2023; accepted 17 September 2023