QuCloud: A New Qubit Mapping Mechanism for Multi-programming Quantum Computing in Cloud Environment

Lei Liu*, Xinglei Dou
Sys-Inventor Lab, SKLCA, ICT, CAS

***Abstract*—For a specific quantum chip, multi-programming improves overall throughput and resource utilization. Previous studies on mapping multiple programs often lead to resource under-utilization, high error rate, and low fidelity. This paper proposes QuCloud, a new approach for mapping quantum programs in the cloud environment. We have three new designs in QuCloud. (1) We leverage the community detection technique to partition physical qubits among concurrent quantum programs, avoiding the waste of robust resources. (2) We design X-SWAP scheme that enables inter-program SWAPs and prioritizes SWAPs associated with critical gates to reduce the SWAP overheads. (3) We propose a compilation task scheduler that schedules concurrent quantum programs to be compiled and executed based on estimated fidelity for the best practice. We evaluate our work on publicly available quantum computer IBMQ16 and a simulated quantum chip IBMQ50. Our work outperforms the state-of-the-art work for multi-programming on fidelity and compilation overheads by 9.7% and 11.6%, respectively.**

I. Introduction

Quantum computers have gradually entered our field of vision. Due to its potential in various critical applications, such as machine learning [5], [19], database search [11] and chemistry simulation [14], [25], many companies, universities and institutes are actively working to develop prototypes of quantum computer systems. In recent years, Google has released their quantum chip with 72 quantum bits (qubits) [15], which has more qubits than those of IBM (50 qubits) [16] and Intel (49 qubits) [12]. IBM announced its quantum chip with 53 qubits accessible via the cloud [10]. The latest photonic quantum computer Jiuzhang [33] generates up to 76 output photon clicks. However, modern quantum chips belong to the Noisy Intermediate-Scale Quantum (NISQ) category [26] – the qubits and the links between them are with variational reliability and are easily disturbed; therefore, quantum computers are susceptible to errors. Though quantum computers can be made fault-tolerant leveraging quantum error correction (QEC) codes, such codes are too expensive (20 to 100 physical qubits to form a fault-tolerant logical qubit) to be implemented on NISQ computers [9].

The emergence of quantum cloud services enables users to easily access quantum computers, but it also brings new challenges. As NISQ computers exhibit low fidelity, the small-sized programs using a few qubits can have high-quality results. NISQ computers tend to have a lower resource utilization as a result. With the rapid development of quantum computers, hundreds of qubits would be on a specific quantum chip. Still, qubits' reliability is challenging to be further improved. Moreover, as more and more people would like to use quantum computers, the growing access to quantum computing cloud service prolongs the service time. Therefore, it is necessary to increase the resource utilization and throughput of quantum computers. Multi-programming can be an effective way of doing this. Although mapping multiple quantum programs onto a specific quantum chip improves the throughput, the activity of a program can adversely affect the reliability of co-located programs because of (i) a limited number of qubits with high fidelity, (ii) cross-talk noise caused by simultaneously executed quantum gates [22] and (iii) long SWAP paths. Previous study [7] on multi-programming shows that running multiple quantum programs on a specific quantum chip incurs a 12.0% reduction on fidelity, on average.

In this paper, we propose solutions to improve the throughput and utilization of NISQ machines in cloud environment while reducing the negative impacts on multi-programming NISQ computers' reliability. We find the previous qubit mapping policies have several shortcomings when handling multi-programming cases. (1) The existing mapping policies often divide a large area of robust on-chip qubits into many small-scale segments that other programs cannot map onto. In many cases, over 20% of the robust qubits are wasted during the initial mapping. (2) When a specific quantum chip is partitioned for mapping multiple quantum programs, post-compilation SWAP operations for each quantum program can increase, leading to an unpredictable impact for fidelity and reliability. For instance, additional SWAPs can be involved when two quantum programs with tens of CNOT gates are compiled together. (3) Scheduling concurrent quantum applications for multi-programming on a specific quantum chip can be a challenging job, which may lead to fidelity degradation and qubit resource under-utilization in the cloud environment.

To this end, we design QuCloud, a new qubit mapping mechanism for multi-programming in the cloud. QuCloud has several key features. (1) It partitions the physical qubits for concurrent quantum programs leveraging community detection technique [23], avoiding the waste caused by the topology-unaware algorithms. It also provides a better initial-mapping, which reduces the SWAP overheads. (2) QuCloud is the first work that clearly enables the inter-program SWAPs to solve the qubit mapping problem in multi-programming cases that reduces the overall SWAP overheads. The experimental results show that QuCloud outperforms the latest solution [7] by 9.7% on fidelity and 11.6% on compilation overheads. (3) QuCloud has a scheduler for scheduling compilation tasks for the best practice of multi-programming, avoiding the performance degradation caused by randomly selected workloads. The scheduler improves the throughput by 42.9% and enhances the fidelity by 5.0% over random workloads chosen in our experiments. In essence, we think that QuCloud contains the prototype of the OS for quantum computers – QuOS.

*Corresponding author (PI): lei.liu@zoho.com; liulei2010@ict.ac.cn.

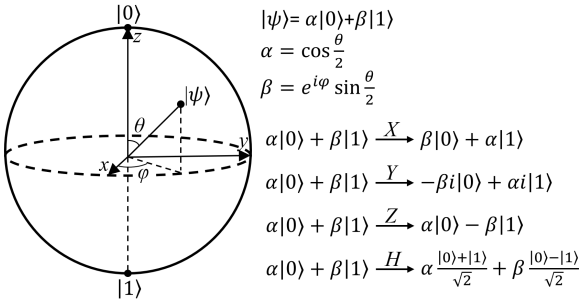

Figure 1. Bloch sphere representation of a qubit.

To sum up, we make the following contributions in QuCloud. (1) We observe that the previous qubit mapping policies cannot make full utilization of resources when multiple quantum programs are mapped on a specific quantum chip simultaneously. (2) We reveal that, rather than the separate execution cases, the number of SWAP operations increases significantly when multiple quantum programs are running together. (3) We propose a new qubit mapping approach (i.e., CDAP), which provides better mapping solutions than the latest work for co-located quantum programs. Furthermore, we design X-SWAP, the first inter-program SWAP mechanism, significantly reducing the SWAP overheads. (4) We propose a compilation task scheduler that provides optimal quantum program combinations for multi-programming, improving the fidelity and resource utilization on quantum chips.

II. BACKGROUND

In this section, we introduce some concepts about quantum computing and quantum computers. We also present the relevant background of quantum cloud services to illustrate why multi-programming is necessary.

A. Quantum mechanism

Quantum computing can solve conventionally hard problems leveraging quantum mechanism [28]. The foundation of quantum computing lies on qubits. As shown in Figure 1, the Bloch sphere is used to represent the state of a single qubit. The north and south poles of the sphere represent the $|0\rangle$ and $|1\rangle$ states of the qubit, respectively, corresponding to the 0 and 1 in classical bits. The state of a qubit can be the linear combinations of $|0\rangle$ and $|1\rangle$ as $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where α, β are complex numbers with $|\alpha|^2 + |\beta|^2 = 1$. The state of a qubit can be manipulated with single-qubit gates, e.g., *H*, *X*, *Z*, etc. When the qubit is measured, the state of the qubit collapses to either $|0\rangle$ with the probability of $|\alpha|^2$, or $|1\rangle$ with the probability of $|\beta|^2$. Two or more qubits can be entangled with two-qubit gates, i.e., 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: $|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle$. A quantum gate involving three or more qubits can be decomposed with single- and two-qubit gates [4].

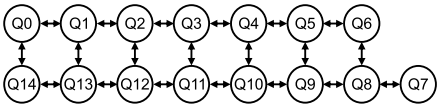

Figure 2. Architecture of IBM Q16 Melbourne.

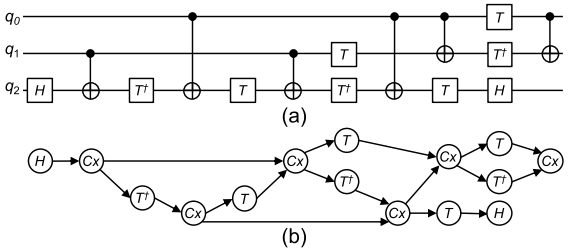

Figure 3. The Quantum circuit and DAG of decomposed Toffoli gate.

B. Quantum computers

There are several competing technologies for the implementation of quantum computers, e.g., ion-trap [8], superconducting quantum circuits [17] and photonic quantum devices [33]. IBM quantum processors are superconducting quantum chips with Josephson-junction-based transmon qubits [17] and microwave-tunable two-qubit gates [27]. Unlike the all-to-all connectivity of qubits on ion-trap quantum computers, the physical qubits on IBM quantum chips only have connections to neighboring qubits. For example, Figure 2 shows the architecture of IBM Q16 Melbourne (denoted as IBMQ16 hereafter).

C. Errors on quantum computers

NISQ computers face reliability challenges. As the physical qubits are fragile and susceptible to interference, the following kinds of errors may occur in quantum programs running on quantum computers. (i) Coherence errors caused by short qubit state retention time. (ii) Operational errors caused by error-prone quantum gates. (iii) Readout errors caused by measurement operations. In practice, the error rate and the coherence time are all reported in IBM backends calibration data [1]. The error rates are variational for different qubits, links, and days. Variation-aware mapping policies tend to map quantum programs onto the region with the lowest error rate.

D. Quantum programs and Quantum circuits

Quantum programs can be converted into quantum circuits composed of a series of quantum gates. For example, Figure 3-(a) shows the decomposed Toffoli gate's quantum circuit, where each horizontal line represents a logical qubit, each block and vertical line represents a single- and two-qubit gate, respectively. Figure 3-(b) shows the Directed Acyclic Graph (DAG) of the circuit. The depth of the circuit equals 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 logical qubits involved are mapped physically adjacent. A SWAP operation changes the qubit mapping by exchanging the states between two qubits [18], [20]. Generally, it takes two steps for a quantum program compiler to solve the mapping problem. (1) Initial mapping generation. The compiler maps each program's logical qubits onto physical qubits. (2) Mapping transition. The compiler meets all two-qubit constraints by inserting SWAPs to the quantum circuit so that every two-qubit gate in the quantum program can be executed physically.

E. Quantum cloud services

It takes a considerable cost to maintain a quantum computer. The vendors would like to provide easy-to-use interfaces

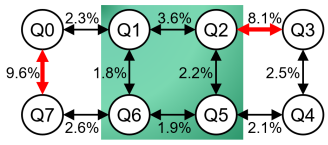

Figure 4. A case for qubit mapping/allocation. The value on the edge represents the CNOT error rate of the link. Edges highlighted in red indicate links with high CNOT error rate. Physical qubits Q_1, Q_2, Q_5, and Q_6 are mapped.

for end-users to access quantum computers through cloud services conveniently. Take IBM Q Experience [1] as an example, the cloud service provides multiple back-ends, and each of them connects a specific quantum computer. Quantum programs can be compiled and executed on the target quantum computer using qiskit framework [2]. However, due to the scarcity of quantum computing resources and the growing need for accessing quantum computers, contentions for accessing quantum computers increases. Take the back-end, IBMQ Vigo, as an example; we observe an average of more than 120 queued jobs per day. The throughput, utilization, and reliability of quantum computers need to be improved to provide high-quality services.

III. Motivation

Quantum computers in the NISQ era are susceptible to errors, and 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 of the qubits and links on NISQ computers; not all of the qubits have similar reliability, and not all of the links between qubits are with the same error rate. To map a specific quantum program, previous noise-aware mapping techniques for a single quantum program [21], [24], [30] usually employ greedy or heuristic approaches to discover the mapping policies that have the most reliable qubits and links. For example, in Figure 4, Q_1, Q_2, Q_5, and Q_6 are mapped as they have higher reliability, and the links between them have lower error rate.

With the rapid development of quantum computing, multi-programming is introduced, which improves the qubits utilization and the overall throughput of an expensive quantum computer. For the quantum computing service providers, multi-programming becomes increasingly useful as qubit counts grow and the global quantum experimentation and quantum computing demands increase. Multi-programming on quantum computers is as essential as it on classical computers; however, it brings new challenges for mapping qubits.

Although it is possible to combine multiple quantum programs into one quantum circuit and then compile it using the compilers dedicated to single quantum programs, the following problems exist. (1) No fairness among co-located quantum programs is guaranteed. Reliable resources on a specific quantum chip are limited. It is not easy to maintain fairness and the overall reliability for programs with different characteristics. (2) 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; when the resources are sufficient, other programs cannot be launched concurrently, causing resource under-utilization.

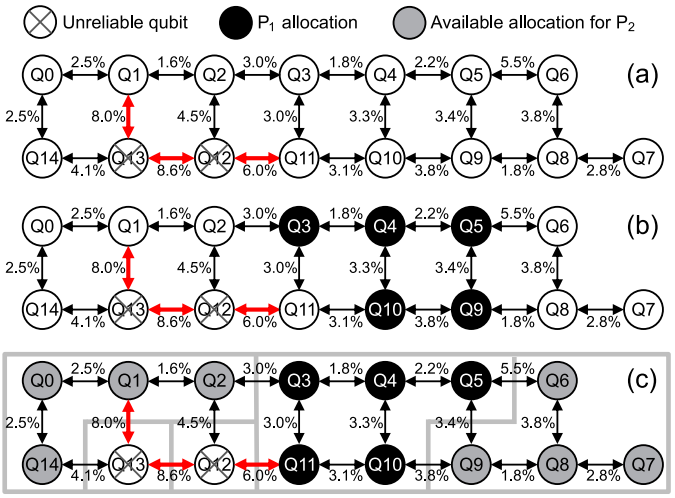

Figure 5. (a) Coupling map of IBMQ16, the calibration data is collected on 12.27.2019. Links with high error rate of CNOT are highlighted with red color. Two quantum programs (i.e., P_1 and P_2) need to be mapped on the chip. P_1 has 5 qubits and P_2 has 4 qubits. (b) Allocation with FRP approach proposed in [7]. (c) Better qubit mapping solutions for P_1 and P_2, providing higher qubit utilization.

(3) New optimization opportunities for multi-programming are missed. For example, the compilation overheads can be reduced by the multi-programming mapping policies and inter-program SWAPs. To sum up, it is necessary to design a new qubit mapping strategy for Multi-programming Quantum Computers.

How to design an ideal scheme for mapping multiple quantum programs on a specific chip simultaneously? – The state-of-the-art multi-programming technique [7] supports to co-locate two quantum programs. In practice, when more than two quantum programs are running concurrently, it needs to be enhanced to schedule them and provide a lower error rate and high reliability. Some previous studies propose solutions for mapping single quantum program [18], [30]. They mainly rely on heuristic policies or greedy algorithms. Nevertheless, mapping multiple programs is quite different from the scheme that maps a single program. We summarize the challenges of multi-programming and introduce our insights as follows.

A. *Resource under-utilization – wasting qubits*

Existing mapping policies face the challenge of resource under-utilization. The available qubits are subject to be divided into smaller scale segments. Some of them support mapping a quantum program, but some cannot due to the high error rate and weak links. Hence, some programs might have weak qubits when mapping multiple quantum programs, leading to unreliable results and increasing error rate for concurrent quantum programs. The latest study [7] proposes how to map multiple quantum programs onto a specific quantum chip. Figure 5 shows an example on how to map quantum programs P_1 (5 qubits) and P_2 (4 qubits) on IBMQ16. The unreliable qubits and weak links are highlighted in Figure 5-(a). Figure 5-(b) illustrates the results of using the mapping approach FRP in [7]. For P_1, FRP tries to have the most reliable qubits and reliable links started from a reliable root that has enough neighbors with high *utility*, which is defined as the ratio of the number of links belong to a specific qubit and the sum of

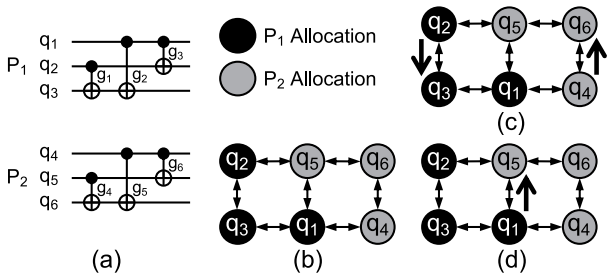

Figure 6. A case for an inter-program SWAP.

the error rate of CNOT operations of the qubit. As a result, started from Q_4, FRP involve Q_3 – Q_5, Q_9, and Q_{10} for initial mapping of P_1. However, after the allocation for P_1, none of the qubits in Q_0 – Q_2, Q_{11} – Q_{14} have enough neighbors with high *utility*; FRP cannot find another reliable root for mapping the other quantum program. Q_6 – Q_8 are not enough to map a program that has four qubits. Thus, some reliable qubits and links are wasted, e.g., Q_0 and Q_8. In fact, there exists better solutions for mapping P_1 and P_2. Figure 5-(c) illustrates a better solution, in which P_1 and P_2 are both mapped on qubits with reliable links. Apparently, the existing qubit mapping scheme incurs qubits under-utilization, therefore wasting resources.

B. More SWAP operations – high SWAP overheads

SWAP exchanges the physical mapping between two logical qubits. We can enable more than one SWAP operation to move a logical qubit to an arbitrary physical qubit location. In the cases where two qubits are not nearby, we can move them together using SWAPs and then execute the two-qubit gate in the circuit [7], [18]. When multiple quantum programs are running concurrently, the number of SWAP operations involved for a specific application can be higher than that in the separate execution cases. The previous multi-programming approach [7] compiles quantum programs one by one. If a program occupies any qubits on the shortest SWAP path for all co-located programs, the SWAP process has to suffer higher overheads, i.e., involving more SWAP operations across more qubits. More SWAP operations bring higher error rate [30]; therefore, the overall fidelity is negatively affected.

Figure 6 shows a case where two quantum programs are mapped on a specific quantum chip. Figure 6-(a) shows the circuits of 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 involved. However, the g_3 cannot be executed directly unless a SWAP operation between q_2 and q_3 is executed first. 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, for such a mapping case where two programs are involved, two SWAPs are needed. Figure 6-(c) shows the SWAP operations.

By contrast, if the two programs could be compiled together, and global information could be considered, the inter-program SWAP operation could be enabled. Figure 6-(d) shows a new policy that enables inter-program SWAPs. Illustrated in (d), only one inter-program SWAP $\{q_1, q_5\}$ is needed, and thus all of the quantum gates can be executed without other overheads. Each SWAP comprises three CNOT gates with relatively higher error rates [7], [18], [30]. Therefore, more SWAPs would incur unreliability; fewer SWAPs can benefit the overall performance for a specific quantum computer and computing results.

C. Multi-programming workloads can be harmful

Quantum programs often exhibit differently on quantum circuit depth, number of qubits, number of CNOTs, etc. There are mutual interferences among programs for a multi-programming workload containing more than one quantum program, negatively affecting the workloads' overall fidelity. For instance, let's suppose that we have a new workload consists of P_1 and P_2 that will be co-located on a quantum chip. The circuit depth of P_1 and P_2 are 40 and 170, respectively. The co-location prolongs P_1's execution time, as the measurement operations for qubits in P_1 cannot be conducted until all gates in P_2 have been executed; otherwise, the measurement operations would interfere with the state of other qubits and cause fidelity degradation [7], [13]. As a result, severe coherence error and fidelity degradation happen for P_1 due to a long time passed [7], [30]. In brief, a randomly generated workload may bring unpredictable performance degradation. Therefore, an ideal task scheduler for multi-programming workloads on quantum chips is needed in cloud environments.

D. Related work

Recent studies towards mapping quantum programs onto quantum chips mainly rely on heuristic search schemes. IBM's compilation framework [2] implements noisy adaptive mapping [21] and Stochastic SWAP. The work in [32] generates mapping transition by inserting SWAP layers into adjacent data dependency layers. The study in [30] enhances the design in [32] by introducing noise variation awareness. SABRE [18] brings exponential speedup in the search complexity by reducing the search space. QURE [3] finds reliable partitions by searching isomorphic sub-graphs. These approaches focus on mapping problems for only one quantum program. However, new policies are expected to map multiple quantum programs and ensure fairness and reliability at the same time. For multi-programming, the effort in [7] proposes the FRP algorithm to assign reliable regions for each quantum program. It enhances SABRE [18] with noise awareness to generate the mapping transition. This design can be further improved.

In this paper, we devise a new policy to generate a better initial mapping for multi-programming quantum workloads and develop a new SWAP approach, i.e., inter-program SWAP. Our work makes better use of the qubit resources and significantly reduces the compilation overheads.

E. Thinking about a New OS for the Quantum Computer

It is about the right time to have a new OS – *QuOS* – for the quantum computer. Our work is among the first step studies 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 not compatible with the quantum architecture. Therefore,

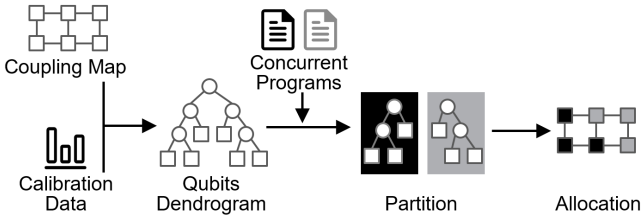

Figure 7. Qubit mapping by using CDAP algorithm in a nutshell.

OS's design principles and implementation strategies for quantum computers should be different from traditional ones. QuCloud project, including QuOS, will continue to explore OS and other run-time stack technologies for quantum computing.

IV. The Art of Our Design

A. 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 can also make full use of the robust qubits and links on the quantum chips. In terms of the concurrent quantum programs, an excellent initial mapping reduces the SWAP cost during mapping, reduces the interference between multiple concurrent programs, and improves the overall fidelity.

For the quantum chip and multi-programming, there are following observations. (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 2, Q_1 has links to the three adjacent physical qubits, while Q_7 has a link to only one qubit. (3) The qubits needed for a single program should be closely allocated; the allocations for qubits belonging to different quantum programs should avoid mutual interference, fairly leveraging robust resources.

In this paper, we propose a new technology – Community Detection Assistant Partitioning (CDAP) – to construct a hierarchy tree consisted of qubits for searching the robust qubits that are tightly connected for initial allocation. Figure 7 illustrates how CDAP works. CDAP creates a hierarchy tree according to the coupling map and calibration data obtained from IBMQ API [1]. In the hierarchy tree's dendrogram, a leaf node denotes a specific physical qubit; a circle node represents the union of its sub-nodes. CDAP then iterates the hierarchy tree from bottom to top to find available regions for initial allocation. Finally, the quantum circuits are allocated by greedy policy to corresponding regions. We show the details as below.

1) Hierarchy tree construction: The hierarchy tree is a profile of a quantum chip, which helps to locate reliable qubit resources on the quantum computer. Algorithm 1 constructs the hierarchy tree based on FN community detection algorithm [23]. The algorithm clusters the physical qubits on a specific quantum chip into communities. Qubits in a community have reliable and close interconnections. By contrast, the links between communities have relatively low reliability.

When the algorithm starts, each physical qubit is a community and is a leaf node in the hierarchy tree. The algorithm keeps merging two communities that can maximize the reward function F until there is only one community containing all qubits. Each community during the process corresponds to a node in the hierarchy tree and is a candidate region for allocating qubits. The reward function F is defined as the benefit of merging two communities –

$$F = Q_{\text{merged}} - Q_{\text{origin}} + \omega EV, \tag{1}$$

in which Q is the modularity of a partition (i.e., $Q = \sum_i (e_{ii} - {a_i}^2)$ [23], 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 more proper partition. Q_{origin} and Q_{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., one minus the error rate of the operation) of CNOTs on the between-group edges of the two communities, and V denotes the average reliability of readout operations on the qubits of the two communities. The reward function F makes CDAP to take both physical topology and error rates into account when performing partitioning. ω is a weight parameter. For a specific quantum chip, we can change the value of ω for adjusting the weight of physical topology and the error rate. If $\omega = 0$, CDAP conducts partitioning completely according to physical topology. Noise-awareness is introduced as ω increases. If ω keeps increasing, the weight of the error rate will exceed the weight of the physical topology, resulting in the degradation of CDAP to a greedy algorithm that is mainly based on error rate. The mapping results of programs with fewer qubits are more sensitive to ω. Because the variation of ω changes the error-rate awareness in CDAP, obviously changing the qubit merging order. By contrast, the program with more qubits is less sensitive. More details on how the value of ω is selected are discussed in IV-A3.

The hierarchy tree has several features: (1) Every node in the hierarchy tree is a candidate region 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 will be. Whether the hierarchy tree is balanced or not doesn't impact the qubit mapping result, as the most reliable region 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.

Algorithm 1: Hierarchy tree construction.

Input: Quantum chip coupling graph w/ calibration data
Output: Hierarchy tree (HT)
Initialize HT by setting it empty;
Add a leaf node to HT for each qubit on chip;
while *not all items in HT are merged for a larger community* **do**
    Take two items A and B that are not merged and are with the max value of $F(A, B)$ in HT;
    Create a New_Node by setting A and B as the New_Node's left subtree and right subtree, respectively;
    Append New_Node to HT;
end
Return HT.

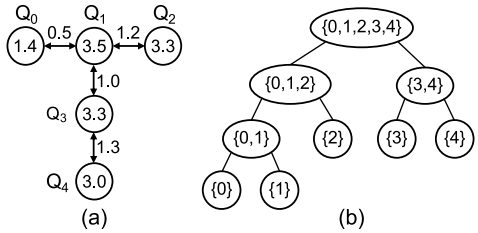

Figure 8. (a) Architecture of IBM Q London. The value in a node represents the readout error rate (in %) of the qubit, and the value on a link means the error rate (in %) of the CNOT operation. (b) The dendrogram generated by Algorithm 1. The values in nodes are the index of the physical qubits.

Further, we explain why the hierarchy tree helps to select the initial allocation with an example in Figure 8. (i) Q_0 and Q_1 are firstly merged due to the link between them is with 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, as illustrated in Figure 8-(b). The algorithm avoids wasting robust resources caused by the topology-unaware greedy algorithm and supports more quantum programs to be mapped on a specific quantum chip.

As the calibration data doesn't change frequently (e.g., IBM calibrates the devices once a day [1]), the hierarchy tree only needs to be constructed once in each calibration cycle. It can be saved for possible reuse within 24 hours, without incurring more compilation overheads.

2) Partition and allocation: Algorithm 2 partitions the qubits into multiple regions for concurrent quantum programs according to the hierarchy tree. The algorithm prioritizes the quantum programs with a higher value of the *CNOT density*, which is defined as: (the number of CNOT instructions) / (the number of qubits in the quantum program). For each quantum program, the algorithm searches the hierarchy tree from bottom to top to find available candidate set of physical qubits. Then, the candidate with the lowest average error rate is assigned to the program for initial allocation. Finally, all of the quantum programs are allocated to regions using Greatest Weighted Edge First strategy [21]. The policy maps two logical qubits with the highest weighted edge (i.e., CNOTs are invoked most frequently between them) to the most robust link on hardware. It helps to generate an initial mapping with high reliability and low compilation overheads.

3) Discussion: In Equation (1), E stands for the reliability of links between qubits, and V denotes the reliability of readout operations on qubits. Our design merges the reliable qubits with robust links and the lower read-out error rate 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 initial allocation. Unreliable qubits are less likely to be selected, thereby improving the overall fidelity.

Using CDAP might lead to a case where the allocated qubits for a quantum program exceed the qubits that the program needs. For example, if a 4-qubit quantum program is mapped on the quantum chip in Figure 8-(a), the only available region (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 and add them to adjacent communities.

Algorithm 2: Qubit partitioning.

Input: Hierarchy tree, Quantum programs
Output: Partition

Sort quantum programs in descending order of *CNOT density*;
for *each quantum program* **do**
  Initialize candidate nodes (C) as an empty set;
  for *each leaf node in the hierarchy tree* **do**
    Search a node that has enough number of on-chip qubits for allocating the program from the leaf to its parent nodes;
    Add the node to C;
  end
  if *C is empty* **then** # The current qubit state cannot meet the requirements
    Fail and revert to separate execution;
  end
  Find the node in C w/ the highest average fidelity;
  Add the node to Partition;
  Remove qubits in the node from all other nodes in the hierarchy tree;
  if *the node's sibling node is isolated* **then** # The sibling node has no path to other nodes in the hierarchy tree
    Remove qubits in the sibling node from its parent nodes;
    Remove the link from the sibling node to its parent;
  end
end
Return Partition.

For a specific node in the hierarchy tree, we define the term maximum redundant qubits, 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 ω 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. The

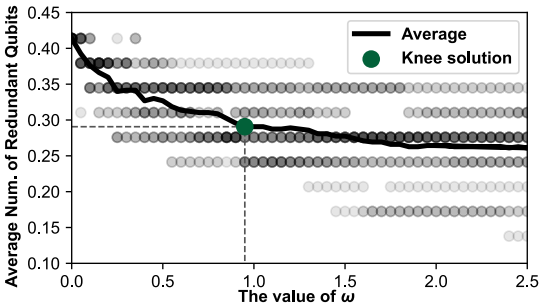

Figure 9. The average number of redundant qubits in the hierarchy tree varies with ω. A gray dot with darker color represents more cases are overlapping in this result. The knee solution refers to the value of ω, which makes the change of redundant qubits slow down with the increase of ω. We use the knee solution because it can reduce the redundant qubits as much as possible, without making the community partitioning depend too much on error rate.

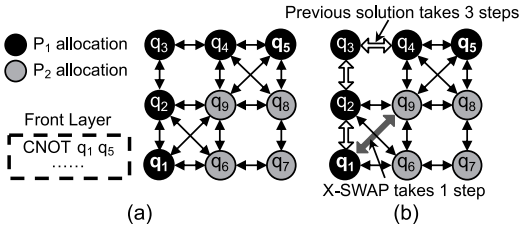

Figure 10. (a) P_1 and P_2 are mapped on a quantum chip with 9 qubits. The next gate to be solved is CNOT that involves q_1 and q_5. (b) X-SWAP scheme takes shortcuts to satisfy the constraint of CNOT q_1, q_5.

number of maximum redundant qubits of the new community is 0 in this case. Thus, the increase of ω leads to a reduction in average redundant qubits. We collected the calibration data of IBMQ16 for 21 days. The ω varies from 0 to 2.5 every day, and the average number of the redundant qubits in the hierarchy tree is recorded, as illustrated in Figure 9. We take the knee solution, in which the value of ω is 0.95. In this case, CDAP takes both physical topology and the error rate into account, and the average redundant quantum number of the hierarchy tree is 0.29. The same experiment is conducted on IBMQ50, and ω is 0.40. Briefly, the number of redundant (unused) qubits is quite low in reality. Moreover, CDAP can also handle mapping cases where just a single program is involved. CDAP maps the program to the most reliable qubit region.

B. The design of inter-program SWAP

Multi-programming brings new challenges for mapping transition. In this paper, we design the X-SWAP, which includes both inter- and intra-program SWAP operations. It provides the best SWAP solution by considering the SWAP possibilities across all of the qubits. In practice, the inter-program SWAP can be enabled when two quantum programs are close to each other. The cost of inter-program SWAPs can be less than the cost in the cases where only intra-program SWAPs are used. The below section shows the details.

1) The advantages of inter-program SWAP: In our study, we find there are two main advantages. (1) An inter-program SWAP can replace two or more intra-program SWAPs. As demonstrated in Figure 6-(c) and (d), the intra-program SWAPs, i.e., $\{q_2, q_3\}$, $\{q_4, q_6\}$, can be replaced by one inter-program SWAP across q_1 and q_5. Obviously, using the inter-program SWAP achieves the same goal but has lower overheads. (2) Inter-program SWAPs take shortcuts. For instance, Figure 10-(a) shows two quantum programs are co-located (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 10-(b), an inter-program SWAP i.e., $\{q_1, q_9\}$, takes only one step (swap operation) to move q_1 and q_5 adjacent. By contrast, to achieve the same goal, 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 where multiple quantum programs are mapped as neighbors on a specific quantum chip, therefore reducing the SWAP overheads and benefiting the overall fidelity.

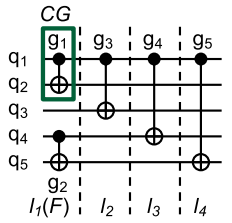

Figure 11. Critical gates (CG).

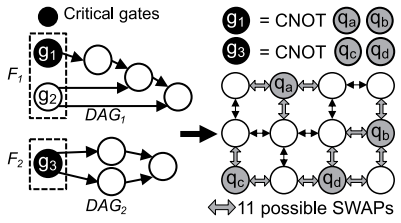

Figure 12. Example of the SWAP candidates.

2) The design of X-SWAP: Instead of generating a schedule for each quantum program separately and then merge them, which is done by the previous work, we are the first to design an approach for generating the global scheduling solution for all of the programs simultaneously. In our work, the SWAP-based heuristic search scheme in previous work [18] is used as the baseline. The design details are shown as follows.

Heuristic search space. To show our design, we illustrate a circuit for a quantum program P in Figure 11. The CNOT gates of the circuit can be divided into 4 layers (i.e., l_1-l_4). As 1-qubit gates can be performed directly without SWAPs, only CNOTs are considered here. Gates in a layer can be executed in parallel. l_1 is the Front layer (denoted as F) of P, which denotes the set of all gates without unexecuted predecessors in the Directed Acyclic Graph (*DAG*) of P. The *DAG* shows P's data dependency. Critical Gates (*CG*) denotes the set of CNOT gates in F that have successors on the second layer (l_2). For example, in F, g_1 has a successor g_3 on l_2, 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 g_3 is resolved and the front layer F will be updated. By contrast, handling g_2 firstly doesn't help to update F.

We reserve the program context (i.e., qubit mapping, *DAG*, F, CG, etc.) for each quantum program in the context list. For each program P_i, we remove hardware-compliant gates that can be executed directly in F_i from *DAG*$_i$. When there are no hardware-compliant gates, SWAPs are needed to make hardware-incompliant gates executable. Among all of the hardware-incompliant gates, the data dependency of critical gates need to be handled firstly for the purpose of updating the Front Layer and reducing the post-compilation circuit depth. Thus, we only search the SWAPs associated with qubits in critical CNOT gates. To help understand, Figure 12 shows other examples, in which g_1 and g_3 are critical gates illustrated in DAGs. The qubits involved in g_1 and g_3 are not mapped as the close neighbors. All SWAPs associated with the critical gates are SWAP candidates. They are highlighted on the coupling map of the quantum chip. The best SWAP among candidates is selected according to the heuristic cost function (details refer to the following). The mapping is updated as the SWAP is inserted into the circuit. Some hardware-incompliant CNOTs become executable when their constraints are eliminated by the SWAPs. The procedure repeats until the constraints of all CNOTs in the DAG are satisfied.

Design of the heuristic cost function. The heuristic cost function helps to get the best SWAP from all SWAP candidates (including inter/intra-program SWAPs). We show its core idea.

The concept *locality* for mapping a quantum program is critical. It indicates that 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 required for a CNOT operation. Nearest Neighbor Cost (NNC) is the length of the shortest path between two logical qubits mapped on a quantum chip. NNC-based heuristic function is used in SABRE [18] to choose the best SWAP among the SWAP candidates. We also use the NNC-based heuristic function *H* as a component in our approach. *H* represents the sum of the cost in the front layers and the cost in the extended sets [18]. Each set's cost is calculated as the averaged NNC of all CNOT gates in the set.

We prioritize inter-program SWAPs on the shortest SWAP path. Given the coupling map of a quantum chip and the qubit allocations, we define the term distance matrix D, in which each cell denotes the length of the shortest path between two physical qubits on the quantum chip. For each program P_i, we define $D_i^{'}$ as the shortest path matrix for qubits that have not been occupied by other programs. i.e., unmapped physical qubits and the physical qubits on which P_i is mapped. In essence, D represents the shortest path matrix to perform mapping transition for concurrent quantum programs with inter-program SWAPs enabled; $D_i^{'}$ represents the shortest path matrix to perform mapping transition for P_i. For a two-qubit gate g, we denote the two logical qubits involved in g as $g.q_1$ and $g.q_2$. We define the physical qubit on which a logical qubit q is mapped as $\sigma(q)$. 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 $D_i^{'}[\sigma(g.q_1)][\sigma(g.q_2)]$ is greater than $D[\sigma(g.q_1)][\sigma(g.q_2)]$ for a two-qubit gate g in a specific quantum program P_i, it means that inter-program SWAPs outperforms 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 10-(b), as it takes either 1 inter-program SWAP or 3 intra-program SWAPs to satisfy the constraint of CNOT q_1, q_5, it delivers $D[\sigma(q_1)][\sigma(q_5)] = 2$ and $D_1^{'}[\sigma(q_1)][\sigma(q_5)] = 4$. In terms of inserting SWAPs to satisfy g's constraint, we define the number of SWAPs saved by X-SWAP scheme as:

$$gain(g) = D[\sigma(g.q_1)][\sigma(g.q_2)] - D_i^{'}[\sigma(g.q_1)][\sigma(g.q_2)], \quad (2)$$

and, we define the heuristic cost function as:

$$score(\textit{SWAP}) = H(\textit{SWAP}) + \sum_{F_i \in F} \frac{1}{|F_i|} \sum_{g \in F_i} gain(g) I(\textit{SWAP}, g). \quad (3)$$

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 both logical qubits involved in the *SWAP* is on the shortest SWAP path, $I(\textit{SWAP}, g) = 1$. Otherwise, $I(\textit{SWAP}, g) = 0$. This indicates only the SWAPs on the shortest SWAP path are prioritized. The SWAP with the minimum value of $score$ is the best among the candidates. Algorithm 3 shows the overall logic of X-SWAP.

Algorithm 3: X-SWAP mechanism.

```
Input: Quantum chip coupling graph, Quantum programs,
       Initial mapping
Output: Final Schedule (FS)
Generate a DAG for each program;
Generate a Front Layer for each DAG;
while not all gates' constraints are satisfied do
    if hardware-compliant gates exist then
        Append hardware-compliant gates to FS;
        Remove hardware-compliant gates from the DAG and
        update the Front Layer;
    else
        for each Front Layer do
            Append CNOTs in the Front Layer that have
            subsequent CNOTs on the second layer to
            Critical Gates;
        end
        Add SWAPs associated with the qubits in Critical
        Gates to SWAP candidates;
        Find a SWAP from SWAP candidates with the
        minimum value of score(SWAP);
        Append the SWAP to FS and update mapping;
    end
end
Return FS.
```

C. The design of the compilation task scheduler

Although there are some mapping mechanisms for concurrent quantum programs, selecting appropriate quantum programs to form a combination for the multi-programming workload is still a challenging job. An ineffective approach often brings problems. (1) Qubits are often under-utilized. (2) The program combinations formed by randomly selected programs may lead to a significant reduction in fidelity. (3) A complicated verification mechanism must be introduced to ensure fidelity, bringing additional system modification overheads. To this end, we design a scheduler that selects appropriate concurrent quantum programs for multi-programming.

Our design selects optimal quantum program combinations, maximizing quantum chips' fidelity and resource utilization. For each task in the scheduling queue, the scheduler checks whether other quantum programs in the queue can be co-located on the quantum chip with the current task without incurring unacceptable fidelity reduction. If so, they are mapped simultaneously for multi-programming. Otherwise, the task will be executed separately.

The Estimated Probability of a Successful Trial (EPST) is proposed to estimate the fidelity of the execution of a quantum program on a specific quantum chip. Separate EPST is the maximum EPST that a program can achieve. To obtain the separate EPST, Algorithm 2 is called for every specific quantum program to allocate a set of physical qubits for it. Co-located EPST represents the EPST when multiple quantum programs are co-located on a quantum chip. Algorithm 2 is involved to generate a partition for all of the programs. The EPST of a quantum program on the set of allocated physical qubits is

Algorithm 4: Compilation task scheduling.

```
Input: List of incoming jobs (IJ), Hierarchy tree
while not all jobs in IJ are scheduled do
    Initialize cur_job_set as a list having the first item in IJ;
    Initialize idx as 1;
    while idx<IJ's length and idx<N and cur_job_set's
      length<MAX_COLOCATE_NUM do
        Append IJ[idx] to cur_job_set;
        for each job in cur_job_set do
            Calculate job's sep_EPST;
            Calculate job's co_EPST;
            Calculate job's EPST_violation as 1-(co_EPST/
            sep_EPST);
            if EPST_violation > ε then
                Remove IJ[idx] from cur_job_set;
                Break;
            end
        end
        Set idx as idx+1;
    end
    Algorithm 3 is called to compile programs in cur_job_set;
    cur_job_set is submitted to execute;
    Remove all programs in cur_job_set from IJ.
end
```

defined as bellow:

$$EPST = r_{2q}^{|CNOTS|} r_{1q}^{|1q\ gates|} r_{ro}^{|qubits|}, \quad (4)$$

in which r_{2q}, r_{1q} and r_{ro} denotes 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. $|CNOTS|$, $|1q\ gates|$ and $|qubits|$ denotes the number of CNOT gates, the number of single-qubit gates, and the number of qubits of the quantum program, respectively. For example, when a program containing 3 qubits with 2 CNOTs and 1 single-qubit gate is allocated to a region with $r_{2q} = 0.9$, $r_{1q} = 0.95$ and $r_{ro} = 0.85$, it delivers $EPST = 0.9^2 * 0.95 * 0.85^3$. A higher EPST indicates the quantum program is mapped to a region with more robust resources and also indicates a higher PST may be obtained during the real execution in practice.

For a specific quantum program combination, our approach generates the EPST violation according to the separate EPST and the co-located EPST. If EPST violation is less than the threshold ε, these quantum programs can be co-located on the chip. The scheduler supports to co-locate more than two programs on a quantum computer. To ensure our approach's efficiency and the scheduling's fairness, we only search the first N tasks (10 by default in practice) in the queue. More details are in Algorithm 4.

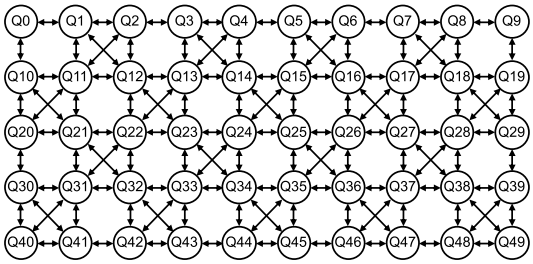

Figure 13. Architecture of IBMQ50.

Table I NISQ BENCHMARKS.

Type	Benchmarks
tiny-sized	bv_n3, bv_n4, peres_3, toffoli_3, fredkin_3
small-sized	3_17_13, decod24-v2_43, 4mod5-v1_22, mod5mils_65, alu-v0_27
large-sized	aj-e11_165, 4gt4-v0_72, alu-bdd_288, ex2_227, ham7_104, bv_n10, ising_model_10, qft_10, sys6-v0_111, rd53_311, qft_16, alu-v2_31, C17_204, cnt3-5_180, sf_276, sym9_146

V. EVALUATION

A. Methodology

1) Metrics: The following metrics are used for evaluations.

Probability of a Successful Trial (PST). PST is used to evaluate the fidelity of the quantum program execution [7], [29], [30]. PST is defined as the fraction of trails that produce a correct result. To get PST, we compile and run each workload on the target quantum chip for 8024 trials.

Number of post-compilation gates. We use the number of post-compilation CNOT gates to evaluate the compiler's ability to reduce the compilation overheads when compiling multiple quantum programs.

Post-compilation circuit depth. The compiled quantum program's circuit depth is used to evaluate the compiler's capability for reducing the coherence error.

Trial Reduction Factor (TRF). TRF is used to evaluate the improvement of the throughput brought by multi-programming policies [7]. TRF is defined as the ratio of needed trails when programs are executed separately to the trails needed when multi-programming is enabled.

2) Quantum chips: We evaluate our work on IBMQ16 [1] and IBMQ50 [16]. Their architectures are illustrated in Figure 2 and Figure 13, respectively. IBMQ16 is publicly available; IBMQ50 is not. We simulate IBMQ50 for evaluations. The simulated chip consists of: (1) topology information, and (2) calibration data. The topology of the IBMQ50 is provided by IBM. We generate the value of each attribute in the calibration data within the range of its maximum and minimum value on IBMQ16 using a uniform random model.

3) Benchmarks: We employ the benchmarks (in Table I) used in previous studies – SABRE [18], QSAM-Bench [6], RevLib [31] and examples in [32]. 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. For today's quantum chips (e.g., IBMQ16/50), using these programs can be sufficient to validate our work.

4) Baseline: **Separate execution.** It compiles and executes each program in a workload separately using the algorithm with the highest optimization level in qiskit [2]. They are the most reliable cases without interference caused by multi-programming.

Multi-programming baseline. It uses the policy proposed in [7], which generates initial mapping for concurrent quantum programs with FRP strategy and generates mapping transition with the enhanced noise-aware SABRE strategy.

SABRE. Multiple programs are merged into one quantum circuit and compiled using SABRE [18]. SABRE is a noise-

Table II PST COMPARISON BETWEEN MULTIPLE STRATEGIES ON IBMQ16.

Workloads		Separate			SABRE			Baseline			CDAP+X-SWAP			CDAP-only			X-SWAP-only		
W1	**W2**	**PST1**	**PST2**	**avg**	**PST1**	**PST2**	**avg**	**PST1**	**PST2**	**avg**	**PST1**	**PST2**	**avg**	**PST1**	**PST2**	**avg**	**PST1**	**PST2**	**avg**
bv_n3	bv_n3	83.4	84.0	83.7	50.8	75.4	63.1	57.5	61.4	59.5	69.7	66.2	68.0	68.7	67.1	67.9	19.3	54.9	37.1
bv_n3	bv_n4	83.4	63.6	73.5	52.6	34.5	43.5	35.9	25.0	30.4	52.5	17.7	35.1	27.2	12.8	20.0	37.4	53.7	45.5
bv_n3	peres_3	83.9	81.2	82.5	56.3	51.2	53.8	49.3	61.0	55.2	63.9	71.1	67.5	65.1	70.8	67.9	67.9	76.8	72.4
bv_n3	toffoli_3	83.1	82.0	82.5	59.0	49.5	54.3	65.7	41.9	53.8	65.3	81.7	73.5	63.8	70.5	67.1	48.5	76.5	62.5
bv_n3	fredkin_3	83.2	46.2	64.7	40.9	56.1	48.5	69.7	39.1	54.4	66.1	81.9	74.0	64.9	82.1	73.5	46.4	64.1	55.2
avg		77.4			52.6			50.7			63.6			59.3			54.5		
3_17_13	3_17_13	43.1	44.6	43.8	12.9	10.8	11.9	14.0	11.6	12.8	33.3	11.3	22.3	7.0	22.2	14.6	23.0	15.6	19.3
3_17_13	4mod5-v1_22	45.0	28.3	36.7	10.4	18.7	14.5	12.0	21.3	16.7	33.5	17.3	25.4	13.7	29.6	21.7	16.8	25.4	21.1
3_17_13	mod5mils_65	22.4	29.2	25.8	9.3	3.6	6.5	18.5	19.1	18.8	32.0	16.9	24.5	13.7	7.6	10.7	13.2	16.7	15.0
3_17_13	alu-v0_27	44.0	14.1	29.0	9.0	7.3	8.1	14.0	7.4	10.7	18.3	15.2	16.8	20.9	21.9	21.4	9.3	6.3	7.8
3_17_13	decod24-v2_43	43.6	6.2	24.9	18.2	7.8	13.0	11.5	9.3	10.4	14.7	11.2	13.0	27.2	6.6	16.9	7.6	19.1	13.4
avg		32.1			10.8			13.9			20.4			17.0			15.3		

Table III COMPILATION OVERHEADS COMPARISON OF 4-PROGRAM WORKLOADS ON IBMQ50.

Mixes	Benchmarks in the workload	SABRE		Baseline		CDAP+X-SWAP		CDAP-only		X-SWAP-only	
		CNOTs	**depth**	**CNOTs**	**depth**	**CNOTs**	**depth**	**CNOTs**	**depth**	**CNOTs**	**depth**
Mix_1	aj-e11_165, alu-v2_31, 4gt4-v0_72, sf_276	1386	768	1303	847	1167	586	1141	530	1177	629
Mix_2	alu-bdd_288, ex2_227, ham7_104, C17_204	1222	596	1266	717	1236	671	1225	581	1242	717
Mix_3	bv_n10, ising_model_10, qft_10, sys6-v0_111	387	189	382	178	351	160	419	236	341	224
Mix_4	aj-e11_165, alu-v2_31, ising_model_10, cnt3-5_180	979	450	1008	437	953	480	1005	450	950	496
Mix_5	4gt4-v0_72, sf_276, sym9_146, rd53_311	1365	672	1429	759	1255	536	1237	497	1089	450
Mix_6	alu-bdd_288, ex2_227, qft_10, sys6-v0_111	871	711	862	655	780	563	862	578	809	579
Mix_7	ham7_104, C17_204, bv_n10, ising_model_10	713	370	963	496	668	413	723	423	677	399
Mix_8	aj-e11_165, 4gt4-v0_72, rd53_311, cnt3-5_180	996	448	1052	537	916	441	894	368	940	390
Mix_9	alu-v2_31, sf_276, sym9_146, qft_16	1481	738	1499	769	1478	718	1545	779	1325	482
Mix_10	alu-bdd_288, ham7_104, ising_model_10, sys6-v0_111	659	392	662	315	622	248	656	306	617	325
Mix_11	ex2_227, C17_204, bv_n10, qft_10	955	537	1049	630	900	569	964	531	985	599
Mix_12	aj-e11_165, sf_276, C17_204, sys6-v0_111	1438	834	1337	828	984	506	1267	773	1063	467

unaware approach for reducing compilation overheads.

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 with SABRE. Moreover, the X-SWAP-only strategy employs the identical initial mapping strategy as SABRE.

B. Evaluation results

1) Evaluations on fidelity: We use tiny-sized and small-sized benchmarks for fidelity evaluation. We show PST of two-program workloads executed on IBMQ16 in Table II. The experiments are conducted within a calibration cycle of IBMQ16, indicating the calibration data are the same. The combination of two programs can double the throughput of the quantum computer. However, multi-programming may reduce reliability due to resource conflicts and cross-talk errors. In most cases, the PST of multi-programming quantum programs is lower than that in separate execution cases. Our approach incurs less fidelity reduction. The average PST of our approach (CDAP+X-SWAP), separate execution, SABRE, and multi-programming baseline for tiny-sized workloads are 63.6%, 77.4%, 52.6% and 50.7%, respectively. For small-sized workloads, they are 20.4%, 32.1%, 10.8% and 13.9%, respectively. The fidelity of our approach outperforms SABRE and the multi-programming baseline by 10.3% and 9.7%, on average, respectively. It indicates that our approach provides a more reliable result and incurs less fidelity reduction.

The benefit of our approach mainly comes from CDAP strategy. CDAP improves the fidelity by providing a better initial mapping, which makes the fidelity in multi-programming close to or even exceed that in separate execution cases. For example, as shown in Table II, in the benchmark combination of bv_n3 and fredkin_3, CDAP-only improves the fidelity significantly by 19.1% over the multi-programming baseline by providing a better initial mapping. The advantages of CDAP over multi-programming baseline originate from two aspects. (1) Gates performed on a reliable region have lower error rate. (2) A better initial allocation reduces mapping transition SWAP cost. On average, CDAP-only strategy reduces the adverse impact of multi-programming and outperforms baseline by 5.9% in fidelity.

X-SWAP-only strategy doesn't exhibit significant advantages on fidelity improvement due to following reasons. (1) Few SWAPs are needed to compile small-sized quantum programs, so X-SWAP scheme can hardly save SWAPs in these cases. (2) The initial mapping has a major impact on reliability for small-sized quantum programs on a quantum chip with simple architecture like IBMQ16. (3) The allocation of the quantum programs may not be adjacent, and inter-program SWAPs are unlikely to be performed. But, X-SWAP performs better on the chip with more qubits.

2) Evaluations on compilation overheads: X-SWAP performs better in an enlarged SWAP search space when larger-sized programs are co-located on a quantum chip with more qubits. We evaluate compilation overheads of 4-program workloads on IBMQ50 by comparing the number of post-compilation gates and circuit depth. The workloads are randomly selected aiming to cover as many orthogonal program combinations as possible. For each workload and each policy, we report the best result out of 5 attempts (similar with [18]).

The experimental results are shown in Table III.

For SABRE [18], using the reverse traversal technique and the heuristic search scheme, it tries to minimize the number of SWAPs inserted for compiling quantum programs. However, SABRE cannot achieve the optimal solution when compiling multi-programming workloads, as the locality is not involved. By contrast, multi-programming baseline takes the locality into account when partitioning qubits, but resource conflicts are introduced between co-located quantum programs, leading to redundant SWAPs involved. Therefore, the experimental results show that the number of post-compilation CNOT gates compiled with multi-programming baseline is 4.0% higher than that compiled with SABRE, on average. The circuit depth of the baseline exceeds SABRE by 6.8%, on average.

CDAP-only outperforms SABRE by 2.7% in the number of post-compilation CNOT gates and 6.8% in circuit depth, on average. The reduction of compilation overheads origins from a closely inter-connected initial mapping. The reduction is not that significant, and in some cases, e.g., Mix_9, CDAP-only shows higher compilation overheads than other techniques. The underlying reason is that CDAP mainly aims at obtaining the initial mapping with the highest fidelity and improving resource utilization, reducing the compilation overheads (i.e., compiling quantum programs with the minimum number of SWAPs) is not CDAP's primary task. X-SWAP-only employs the identical initial mapping strategy as SABRE. By enabling inter-program SWAPs and prioritizing critical gates, the X-SWAP-only effectively reduces the number of post-compilation gates by 8.8%. It also reduces the post-compilation circuit depth by 9.2%, on average. The reasons are multi-folds. (1) X-SWAP leverages the inter-program SWAPs, taking shortcuts to save compilation overheads. (2) By searching the SWAPs that are associated with critical gates, X-SWAP provides more efficient SWAPs to reduce the compilation overheads and circuit depth.

In our design, CDAP generates a reliable and closely inter-connected initial mapping; X-SWAP helps to reduce the compilation overheads. CDAP and X-SWAP work together to benefit the performance – reducing the number of post-compilation gates by 11.6% compared with baseline, and 8.6% compared with SABRE. The circuit depth is reduced by 16.0% and 10.3% compared with baseline and SABRE, respectively. More results can be found in Table III.

Moreover, our work exhibits scalability. It reduces the compilation overheads for 4-program workloads on IBMQ50. It also can be used on a larger quantum chip with more qubits, because – (1) The community detection approach in CDAP has been proved to be effective for large networks. (2) X-SWAP reduces compilation overheads when quantum programs are mapped adjacently. They both work well regardless of the scale of a specific quantum chip.

3) Evaluations on the task scheduler (i.e., QuCloud scheduler): We construct a task queue that includes the tiny-sized and small-sized quantum programs in Table I. We use the task scheduler to schedule the workloads with the estimated fidelity (EPST) violation threshold ε ranging from 0.05 to 0.2. The average Probability of a Successful Trial (PST) and Trial Reduction Factor (TRF) of the workloads are shown in Figure 14. Figure 14 also shows the performance in separate execution cases and the randomly selected two-programmed combination cases.

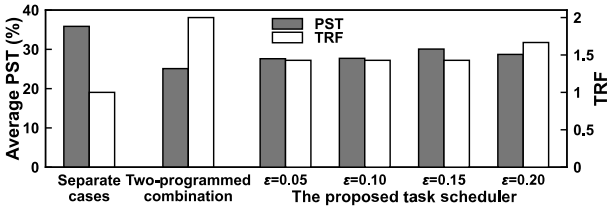

Figure 14. Performance of the task scheduler. PST and TRF stand for fidelity and throughput, respectively. Higher is better for both. The increase of ε leads to higher throughput, but may cause fidelity reduction.

Separate execution can support the best average PST of 35.9% with a TRF of 1 (no parallelism). Randomly selected two-programmed combination cases have the lowest average PST of 25.1%, but the TRF is 2 (i.e., 2 programs in parallel all the time). By contrast, our scheduler performs best when ε 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 30.1%, outperforming the randomly selected workloads by 5.0%. The TRF is 1.429, indicating the throughput is improved by 42.9% compared to separate execution cases. The experimental results show that our QuCloud scheduler can provide better solutions to balance the quantum computers' throughput and fidelity.

VI. Conclusion

Quantum computers attract more and more attention. With the trend of putting everything on the cloud, quantum computers face the problems of resource under-utilization, lower fidelity, higher error rates, etc. Our work presents a new qubit mapping policy for multi-programming cases, improving the fidelity and resource utilization when multiple quantum programs are running on a specific quantum chip. Our approach outperforms the state-of-the-art multi-programming strategy by improving the fidelity and reducing the SWAP overheads. As multi-programming is gaining importance in the cloud, we hope our efforts could help future researchers in the related field.

Acknowledgement

We thank the reviewers for their valuable comments. We thank Prof. Mingsheng Ying, Prof. Wei Zhao, and Academician Guojie Li for their invaluable suggestions and attention. This work is supported by NSFC under grant No. 62072432, 61502452. L. Liu thanks Yun He. X. Dou is a student member in Sys-Inventor Lab led by L. Liu.

References

[1] "IBM Quantum Experience," https://quantum-computing.ibm.com/, Accessed: April 2, 2020.

[2] H. Abraham, AduOffei, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, E. Arbel, A. Asfaw, C. Azaustre, AzizNgoueya, P. Barkoutsos, G. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, L. S. Bishop, S. Bolos, S. Bosch, S. Bravyi, D. Bucher, A. Burov, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, R. Chen, J. M. Chow, C. Claus, C. Clauss, R. Cocking, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, S. Dague, T. E. Dandachi, M. Dartiailh, DavideFrr, A. R. Davila, A. Dekusar,

D. Ding, J. Doi, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, A. H. Ferrera, FranckChevallier, A. Frisch, A. Fuhrer, M. GEORGE, J. Gacon, B. G. Gago, C. Gambella, J. M. Gambetta, A. Gammanpila, L. Garcia, S. Garion, A. Gilliam, J. Gomez-Mosquera, S. de la Puente González, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, M. Haglund, I. Haide, I. Hamamura, O. C. Hamido, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, JamesSeaward, A. Javadi, A. Javadi-Abhari, Jessica, M. Jivrajani, K. Johns, Jonathan-Shoemaker, T. Kachmann, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, S. King, Knabberjoe, Y. Kobayashi, A. Kovyrshin, R. Krishnakumar, V. Krishnan, K. Krsulich, G. Kus, R. LaRose, E. Lacal, R. Lambert, J. Latone, S. Lawrence, G. Li, D. Liu, P. Liu, Y. Maeng, A. Malyshev, J. Manela, J. Marecek, M. Marques, D. Maslov, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, D. McPherson, S. Meesala, T. Metcalfe, M. Mevissen, A. Mezzacapo, R. Midha, Z. Minev, A. Mitchell, N. Moll, M. D. Mooring, R. Morales, N. Moran, MrF, P. Murali, J. Müggenburg, D. Nadlinger, K. Nakanishi, G. Nannicini, P. Nation, E. Navarro, Y. Naveh, S. W. Neagle, P. Neuweiler, P. Niroula, H. Norlen, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, S. Oud, D. Padilha, H. Paik, S. Perriello, A. Phan, F. Piro, M. Pistoia, C. Piveteau, A. Pozas-iKerstjens, V. Prutyanov, D. Puzzuoli, J. Pérez, Quintiii, N. Ramagiri, A. Rao, R. Raymond, R. M.-C. Redondo, M. Reuter, J. Rice, D. M. Rodríguez, RohithKarur, M. Rossmannek, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, H. Sargsyan, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, J. Schwarm, I. F. Sertage, K. Setia, N. Shammah, Y. Shi, A. Silva, A. Simonetto, N. Singstock, Y. Siraichi, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, SooluThomas, Starfish, D. Steenken, M. Stypulkoski, S. Sun, K. J. Sung, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, M. Tomasik, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. C. Vazquez, V. Villar, D. Vogt-Lee, C. Vuillot, J. Weaver, R. Wieczorek, J. A. Wildstrom, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, S. Wood, J. Wootton, D. Yeralin, D. Yonge-Mallo, R. Young, J. Yu, C. Zachow, L. Zdanski, H. Zhang, C. Zoufal, Zoufalc, a matsuo, adekusar drl, bcamorrison, brandhsn, chlorophyll zz, dekel.meirom, dekool, dime10, drholmie, dtrenev, elfrocampeador, faisaldebouni, fanizzamarco, gadial, gruu, jagunther, jliu45, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, michelle4654, ordmoj, rmoyard, saswati qiskit, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, welien, willhbang, yang.luh, yotamvakninibm, and M. Čepulkovskis, "Qiskit: An open-source framework for quantum computing," 2019.

[3] A. Ash-Saki, M. Alam, and S. Ghosh, "QURE: Qubit re-allocation in noisy intermediate-scale quantum computers," in Proceedings of the 56th IEEE/ACM Design Automation Conference (DAC), 2019.

[4] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," in Physical review A, 1995.

[5] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," in Nature, 2017.

[6] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," in arXiv 1707.03429.

[7] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi, "A Case for Multi-Programming Quantum Computers," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (Micro), 2019.

[8] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, "Demonstration of a small programmable quantum computer with atomic qubits," in Nature, 2016.

[9] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, "Surface codes: Towards practical large-scale quantum computation," in Physical Review A, 2012.

[10] M. Giles, "IBM's new 53-qubit quantum computer is the most powerful machine you can use," https://www.technologyreview.com/f/614346/ibms-new-53-qubit-quantum-computer-is-the-most-powerful-machine-you-can-use/, Accessed: April 2, 2020.

[11] L. K. Grover, "A fast quantum mechanical algorithm for database search," in Proceedings of the 28th annual ACM symposium on Theory of computing (STOC), 1996.

[12] J. Hsu, "CES 2018: Intel's 49-qubit chip shoots for quantum supremacy," in IEEE Spectrum Tech Talk, 2018.

[13] C. John, F. K. Wilhelm, "Superconducting quantum bits," in Nature, 2008.

[14] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," in Nature, 2017.

[15] J. Kelly, "A Preview of Bristlecone, Google's New Quantum Processor," https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, Accessed: April 2, 2020.

[16] W. Knight, "IBM raises the bar with a 50-qubit quantum computer," in Sighted at MIT Review Technology, 2017.

[17] J. Koch, M. Y. Terri, J. Gambetta, A. A. Houck, D. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the Cooper pair box," in Physical Review A, 2007.

[18] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for NISQ-era quantum devices," in Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2019.

[19] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum algorithms for supervised and unsupervised machine learning," in arXiv 1307.0411.

[20] N. D. Mermin, "Quantum computer science: an introduction," Cambridge University Press, 2007.

[21] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2019.

[22] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," in Proceedings of the 25th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2020.

[23] M. E. Newman, "Fast algorithm for detecting community structure in networks," in Physical review E, 2004.

[24] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. Van Meter, "Extracting Success from IBM's 20-Qubit Machines Using Error-Aware Compilation," in ACM Journal on Emerging Technologies in Computing Systems, 2020.

[25] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," in Nature communications, 2014.

[26] J. Preskill, "Quantum Computing in the NISQ era and beyond," in Quantum, 2018.

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

[28] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," in SIAM review, 1999.

[29] S. S. Tannu and M. K. Qureshi, "Mitigating measurement errors in quantum computers by exploiting state-dependent bias," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (Micro), 2019.

[30] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: a case for variability-aware policies for NISQ-era quantum computers," in Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), 2019.

[31] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in 38th IEEE International Symposium on Multiple Valued Logic (ISMVL), 2008.

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

[33] H.-S. Zhong *et al.*, "Quantum computational advantage using photons," in Science, 2020.