Towards Efficient Superconducting Quantum Processor Architecture Design

Gushu Li
University of California
Santa Barbara, USA
gushuli@ece.ucsb.edu

Yufei Ding
University of California
Santa Barbara, USA
yufeiding@cs.ucsb.edu

Yuan Xie
University of California
Santa Barbara, USA
yuanxie@ece.ucsb.edu

Abstract

More computational resources (i.e., more physical qubits and qubit connections) on a superconducting quantum processor not only improve the performance but also result in more complex chip architecture with lower yield rate. Optimizing both of them simultaneously is a difficult problem due to their intrinsic trade-off. Inspired by the application-specific design principle, this paper proposes an automatic design flow to generate simplified superconducting quantum processor architecture with negligible performance loss for different quantum programs. Our architecture-design-oriented profiling method identifies program components and patterns critical to both the performance and the yield rate. A follow-up hardware designfl ow decomposes the complicated design procedure into three subroutines, each of which focuses on different hardware components and cooperates with corresponding profiling results and physical constraints. Experimental results show that our design methodology could outperform IBM's general-purpose design schemes with better Pareto-optimal results.

CCS Concepts. • **Hardware → Quantum computation**; • **Computer systems organization → Quantum computing**.

Keywords. quantum computing; superconducting quantum circuit; architecture design; application-specific architecture

ACM Reference Format:
Gushu Li, Yufei Ding, and Yuan Xie. 2020. Towards Efficient Superconducting Quantum Processor Architecture Design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20), March 16–20, 2020, Lausanne, Switzerland.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3373376.3378500

ASPLOS '20, March 16–20, 2020, Lausanne, Switzerland

ACM ISBN 978-1-4503-7102-5/20/03…$15.00
https://doi.org/10.1145/3373376.3378500

1 Introduction

As a promising computation paradigm, Quantum Computing (QC) has been rapidly growing in the last two decades and found its strong potential in many important areas, including machine learning [14, 20], chemistry simulation [32, 39], etc. In particular, the superconducting quantum circuit [13] has become one of the most promising technique candidates for building QC systems [5, 9, 37] due to the ever-increasing qubit coherence time, individual qubit addressability, fabrication technology scalability, etc. Towards efficient superconducting quantum circuit based QC systems, significant research has recently been conducted, ranging from compiler optimization [34, 47] to periphery control hardware support [16, 50] and device innovation [27, 33].

Despite these system optimizations, the performance of a superconducting quantum processor is still highly limited by the amount of computation resource on it. Researchers have been trying to integrate more qubits and qubit connections on one superconducting quantum processor substrate. For example, IBM'sfi rst superconducting quantum chip on the cloud has 5 qubits with 6 qubit connections, while its latest published chip has 20 qubits with 37 qubit connections [10]. Increasing the number of physical qubits on a superconducting quantum processor allows programs with more logical qubits to be executed. Denser qubit connections can increase the overall chip performance by reducing the overhead of qubit mapping and routing [28, 35, 48, 56].

Nevertheless, more qubits and qubit connections will, unfortunately, increase the probability of defect occurrence on a chip, leading to lower yield rate and blocking future development of larger-scale superconducting quantum processor. For example, the yield rate of a 17-qubit chip can be lower than 1% under IBM's state-of-the-art technology [43]. Such a low yield rate comes from *frequency collision*, a unique defect on superconducting quantum processors [6, 30]. The frequencies of physically connected qubits may 'collide' with each other when their values satisfy some specific conditions. More qubit connections naturally increase the probability of frequency collision and lower the yield rate.

To optimize both the yield rate and performance would be desirable, but it is difficult in general due to the inherent trade-off between these two objectives. Most previous efforts on them are direct device-level improvement [26, 27, 33, 44],

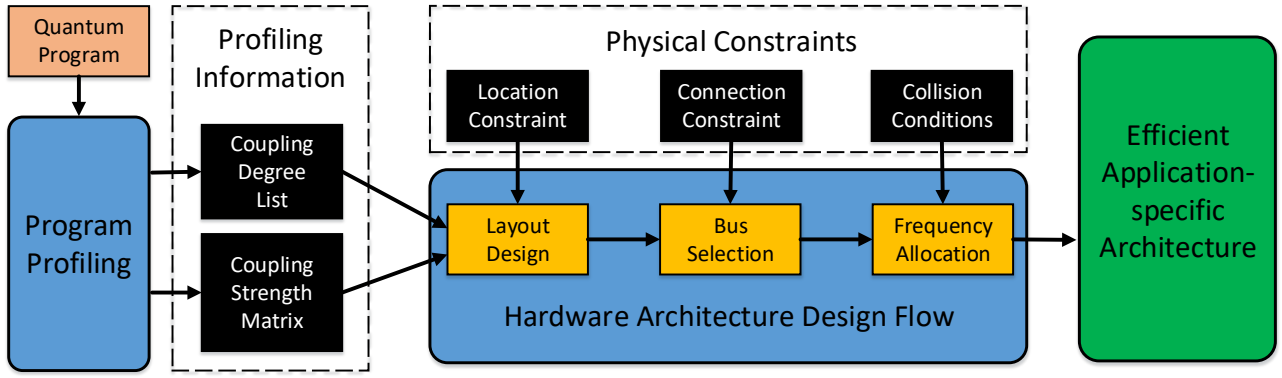

Figure 1. Overview of the Proposed Architecture Design Flow

while little attention has been given to the architectural design of a superconducting quantum processor. This paperfills the gap by exploring the possibility of efficient *application-specific architecture design* to reach an optimized balance between yield rate and performance. We vision that an array of QC accelerators, each of which is tailored to a specific application, is much more likely to be adopted in the near term where computation resources are still limited before we can reach a universal quantum computer (i.e., one quantum computer that runs all kinds of quantum programs). Our design shares the same high-level spirit with the hardware architecture designs in classical computing (e.g., machine learning [8, 19], graph processing [1, 18]), but faces different scenarios because both the program patterns and the hardware design space are different in QC.

In particular, we highlight two key challenges to be addressed before the application-specific principle can be applied in superconducting quantum processor design. **First**, we need to identify and abstract the computation pattern of quantum programs that can guide the hardware architecture design. Prior quantum program analysis studies [21, 24, 38, 53–55] mainly focused on software or compiler optimization and cannot extract appropriate information for hardware architecture optimization. **Second**, the abstracted computation pattern must give guidance to efficient architectural designs, which employ fewer computation resources with physical constraints satisfied to achieve both high yield rate and performance. Existing superconducting quantum processor design schemes cannot handle such irregular/complicated application-specific architecture design tasks [7, 12, 29, 43].

To overcome these two challenges, we design a systematic designfl ow to automatically generate efficient superconducting quantum processor architecture designs for different quantum programs (shown in Figure 1). Wefi rst identify two key computation patterns in quantum programs, ***coupling degree list*** and ***coupling strength matrix***. A profiler is built to automatically extract them from an input quantum program. Both of them are critical to the program performance and hardware yield rate, and thus optimizing their underlying architecture support can potentially achieve a better balance between the performance and yield rate. We then propose an architecture designfl ow, which comes with three key subroutines, ***layout design***, ***bus selection***, and ***frequency allocation***. Each subroutine focuses on different hardware resources and must cooperate with corresponding profiling results and physical constraints. We further propose an array of heuristics to ensure the scalability and effectiveness of the architecture search process. Empirical studies show that these heuristics canfi nd 'near-optimal' solution in the reduced search space.

In summary, this paper makes the following contributions:

- We are thefi rst to identify the optimization opportunity from the architecture level to push forward the balance between performance and hardware yield rate for superconducting QC processors.
- We formalize an end-to-end designfl ow, equipped with a set of novel algorithmic primitives, to automatically generate a series of application-specific architectural designs under different hardware resource limits.
- Comprehensive experiments show that our designfl ow could outperform IBM's general-purpose designs with better Pareto-optimal results, e.g., magnitudes of yield improvement with negligible performance loss.

2 Background

In this section, we will introduce the necessary QC basics for understanding the following program profiling and superconducting quantum processor architecture design.

2.1 QC Program Basics

A quantum program can be represented in the well adopted quantum circuit model [36]. We will start from the basic components in a quantum circuit and then illustrate how they compose a quantum circuit.

Logical Qubit and Quantum Operation A quantum program consists of some logical qubits as variables and some quantum operations which can modify the state of the qubits. Qubit is the basic information processing unit in QC, which has two basis states denoted as $|0\rangle$ and $|1\rangle$. One qubit can be not only the basis states themselves but also their linear combinations which can be depicted by a vector in the Hilbert space. The state of the qubits can be modified by quantum operations. Thefi rst type of quantum operation is unitary operation, also known as quantum gates in the circuit model, which can implement a unitary transformation on the qubit state. Quantum gates can be applied on single qubit or multiple qubits. The second type is measurement operation, which forces the qubits to collapse to basis states.

Quantum Circuit Quantum circuit is a model of QC in which the computation is a sequence of quantum gates and measurement operations. The state of the qubits isfi rst initialized and then manipulated by a sequence of operations. Single-qubit gates and measurement operations are applied on individual qubits while two-qubit gates are applied on two logical qubits. It has been proved that any multi-qubit gate can be decomposed into a series of single-qubit gates and CNOT gates (a specific two-qubit gate) [4]. This is also the basic gate set directly supported on IBM's devices. As a result, this paper assumes that the quantum circuit has been decomposed and gates with three or more qubits are not considered.

2.2 Superconducting Quantum Circuit Basics

All the qubits and quantum operations in a quantum circuit must be implemented in a real physical QC system to execute the program. In this paper, we focus on superconducting quantum processors withfi xed-frequency Josephson-junction-based transmon qubits [27] and all-microwave cross-resonance two-qubit gates [41] adopted by IBM [43].

Physical Qubit and Frequency Figure 2 shows the physical circuit and energy levels of a transmon qubit [27]. Due to the nonlinearity of the Josephson junction, the gaps between the energy levels in this quantum anharmonic oscillator are different, which allows us to use the ground state $|0\rangle$ and thefi rst-excited state $|1\rangle$ as the computation basis without populating other states. Suppose the energy gap between $|0\rangle$ and $|1\rangle$ for a qubit is E_{01}. The *frequency* of this qubit f_{01} is defined as $f_{01} = E_{01}/h$, where h is the Planck constant. Similarly, we use f_{12} to represent the energy gap between $|1\rangle$ and $|2\rangle$. For a typical qubit design with effective operations [30], f_{01} and f_{12} are about $5GHz$ and $4.66GHz$, respectively. The anharmonicity of this qubit is defined to be $\delta = f_{12} - f_{01}$, which is $-340MHz$ under this typical design [7, 46].

Qubit Layout The superconducting physical qubits are confined on a 2-dimensional planar substrate. Although the qubit placement can befl exible, major vendors fabricate the qubits in a regularized structure to ensure scalability and reduce the fabrication complexity. For example, IBM's 16-qubit and 20-qubit chips [23] placed their qubits on the nodes of 2×8 and 4×5 lattices, respectively. Google's 72-qubit chip placed its qubits on some nodes of an 11×12 lattice [25].

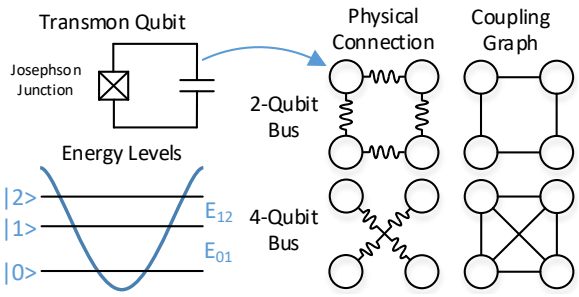

Figure 2. Superconducting Qubit and Connection

Qubit Connection To enable two-qubit gates between two physical qubits, resonators, also known as qubit buses, are employed to connect nearby qubits [41]. For examples, Figure 2 shows two types of commonly used buses. Thefirst one is a 2-qubit bus connecting two physical qubits. The second one is a 4-qubit bus, which connects four physical qubits in a square together. The coupling graphs of these two types of buses are shown on the right. Compared with a 2-qubit bus, 4-qubit bus support two-qubit gates on not only the four qubit pairs on the edges but also two qubit pairs on the diagonals.

Qubit Mapping It is usually assumed that a two-qubit gate can be applied on arbitrary two logical qubits in a quantum program but some two-qubit gates may not be executable due to the limited qubit connection on a superconducting quantum processor. On the hardware side, this problem can be relieved by employing more physical qubit connections so that two-qubit gates can be directly supported on more qubit pairs. On the software side, a qubit-remapping compiler [31] can resolve the dependency of the remaining unexecutable two-qubit gates while additional operations must be introduced with longer execution time and higher error rate. Therefore, more physical qubit connections can help with the overall performance by allowing native two-qubit gates on more physical qubit pairs.

Fabrication Variation Variation is inevitable when fabricating a superconducting quantum processor. If a qubit is designed to have frequency f, the actual frequency after fabrication will be $f' = f + n_f$, where n_f satisfies Gaussian distribution $N(0, \sigma)$. σ is the fabrication precision parameter, which is around $130MHz \sim 150MHz$ under IBM's state-of-the-art technology [43]. Such noise makes it hard to predict the post-fabrication frequency precisely, which brings the probability of frequency collision.

Frequency Collision When two or three qubits are connected, *frequency collision* may happen and cause defects on

	Conditions	Thresholds
1	$f_j \cong f_k$	$\pm 17MHz$
2	$f_j \cong f_k - \delta/2$	$\pm 4MHz$
3	$f_j \cong f_k - \delta$	$\pm 25MHz$
4	$f_j > f_k - \delta$	
5	$f_i \cong f_k$	$\pm 17MHz$
6	$f_i \cong f_k - \delta$	$\pm 25MHz$
7	$2f_j + \delta \cong f_k + f_i$	$\pm 17MHz$

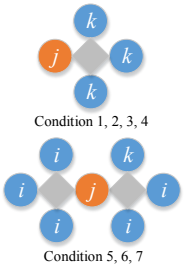

Figure 3. Frequency Collision Conditions [6, 43]

the device. Figure 3 summaries seven qubit frequency collision conditions in IBM's devices [6, 43]. On the left is a table showing the conditions and thresholds of different collision situations. Condition 1, 2, 3, and 4 involve two connected qubits (j and k). Condition 5, 6, and 7 involve three qubits of which two qubits (k and i) both connect to the other qubit j. The approximate equations and the corresponding thresholds determine whether one frequency collision happens. For example, if qubit j and k are connected and $|f_j - f_k| < 17MHz$, then thefi rst condition is satisfied and frequency collision occur. Note that the fourth condition has no threshold because it is an inequality rather than an approximate equation. On the right is a graphical illustration, showing the geometric locations of the qubits that may have frequency collisions of different conditions in two subfigures. Each circle represents one qubit and the gray square represent a 4-qubit bus connecting the four surrounding qubits.

3 Quantum Program Profiling

Thefi rst step towards the development of an architecture-specific quantum processor for both high performance and yield rate is to determine what program information we should focus on. There are several different types of components in a quantum circuit but not all of them will significantly affect the hardware design. Our target program component(s) should satisfy two conditions: 1) the component's execution is a performance bottleneck which can be dramatically improved with optimized hardware support, and 2) the component's required hardware should significantly affect the yield rate.

We found that two-qubit gates can be a key factor to bridge performance and yield. To execute two-qubit gates on a quantum processor with limited qubit-to-qubit coupling, a large number of additional operations are introduced to satisfy their dependencies. But implementing two-qubit gates on two physical qubits require on-chip qubit connections which can lower the yield rate through increasing the probability of frequency collision. Therefore, we give logical qubits and qubit pairs priorities based on the number of involving two-qubit gates to help with the following architecture design. Critical qubits and qubit pairs will have more hardware support to improve the efficiency of the generated architectures.

These remaining components, single-qubit gates, initialization, and measurement operations, do not involve qubit-to-qubit interactions and all happen locally on individual qubits when they are implemented on hardware. As a result, hardware support for these components will not affect the chip yield through frequency collision.

3.1 Profiling Method

As discussed above, our profiling will focus on the logical qubits and the two-qubit gates. Figure 4 shows an example to illustrate the profiling procedure. Suppose we have a quantum circuit as shown in Figure 4 (a). It has 5 logical qubits denoted by $q_{0,1,2,3,4}$. All of them are initialized to be $|0\rangle$. Then some single-qubit gates and two-qubit gates are applied. Measurement operations are at the end.

Wefi rst ignore all single-qubit gates, initialization, and measurement operations. Then we create a logical coupling graph, in which each vertex represents one logical qubit in the circuit. Two vertices are connected by an undirected edge if there exists two-qubit gates applied on the two corresponding logical qubits. The weight of an edge is the number of two-qubit gate instances on the two connected vertices. In this example, Figure 4 (b) shows the generated graph for the example circuit. The weight of the edge between vertex q_0 and vertex q_4 is 2 since there are two two-qubit gates on q_0 and q_4. For all other edges, the weight is 1 because there is only one two-qubit gate on each of those qubit pairs. The first profiling result is the weighted adjacency matrix of the logical coupling graph, namely the *coupling strength matrix*. The element with indices (i,j) represents the number of two-qubit gates between q_i and q_j. Figure 4 (c) shows the *coupling strength matrix* for the example circuit. Note that *coupling strength matrix* is always a symmetric matrix.

The second result is *coupling degree list*. For each qubit, we sum the weights of edges that connect to its corresponding vertex and define the number of two-qubit gates applied on it as the *coupling degree* of one qubit. If one qubit is associated with more two-qubit gates in a quantum circuit than other qubits, this qubit will use the physical qubit connections more frequently when executing on the chip. Naturally, we should pay more attention to those qubits with larger coupling degree. Therefore, all qubits are placed in a sorted list, namely the *coupling degree list*. Figure 4 (d) is the *coupling degree list* in this example. Thefi rst one in this list is q_4 because it has the largest coupling degree. All qubits are in a descending order.

3.2 Gate Pattern Examples

In this section, we show the existence of distinct two-qubit gate patterns and discuss the opportunity for application-specific architecture design with two examples. Figure 5 shows their *coupling strength matrices*. On the left is an 8-qubit UCCSD ansatz for VQE, a quantum simulation algorithm [39]. The high coupling strength qubit pairs form a

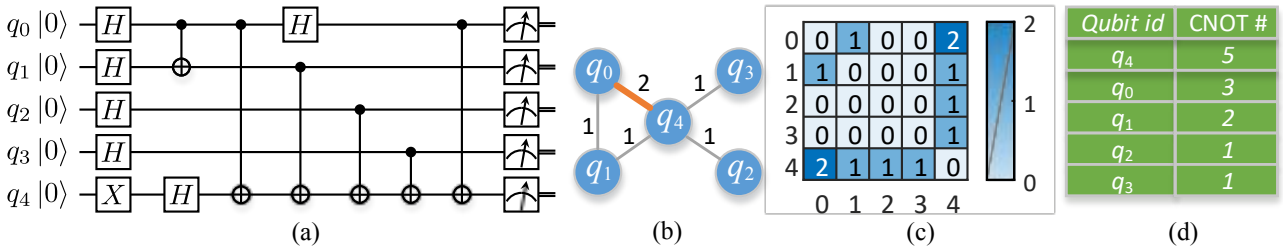

Figure 4. Example of the Profiling Method

chain structure marked by a red rectangle. Q_0 and Q_1 have a large number of two-qubit gates between them, as well as $\{Q_1Q_2, Q_2Q_3, \cdots, Q_6Q_7\}$. For other qubit pairs, the coupling strength is much lower (only about 10%). On the right is a 15-qubit quantum arithmetic function [52]. The coupling strength among $Q_0Q_1 \cdots Q_5$ are 0 since there are no two-qubit gates on any two of them. However, there is a large number of two-qubit gates where one qubit is in the set $Q_{7,8,9,10}$ and the other qubit is in the set $Q_{10,11,12}$ (marked by a red circle). The analysis of these two motivating examples provides us two observations:

1. The numbers of two-qubit gates on different logical qubit pairs can vary dramatically in a real quantum program.
2. Different types of quantum programs can have different two-qubit gate patterns.

These observations suggest that quantum processors can be customized for different programs with different patterns. An efficient architecture can focus on supporting the high-density coupling in a quantum program to reduce the number of connections on-chip. For example, a quantum processor with an 8-qubit chain structure (8 qubits and 7 qubit connections) can immediately support most of the two-qubit gates in the 8-qubit UCCSD ansatz program. The rest two-qubit gates can be supported through remapping without introducing too many additional operations because the total number of the remaining two-qubit gates is relatively small. Such application-specific QC accelerators with simplified architectures can be a more realistic goal in the near term than a general-purpose quantum processor with a large number of hardware resources.

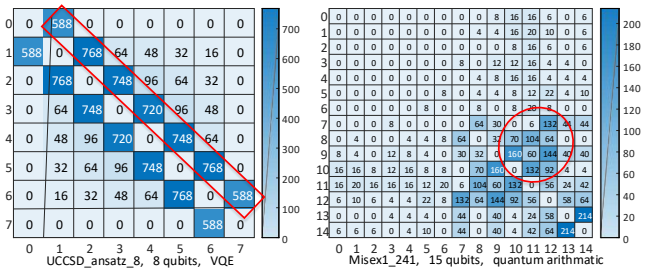

Figure 5. Qubit Coupling Strength Pattern Examples

4 Architecture Design

After a quantum circuit is profiled, a straightforward quantum processor architecture for such a circuit is to organize the on-chip qubits and qubit connections directly based on the logical coupling graph. However, we must consider the physical constraints for a practical architecture. For example, a logical coupling graph may not be perfectly fabricated on hardware since the allowed connections among superconducting qubits are very limited. Moreover, we hope to improve the yield rate by delivering architecture designs with fewer hardware resources. Therefore, the proposed hardware designfl ow must not only invest more hardware resource on frequent operations based on the profiling results, but must also obey the physical constraints on the hardware components arrangement.

To accomplish such a complicated task in a scalable way, we decouple the hardware design procedure into three subroutines and each subroutine focuses on different architecture components, i.e., qubit layout, connection, and frequency. For each subroutine, wefi rst review the difficulty and the physical constraints considered. Then we discuss the design objectives, and how they are achieved in the proposed design algorithms.

4.1 Layout Design

Thefi rst step is to determine where to place the qubits. To ensure scalability and modularity, we follow the convention from major vendors introduced in Section 2 and will only place qubits on the nodes of a 2D lattice. We start from a large 2D lattice, in which each node is initialized to be empty (Figure 6 (a)). Then physical qubits can be placed in the empty nodes and one node can contain at most one qubit.

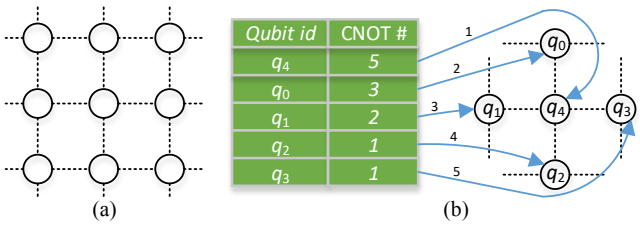

Figure 6. (a) Empty Lattice, (b) Qubit Placement Example

There are many ways to place a given number of qubits on a 2D lattice. For example, 16 qubits can constitute a 4×4 lattice, a 2×8 lattice, or other more irregular structures. But we need to select one qubit layout that is most suitable for executing the program, i.e., most operations can be directly supported or indirectly supported with low overhead. The objectives of this qubit layout design subroutine are summarized as follows.

- Since we need to consider the profiling information, we create a pseudo mapping between logical qubits in the profiled program and the physical qubits in hardware architecture to be delivered. For two logical qubits with a large number of two-qubit gates between them, we hope to place their corresponding physical qubits in adjacent nodes so that later those two-qubit gates can be directly supported by the connection between the two physical qubits.
- One physical qubit can only have a limited number of directly connected qubits. For those two-qubit gates that cannot be directly supported, we hope to reduce the amount of additional operations introduce for remapping the qubits.

We propose a *coupling-based* qubit placement algorithm to determine the geometric locations of the qubits on a 2D lattice (pseudocode shown in Algorithm 1). We illustrate the algorithm with an example in Figure 6. First, we put thefirst qubit in the *coupling degree list*, q_4, on one node of the 2D lattice. Since the initial 2D lattice is empty, the location of q_4 does not matter. We set the geometric coordinate of the first qubit to be $(0, 0)$ and then place the rest qubits around q_4. q_4 has four neighbors, $q_{\{0,1,2,3\}}$, in the logical coupling graph. We need to select the next one to place. By checking the *coupling degree list*, we can see that q_0 is the one with the largest coupling degree. The node occupied by q_4 has four equivalent adjacent nodes and we can place q_0 on any of them. In this example, we select the node on the north of q_4 with coordinate $(0, 1)$. Such an algorithm design ensures that the strongly coupled qubit pairs are given higher priority and placed on adjacent nodes, accomplishing thefi rst objective mentioned above.

Then we need to place q_1 since its coupling degree is larger than that of q_2 and q_3. q_1 is connected to both q_4 and q_0 so that we need a more sophisticated way to evaluate all potential nodes for q_1. We use the function in line 13 of Algorithm 1 tofi nd the node that can make q_1 close to its strong coupled neighbors in the logical coupling graph. This function is the summation over all q_1's placed neighbors. Each term in the summation is the product of the coupling strength between q_1 and one logical coupling neighbor q' and the Manhattan distance between the evaluated node location and the location of q'. After evaluating all the empty nodes that are adjacent to placed nodes q_4 and q_0, we willfind that the nodes on the east and west of q_4 are the best ones because they are closest to q_4 but not far away from q_0. Here we select the one on the west of q_4 with coordinate $(-1, 0)$. This summation function can help reduce the number of operations for later remapping and achieve the second design objective.

Algorithm 1: Qubit Placement on 2D Lattice

Input: *coupling degree list L, coupling strength matrix M*
Output: Geometric coordinates of placed qubits

1 Place the qubit with the largest coupling degree in L at one node with coordinate $(0, 0)$;
2 R = all the qubits remaining; `// R is the set of qubits that has not been placed yet.`
3 **while** *R is not empty* **do**
  `/* Find the next qubit to place */`
4 *qubit_candidate_list* = $\varnothing$;
5 **for** $\boldsymbol{q}$ *in R* **do**
6 **if** $\boldsymbol{q}$ *is connected to any placed qubits* **then**
7 *qubit_candidate_list.append*($\boldsymbol{q}$);
8 **end**
9 **end**
10 Find the qubit $\boldsymbol{q}$ with the largest coupling degree in *qubit_candidate_list*;
11 *node_cost* = [ ];
  `/* Determine the placement location */`
12 **for** ***location*** *of the nodes that are empty and connected to at least one occupied node* **do**
  `/* Heuristic Cost function */`
13 $node\_cost[\boldsymbol{location}] = \sum_{q' \in \boldsymbol{q}.neighbors} M[\boldsymbol{q}, q'] * distance[\boldsymbol{location}, q'.node]$
14 **end**
  `/* q' must be placed neighbor qubits */`
15 Place $\boldsymbol{q}$ in the ***location*** with the minimal score;
16 *R.remove*($\boldsymbol{q}$);
17 **end**

The remaining qubits can be placed in a similar procedure until all the qubits have been placed on the 2D lattice. In this example, q_2 and q_3 are placed on the nodes with coordinates $(0, -1)$ and $(1, 0)$, respectively. All the qubits have their locations (coordinates) on a 2D lattice where we can fabricate one physical qubit on each occupied node. Finally, the nodes with no qubits are removed.

4.2 Bus Selection

In the second step, we need to connect the placed physical qubits to enable two-qubit gates. The difficulty comes from the large size of the design space. For N qubits, there are $\binom{N}{2}$ distinct qubit pairs. Any of them can be either connected or disconnected so that there are $2^{\binom{N}{2}}$ different cases. Even

Algorithm 2: 4-qubit Bus Selection

Input: Geometric coordinates of placed qubits, *coupling strength matrix*, Maximum number of 4-qubit buses K
Output: Locations of 4-qubit Buses

```
Calculate the cross coupling weight for each square;
while K > 0 do
    // Select one square in each iteration
    for square(i, j) in all squares do
        filtered_weight(i, j) = weight(i, j) - weight(i+1, j)
          - weight(i, j+1) - weight(i-1, j) - weight(i, j-1);
    end
    if no square available for 4-qubit bus then
        Break;
    end
    Select the square with the highest
      filtered_weight;
    Set the weights of squares (i+1, j), (i, j+1), (i-1, j),
      and (i, j-1) to be 0 and mark them to be blocked;
    K = K − 1;
end
```

after considering the nearest-neighbor coupling constraint in which one qubit can only connect with few qubits around it on the lattice, the size of the design space is still $O(exp(N))$. More importantly, more qubit connections will improve the performance but lower the yield rate in general so that we need to identify those connections with the most potential performance benefit in a very large design space.

This paper simplifies the connection design problem by considering two types of common buses, 2-qubit bus and 4-qubit bus (shown in Figure 2). These two types of buses naturallyfi t in the 2D lattice qubit layout and can be easily fabricated because at most 4 nearby qubits are connected by one bus. After placing the qubits on a 2D lattice in thefirst step, 2-qubit buses can be directly generated on the edges that connect two occupied nodes but the qubits on a diagonal

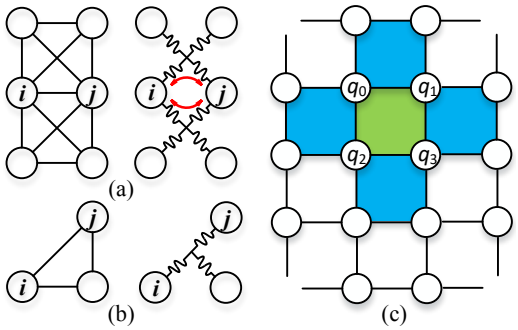

Figure 7. (a) Prohibited Condition, (b) Corner Case, (c) Filtered Weight

of a 4-qubit square can never be connected with only 2-qubit buses. Replacing some 2-qubit buses with 4-qubit buses could provide more qubit connection by trading in yield rate while it is not yet clear where to apply the 4-qubit buses can achieve the Pareto-optimal results. The bus selection subroutine was proposed to identify the locations for 4-qubit buses. Other potential bus designs are left as future research directions and will be discussed in Section 6.

Instead of considering the nodes in a 2D lattice, we consider the squares that are naturally formed by the edges in the 2D lattice. Each square can be configured to 2-qubit bus or 4-qubit bus. Now the problem is on which squares we should use 4-qubit buses. The size of search space, even for this 4-qubit bus square selection problem, is still $O(exp(N))$. But the simplification allows us to design high-quality heuristics to guide the selection. Before introducing our solution, one additional prohibited condition must be considered.

Prohibited Condition One physical constraint that we must consider when applying 4-qubit buses is that we cannot have 4-qubit buses in two adjacent squares. The reason is explained with the example in Figure 7 (a). Suppose we have two adjacent squares and both of them are using 4-qubit buses. Then there will be two physical connections between qubit i and j. When we use one of the connections, the other one will bring unexpected effects so that employing 4-qubit bus in one square will immediately block using 4-qubit buses in any of its adjacent squares.

Considering the physical constraints mentioned above, the objectives of this step are summarized as follows:

- Since adding more qubit connections will increase the probability of frequency collision and lower the yield, we hope to apply 4-qubit buses on those squares that can benefit the performance most. In other words, the additional connections are expected to directly support as many two-qubits gates as possible.
- Applying 4-qubit bus in one square will block adjacent squares, making it impossible to directly support some two-qubit gates in those blocked squares. This effect should also be considered when selecting the 4-qubit squares.

We propose a 4-qubit bus selection algorithm to select some squares for 4-qubit buses (pseudocode shown in Algorithm 2). In each iteration, one square that could benefit most from a 4-qubit bus will be selected. Users can specify the maximum number of 4-qubit buses they hope to have. By varying the number of selected squares, a series of architectures can be generated with a trade-off between yield and performance.

Tofi nd the mostfi tting square, wefi rst need to calculate how much one square could benefit from a 4-qubit bus. Since the difference between a 2-qubit bus square and a 4-qubit bus square is whether the qubit pairs on the diagonals are connected, we define the cross-coupling weight for each

square as the sum of the coupling strength of the qubit pairs on the diagonals. For the example in Figure 7 (c), the cross-coupling weight of the green square is the coupling strength of (q_0, q_3) plus that of (q_1, q_2). A corner case in the coupling weight computation is the square with only 3 qubits (shown in Figure 7 (b)). In such squares, 4-qubit buses can naturally reduce to 3-qubit buses which support coupling between any two of the three connected qubits. The weight of a 3-qubit square is only the weight of logical coupling between the two qubits on one diagonal since the other diagonal only has one qubit. For example, the weight of the 3-qubit square in Figure 7 (b) is the (i,j) element in the *coupling strength matrix*. Except for this small modification, 3-qubit squares are treated equally as other 4-qubit squares in our bus selection step. This cross coupling weight can estimate the potential benefit of applying 4-qubit bus in one square and realize the first objective.

However, the cross-coupling weight is not accurate enough to evaluate the benefit of 4-qubit for a square because the prohibited condition is not yet considered. We design afilter to apply this constraint. For each square, thefi ltered weight is its original cross-coupling weight minus all its neighbors' weights. For example in Figure 7 (c), thefi ltered weight of the green square is its original weight minus the weights of the four blue squares. Thisfi lter can take the prohibited condition into consideration and achieve the second objective.

After applying thefi lter, we will select one square with the highestfi ltered weight. Then we will label the selected square and its adjacent neighbors so that it will no longer be available for future 4-qubit buses. We also change their weights to zero because they should not affect the 4-qubit selection among the remaining squares. The algorithm will iterate again to select the next square until there are not more squares available or we have already applied enough number of 4-qubit buses.

4.3 Frequency Allocation

After the two steps above, we now have a complete coupling topology design of a superconducting quantum processor. In the third step, we need to designate the pre-fabrication frequency of each qubit. IBM's 5-frequency scheme is a regular frequency designation [43]. However, the generated qubit layout and connection in our designfl ow can be irregular since more hardware sources are invested in locations that can benefit the performance most. Thus, we need a more flexible frequency allocation scheme to leverage this unbalanced qubit layout and connection. The objective of this step is to minimize the probability of post-fabrication frequency collision and improve the yield rate. The physical constraints are the frequency collision conditions in Figure 3.

Finding the qubit frequency allocation plan to maximize the yield rate is a hard problem. The complex collision conditions make it difficult tofi nd an analytic expression for the yield rate and a brute-force search over all possible frequency configurations will be very time-consuming. For example, if there are M candidate frequencies for each qubit and we have N qubits in total, the total number of possible frequency configurations is M^N. For each of these potential configurations, we need to run a yield simulation (introduced in Section 4.3.1) and then select the one with maximal yield rate. This method is not acceptable due to its high complexity. We propose to optimize the qubit frequency allocation algorithm based on the facts that 1) the physical qubits in the geometric center of the qubit lattice are more likely to involve in a frequency collision since they usually have more qubit connections, and 2) frequency collision only happens among nearby qubits.

Algorithm 3: Frequency Allocation

Input: Qubit Location and Connection
Output: Frequency Configuration of Each Qubit

```
Select the qubit in the geometric center of the placed qubits and set its frequency to be the middle of the allowed frequency range;
repeat
    Find the next qubit q_i in breadth-first traversal order;
    for temp_freq in all frequency samples do
        Set the frequency of q_i to be temp_freq;
        Simulate the yield rate within q_i's local region;
    end
    Assign the frequency with maximal yield rate to q_i;
until the frequencies of all qubits are determined;
```

Our algorithm determines the qubit frequencies from the center to the periphery (pseudocode shown in Algorithm 3). Since this step is purely about hardware, the input of our algorithm is only the qubit location and connection generated from the previous two subroutines. To reduce the manufacturing difficulty and help prevent the collision condition 4, we follow the convention from IBM and set an allowed frequency interval $5.00GHz$ to $5.34GHz$. All pre-fabrication frequencies are limited within this interval. First, we locate the qubit that is closest to the center of the qubit lattice and assign its frequency to be the center of the allowed frequency

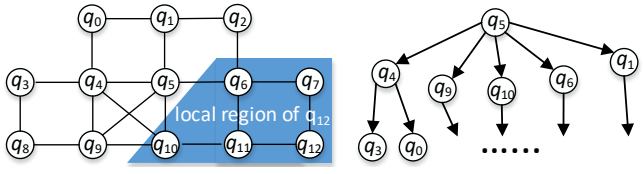

Figure 8. Breath First Frequency Allocation

interval. Then we apply breadth-first traversal on the coupling graph from thefi rst qubit in the center. For example, q_5 is the center qubit in the example shown in Figure 8. In the breadth-first traversal, we willfi rst access $q_{4,9,10,6,1}$ as shown on the right. Each time we access one new qubit, we will immediately determine its frequency. A list of candidate frequencies is prepared. In this paper, the candidate frequencies are $5.00, 5.01, 5.02, \ldots, 5.33, 5.34GHz$ to achieve an accuracy of $0.01GHz$. We can also have more candidate frequencies but it will take more time to evaluate all of them.

To evaluate a candidate frequency on a new qubit, we temporarily assign the candidate frequency to the new qubit and then simulate the yield rate within its local region. The local region of a qubit is defined as a sub-graph of the original chip coupling graph in which a qubit may collide with the new qubit. For example in Figure 8, when we are searching for the best frequency of q_{12}, the local region is marked in blue. Note that it is necessary to consider two hops when allocating frequency for one qubit because the frequency collision conditions in row 5, 6, and 7 of Figure 3 involve 3 connected physical qubits. Qubits not in this region like q_5 cannot collide with q_{12}. We will select the frequency with the maximal yield rate and assign it to the new qubit. Now the time complexity of the frequency allocation algorithm is $O(MN)$ where M is the number of candidate frequencies and N is the number of qubits.

4.3.1 Yield Simulation. We developed a yield simulator based on IBM's yield model [6, 43]. The fabrication process can be modeled by adding a Gaussian noise $N(0, \sigma)$ to the pre-fabrication frequency of a qubit to generate its post-fabrication frequency where σ is the fabrication precision parameter. For a given superconducting quantum processor design, we estimate its yield rate through Monte Carlo simulation. Each time we will simulate if one fabrication is successful. Wefi rst generate the post-fabrication frequencies by adding a random noise sampled from Gaussian distribution mentioned above. Then we check if any frequency collision condition listed in Figure 3 occurs in the post-fabrication frequencies. If so, this fabrication fails. Otherwise, it is successful. All possible cases are taken into account. For example, we will examine the two frequencies of all connected physical qubit pairs for condition 1, 2, 3, and 4. If they meet any one of the inequalities of the conditions, frequency collision is considered to occur in this simulation. This simulation process is repeated many times. The yield rate can be estimated by the ratio between the number of successful simulations and the total number of simulations.

5 Evaluation

To demonstrate that the proposed application-specific architecture designfl ow can deliver hardware designs with better Pareto-optimal results in terms of performance and yield rate, we conduct experiments over various benchmarks to show not only the overall improvement but also the breakdown of benefits from each of our hardware design subroutines.

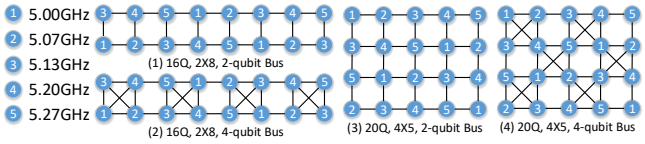

Figure 9. Baseline Qubit Frequency, Layout, and Connection Designs

5.1 Experiment Setup

Benchmarks Twelve quantum programs are collected from IBM's QISKit [2] and RevLib [52], or compiled from ScaffCC [24]. These benchmarks cover several important domains (e.g., simulation, arithmetic) and have various sizes (from 7- to 16-qubit) for a versatility test of the proposed designfl ow.

Metrics To evaluate the efficiency of an architecture, we need both the yield rate and performance. An architecture with a higher yield rate can be successfully fabricated with fewer attempts, indicating a lower hardware cost. In our experiments, the yield rate is simulated with IBM's yield model [6, 43] as introduced in Section 4.3.1. For the performance evaluation, we adopt the total post-mapping gate count metric widely used in previous studies [28, 48, 56]. More gates lead to longer execution time and a larger probability of error on QC devices. If a hardware architecture could execute the program with fewer gates, then its performance is considered to be better.

Yield Simulation Configuration The number of trials in the Monte-Carlo simulation for each architecture is 10,000∼100,000, which is $10 \sim 100\times$ of that used in IBM's experiments [6, 7, 22] to ensure the simulation accuracy. The fabrication precision parameter σ is set to be $30MHz$, a realistic extrapolation of progress in hardware by IBM [7, 43]. IBM has improved the σ from $200MHz$ [42] to $130MHz$ [43] in the last few years and $30MHz$ is a reasonable projection to achieve a useful yield as predicted by IBM [7].

5.2 Experiment Methodology

To illustrate the benefit of our designfl ow,fi ve experiment configurations are designed to show the overall improvement and the performance/yield trade-off gain at each of the three subroutines in Section 4. Among them, **ibm** is a set of general-purpose architectures from IBM and they are not tailored for any applications. The remaining four configurations are application-specific architectures generated by the entire or part of the proposed designfl ow.

ibm We use IBM's design scheme as the baseline configuration. It has two layout options, a 2×8 lattice with 16 qubits, and a 4×5 lattice with 20 qubits. The qubit connection design can be either 2-qubit bus only or using 4-qubit buses

as many as possible. In total, there are four architectures combining the layout and connection options (shown in Figure 9). The frequency allocation scheme is a 5-frequency scheme [7, 43]. Thefi ve frequencies are an arithmetic progression from $5GHz$ to $5.27GHz$ and their arrangement is also in Figure 9.

eff-full We apply all three subroutines and generate a series of efficient superconducting quantum processor architectures by varying the number of 4-qubit buses. The number of designs we can obtain for a quantum program depends on the number of qubits as more qubits can provide more squares to apply 4-qubit buses in the generated layout. In this paper, we obtain the **eff-full** data series through iterating over all possible numbers of 4-qubit buses in the second subroutine for bus selection. This experiment can show the overall architecture design improvement when comparing with the baseline **ibm**.

eff-5-freq We only apply thefi rst two subroutines to generate qubit layout and connection design but the frequency allocation is done with IBM's 5-frequency scheme. The yield benefit from the proposed frequency allocation algorithm can be demonstrated by comparing with results from **eff-full**.

eff-rd-bus We keep thefi rst and the third subroutines but randomly select some squares to employ 4-qubit buses with the prohibited condition constraint satisfied. This will demonstrate the effect of ourfi ltered-weight-based 4-qubit bus selection algorithm by comparing with results from **eff-full**.

eff-layout-only We apply our profiling method and perform a layout design. The connection design has two options. One is only using 2-qubit buses. The other is using 4-qubit buses as much as possible. The frequency design follows the baseline **ibm**. The benefit of our layout optimization can be shown when comparing with the results from **ibm**.

For each benchmark, we run all thefi ve configurations to generate different superconducting quantum processor architectures with different yield rates. Then we apply one state-of-the-art qubit mapping algorithm [28] on these architectures to obtain the total number of gates when running the generated or baseline architectures.

5.3 Overall Improvement

Figure 10 shows the result of yield and performance for all benchmarks and thefi ve experiment configurations. There are 12 subfigures and one subfigure contains the results of thefi ve experiment configurations for one benchmark. The X-axis represents the normalized reciprocal of post-mapping gate count and data points on the **right** have better performance. The Y-axis represents the yield rate and data points on the **top** have higher yield rates. The legend at the bottom of Figure 10 shows the markers for thefi ve configurations. The data points for the four designs in the baseline are labeled by (1), (2), (3), and (4), according to Figure 9.

Optimality The optimal solution in this paper means the Pareto-optimal solution in terms of post-mapping gate count and yield rate. A series of architectures with better Pareto-optimal results can be generated by our designfl ow as the data of **eff-full** is on the upper right of **ibm**. The most simplified designs (the most left top blue triangle data point in **eff-full**, zero 4-qubit buses) generated by our designfl ow outperforms the 16-qubit baseline design (data point (1) in **ibm**) without 4-qubit buses in both performance ($\sim$ 7.7%) and yield rate ($\sim$ 4×). Compared with the 16-qubit baseline with four 4-qubit buses (data point (2) in **ibm**), our designs with zero 4-qubit buses achieve over 100× better yield rate with $<$ 1% performance loss. On the other side, compared with IBM's 20-qubit chip design with six 4-qubit buses (the baseline design with the most hardware resources, data point (4) in **ibm**), the designs with the maximum number of 4-qubit buses generated from our designfl ow (the data points on the most bottom right in **eff-full**) have over 1000× yield rate improvement on average with only about 3.5% performance loss.

Controllability The proposed designfl ow can easily control the trade-off between yield and performance by only changing the number of 4-qubit buses without traversing across, or sampling a large number of designs in, the entire search space. Depending on the number of qubits in different target programs, we can trade in around 10× $\sim$ 50× yield rate for 10% $\sim$ 33% performance improvement.

5.3.1 Special Case. The results of *ising_model* are significantly different because the logical qubit coupling in this benchmark forms a chain structure. The mapping algorithm can alwaysfi nd the perfect initial mapping without inserting additional operations. As a result, the post-mapping gate count is the same for all tested hardware architectures. All data points for this program lie in one vertical line. Only one architecture is generated from our designfl ow because there is no need to add 4-qubit bus. All the two-qubit gates can be executed through the edges on the 2D lattice. There are no two-qubit gates applied on two qubits on a diagonal because of the chain coupling structure. In this case, 4-qubit buses can only lower the yield rate without improving the performance.

5.4 Effects from Individual Subroutines

The overall improvement has already been discussed, but one interesting question is how much improvement the layout and connection optimization contribute and how much comes from the optimized yield allocation directly. Thefi ve configurations decouple the proposed designfl ow and provide a breakdown of the effect of individual subroutines.

5.4.1 Effect of Layout Design. The difference between **ibm** and **eff-layout-only** illustrates the effect of layout design since the rest two subroutines are the same. An architecture with more hardware resources is expected to provide

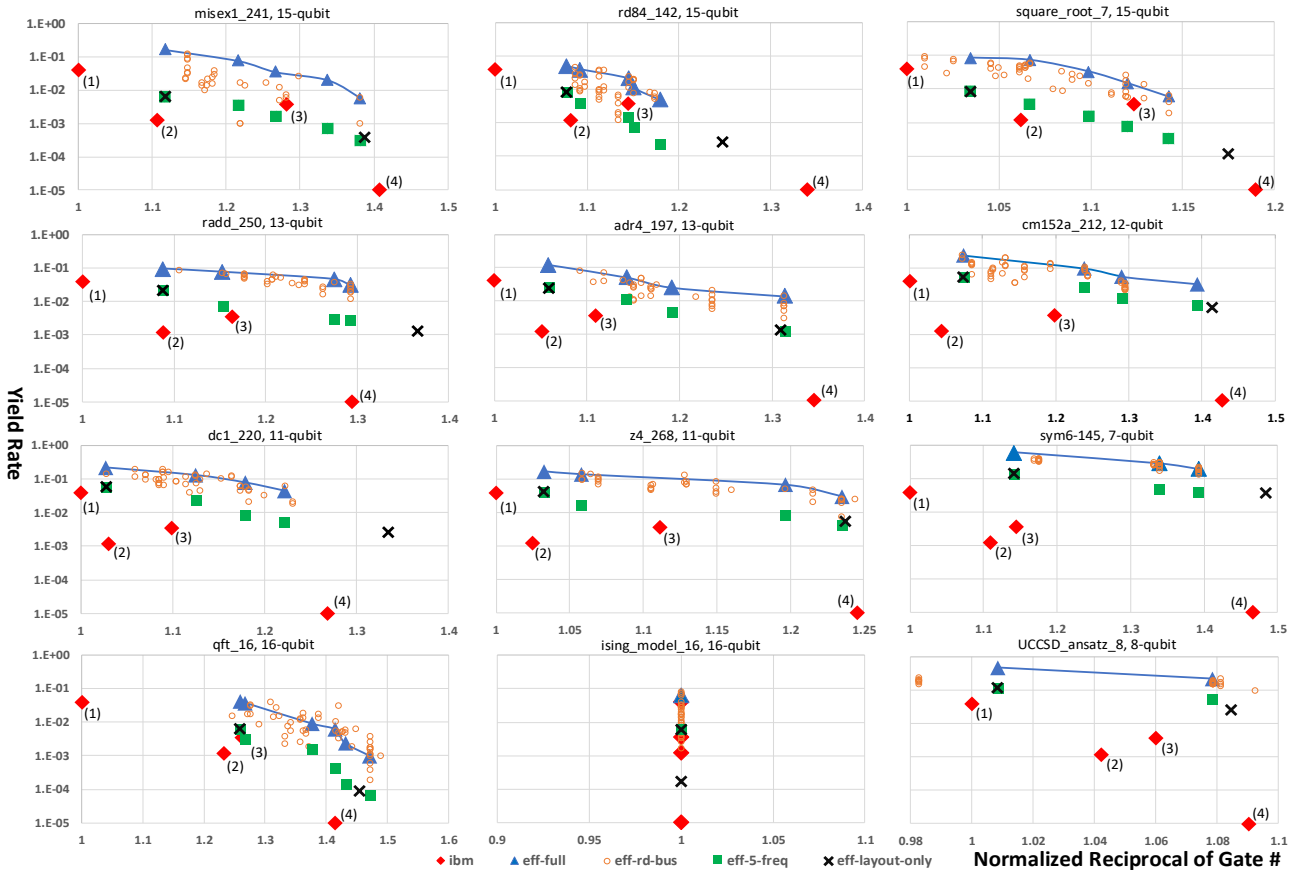

Figure 10. Yield v.s. Normalized Reciprocal of Post-mapping Gate Count

higher performance by allowing morefl exibility in qubit mapping. But our optimized layout design could use comparable or fewer hardware resources while the performance can be even better. For example, we compare the 2-qubit bus only data point (the upper left one) with the 16-qubit baseline with four 4-qubit buses (labeled by (2) in each subfigure). **eff-layout-only** provides better or comparable performance most of the time with about 35× yield improvement on average. The improvement at this step depends on the program size and programs with fewer qubits will use fewer qubits and connections in an optimized architecture. This result proves that our layout design could generate qubit layout with high performance but using much fewer hardware resource for different programs.

5.4.2 4-qubit Bus Selection Quality. By comparing the results from **eff-full** and **eff-rd-bus**, we can see that the architectures generated from our bus selection algorithm are better than that of random selection in trading in yield for performance most of the time. The data points of **eff-rd-bus** reveal the distribution of the yield and performance sampled from random bus designs. Note that the performance of **eff-rd-bus** is usually confined by the two data points in **eff-layout-only** because adding connections can improve the performance most of the time. For most benchmarks except *qft*, the results from **eff-full** are close to the upper bound formulated by the random samples, which shows that our weight-based bus selection could generate a series of near Pareto-optimal hardware architectures with various numbers of qubit connections.

The result of *qft* is much worse than that of other programs due to the unique uniform two-qubit gate pattern in this program. The number of two-qubit gates between arbitrary two logical qubits is always two in *qft*, which makes all the logical qubit pairs are the same in the sense the coupling strength during profiling. Then in bus selection subroutine, all the squares share the same weight and the weight-based selection is the same as random selection.

For the two small benchmarks, *sym6* and *UCCSD_ansatz*, the number of available squares in the generated qubit layout is small and there are very few options when applying 4-qubit buses. Therefore, most of the architectures generated from the random 4-qubit bus selection are the same as those from the proposed designfl ow, which makes the results from **eff-full** and **eff-rd-bus** very close.

5.4.3 Frequency Allocation Optimization. By comparing **eff-full** and **eff-5-freq**, we can see that the proposed

frequency allocation algorithm provides about 10× yield rate improvement on average. This improvement is slightly worse when the yield from the baseline 5-frequency is already high, e.g., results from *sym6* and *UCCSD_ansatz*. The fabrication variance makes the ideal yield 100% unreachable and it is hard to optimize yield when it is already high.

6 Discussion

This paper studies application-specific efficient superconducting quantum processor design. In particular, we formalize the architecture design for superconducting quantum processors with three key steps, each of which comes with an optimization subroutine. This is thefi rst attempt, to the best of our knowledge, to identify the optimization opportunity from the architecture level to push forward the balance between QC performance and hardware yield rate. Effort towards this direction can be of significant demand in the near term QC with limited computation resource and immature fabrication technology.

Although we show that improved Pareto-optimal designs can be generated with a static program analysis and three optimized design algorithms, several future research directions can be explored as with any initial research.

Improving Profiling Method This paper focused on the logical qubit coupling topology in a quantum program but other patterns may also be leveraged. We omitted the temporal information of the two-qubit gates and all information about other program components. But the locations of two-qubit gates in a quantum program may also be leveraged for finer-grained evaluation of the coupling strength for different logical qubit pairs at different times during the execution. The single-qubit patterns can also help with the basic gate set design.

Exploring More Design Space In the proposed design flow, the number of physical qubits is the same as that of logical qubits for higher yield rate. However, we can still add auxiliary physical qubits since they can also be used during the qubit routing, trading in more yield rate for higher performance. How to add auxiliary qubit to appropriate locations and how to connect them are interesting problems to explore in the future. To ensure modularity and scalability, the qubits are forced to be embedded in a 2D lattice and only consider two types of buses lying in the lattice. However, the qubit placement and connection could be morefl exible if we trade in part of the scalability. For example, one bus could also connect more than four qubits [17]. The design space in this direction is not yet explored.

Optimizing Frequency Allocation This paper tried to optimize the qubit frequency selection from the center to periphery and only searched for the optimal frequency for one qubit, resulting in a sub-optimal frequency allocation. A global optimization like formal methods can be explored to further optimize the frequency allocation result. One alternative approach to resolve the frequency collision issue is to usefl ux-tunable transmon qubits [26], of which the frequencies can be dynamically tuned with additional control signals. The design trade-off of different types of qubits is not yet explored and additional signals bring more noise and increase the control complexity. The proposed designfl ow is still valuable even with frequency-tunable qubits because the simplified architectures with fewer the on-chip connections can not only reduce the fabrication complexity but also benefit the overall performance by lowering the crosstalk error.

7 Related Work

This paper ranges across multiple topics, i.e., program profiling, superconducting processor design, application-specific design, qubit mapping. We briefly introduce related work for all of them.

Application-specific Design The closest related work is SPARQS, a superconducting planar architecture proposed by Wilhelm *et al.* [12, 29] targeting a specific Fermi-Hubbard model simulation program. However, they only provide an implementation-independent design from theoretical physics level. This paper formalizes a systematic end-to-end design flow with automatic program profiling and realistic physical constraints included, for thefi rst time. With no limitation on the target program, we can generate a series of Pareto-optimal hardware architecture designs in a controllable way.

Quantum Program Profiling and Analysis Program profiling and analysis are very important for software and compiler optimization. Previous works on quantum program analysis [21, 24, 38, 53–55] have studied entanglement, termination, non-cloning checking, etc. The profiling method in this paper is proposed to guide the hardware design, fulfilling a different goal.

Superconducting Quantum Processors As one of the most promising candidate technology to implement QC, superconducting quantum techniques have been employed in two mainstream QC computation models. The circuit model based processors [23, 25, 40] support quantum circuit model [36] and the quantum annealers [11] can implement adiabatic QC [15]. Their programming model and hardware architecture are different for these two QC approaches. The designfl ow in this paper is proposed for circuit model based quantum processors while efficient quantum annealer design can be a future research direction.

Qubit Mapping Formal and heuristic methods have been attempted to solve this problem [28, 45, 48, 51, 56] and minimize the total gate count. Recently several studies [3, 34, 49] have applied the actual gate error rates forfi ne-grained optimization. All these optimizations are pure software-level

modification. This paper attempts to improve the performance by reducing the mapping overhead from the hardware level. We adopt the gate count metric to estimate the mapping overhead since our experiments are performed on artificial hardware architectures.

8 Conclusion

The demand for larger computation capability in a superconducting quantum processor naturally calls for more hardware resources which will also increase the design complexity and lower the yield rate. This paper explored application-specific architecture design for superconducting quantum processors to achieve both high performance and higher yield rate. Gate patterns in a quantum program can be extracted by the proposed profiling method and then utilized in the follow-up hardware architecture design. Three subroutines are designed to generate the qubit layout, connection, and frequency respectively with physical constraints taken into consideration. Experimental results show that the proposed designfl ow could deliver architectures with both high yield rate and performance automatically for different applications except those with extremely special gate patterns.

Acknowledgments

We thank all anonymous reviewers for their valuable comments. This work was supported in part by NSF 1730309 and 1925717.

References

[1] Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi. 2016. A scalable processing-in-memory accelerator for parallel graph processing. *ACM SIGARCH Computer Architecture News* 43, 3 (2016), 105–117.

[2] Gadi Aleksandrowicz, Thomas Alexander, Panagiotis Barkoutsos, Luciano Bello, Yael Ben-Haim, David Bucher, Francisco Jose Cabrera-Hernádez, Jorge Carballo-Franquis, Adrian Chen, Chun-Fu Chen, Jerry M. Chow, Antonio D. Córcoles-Gonzales, Abigail J. Cross, Andrew Cross, Juan Cruz-Benito, Chris Culver, Salvador De La Puente González, Enrique De La Torre, Delton Ding, Eugene Dumitrescu, Ivan Duran, Pieter Eendebak, Mark Everitt, Ismael Faro Sertage, Albert Frisch, Andreas Fuhrer, Jay Gambetta, Borja Godoy Gago, Juan Gomez-Mosquera, Donny Greenberg, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers,Ł ukasz Herok, Hiroshi Horii, Shaohan Hu, Takashi Imamichi, Toshinari Itoko, Ali Javadi-Abhari, Naoki Kanazawa, Anton Karazeev, Kevin Krsulich, Peng Liu, Yang Luh, Yunho Maeng, Manoel Marques, Francisco Jose Martín-Fernández, Douglas T. McClure, David McKay, Srujan Meesala, Antonio Mezzacapo, Nikolaj Moll, Diego Moreda Rodríguez, Giacomo Nannicini, Paul Nation, Pauline Ollitrault, Lee James O'Riordan, Hanhee Paik, Jesús Pérez, Anna Phan, Marco Pistoia, Viktor Prutyanov, Max Reuter, Julia Rice, Abdón Rodríguez Davila, Raymond Harry Putra Rudy, Mingi Ryu, Ninad Sathaye, Chris Schnabel, Eddie Schoute, Kanav Setia, Yunong Shi, Adenilton Silva, Yukio Siraichi, Seyon Sivarajah, John A. Smolin, Mathias Soeken, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor, Pete Taylour, Kenso Trabing, Matthew Treinish, Wes Turner, Desiree Vogt-Lee, Christophe Vuillot, Jonathan A. Wildstrom, Jessica Wilson, Erick Winston, Christopher Wood, Stephen Wood, Stefan Wörner, Ismail Yunus Akhalwaya, and Christa Zoufal. 2019. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2562110

[3] Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh. 2019. QURE: Qubit Re-allocation in Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 141.

[4] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. 1995. Elementary gates for quantum computation. *Physical review A* 52, 5 (1995), 3457.

[5] R. Barends, J. Kelly, A. Megrant, D. Sank, E. Jeffrey, Y. Chen, Y. Yin, B. Chiaro, J. Mutus, C. Neill, P. O'Malley, P. Roushan, J. Wenner, T. C. White, A. N. Cleland, and John M. Martinis. 2013. Coherent Josephson Qubit Suitable for Scalable Quantum Integrated Circuits. *Phys. Rev. Lett.* 111 (Aug 2013), 080502. Issue 8. https://doi.org/10.1103/PhysRevLett.111.080502

[6] Markus Brink, Jerry M Chow, Jared Hertzberg, Easwar Magesan, and Sami Rosenblatt. 2018. Device challenges for near term superconducting quantum processors: frequency collisions. In *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 6–1.

[7] Christopher Chamberland, Guanyu Zhu, Theodore J Yoder, Jared B Hertzberg, and Andrew W Cross. 2019. Topological and subsystem codes on low-degree graphs withfl ag qubits. *arXiv preprint arXiv:1907.09528* (2019).

[8] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. 2014. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, Vol. 49. ACM, 269–284.

[9] Yu Chen, C Neill, P Roushan, N Leung, M Fang, R Barends, J Kelly, B Campbell, Z Chen, B Chiaro, and A Dunsworth. 2014. Qubit architecture with high coherence and fast tunable coupling. *Physical review letters* 113, 22 (2014), 220502.

[10] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. 2018. Validating quantum computers using randomized model circuits. *arXiv preprint arXiv:1811.12926* (2018).

[11] D-Wave Systems Inc. 2018. D-Wave System Documentation. https://docs.dwavesys.com/docs/latest/.

[12] Pierre-Luc Dallaire-Demers and Frank K Wilhelm. 2016. Quantum gates and architecture for the quantum simulation of the Fermi-Hubbard model. *Physical Review A* 94, 6 (2016), 062304.

[13] Michel H Devoret and Robert J Schoelkopf. 2013. Superconducting circuits for quantum information: an outlook. *Science* 339, 6124 (2013), 1169–1174.

[14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).

[15] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. 2000. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106* (2000).

[16] X Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels. 2017. An experimental microarchitecture for a superconducting quantum processor. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE/ACM, 813–825.

[17] Joydip Ghosh, Andrei Galiautdinov, Zhongyuan Zhou, Alexander N Korotkov, John M Martinis, and Michael R Geller. 2013. High-fidelity controlled-σ z gate for resonator-based superconducting quantum computers. *Physical Review A* 87, 2 (2013), 022309.

[18] Tae Jun Ham, Lisa Wu, Narayanan Sundaram, Nadathur Satish, and Margaret Martonosi. 2016. Graphicionado: A high-performance and energy-efficient accelerator for graph analytics. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[19] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. 2016. EIE: efficient inference engine

on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 243–254.

[20] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. 2009. Quantum algorithm for linear systems of equations. *Physical review letters* 103, 15 (2009), 150502.

[21] Kentaro Honda. 2015. Analysis of Quantum Entanglement in Quantum Programs using Stabilizer Formalism. *arXiv preprint arXiv:1511.01572* (2015).

[22] MD Hutchings, Jared B Hertzberg, Yebin Liu, Nicholas T Bronn, George A Keefe, Markus Brink, Jerry M Chow, and BLT Plourde. 2017. Tunable superconducting qubits withfl ux-independent coherence. *Physical Review Applied* 8, 4 (2017), 044003.

[23] IBM. 2018. IBM Q Experience Device. https://www.research.ibm.com/ibm-q/technology/devices/.

[24] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. 2015. ScaffCC: Scalable compilation and analysis of quantum programs. *Parallel Comput.* 45 (2015), 2–17.

[25] Julian Kelly. 2018. A Preview of Bristlecone, Google's New Quantum Processor. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html.

[26] Julian Kelly, R Barends, AG Fowler, A Megrant, E Jeffrey, TC White, D Sank, JY Mutus, B Campbell, Yu Chen, and Z Chen. 2015. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature* 519, 7541 (2015), 66.

[27] Jens Koch, M Yu Terri, Jay Gambetta, Andrew A Houck, DI Schuster, J Majer, Alexandre Blais, Michel H Devoret, Steven M Girvin, and Robert J Schoelkopf. 2007. Charge-insensitive qubit design derived from the Cooper pair box. *Physical Review A* 76, 4 (2007), 042319.

[28] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1001–1014.

[29] Per J Liebermann, Pierre-Luc Dallaire-Demers, and Frank K Wilhelm. 2017. Implementation of the iFREDKIN gate in scalable superconducting architecture for the quantum simulation of Fermionic systems. *arXiv preprint arXiv:1701.07870* (2017).

[30] Easwar Magesan and Jay M Gambetta. 2018. Effective Hamiltonian models of the cross-resonance gate. *arXiv preprint arXiv:1804.04073* (2018).

[31] Dmitri Maslov, Sean M Falconer, and Michele Mosca. 2008. Quantum Circuit Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 4 (2008), 752–763.

[32] Sam McArdle, Suguru Endo, Alan Aspuru-Guzik, Simon Benjamin, and Xiao Yuan. 2018. Quantum computational chemistry. *arXiv preprint arXiv:1808.10402* (2018).

[33] David C McKay, Stefan Filipp, Antonio Mezzacapo, Easwar Magesan, Jerry M Chow, and Jay M Gambetta. 2016. Universal gate forfi xed-frequency qubits via a tunable bus. *Physical Review Applied* 6, 6 (2016), 064007.

[34] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. 2019. Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*. ACM, 1015–1029.

[35] Prakash Murali, Norbert Matthias Linke, Margaret Martonosi, Ali Javadi Abhari, Nhung Hong Nguyen, and Cinthia Huerta Alderete. 2019. Full-stack, Real-system Quantum Computer Studies: Architectural Comparisons and Design Insights. In *Proceedings of the 46th International Symposium on Computer Architecture* (Phoenix, Arizona) *(ISCA '19)*. ACM, New York, NY, USA, 527–540. https://doi.org/10.1145/3307650.3322273

[36] Michael A Nielsen and Isaac L Chuang. 2010. Quantum Computation and Quantum Information. *Quantum Computation and Quantum Information, by Michael A. Nielsen, Isaac L. Chuang, Cambridge, UK: Cambridge University Press, 2010* (2010).

[37] Hanhee Paik, D. I. Schuster, Lev S. Bishop, G. Kirchmair, G. Catelani, A. P. Sears, B. R. Johnson, M. J. Reagor, L. Frunzio, L. I. Glazman, S. M. Girvin, M. H. Devoret, and R. J. Schoelkopf. 2011. Observation of High Coherence in Josephson Junction Qubits Measured in a Three-Dimensional Circuit QED Architecture. *Phys. Rev. Lett.* 107 (Dec 2011), 240501. Issue 24. https://doi.org/10.1103/PhysRevLett.107.240501

[38] Simon Perdrix. 2008. Quantum entanglement analysis based on abstract interpretation. In *International Static Analysis Symposium*. Springer, 270–282.

[39] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L O'brien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature communications* 5 (2014), 4213.

[40] Rigetti. 2018. The Quantum Processing Unit (QPU). https://www.rigetti.com/qpu.

[41] Chad Rigetti and Michel Devoret. 2010. Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies. *Physical Review B* 81, 13 (2010), 134507.

[42] Sami Rosenblatt, Jared Hertzberg, Markus Brink, Jerry Chow, Jay Gambetta, Zhaoqi Leng, Andrew Houck, JJ Nelson, Britton Plourde, Xian Wu, et al. 2017. Variability metrics in Josephson Junction fabrication for Quantum Computing circuits. In *APS Meeting Abstracts*.

[43] Sami Rosenblatt, Jared Hertzberg, José Chavez-Garcia, Nicholas Bronn, Hanhee Paik, Martin Sandberg, Easwar Magesan, John Smolin, Jeng-Bang Yau, Vivekananda Adiga, Markus Brink, and Jerry M. Chow. 2019. Enablement of near-term quantum processors by architectural yield engineering. *Bulletin of the American Physical Society* (2019).

[44] Sami Rosenblatt, Jason S Orcutt, and Jerry M Chow. 2019. Laser annealing qubits for optimized frequency allocation. US Patent App. 10/340,438.

[45] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. 2014. Qubit placement to minimize communication overhead in 2D quantum architectures. In *Design Automation Conference (ASP-DAC), 2014 19th Asia and South Pacific*. IEEE, 495–500.

[46] Sarah Sheldon, Easwar Magesan, Jerry M Chow, and Jay M Gambetta. 2016. Procedure for systematically tuning up cross-talk in the cross-resonance gate. *Physical Review A* 93, 6 (2016), 060302.

[47] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I Schuster, Henry Hoffmann, and Frederic T Chong. 2019. Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1031–1044.

[48] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintão Pereira. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 113–125.

[49] Swamit S Tannu and Moinuddin K Qureshi. 2019. Not all qubits are created equal: a case for variability-aware policies for NISQ-era quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 987–999.

[50] Jeroen PG van Dijk, Edoardo Charbon, and Fabio Sebastiano. 2018. The electronic interface for quantum processors. *arXiv preprint arXiv:1811.01693* (2018).

[51] Davide Venturelli, Minh Do, Eleanor Rieffel, and Jeremy Frank. 2018. Compiling quantum circuits to realistic hardware architectures using temporal planners. *Quantum Science and Technology* 3, 2 (2018), 025004.

[52] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W Dueck, and Rolf Drechsler. 2008. RevLib: An online resource for reversible functions

and reversible circuits. In *Multiple Valued Logic, 2008. ISMVL 2008. 38th International Symposium on*. IEEE, 220–225.

[53] Mingsheng Ying and Yuan Feng. 2010. Quantum loop programs. *Acta Informatica* 47, 4 (2010), 221–250.

[54] Mingsheng Ying, Nengkun Yu, Yuan Feng, and Runyao Duan. 2013. Verification of quantum programs. *Science of Computer Programming* 78, 9 (2013), 1679–1700.

[55] Shenggang Ying, Yuan Feng, Nengkun Yu, and Mingsheng Ying. 2013. Reachability probabilities of quantum Markov chains. In *International Conference on Concurrency Theory*. Springer, 334–348.

[56] Alwin Zulehner, Alexandru Paler, and Robert Wille. 2018. Efficient mapping of quantum circuits to the IBM QX architectures. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 1135–1138.