Contents lists available at ScienceDirect

Results in Physics

journal homepage: www.elsevier.com/locate/rinp

A superconducting quantum processor architecture design method for improving performance and reducing frequency collisions

Tian Yang [a], Weilong Wang [a,*], Lixin Wang [a], Bo Zhao [a], Chen Liang [a,b], Zheng Shan [a,*]

[a] State Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China
[b] School of Cyber Science and Engineering, Zhengzhou University, Zhengzhou 450002, China

ARTICLE INFO

Keywords:
Architecture design
Frequency collisions
Graph theory
Optimization problem
Performance
Superconducting quantum processor

ABSTRACT

More physical qubits and qubit connections integrated on a superconducting quantum processor can improve the ability to execute quantum programs, but on the other hand, they might increase the probability of frequency collisions. Considering a performance and frequency collisions trade-off, it is a feasible method to optimize processor architecture aiming at running specific quantum programs. To this end, we propose a method for designing superconducting quantum processor architectures for the purpose of running specific quantum programs. Different from existing methods, our method is mainly based on graph theory to optimize processor architecture design. First, we convert the trade-off problem into the optimization of the distance between two points as well as the maximum degree in the processor architecture graph. Second, we consider the actual physical routing constraints and build a mathematical model for the optimization problem. Finally, we propose an automatic processor architecture design flow based on the mathematical model, which is implemented with an improved genetic algorithm. To show the effectiveness of our method, we selected sixteen quantum programs with different qubit numbers and different functions for comparison. Simulation results show that the architecture schemes of our method outperform IBM's general-purpose square lattice architecture schemes. Compared to the general-purpose architecture schemes, the architecture schemes of our method have an average performance improvement of 15.61% and a minimum reduction of 21.33% in the probability of frequency collisions. Furthermore, in most of the selected quantum programs, our architecture schemes perform better than the eff-5-freq's architecture schemes, with an average 6.58% improvement in performance and a minimum 6.45% reduction in the probability of frequency collisions. Therefore, our method can provide superconducting quantum processor architecture design with better performance and lower probability of frequency collisions for quantum programs.

Introduction

As a new generation of information processing technology, quantum computing (QC), by using superposition, entanglement, and other quantum mechanical phenomena, makes it possible to deal with quantum simulation [1], quantum chemistry [2], combinatorial optimization [3], machine learning [4], cryptography [5] and some other complex problems with unparalleled acceleration advantages over classical computing. Up to now, superconductors [6], trapped ions [7], optics [8], semiconductor quantum dots [9], and other physical systems have been used for the realization of physical quantum computers. Among them, the superconducting quantum system has become a leading candidate in the quantum computing commercial space due to its high compatibility with existing integrated circuit technologies in terms of design, fabrication, and measurement [10,11]. In particular, Google and IBM have recently made breakthroughs in solving the quantum noise problem of superconducting quantum systems [12,13]. This will further advance superconducting quantum systems to realize universal quantum computation [14–16].

To make the superconducting quantum computer run efficiently, researchers have not only worked on improving qubit structure [17,18], coupling control [19], measurement and control devices [20,21], and software compilation [22–26], but also tried to add more physical qubits and qubit connections [14–16] on a single processor to increase the diversity and scale of problems that the processor can solve. However, more qubits and connections will increase the probability of frequency collisions between physical qubits on the processor, leading to an increase in gate error rate and a decrease in yield rate (an evaluation metric to estimate the probability that no frequency collisions

* Corresponding authors.
E-mail addresses: ly_yangtian@163.com (T. Yang), wlwang19888@163.com (W. Wang), wlx715@163.com (L. Wang), zhaob07@tsinghua.org.cn (B. Zhao), lc18937658696@163.com (C. Liang), zzzhengming@163.com (Z. Shan).

https://doi.org/10.1016/j.rinp.2023.106944
Received 7 March 2023; Received in revised form 21 August 2023; Accepted 2 September 2023
Available online 6 September 2023

Table 1
Comparison of design methods for superconducting quantum processor architecture.

Reference	Performance improvement	Frequency collisions reduction
[33]	×	×
[34,35]	✓	×
[32]	✓	✓

occur) [27,28]. Therefore, there exists a trade-off between performance and yield rate to ensure that the processor has strong computational power together with a low probability of frequency collisions.

Currently, most studies have focused on the improvement of the physical device, fabrication technology and frequency allocation method to solve the above problem [19,28–32]. On the other hand, the trade-off between performance and yield rate could be well solved if one can optimize the architectural design of the superconducting quantum processor. Scholars have also conducted some related studies. In Ref. [33], the authors compared the influence of different general-purpose architectures on the performance of various quantum programs and proposed the idea of designing processor architectures combined with quantum programs. However, they do not give a specific design method. In Ref. [34], the authors firstly proposed an optimization algorithm for the connections of general-purpose processor architectures to reduce the mapping overhead of different quantum programs and improve the overall performance of general-purpose processor architectures. Secondly, they proposed a method of designing processor architecture combined with quantum programs to make the quantum program better match the processor architecture. Ref. [35] integrated architecture optimization with an optimal compiler to provide performance guarantees for given quantum programs. Unfortunately, the optimization ideas of the two works [34,35] do not take into account the actual physical factors such as frequency collisions. In Ref. [32], a heuristic processor architecture design algorithm that balances performance and the yield rate was proposed based on the coupling strength matrix of the quantum program and the practical constraints of physical wiring. Compared with IBM's general-purpose design schemes, the processor architecture designed by this algorithm has better Pare-to-optimal results. However, in its algorithm, qubits with the same weight are not further discussed in the processor architecture design. There is still room for optimization in its designed architecture scheme. The comprehensive comparison of Ref. [32–35] is shown in Table 1. It can be seen that designing processor architectures based on quantum programs can improve performance. But for most previous works, they did not consider the actual physical factors such as frequency collisions in the processor architecture design, which will affect the actual effectiveness of the processor architecture. It could be further optimized in processor architecture design method that consider frequency collisions. Therefore, we should systematically build mathematical model for the design problem of superconducting quantum processor architecture that trade-off performance and frequency collisions to find better automated design methods.

Focusing on the shortcomings of the existing research, in this paper, we propose a method to optimize the superconducting quantum processor architecture design by building the mathematical model based on graph theory technology, to better solve the performance and frequency collisions (yield rate) trade-off problem. Currently, in qubit mapping, scholars have used graph theory techniques to optimize mapping algorithms in order to reduce the mapping overhead of quantum programs on processor architectures and improve processor performance [23,36–38]. However, these are not the same as our method. They improved the mapping algorithm and took the architectures as invariant. While we consider the actual physical wiring constraints and design special-purpose processor architectures for quantum programs based on graph theory to reduce the mapping overhead of existing mapping algorithm, improve the performance, lower the probability of frequency collisions and enhance the reliability of the architectures. The basic ideas of our method are shown in Fig. 1. First, we convert the improvement of performance and yield rate into the optimization of the distance of two points and the maximum degree in the processor architecture graph, respectively. Then, we design the objective function based on the converted metrics and establish all the constraints for the processor architecture design, modeling the problem of superconducting quantum processor architecture design as an optimization problem that trades off performance and frequency collisions. Lastly, based on this mathematical model, we improve the genetic algorithm and propose an automatic processor architecture design flow. The automatic design flow iteratively optimizes the processor architecture by an improved genetic algorithm (improving selection, crossover and mutation operations) and finally outputs an optimal processor architecture. To show the effectiveness of our method, we selected sixteen quantum programs with different qubit numbers and different functions for the simulation experiments of architecture comparison. In addition to the architecture schemes of our method, the compared architectures also include four general-purpose square-lattice design schemes of IBM and the design schemes of the eff-5-freq algorithm [32]. Through the comparison of simulation experiments, it is proved that the architecture schemes of our method can improve the performance and yield rate.

In summary, our contributions can be concluded as the following:

- We convert the improvement of performance and yield rate into the optimization of the distance between two points and maximum degree based on graph theory. And considering the actual physical wiring constraints, the superconducting quantum processor architecture design problem that trades off performance and frequency collisions is constructed as a mathematical model and systematically described as an optimization problem.
- Based on this mathematical model, we improve the genetic algorithm and propose an automatic processor architecture design flow. It can automate the design of superconducting quantum processor architectures that trade off performance and frequency collisions according to quantum programs.
- Comprehensive simulation results show that the architecture schemes generated by our method outperforms not only IBM's general-purpose design schemes based on the square lattice, but also the schemes of the eff-5-freq algorithm [32] in most quantum programs. Compared to the general-purpose architecture schemes, our architecture schemes in terms of performance show an average improvement of 15.61% and a minimum reduction of 21.33% in the probability of frequency collisions. Compared to the architecture schemes generated by the eff-5-freq method, in most of the selected quantum programs, our architecture schemes demonstrate an average performance improvement of 6.58% and a minimum reduction of 6.45% in the probability of frequency collisions.

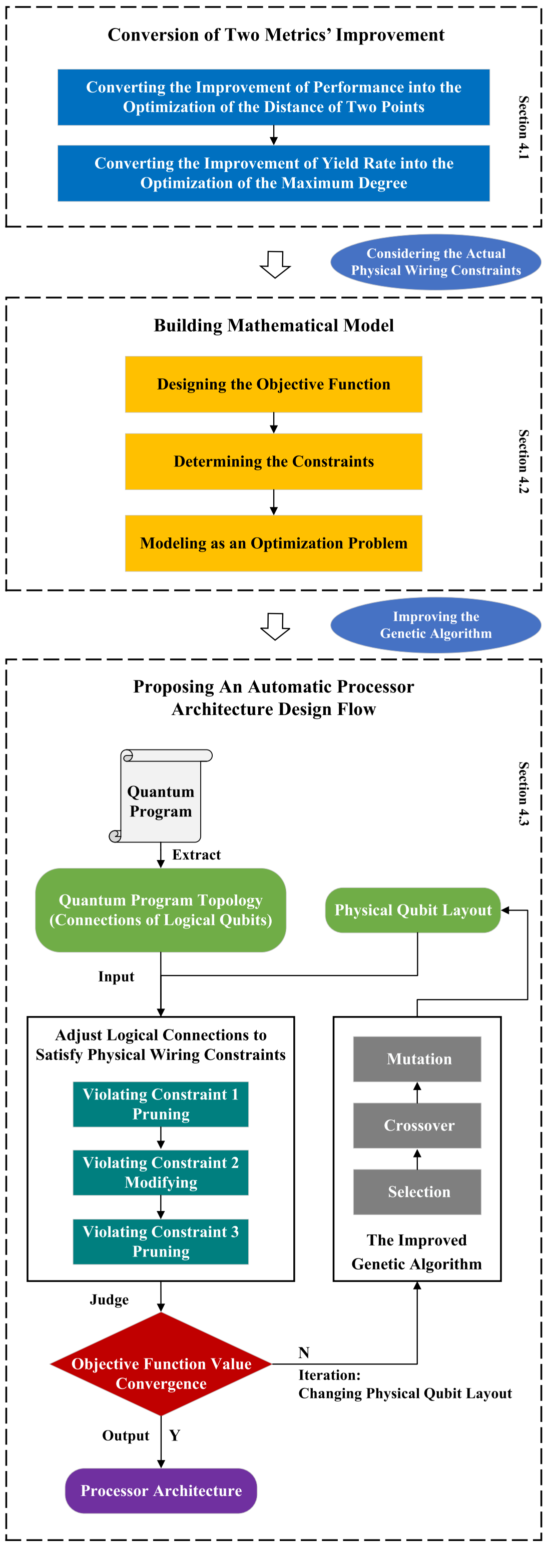
Our work can be applied to design superconducting quantum processor architectures for quantum programs with high performance and low probability of frequency collisions. Moreover, our processor architecture design method can be integrated into superconducting quantum processor design automation, which could promote development of quantum electronic design automation (QEDA) [39,40].

Preliminaries

In this section, we will briefly review the necessary QC and genetic algorithm basics related to our research.

Qubit layout and connections

The qubits on superconducting quantum processors are mostly fabricated on a 2D planar substrate. Although qubits can be placed at any position on the planar substrate, to facilitate the scalability and



Conversion of Two Metrics' Improvement
Converting the Improvement of Performance into the Optimization of the Distance of Two Points
Converting the Improvement of Yield Rate into the Optimization of the Maximum Degree
Section 4.1
Considering the Actual Physical Wiring Constraints
Building Mathematical Model
Designing the Objective Function
Determining the Constraints
Modeling as an Optimization Problem
Section 4.2
Improving the Genetic Algorithm
Proposing An Automatic Processor Architecture Design Flow
Section 4.3
Quantum Program
Extract
Quantum Program Topology (Connections of Logical Qubits)
Physical Qubit Layout
Input
Adjust Logical Connections to Satisfy Physical Wiring Constraints
Violating Constraint 1 Pruning
Violating Constraint 2 Modifying
Violating Constraint 3 Pruning
Mutation
Crossover
Selection
The Improved Genetic Algorithm
Judge
Objective Function Value Convergence
N
Iteration: Changing Physical Qubit Layout
Output
Y
Processor Architecture

Fig. 1. Overview of basic ideas of the method.

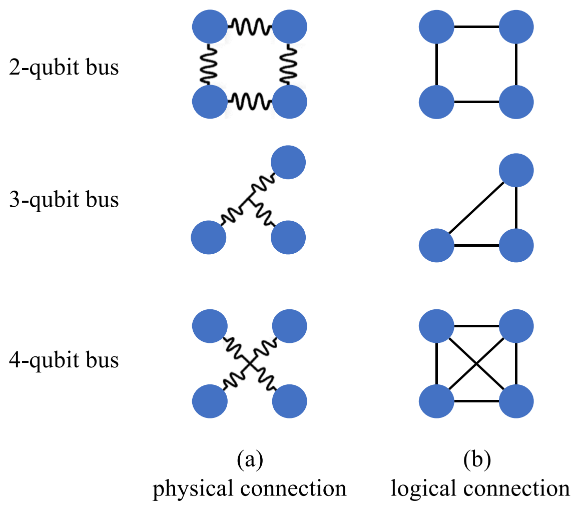


Fig. 2. Three types of qubit buses.

feasibility of qubits, researchers mainly choose to design the processor architecture based on square lattice topology [14–16]. As for the qubit connection, it has been proved in Ref. [41] that any multi-qubit gate in a quantum program can be decomposed into a series of single-qubit gates and two-qubit CNOT gates. Therefore, this paper mainly considers two-qubit connections. One of the most widely used approaches is to couple two qubits with qubit buses (*i.e.*, resonators). Fig. 2 shows three types of buses, where columns (a) and (b) show the physical connection structure and the logical connection structure, respectively. The first row is a 2-qubit bus, which connects two physical qubits. The second and third rows are the 3-qubit bus and the 4-qubit bus, which connect three and four physical qubits in a square lattice, respectively. From Fig. 2, it is obvious that the 4-qubit bus can support two-qubit gates not only for four qubit pairs on the edges but also for diagonal two-qubit pairs. In this paper, we focus on superconducting quantum processors based on fixed-frequency transmon qubits and all-microwave cross-resonance two-qubit gates [42,43] and describe the processor architecture using logical connections.

Qubit mapping

If a quantum program (*i.e.*, quantum circuit) is executed on an actual superconducting quantum processor, the logical qubits must be mapped to the physical qubits of the processor. In the quantum circuit, users can perform a two-qubit gate operation for any two defined logical qubits and do not need to consider any physical constraints. However, in the actual superconducting quantum processor, it is quite difficult to perform a two-qubit gate operation directly between any two qubits due to limited qubit connections. A common way to deal with this problem is to add the SWAP gate operations [23–26]. For example, Fig. 3(a) is a quantum circuit, which contains five CNOT gate operations (g_1–g_5). Fig. 3(b) is the logical architecture of a superconducting quantum processor (vertices represent qubits and edges represent logical connections), which is now used to execute the quantum program in Fig. 3(a). By mapping each logical qubit q_i directly to the physical qubit Q_i, the first four CNOT gate operations can be supported. In contrast, the CNOT gate operation g_5 cannot be supported because the connection between Q_2 and Q_3 does not exist. For this, an additional SWAP gate (the red operation in Fig. 3(c)) can be added before g_5 to transfer the quantum state in Q_2 to Q_0 and the CNOT gate operation of q_2 and q_3 can be realized using the connection between Q_0 and Q_3. However, additional gate operations increase the execution time of quantum programs and reduce the fidelity of the whole execution process. Therefore, if more qubit connections

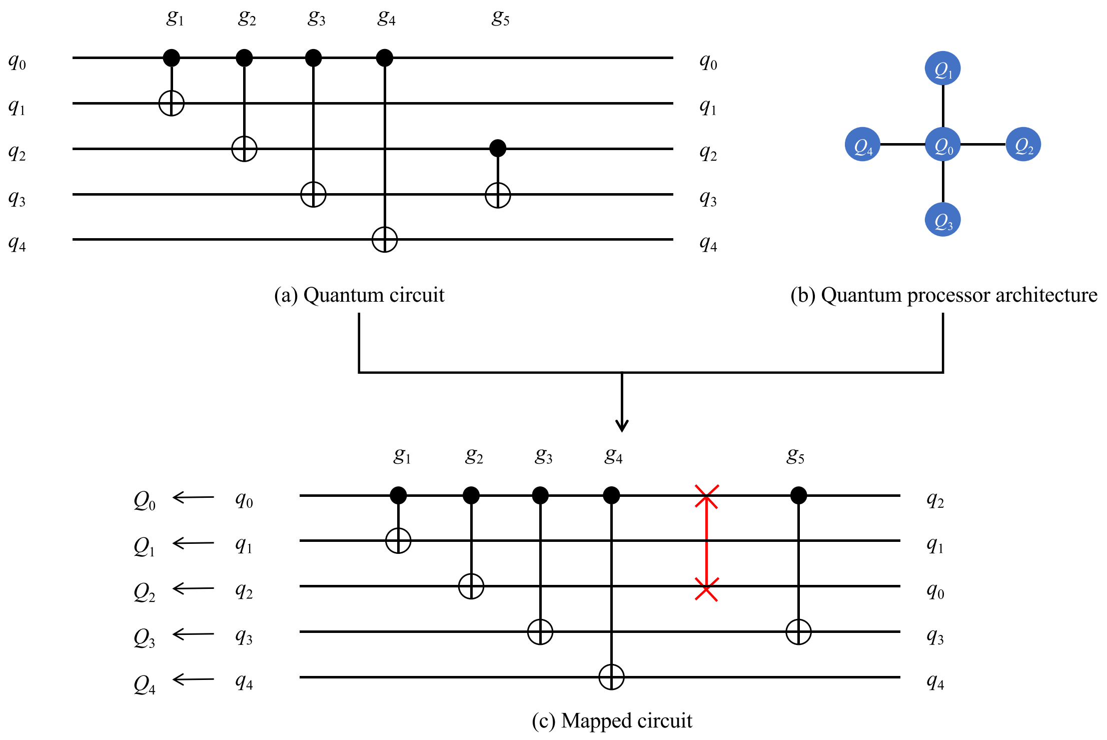


Fig. 3. Example of qubit mapping.

Table 2
Frequency Collision Conditions ($\delta = -340$ MHz).

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

are added to the processor architecture, it will help to improve the overall performance of the processor. However, more connections could increase the probability of other problems which we will explain in the next subsection.

Frequency collisions and yield rate

Due to the limitation of current fabrication technologies, the frequency of the qubit will have some uncontrollable deviations during the fabrication process. For example, if the designed frequency of the qubit is f, then the actual frequency after fabrication could be $f + N(0,\sigma_f)$. $N(0,\sigma_f)$ is a fabrication error conforming to Gaussian distribution, and the parameter σ_f is the standard deviation. Due to the uncertainty variation of qubit frequencies after fabrication, frequency collisions occur when two or three qubits are connected and their frequencies satisfy some specific conditions. Table 2 summarizes seven qubit frequency collision conditions proposed by IBM [44], where the second column is the mathematical expression, the third column is the threshold that the expression must satisfy with the anharmonicity of qubit $\delta = -340$ MHz. Conditions 1–4 are used to judge whether a frequency collision occurs when two qubits j and k are connected and conditions 5–7 are used to judge the three-qubit case of qubits i, k connected to qubit j.

The parameter 'yield rate' is proposed to estimate the probability that no frequency collisions occurs [27,28]. The detailed calculation is as follows: First, allocate a fixed frequency to each qubit in a given processor architecture as the design frequency f, and then repeat the Monte Carlo simulation more than 1000 times. In each Monte Carlo simulation, $f + N(0,\sigma_f)$ is simulated as the frequency after fabrication, and it is judged whether $f + N(0,\sigma_f)$ satisfies any of the conditions in Table 2. If the judgment condition is satisfied, then the frequency collision is considered to occur. The yield rate is defined as the ratio between the number of times that no collisions occur and the total number of simulation runs. That is, the higher the yield rate, the lower the probability of frequency collisions. In this paper, we make use of the frequency allocation algorithm of Ref. [32] to calculate the yield rate, and the optional frequency range is taken from the IBM's 5-frequency scheme [27].

Genetic algorithm

Genetic Algorithm is a random global search optimization algorithm, which is designed and proposed according to the evolution law of organisms in nature [45]. It simulates the selection, crossover, variation and other phenomena of heredity and evolution in nature, and converts the process of solving problem into the evolutionary process similar to chromosome (individual). Here, a chromosome (individual) is encoded using multiple genes with each gene being a part of the solution. The algorithm starts from the initial population containing multiple random individuals (candidate solutions) and performs genetic operations such as selection, crossover and mutation. After a number of iterations, it finally converges to a group of individuals which are the most suitable for the environment (problem), thus obtaining a high-quality solution of the problem.

Motivation

In the previous section, we introduced the background knowledge involved in this paper. In this section, we will compare several processor architecture examples to illustrate the motivation of our research.

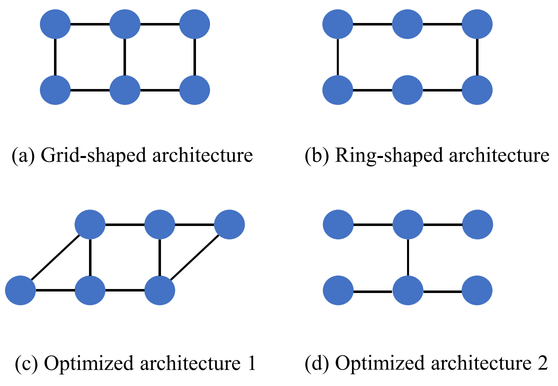

(a) Grid-shaped architecture
(b) Ring-shaped architecture
(c) Optimized architecture 1
(d) Optimized architecture 2

Fig. 4. Four different square lattice processor architectures.

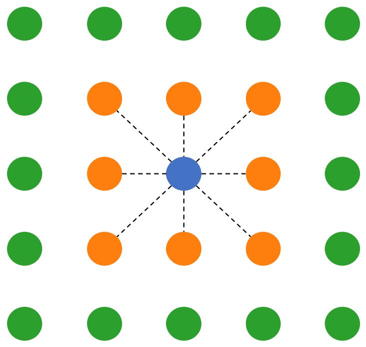


Fig. 5. Example of **Constraint1**.

We select four different square lattice processor architectures (see Fig. 4) and compare the execution performance of two 6-qubit quantum programs (QAOA and Deutsch–Jozsa Algorithm) as well as the yield rates on four architectures. The comparison results are shown in Table 3. The total post-mapping gate count is used as the performance metric [23–26,32]. A lower gate count indicates that the program can complete the computation with a shorter execution time and a lower error probability, meaning higher performance. The yield rate is calculated according to the methods introduced in Section “Frequency Collisions and Yield Rate”. Fig. 4(a) and (b) show grid-shaped and ring-shaped general-purpose architectures, respectively. Fig. 4(c) and (d) are optimized architectures. As can be seen from Table 3, in terms of performance in 6-qubit QAOA, the optimized architecture shown in Fig. 4(c) outperforms the general-purpose architectures shown in Fig. 4(a) and (b) by 13.1% and 16.9%, respectively. As for the yield rate, the result obtained in Fig. 4(c) is only 4.4% different from Fig. 4(b). The optimized architecture shown in Fig. 4(d) has better performance in 6-qubit Deutsch–Jozsa Algorithm and a better yield rate than those in Fig. 4(a) and (b). That is to say, the processor architecture optimization has great importance of improving the performance and yield rate of a superconducting quantum processor. However, Fig. 4(c) and (d) have differences in performance of different programs, as well as in yield rate. Therefore, to achieve higher performance and yield rate, the processor architecture should be optimized for running specific quantum programs.

In addition, Ref. [32] suggests that before the realization of the universal quantum computer, it is more likely to adopt an array of QC accelerators customized for quantum programs (*i.e.*, Special-Purpose Quantum Processor). At present, researchers have begun to optimize the processor architectures closely compatible with quantum programs [32,34]. In particular, Ref. [32] proposed a processor architecture design method that balances the performance and yield rate

Table 3
Performance and yield rate comparison of four processor architectures.

Processor architecture	Total Post-Mapping gate count		Yield rate
	QAOA	Deutsch–Jozsa Algorithm	
Fig. 4(a)	153	22	0.2788
Fig. 4(b)	160	32	0.3227
Fig. 4(c)	133	28	0.3085
Fig. 4(d)	165	22	0.5027

according to quantum programs. Different from Ref. [32], we use a graph theory based technique to solve the performance and yield rate trade-off problem. Specifically, we improve the performance and yield rate by adjusting the distance of two points and the maximum degree in the processor architecture graph. In most simulation experiments of quantum programs, our architecture schemes perform better than the eff-5-freq’s [32] architecture schemes.

Our method and simulation experiments will be explained in the next sections in detail.

Method

In this section, we will introduce the main idea and specific implementation of the superconducting quantum processor architecture design method. First, we convert the improvement of performance and yield rate into the optimization of the distance between two points and maximum degree based on graph theory. Then, we construct a mathematical model of the optimization problem with actual physical wiring constraints. Finally, based on this mathematical model, we improve the genetic algorithm and propose an automatic processor architecture design flow with the joint optimization of performance and yield rate. This will be explained in detail below.

Conversion of two metrics’ improvement

In this subsection, we will explain the conversion corresponding to each metric improvement based on graph theory.

Conversion of performance improvement

Since our work is based on the square lattice structure for laying out qubits and the actual physical wiring with 2-qubit bus, 3-qubit bus, and 4-qubit bus, we first introduce the relevant physical wiring constraints and then illustrate the coping strategy and the performance-improved conversion with a specific case.

Constraint1 Limited by the square lattice, qubits can only be connected to the neighboring qubits. As shown in Fig. 5, the blue qubit can only be connected with the orange qubits.

Constraint2 When two diagonals within the square lattice are connected, a 4-qubit bus is used to connect four qubits.

Constraint3 There cannot be two physical connections between two shared qubits in adjacent square lattices [32]. The reason is shown in Fig. 6. Taking Fig. 6(a) as an example, when both adjacent square lattices adopt 4-qubit bus, there are two physical connections between qubit i and qubit j. When one of the connections is used, the other will bring unexpected effects [32]. Similarly, when the 4-qubit bus is adjacent to the 3-qubit bus or the 3-qubit bus is adjacent to the 3-qubit bus, it may also make the case that there are two physical connections of qubit i and qubit j occur, as shown in Fig. 6(b) and (c).

In the following, a specific quantum program is used to illustrate a processor architecture wiring strategy that satisfies the physical constraints and the optimized conversion of performance. First, we extract the connections of logical qubits from the quantum circuit describing the quantum program, and represent it with an upper triangular matrix, denoted by $\mathbf{A}$, see Fig. 7(a), Fig. 7(b). In this paper, it is assumed that the gate operations in the quantum program have been decomposed

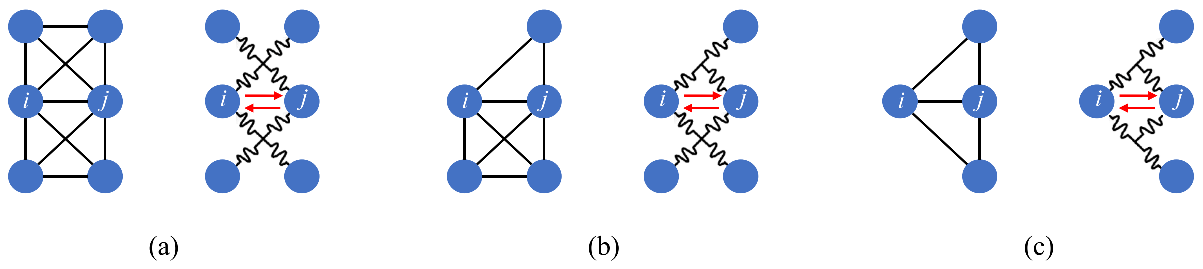
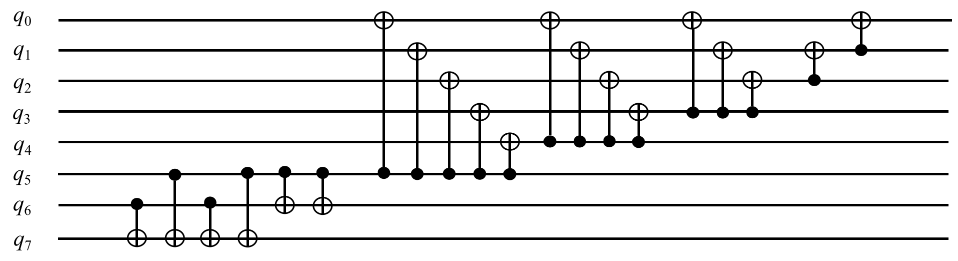


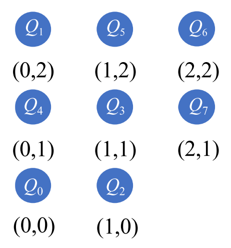
Fig. 6. Example of qubit mapping.



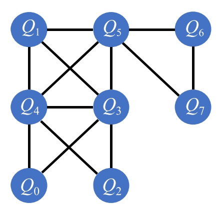
(a) Quantum circuit

$$\begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 2 & & & 0 & 1 & 1 & 1 & 0 & 0 \\ 3 & & & & 0 & 1 & 1 & 0 & 0 \\ 4 & & & & & 0 & 1 & 0 & 0 \\ 5 & & & & & & 0 & 1 & 1 \\ 6 & & & & & & & 0 & 1 \\ 7 & & & & & & & & 0 \end{array}$$

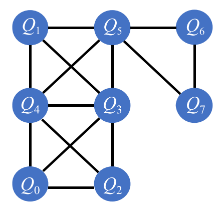
(b) Quantum program connection matrix A



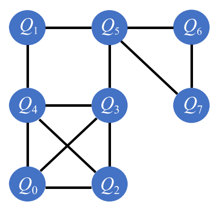
(c) Physical qubits logic layout



(d) Constraint 1 pruning



(e) Constraint 2 modification



(f) Constraint 3 pruning

$$\begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 0 & 0 & \color{red}{0} & \color{blue}{1} & 1 & 1 & \color{red}{0} & 0 & 0 \\ 1 & & 0 & \color{red}{0} & \color{green}{0} & 1 & 1 & 0 & 0 \\ 2 & & & 0 & 1 & 1 & \color{red}{0} & 0 & 0 \\ 3 & & & & 0 & 1 & 1 & 0 & 0 \\ 4 & & & & & 0 & \color{green}{0} & 0 & 0 \\ 5 & & & & & & 0 & 1 & 1 \\ 6 & & & & & & & 0 & 1 \\ 7 & & & & & & & & 0 \end{array}$$

(g) pruning-modified connection matrix B

Fig. 7. Example of generating superconducting quantum processor architecture.

into a series of single-qubit gates and two-qubit CNOT gates, and the connections are only related to the CNOT gates, so Fig. 7(a) only shows the two-qubit gates. **A**(i, j)=1 represents that there exists a two-qubit gate between the logical qubit q_i, and q_j. Then, the logical layout of physical qubits is randomly generated. That is, a plane coordinate $p_i(x_i, y_i)$ is allocated to each physical qubit Q_i (for convenience, the coordinates are taken in a range of natural numbers), as shown in Fig. 7(c). Since the processor architecture is designed according to the quantum program, q_i corresponds to Q_i one-to-one. Finally, wirings are formed according to three constraints. The specific strategy is as follows:

1. If **A**(i, j)=1, and Q_i, Q_j satisfy **Constraint1**, then connect Q_i and Q_j. If **A**(i, j)=1, and Q_i, Q_j violate **Constraint1**, then Q_i and Q_j are not connected. The corresponding pseudo code is shown in Algorithm 1;
2. If two diagonals in a square lattice are connected, use a 4-qubit bus in the physical wiring, and modify the logical connection in the square lattice. That is, connect the unconnected edges in the square lattice to satisfy **Constraint2**. The corresponding pseudo code is shown in Algorithm 2;
3. If two adjacent square lattices violate **Constraint3**, the diagonal in one of the square lattices will be pruned. The detailed strategy determining which square lattice should be selected for diagonal pruning will be explained in detail in Section "Fitness Function".

The three steps of the strategy are presented in Fig. 7(d), (e), and (f), respectively. The pruning-modified connection matrix is denoted by **B**, as shown in Fig. 7(g), where the red number 0 indicates the set of edges pruned for **Constr-aint1**, the blue number 1 indicates the set of edges modified for **Constraint2**, and the green number 0 indicates the set of edges pruned for **Constraint3**. Let us take n qubits as the vertex set **V**, the connections stored in **B** as the edge set **E**, and store the logical coordinates of each qubit with the coordinate vector **P**. In this way, the processor architecture graph **G**(**V**, **E**, **P**) is generated.

Algorithm 1 Adjusting **Constraint 1** Logical Connection Algorithm

Input: quantum program connection matrix **A**, physical qubits logic layout **Q**, number of qubits n

Output: temporary connection matrix $\mathbf{C}_{\text{temp}}$ that satisfies **Constraint 1**

```
 1: Initialization:$\mathbf{C}_{\text{temp}} \leftarrow \mathbf{A}$ // Initialize the $\mathbf{C}_{\text{temp}}$
 2: for $i \leftarrow 1$ to $n$-1 do
 3:   for $j \leftarrow i+1$ to $n$ do
 4:     if $\mathbf{A}(i,j) == 1$ and $\mathbf{Q}_i$, $\mathbf{Q}_j$ are not adjacent then
 5:       $\mathbf{C}_{\text{temp}}(i,j) = 0$ // Violating Constraint 1 (Pruning)
 6:     else if $\mathbf{A}(i,j) == 1$ and $\mathbf{Q}_i$, $\mathbf{Q}_j$ are adjacent then
 7:       $\mathbf{C}_{\text{temp}}(i,j) = 1$
 8:     end if
 9:   end for
10: end for
```

Algorithm 2 Adjusting **Constraint 2** Logical Connection Algorithm

Input: temporary connection matrix $\mathbf{C}_{\text{temp}}$ that satisfies **Constraint 1**, physical qubits logic layout **Q**, the number of square lattices K

Output: temporary connection matrix $\mathbf{C}_{\text{temp}}$ that satisfies **Constraint 1** and **Constraint 2**

```
 1: for $i \leftarrow 1$ to $K$ do
 2:   if $\mathbf{C}_{\text{temp}}(\text{squarelattice}(i)_a, \text{squarelattice}(i)_c) == 1$ and
      $\mathbf{C}_{\text{temp}}(\text{squarelattice}(i)_b, \text{squarelattice}(i)_d) == 1$ then
 3:     /*a,b,c,d are the four vertices on the square lattice($i$)*/
 4:     /*Using the 4-qubit bus in the square lattice($i$) and modifying
        the logical connection*/
 5:     $\mathbf{C}_{\text{temp}}(\text{squarelattice}(i)_a, \text{squarelattice}(i)_b) = 1$
 6:     $\mathbf{C}_{\text{temp}}(\text{squarelattice}(i)_b, \text{squarelattice}(i)_c) = 1$
 7:     $\mathbf{C}_{\text{temp}}(\text{squarelattice}(i)_c, \text{squarelattice}(i)_d) = 1$
 8:     $\mathbf{C}_{\text{temp}}(\text{squarelattice}(i)_d, \text{squarelattice}(i)_a) = 1$
 9:   end if
10: end for
```

The relevant background in Section "Qubit Mapping" shows that if there is a direct connection between the physical qubits corresponding to two logical qubits, a two-qubit gate operation can be performed directly. Otherwise, one can use SWAP gates to solve this problem. And in the processor architecture graph **G**, the number of swap gates is proportional to the distance $d(i,j)$ between two qubits (vertices) i and j (the length of the shortest path between two vertices, for the unweighted graph [46]). When $d(i,j) = 1$, it represents two qubits directly connected. The number of additional SWAP gates required for performing two-qubit gate operations is 0. When $d(i,j) > 1$, the number of additional SWAP gates required for performing two-qubit gate operations is $d(i,j)-1$ [38]. That is, as $d(i,j)$ increases, more SWAP gates are needed, resulting in an increase in the total post-mapping gate count and thus a decrease in performance. Therefore, we convert the performance improvement to the case where a two-qubit gate operation is required, but there is no direct connection between the corresponding qubits in **G**. Take the sum of distances in such cases as the optimization objective:

$$\sum d(i,j)\ ,\ \forall i,j \in \mathbf{V}(\mathbf{G}) \wedge \mathbf{A}(i,j)= 1 \wedge \mathbf{B}(i,j)= 0 \tag{1}$$

If $\sum d(i,j)$ is minimized, SWAP gates in such cases and total post-mapping gate count are reduced, thereby improving the overall performance.

Conversion of yield rate improvement

Denser qubit connections in the processor architecture can improve the performance but may also increase the probability of frequency collisions and reduce the yield rate. For this reason, the number of direct qubit connections should be optimized to improve the yield rate. Based on graph theory, we convert the improvement of yield rate to the optimization of vertex degree (the number of vertices adjacent to the vertex i [46]) in **G**. That is, the maximum degree (the largest vertex degree of **G** [46]) of the qubit is used as the optimization objective:

$$\Delta(\mathbf{G}) = max\{\deg_{\mathbf{G}}(i) | i \in \mathbf{V}(\mathbf{G})\} \tag{2}$$

Here, the average degree is not chosen as the optimization objective because it does not necessarily reflect the reasonable number of connections for each qubit in **G**. For example, in a star architecture, although the average degree is small, the central qubit has a higher number of connections. This increases the probability of frequency collisions occurring at the central qubit and affects the yield rate. While $\Delta(\mathbf{G})$ is the maximum degree (the largest vertex degree of **G**). When $\Delta(\mathbf{G})$ decreases, it indicates that the degree of each vertex in **G** (*i.e.*, the connections for each qubit) is decreasing. As the number of qubit connections decreases, it also reduces the probability of frequency collisions occurring for each qubit. Therefore, the yield rate is improved.

Mathematical model

According to the descriptions in Section "Conversion of Two Metrics' Improvement", the optimization problem can be explained as: by extracting the connection matrix **A** of the quantum program, continuously iterating the coordinate vector **P** of qubits, and minimizing $\sum d(i,j)$ and $\Delta(\mathbf{G})$ under the constraints of the actual physical wiring, one can obtain a processor architecture with better performance and yield rate. The optimization problem is modeled as Eq. (3),

$$\begin{aligned}
&\min_{\mathbf{P},\mathbf{A}} f(\mathbf{G})\\
&f(\mathbf{G}) = \begin{cases} \alpha \times \sum d(i,j) + \beta \times \lambda \times \Delta(\mathbf{G})\ , \\ \qquad \forall i,j \in \mathbf{V}(\mathbf{G}) \wedge \mathbf{A}(i,j)= 1 \wedge \mathbf{B}(i,j)= 0 \wedge \mathbf{G} \text{ is connected} \\ \infty\ ,\ \mathbf{G} \text{ is disconnected} \end{cases}\\
&\mathbf{P} = [p_0, p_1, \ldots, p_i, \ldots, p_{n-1}], \forall n \in \mathbf{N}^*\\
&p_i = (x_i, y_i), \forall x_i, y_i \in \mathbf{N}\\
&s.t.\\
&C_1 : 0 \le \alpha, \beta \le 1\\
&C_2 : \alpha + \beta = 1\\
&C_3 : \mathbf{G} \subsetneqq \mathbf{S} \cap \mathbf{G} \subsetneqq \mathbf{T} \cap \mathbf{G} \subsetneqq \mathbf{U}
\end{aligned} \tag{3}$$

where $f(\mathbf{G})$ is the objective function, and the optimization of performance and yield rate is defined as an overall objective function by the multi-objective linear weighted sum method, α and β are weight coefficients, representing the level of concern to performance and yield rate in the optimization process, whose values range in C_1 and C_2. In this paper, we give equal weights to performance and yield rate. That is, both α and β are set to be 0.5. λ is an order of magnitude adjustment factor, which ensures that $\sum d(i,j)$ and $\Delta(\mathbf{G})$ have the same order of magnitude and can be optimized together. λ takes different values depending on different quantum programs. In this paper, the value range of λ is 1~6. Note that during pruning, **G** may be disconnected due to different coordinate vectors, which affects the calculation of f. Therefore, when **G** is a disconnected graph, the value of f is set to be positive infinity (*i.e.*, the maximum value). **S** is the set of processor architectures satisfying **constraint1**, **T** is the set of processor architectures satisfying **constraint2**, **U** is the set of chip architectures satisfying **constraint3**, and C_3 indicates that **G** should satisfy the actual physical wiring constraints.

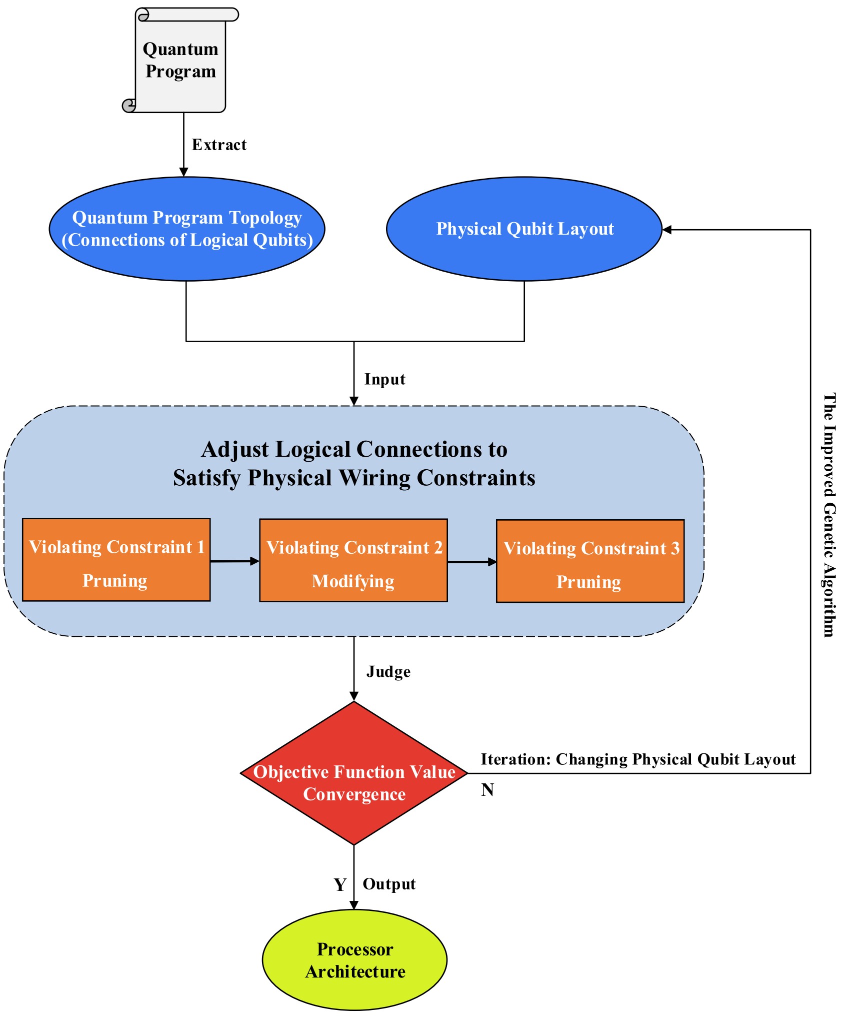

Quantum Program
Extract
Quantum Program Topology (Connections of Logical Qubits)
Physical Qubit Layout
Input
Adjust Logical Connections to Satisfy Physical Wiring Constraints
Violating Constraint 1 Pruning
Violating Constraint 2 Modifying
Violating Constraint 3 Pruning
The Improved Genetic Algorithm
Judge
Objective Function Value Convergence
Iteration: Changing Physical Qubit Layout
N
Y
Output
Processor Architecture

Fig. 8. The proposed automatic processor architecture design flow.

Automatic processor architecture design flow

Based on the mathematical model described in Section "Mathematical Model", we improve the genetic algorithm and propose an automatic processor architecture design flow. The automatic processor architecture design flow is shown in Fig. 8. According to the input quantum program and initial qubit layout, the automatic design flow minimizes the objective function value by continuously iterating the coordinate vector **P** with the improved genetic algorithm. Finally, the automatic design flow generates a processor architecture with better performance and yield rate while satisfying the physical wiring constraints.

Below, we will introduce the steps of the improved genetic algorithm and the automatic processor architecture design flow in detail.

Encoding

The logical coordinates of n qubits (*i.e.*, the coordinate vector **P**) are represented as an individual, and the coordinate of a qubit is the gene of the individual. To accelerate the convergence process of the algorithm, we create the search space based on the number of qubits. For example, when $0 < n \leq 4$, the algorithm searches in a 2×2 coordinate matrix; when $4 < n \leq 9$, the algorithm searches in a 3×3 coordinate matrix, and so on. Moreover, during population initialization, the algorithm does not allow repeated coordinates between qubits to avoid the existence of unrealistic individuals.

Fitness function

Genetic Algorithm uses fitness to evaluate an individual. The higher fitness value, the better individual. However, in our optimization problem, the smaller value of the objective function $f(\mathbf{G})$, the better performance and yield rate of the processor architecture represented by **G**. In order to solve our optimization problem using the genetic algorithm, we need to combine the fitness function with $f(\mathbf{G})$. Therefore, we use -$f(\mathbf{G})$ as the fitness function,

$$Fit(\mathbf{G}) = -f(\mathbf{G}) \tag{4}$$

The smaller value of the objective function, the larger value of individual fitness. The algorithm iteratively searches for individuals with larger fitness value to find processor architecture with better performance and yield rate. Here, according to the experimental situation, the objective function value under the disconnected graph is set to be 200. In addition, the pruning scheme of **Constraint3** in Section "Conversion of Two Metrics' Improvement" can be selected according to the individual fitness value. The specific strategy is as follows:

The square lattices in the search space are divided into two categories: one is the orange square lattice, denoted by "Sequence 1", and the other is the green square lattice, denoted by "Sequence 2", as shown in Fig. 9. It can be seen that there are two pruning schemes to satisfy **Constraint3**:

1. Keep the diagonals of square lattice in "Sequence 1" and prune the diagonals of square lattice in "Sequence 2";

Algorithm 3 Adjusting **Constraint 3** Logical Connection Algorithm

```
Input: coordinate vector $\mathbf{P}$, the number of square lattices $K$, temporary connection matrix $\mathbf{C}_{temp}$ that satisfies Constraint 1 and Constraint 2
Output: pruning-modified connection matrix $\mathbf{B}$, processor architecture graph $\mathbf{G}(\mathbf{V}, \mathbf{E}, \mathbf{P})$
 1: Initialization:$\mathbf{C}_{\prime 1'} \leftarrow \mathbf{C}_{temp}$, $\mathbf{C}_{\prime 2'} \leftarrow \mathbf{C}_{temp}$ // Initialize the "Sequence 1" and "Sequence 2" schemes
 2: for $i \leftarrow 1$ to $K$ do
 3:   if square lattice($i$) has Constraint 3 conflict with adjacent square lattices and square lattice($i$)$\in$ "Sequence 1" then
 4:     /*Keep the diagonals of "Sequence 1" square lattice, prune the diagonals of "Sequence 2" square lattice*/
 5:     prune the diagonals of adjacent square lattices in $\mathbf{C}_{\prime 1'}$
 6:   else if square lattice($i$) has Constraint 3 conflict with adjacent square lattices and square lattice($i$)$\in$ "Sequence 2" then
 7:     /*Keep the diagonals of "Sequence 2" square lattices, prune the diagonals of "Sequence 1" square lattices*/
 8:     prune the diagonals of adjacent square lattices in $\mathbf{C}_{\prime 2'}$
 9:   end if
10: end for
11: if $Fit(\mathbf{G}_{\prime 1'}(\mathbf{V}, \mathbf{E}_{\prime 1'}, \mathbf{P})) >= Fit(\mathbf{G}_{\prime 2'}(\mathbf{V}, \mathbf{E}_{\prime 2'}, \mathbf{P}))$ then
12:   $\mathbf{B} = \mathbf{C}_{\prime 1'}$
13:   $\mathbf{G}(\mathbf{V}, \mathbf{E}, \mathbf{P}) = \mathbf{G}_{\prime 1'}(\mathbf{V}, \mathbf{E}_{\prime 1'}, \mathbf{P})$
14: else
15:   $\mathbf{B} = \mathbf{C}_{\prime 2'}$
16:   $\mathbf{G}(\mathbf{V}, \mathbf{E}, \mathbf{P}) = \mathbf{G}_{\prime 2'}(\mathbf{V}, \mathbf{E}_{\prime 2'}, \mathbf{P})$
17: end if
```

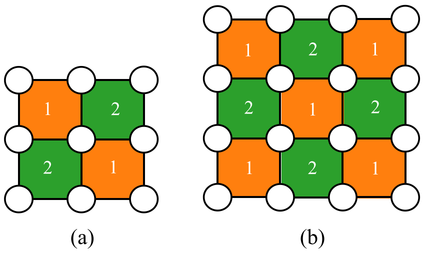


Fig. 9. Square lattice classification examples of different search spaces, '1' and '2' represent "Sequence 1" and "Sequence 2", respectively.(a) is 3×3 search space,(b) is 4×4 search space.

2. Keep the diagonals of the square lattice in "Sequence 2" and prune the diagonals of the square lattice in "Sequence 1".

Finally, the fitness value is calculated according to different pruning schemes, and the pruning scheme with the largest fitness value is selected. The corresponding pseudo code is shown in Algorithm 3.

Selection

To accelerate the optimization problem, we use the binary tournament selection method. The main idea is as follows: First, select two individuals from the offspring each time (sampling with replacement). Then compare their fitness values and put the individual with larger fitness into the next generation until the next generation reaches the current population size.

In addition, to prevent the loss of the optimal individual of the current population in the next generation, which causes the genetic algorithm to fail to converge globally, we need to save the best individual so far. For this, we further adopt the elite selection strategy. That is, we compare the fitness value of the best individual in the current population to that of the best individual in the next generation. If the former value is larger, the best individual will be copied directly to the next generation without crossover and mutation. There are two approaches to realize elite selection strategy: one is to directly add the best current individual to the next generation but expand the population size; the other is to replace the individual with the smallest fitness value in the next generation with the best current individual to maintain the population size. To prevent the population size from increasing, which will affect the calculation and convergence speed of the algorithm, we adopt the second approach. The corresponding pseudo code is shown in Algorithm 4.

Algorithm 4 Selection Algorithm

```
Input: current population current_pop, population size pop_size, offspring children
Output: next generation next_pop
 1: Initialization:next_pop $\leftarrow \emptyset$ // Initial next generation is an empty set
 2: /*Binary Tournament Selection*/
 3: while length(next_pop) != pop_size do
 4:   $i$ , $j$ = random(range(1,length(children)),2) // Two individuals were randomly selected
 5:   if $Fit(\mathbf{G}_{children[i]}) >= Fit(\mathbf{G}_{children[j]})$ then
 6:     next_pop.append($children[i]$)
 7:   else
 8:     next_pop.append($children[j]$)
 9:   end if
10: end while
11: /*Elite Tournament Selection*/
12: $h$ = find_max_fit_idx(current_pop) // Find the best current individual
13: $l$ = find_max_fit_idx(next_pop) // Find the best individual in next generation
14: if $Fit(\mathbf{G}_{current\_pop[h]}) >=$ Fit $(Fit(\mathbf{G}_{next\_pop[l]})$ then
15:   $M$ = find_min_fit_idx(next_pop) // Find the worst individual in next generation
16:   next_pop[$m$]= current_pop[$h$]
17: end if
```

Crossover

We adopt the single-point crossover method and judge the child after crossover. If there are repeated coordinates between qubits in the child, the crossover is determined to fail, and the parent is regarded as the child. There are two cases:

1. If only one child fails the crossover, a parent is randomly selected to replace the failed child;
2. If both children fail the crossover, both parents are used as children.

The corresponding pseudo code is shown in Algorithm 5.

Mutation

The mutation operation is divided into two categories:

1. If the search space does not contain unoccupied points, the qubit's coordinate of the mutation point is exchanged with any

Algorithm 5 Crossover Algorithm

```
Input: parents $parent_1$, $parent_2$, number of qubits $n$
Output: children $child_1$, $child_2$
1: Initialization:$child_1 \leftarrow \emptyset$, $child_2 \leftarrow \emptyset$
2: $cross\_point$ = random(range(1, $n$), 1) //Select crossover point
3: for $i \leftarrow 1$ to $n$ do
4:   if $i < cross\_point$ then
5:     $child_1$.append($parent_1[i]$)
6:     $child_2$.append($parent_2[i]$)
7:   else
8:     $child_1$.append($parent_2[i]$)
9:     $child_2$.append($parent_1[i]$)
10:  end if
11: end for
12: /*Judge repeated coordinates*/
13: $num$ = 0 //Record the number of crossover failures
14: if $child_1$ has repeated coordinates then
15:   $num$ += 1
16:   $child_1$ = pick one of $parent_1$ and $parent_2$ at random
17: end if
18: if $child_2$ has repeated coordinates then
19:   $num$ += 1
20:   if $num$ == 1 then
21:     $child_2$ = pick one of $parent_1$ and $parent_2$ at random
22:   else if $num$ == 2 then
23:     $child_2$ = parent is not equal to $child_1$
24:   end if
25: end if
```

other qubit's coordinate. That is, the positions of the two qubits are exchanged, as shown in Fig. 10(a);

2. If the search space contains unoccupied points, move the qubit of the mutation point to any unoccupied point with half probability. Otherwise, the qubit position is exchanged as in category 1, as shown in Fig. 10(b).

The corresponding pseudo code is shown in Algorithm 6.

Algorithm 6 Mutation Algorithm

```
Input: individual $\mathbf{P}_m$, index of the mutated qubit $mutate\_idx$
Output: individual after mutation $\mathbf{P}_m$
1: /*Find unoccupied points in the search space*/
2: $unoccupied\_set$ = find_unoccupied_point()
3: if length($unoccupied\_set$) == 0 then
4:   /* Mutation Type 1 */
5:   $i$ = choose a qubit index from $\mathbf{P}_m$ except for $mutate\_idx$
6:   exchange($\mathbf{P}_m[mutate\_idx]$ ,$\mathbf{P}_m[i]$ )
7: else
8:   /* Mutation Type 2 */
9:   $Sel$ = random(range(0,1),1) // Generate a random number between 0 and 1
10:  if $Sel >= 0.5$ then
11:    $j$ = random(range(1,length($unoccupied\_set$)),1)
12:    $\mathbf{P}_m[mutate\_idx]$ = $unoccupied\_set[j]$
13:  else
14:    $i$ = choose a qubit index from $\mathbf{P}_m$ except for $mutate\_idx$
15:    exchange($\mathbf{P}_m[mutate\_idx]$ ,$\mathbf{P}_m[i]$ )
16:  end if
17: end if
```

Based on the above improved genetic algorithm, we propose an automatic processor architecture design flow. First, we take the quantum program topology and randomly generated physical qubit layout as input parameters. Then, we adjust the corresponding logical connections to satisfy the physical wiring constraints. Finally, we judge the design scheme according to the objective function of the optimization problem. Once the objective function value converges, the processor architecture will be output as the final design scheme. Otherwise, we update the physical qubit layout **P** according to the rules of the improved genetic algorithm for the next iteration (see Fig. 8).

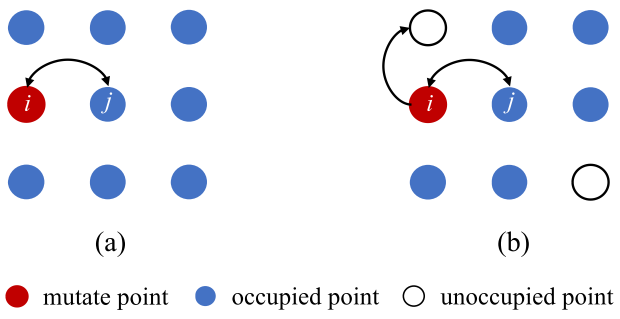


Fig. 10. Mutation operation classification.

Simulation

To evaluate each processor architecture scheme, we compute performance and yield rate metrics by using computational simulation (see Section "Parameter Setting" for details). Computational simulation is the process of performing simulated calculations through computational programs, and it has been widely applied in various fields such as physics [47,48], engineering [49–52], medical [53–55], and mathematics [56]. Compared to analytical and experimental study, computational simulation can bring several advantages. First, computational simulation has a lower economic cost [57–59]. If one analyzes processor architectures from experimental research, processors need to be fabricated for each architecture. These require a large amount of materials, measurement and control equipment, and human resources. However, by initially evaluating processor architecture schemes through computational simulation, only computational resources and corresponding program codes are required. This greatly reduces the economic cost of research. Second, computational simulation can provide faster results [32–35]. By performing computational simulation, data and results can be obtained in a shorter period of time. This helps to accelerate the research process of processor architecture design. In addition, computational simulation can be an important tool for performing or supporting analytical and experimental study. Computational simulation is an initial study to understand the feasibility and possible results of analytical and experimental study before performing them [60–62]. This can help researchers better understand the problem and develop more focused analytical and experimental plans. Also, computational simulation can be used to support the results of analytical and experimental study. Therefore, we use computational simulation to evaluate different processor architecture schemes.

To validate the efficiency of our method, we selected quantum programs (gate operations have been decomposed) from QASMBench [63] and ibm_qx_mapping [64] as benchmarks for experiments. According to three constraints of wiring, the maximum degree of vertex in **G** is 6. Therefore, we choose the program whose average degree of quantum program topology is less than or equal to 6 to generate a better matching processor architecture. Finally, we selected 16 quantum programs whose average degrees are shown in Table 4. These benchmarks have different qubit numbers (6~16) and different program functions to verify the generality of the method. In addition, to further verify the performance and yield rate improvement brought by the proposed method, four general-purpose square-lattice design schemes (20 qubits) of IBM [32,65] (see Fig. 11) and the eff-5-freq processor architecture

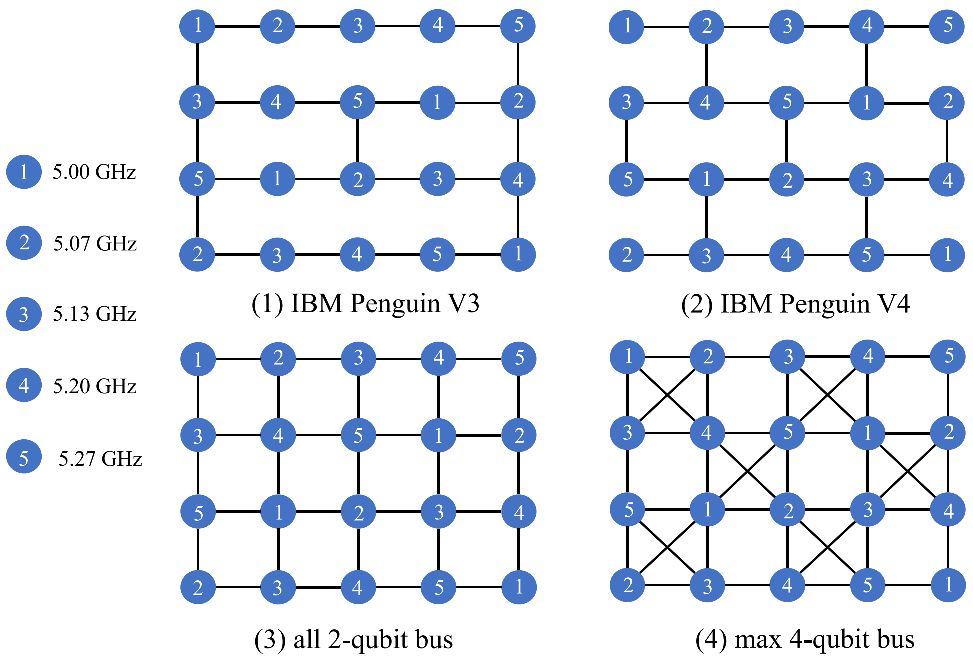


Fig. 11. Four general-purpose architectures and frequency allocation.

Table 4
Average degree of 16 quantum programs.

	Program	Qubits Number	Ave. Degree
1	alu-v2_30	6	5
2	sym6_145	7	6
3	hwb6_56	7	6
4	cm82a_208	8	5.25
5	dnn_n8	8	2
6	rd53_138	8	4
7	qpe_n9	9	4.22
8	mini_alu_305	10	4.4
9	seca_n11	11	3.45
10	wim_266	11	5.64
11	multiply_n13	13	3.385
12	rd53_311	13	5.23
13	0410184_169	14	3
14	multiplier_n15	15	4
15	cnt3-5_179	16	3.75
16	ising_model_16	16	1.875

design algorithm in Ref. [32] are selected for comparison. Among them, (1) is the IBM Penguin V3 architecture; (2) is the IBM Penguin V4 architecture; (3) is the all 2-qubit bus connection architecture; and (4) is the max 4-qubit bus connection architecture. The 5-frequency allocation scheme of four general-purpose architectures is referred to Ref. [32], as shown in Fig. 11. Here, the heavy-square and heavy-hexagon architectures [28,65,66] are not selected for comparison because both architectures use a 3-frequency scheme for frequency allocation. And the eff-full algorithm in Ref. [32] is not considered because the eff-full does not follow IBM's 5-frequency scheme for frequency allocation.

Parameter setting

In our design flow, the population size is set to be 200, the crossover probability is 0.75, the mutation probability is 0.15, and the number of iterations is 300. In the last 100 iterations, if the architecture and objective function value of 50 consecutive iterations do not change, the automatic design will be ended. In addition, we also record different architectures of the same objective function value, and make a better choice through the metrics of comparison.

We use the total post-mapping gate count [23–26,32] to evaluate the performance, which is completed by the IBM Qiskit transpiler [67]. For the parameters of IBM Qiskit transpiler, qubit mapping is realized by the SABRE algorithm in Ref. [24], and the optimization_level is 3. Since in both of our method and eff-5-freq algorithm, processor architectures are designed based on quantum programs, the logical qubit q_i corresponds to the physical qubit Q_i one by one. For general-purpose architectures, we do not specify the corresponding relationship between logical qubits and physical qubits.

For the calculation of yield rate, we use the frequency allocation algorithm in Ref. [32], and the candidate frequencies are selected according to IBM's 5-frequency scheme [27]. The number of trials for each architecture Monte-Carlo simulation is 100,000, which is 100 times of the IBM experiment [27], to ensure the accuracy of the simulation. σ_f is the same as in Ref. [32] and is set to be 30 MHz. For the eff-5-freq algorithm, the input of different quantum programs may cause the algorithm to output multiple chip architectures, and we select the processor architecture with the best performance and yield rate for comparison.

Result comparison

The simulation results of the performance and yield rate for all benchmarks on four general-purpose architectures and two special-purpose architectures are shown in Fig. 12, of which each subfigure contains the results of a benchmark tested under six processor architectures. In each subfigure, the x-axis represents the total post-mapping gate count, and the data points on the left side indicate less total post-mapping gate count, *i.e.*, better performance. The y-axis represents the yield rate, and the top data points indicate higher yield rates, *i.e.*, lower probability of frequency collisions. Therefore, for each benchmark, the data point in the top left indicates that the processor architecture has the best performance and yield rate. The legend at the bottom of Fig. 11 shows the markers for the six architectures.

Overall analysis

It can be seen that since IBM's design scheme (4) uses the max 4-qubit bus design, this architecture has the lowest total post-mapping gate count in each comparison. However, the rich qubit connections

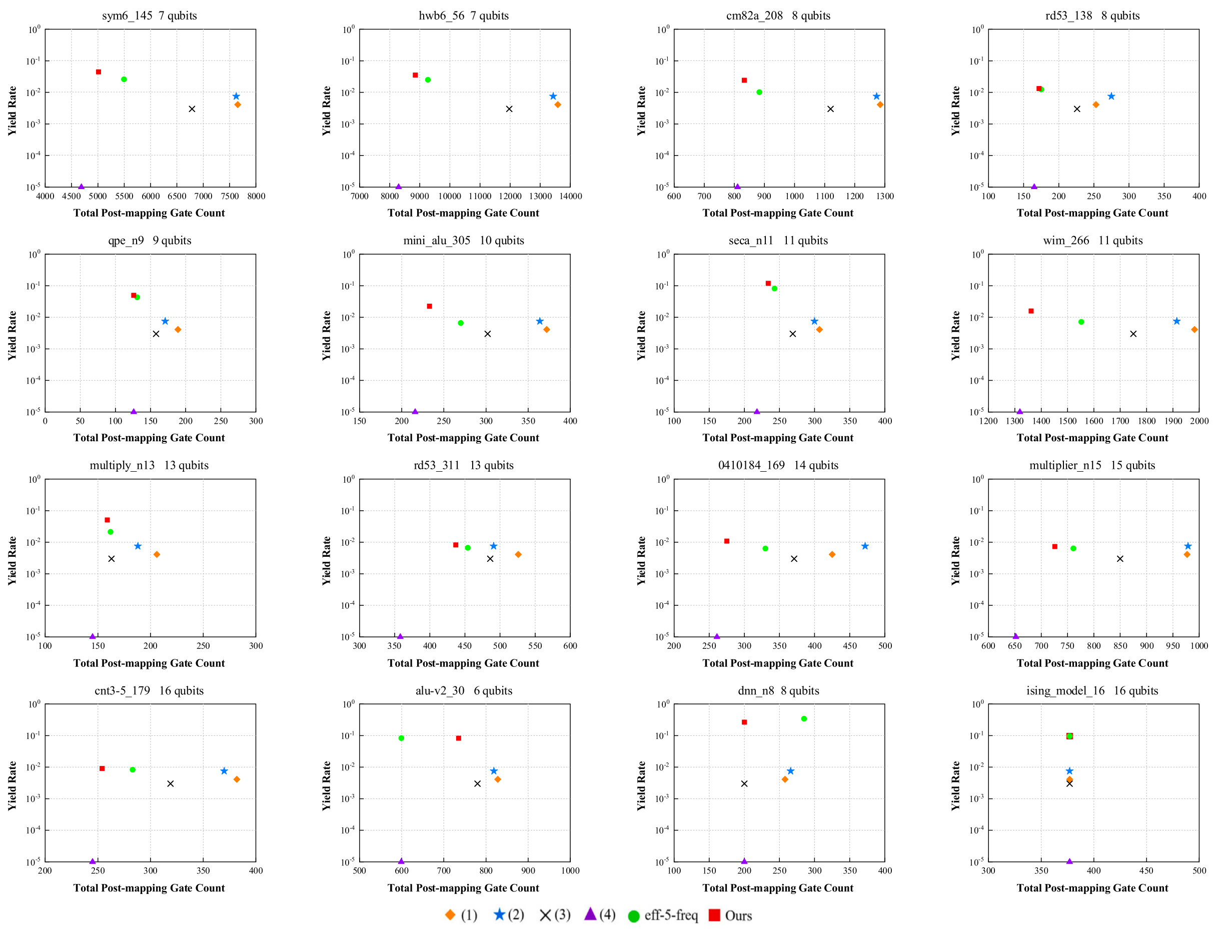


Fig. 12. Simulation comparison of 16 quantum programs.

also result in IBM's design scheme (4) having the lowest yield rate among the six architectures. The general-purpose architectures (1)~(3) improve the yield rate by more than 100 times compared to (4), but their performance rankings are almost always in the bottom in different benchmark tests due to the inability to satisfy more qubit connections.

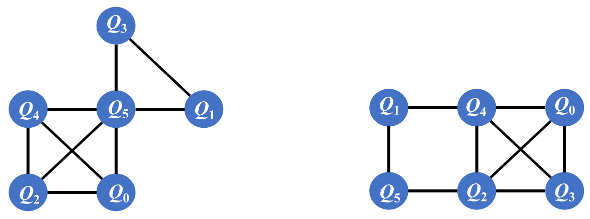
The data points of our method (red squares) and eff-5-freq algorithm (green circles) are located in the upper left of each subfigure except dnn_n8. This is because both algorithms are designed for quantum programs to improve performance and yield rate. For the total post mapping gate count, first, we calculate the percentage change in the architecture schemes of two algorithms compared to the general architecture schemes for each benchmark i, and obtain their arithmetic means $\bar{g}_i^{\mathrm{eff}}$ (for eff-5-freq) and $\bar{g}_i^{\mathrm{our}}$ (for our method). The positive percentage change represents the percentage decrease in the total post-mapping gate count (*i.e.*, the percentage of performance improvement). Conversely, the negative percentage change represents the percentage increase in the total post-mapping gate count, which corresponds to the performance decrease percentage. Then, for all benchmarks, compute the arithmetic means $\bar{g}_{all}^{\mathrm{eff}}$ and $\bar{g}_{all}^{\mathrm{our}}$ of $\bar{g}_i^{\mathrm{eff}}$ and $\bar{g}_i^{\mathrm{our}}$ respectively, obtain the average percentage change in the total post-mapping gate count for the architecture schemes of two algorithms compared to the general-purpose architecture schemes. Compared to the general-purpose architecture schemes, the architecture schemes of eff-5-freq reduce the total post-mapping gate count by an average of 9.2%, and the architecture schemes of our method reduce it by 15.61% on average. These also represent the average percentage improvement in performance. For the yield rate, we compare the minimum percentage improvement in the architecture schemes of two algorithms compared to the general-purpose architecture schemes across all benchmarks. Compared to the general-purpose architecture schemes, the architecture schemes of eff-5-freq provide a minimum 10.67% increase in the yield rate, and the architecture schemes of our method have a minimum increase of 21.33%. Since the yield rate represents the probability of no frequency collisions occurring, the percentage increase in the yield rate also represents the percentage decrease in the probability of frequency collisions. Therefore, under the comprehensive comparison of two metrics, they have better simulation results than four general-purpose design schemes.

Furthermore, in the comparison of the first 13 benchmarks, our method performs better than eff-5-freq. Specifically, compared with the processor architectures generated by eff-5-freq, our schemes have reduced the total post mapping gate count by 6.58% on average, with a maximum reduction of 16.67% in 0410184_169. As for yield rate, the minimum improvement of our scheme is 6.45% in rd53_138. In cm82a_2-08 and mini_alu_305, the yield rate is increased by 2.38× and 3.42×, respectively. This verifies the effectiveness of our method to optimize the processor architecture from the graph theory perspective. Therefore, the feasibility of solving the performance and frequency collision trade-off problem is also demonstrated. However, in the simulations on alu-v2_30, dnn_n8 and ising_model_16, the optimization results of our method are different from expectation. The reasons are analyzed below.

Analysis of special cases

alu – v2_30

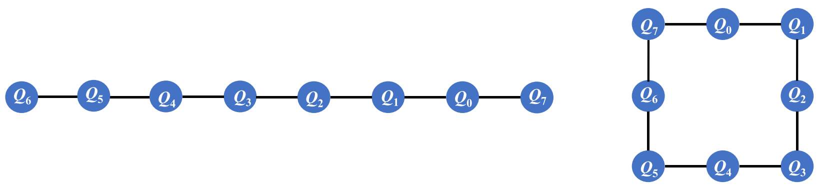
In alu-v2_30, our method and eff-5-freq algorithm perform better than general-purpose architectures (1)~(4). However, in terms of performance, eff-5-freq is even better, mainly due to the different optimization ideas of the two methods. The specific analysis is shown



$$\begin{array}{c|cccccc} & q_0 & q_1 & q_2 & q_3 & q_4 & q_5 \\ \hline q_0 & 0 & 11 & 23 & 8 & 12 & 22 \\ q_1 & 11 & 0 & 4 & 6 & 16 & 22 \\ q_2 & 23 & 4 & 0 & 6 & 21 & 8 \\ q_3 & 8 & 6 & 6 & 0 & 16 & 20 \\ q_4 & 12 & 16 & 21 & 16 & 0 & 28 \\ q_5 & 22 & 22 & 8 & 20 & 28 & 0 \end{array}$$

(a) eff-5-freq (b) Ours (c) coupling strength matrix **M**

Fig. 13. alu-v2_30 architecture optimization analysis.



$$\begin{array}{c|cccccccc} & q_0 & q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 \\ \hline q_0 & 0 & 24 & 0 & 0 & 0 & 0 & 0 & 24 \\ q_1 & 24 & 0 & 24 & 0 & 0 & 0 & 0 & 0 \\ q_2 & 0 & 24 & 0 & 24 & 0 & 0 & 0 & 0 \\ q_3 & 0 & 0 & 24 & 0 & 24 & 0 & 0 & 0 \\ q_4 & 0 & 0 & 0 & 24 & 0 & 24 & 0 & 0 \\ q_5 & 0 & 0 & 0 & 0 & 24 & 0 & 24 & 0 \\ q_6 & 0 & 0 & 0 & 0 & 0 & 24 & 0 & 24 \\ q_7 & 24 & 0 & 0 & 0 & 0 & 0 & 24 & 0 \end{array}$$

(a) eff-5-freq (b) Ours (c) coupling strength matrix **M**

Fig. 14. dnn_n8 architecture optimization analysis.

in Fig. 13. Fig. 13(a) and Fig. 12(b) show the processor architecture generated by eff-5-freq and our method, respectively. Fig. 13(c) shows the coupling strength matrix M (symmetric matrix) of eff-5-freq, where $\mathbf{M}(i, j)$ represents the number of two-qubit gates between the logical qubit q_i and q_j in the quantum program. Following the optimization idea of Ref. [32], the eff-5-freq algorithm first calculates the coupling degree of each q_i based on $\mathbf{M}$ with the calculation formula being $\sum_{j=0,j\neq i}^{n-1}\mathbf{M}(i,j)$. Then the coupling degrees of all logical qubits are sorted in descending order as a priority list of physical qubits. Finally, the layout and connections are made according to the priority list. According to Fig. 13(c), eff-5-freq conducts qubits layout and connections in the order of Q_5, Q_4, Q_0, Q_2, Q_1, and Q_3. That is, eff-5-freq gives priority to the placement and connection of frequently-used qubits in alu-v2_30 to reduce additional gate operations. Therefore, in qubit mapping, architecture (a) has 45 fewer SWAP gate operations than architecture (b). That is, the total post-mapping gate count is lower, and the performance is better. For our optimization scheme, architecture (a) and architecture (b) have the same calculation for $\sum d(i,j)$, and the calculation value of $\Delta(\mathbf{G})$ for architecture (b) is smaller, so the architecture scheme of our method converges to (b).

dnn_n8

In dnn_n8, our method and eff-5-freq obtain the optimal value in the total post-mapping gate count and yield rate, respectively. The main reason is that the two schemes have different qubit layout strategies, resulting in different architectures. The specific analysis is shown in Fig. 14. According to the coupling strength matrix $\mathbf{M}$ (Fig. 14(c)), we know that quantum program topology is a ring-shaped structure. And the 8 qubits in dnn_n8 are calculated to have the same priority. This leads to no discrimination in the selection of each qubit and the placement position. Following the eff-5-freq's qubit by qubit layout rules, the linked architecture (Fig. 14(a)) is finally generated, so the connectivity of qubits is reduced and the yield rate is higher. The yield rate in eff-5-freq is 27.75%, which is higher than that in our method. Our method optimizes the layout of all qubits according to the encoded individual. By minimizing the objective function value, we finally generate a ring-shaped architecture that meets the program requirements (Fig. 14(b)), so the total post-mapping gate count is less. In addition, (3) and (4) can match the 8-qubit ring-shaped structure through the mapping algorithm, so they are the minimum in total post-mapping gate count, as in our scheme.

ising_model_16

In ising_model_16, the total post-mapping gate count is the same for six architectures. That is, the 6 data points for ising_model_16 lie in one vertical line. This is because the quantum program topology extracted by ising_model_16 is a linked structure. The qubit mapping algorithm finds the optimal mapping without inserting additional gate operations. Therefore, the total post-mapping gate count is the same for all processor architectures. Our method generates the same linked processor architecture as the eff-5-freq algorithm, so the difference between the calculations of the yield rate in these two cases is only less than 0.5%. This is mainly affected by the randomness of Monte-Carlo simulation.

Conclusion and outlook

In this paper, we use a graph theory based technique to solve the performance and frequency collisions (yield rate) trade-off problem for designing superconducting quantum processors. Specifically, we convert the improvement of performance and yield rate into the optimization of the distance between two points and maximum degree. Furthermore, we build a mathematical model of the optimization problem with actual physical wiring constraints. Based on this mathematical model, we improve the genetic algorithm and propose an automatic processor architecture design flow with the joint optimization of performance and yield rate. Simulation results show that under the comprehensive comparison of performance and yield rate, the processor architectures designed by our method outperform IBM's general-purpose design schemes based on the square lattice and better than the schemes of the eff-5-freq algorithm in most quantum programs. The effectiveness and generality of our method are verified.

In this article, we mainly consider the cross-resonance gates allowed in both directions. In the future, we will consider one-way architecture optimization to achieve high fidelity [34]. Moreover, a larger number of qubits, number of operations of two-qubit gates, physical connection crosstalk and other factors will be considered to study a more comprehensive and practical processor architecture design method. Finally, we will also focus on the optimization of frequency allocation method. By adjusting the number of candidate frequencies and the frequency step to obtain a higher yield rate in a shorter time. Our work and next steps also promote the research and development of superconducting quantum processor design automation.

CRediT authorship contribution statement

Tian Yang: Conceptualization, Investigation, Methodology, Software, Visualization, Formal analysis, Writing – original draft. **Weilong Wang:** Data curation, Validation, Supervision, Writing – review & editing. **Lixin Wang:** Investigation, Visualization. **Bo Zhao:** Methodology, Supervision. **Chen Liang:** Software, Data curation. **Zheng Shan:** Conceptualization, Resources, Validation, Supervision, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work was supported by the Major Science and Technology Projects in Henan Province(CN), Grant No. 221100–210600.

References

[1] Georgescu IM, Ashhab S, Nori F. Quantum simulation. Rev Modern Phys 2014;86:153–85.
[2] McArdle S, Endo S, Aspuru-Guzik A, Benjamin SC, Yuan X. Quantum computational chemistry. Rev Modern Phys 2020;92:015003.
[3] Pan Y, Tong Y, Yang Y. Automatic depth optimization for a quantum approximate optimization algorithm. Phys Rev A 2022;105:032433.
[4] Biamonte J, Wittek P, Pancotti N, Rebentrost P, Wiebe N, Lloyd S. Quantum machine learning. Nature 2017;549(7671):195–202.
[5] Shor PW. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J Comput 1997;26(5):1484–509.
[6] Bao F, Deng H, Ding D, Gao R, Gao X, Huang C, et al. Fluxonium: An alternative qubit platform for high-fidelity operations. Phys Rev Lett 2022;129:010502.
[7] Murali P, Debroy DM, Brown KR, Martonosi M. Toward systematic architectural design of near-term trapped ion quantum computers. Commun ACM 2022;65(3):101–9.
[8] Arrazola J, Bergholm V, Brádler K, Bromley T, Collins M, Dhand I, et al. Quantum circuits with many photons on a programmable nanophotonic chip. Nature 2021;591:54–60.
[9] Mills AR, Feldman MM, Monical C, Lewis PJ, Larson KW, Mounce AM, Petta JR. Computer-automated tuning procedures for semiconductor quantum dot arrays. Appl Phys Lett 2019;115(11):113501.
[10] Krantz P, Kjaergaard M, Yan F, Orlando T, Gustavsson S, Oliver W. A quantum engineer's guide to superconducting qubits. Appl Phys Rev 2019;6(2):021318.
[11] Kjaergaard M, Schwartz ME, Braumüller J, Krantz P, Wang JI-J, Gustavsson S, et al. Superconducting qubits: Current state of play. Ann Rev Conden Matter Phys 2020;11(1):369–95.
[12] Google Quantum AI. Suppressing quantum errors by scaling a surface code logical qubit. Nature 2023;614(7949):676–81.
[13] Kim Y, Eddins A, Anand S, Wei KX, Van Den Berg E, Rosenblatt S, et al. Evidence for the utility of quantum computing before fault tolerance. Nature 2023;618(7965):500–5.
[14] Arute F, Arya K, Babbush R, Bacon D, Bardin JC, Barends R, et al. Quantum supremacy using a programmable superconducting processor. Nature 2019;574:505.
[15] Gong M, Wang S, Zha C, Chen M-C, Huang H-L, Wu Y, et al. Quantum walks on a programmable two-dimensional 62-qubit superconducting processor. Science 2021;372(6545):948–52.
[16] Wu Y, Bao W-S, Cao S, et al. Strong quantum computational advantage using a superconducting quantum processor. Phys Rev Lett 2021;127(18):180501.
[17] Place APM, Rodgers LVH, Mundada BM, Fitzpatrick M, Leng Z, et al. New material platform for superconducting transmon qubits with coherence times exceeding 0.3 milliseconds. Nat Commun 2021;12(1):1–6.
[18] Park SH, Bang J, An S, Hahn S. Design and performance analysis of hexagonal transmon qubit in a superconducting circuit. IEEE Trans Appl Supercond 2021;31(5):1–5.
[19] McKay DC, Filipp S, Mezzacapo A, Magesan E, Chow JM, Gambetta JM. Universal gate for fixed-frequency qubits via a tunable bus. Phys Rev Appl 2016;6:064007.
[20] Ahmad M, Giagkoulovits C, Danilin S, Weides M, Heidari H. Scalable cryoelectronics for superconducting qubit control and readout. Adv Intell Syst 2022;2200079.
[21] Van Dijk JPG, Patra B, Subramanian S, Xue X, Samkharadze N, Corna A, et al. A scalable cryo-CMOS controller for the wideband frequency-multiplexed control of spin qubits and transmons. IEEE J Solid-State Circuits 2020;55(11):2930–46.
[22] Fu X, Rol MA, Bultink CC, Van S. J, Khammassi N, Ashraf I, et al. An experimental microarchitecture for a superconducting quantum processor. In: Proceedings of the 50th annual IEEE/ACM international symposium on microarchitecture. 2017, p. 813–25.
[23] Zulehner A, Paler A, Wille R. An efficient methodology for mapping quantum circuits to the IBM QX architectures. IEEE Trans Comput-Aided Des Integr Circuits Syst 2019;38(7):1226–36.
[24] Li G, Ding Y, Xie Y. 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. Association for Computing Machinery; 2019, p. 1001–14.
[25] Li S, Zhou X, Feng Y. Qubit mapping based on subgraph isomorphism and filtered depth-limited search. IEEE Trans Comput 2021;70(11):1777–88.
[26] Zhang C, Hayes AB, Qiu L, Jin Y, Chen Y, Zhang EZ. Time-optimal qubit mapping. In: Proceedings of the 26th ACM international conference on architectural support for programming languages and operating systems. Association for Computing Machinery; 2021, p. 360–74.
[27] Brink M, Chow JM, Hertzberg J, Magesan E, Rosenblatt S. Device challenges for near term superconducting quantum processors: frequency collisions. In: 2018 IEEE International Electron Devices Meeting (IEDM). 2018, p. 6.1.1–3.
[28] Hertzberg JB, Zhang EJ, Rosenblatt S, Magesan E, Smolin JA, Yau J-B, et al. Laser-annealing Josephson junctions for yielding scaled-up superconducting quantum processors. npj Quan Inform 2021;7(1):1–8.
[29] Kim Y, Morvan A, Nguyen LB, Naik RK, Jünger C, Chen L, et al. High-fidelity three-qubit iToffoli gate for fixed-frequency superconducting qubits. Nat Phys 2022;1–6.
[30] Zhang EJ, Srinivasan S, Sundaresan N, Bogorin DF, Martin Y, Hertzberg JB, et al. High-performance superconducting quantum processors via laser annealing of transmon qubits. Sci Adv 2022;8(19):eabi6690.
[31] Morvan A, Chen L, Larson JM, Santiago DI, Siddiqi I. Optimizing frequency allocation for fixed-frequency superconducting quantum processors. Phys Rev Res 2022;4:023079.
[32] Li G, Ding Y, Xie Y. Towards efficient superconducting quantum processor architecture design. In: Proceedings of the twenty-fifth international conference on architectural support for programming languages and operating systems. Association for Computing Machinery; 2020, p. 1031–45.
[33] Hu W, Yang Y, Xia W, Pi J, Huang E, Zhang X-D, et al. Performance of superconducting quantum computing chips under different architecture designs. Quantum Inf Process 2022;21(7):1–14.
[34] Deb A, Dueck GW, Wille R. Exploring the potential benefits of alternative quantum computing architectures. IEEE Trans Comput-Aided Des Integr Circuits Syst 2021;40(9):1825–35.
[35] Lin W-H, Tan B, Niu MY, Kimko J, Cong J. Domain-specific quantum architecture optimization. IEEE J Emerg Sel Top Circuits Syst 2022;12(3):624–37.
[36] Shafaei A, Saeedi M, Pedram M. Qubit placement to minimize communication overhead in 2D quantum architectures. In: 2014 19th asia and south pacific design automation conference. 2014, p. 495–500.
[37] Siraichi MY, Santos VFd, Collange C, Pereira FMQ. Qubit allocation. In: Proceedings of the 2018 international symposium on code generation and optimization. CGO 2018, New York, NY, USA: Association for Computing Machinery; 2018, p. 113–25.
[38] Zhou X, Li S, Feng Y. Quantum circuit transformation based on simulated annealing and heuristic search. IEEE Trans Comput-Aided Des Integr Circuits Syst 2020;39(12):4683–94.
[39] Raghunathan S, Stok L. EDA and quantum computing: A symbiotic relationship? IEEE Des Test 2020;37(6):71–8.
[40] Stok L. EDA and quantum computing: The key role of quantum circuits. In: Proceedings of the 2021 international symposium on physical design. New York, NY, USA: Association for Computing Machinery; 2021, p. 111.
[41] Barenco A, Bennett CH, Cleve R, DiVincenzo DP, Margolus N, Shor P, et al. Elementary gates for quantum computation. Phys Rev A 1995;52:3457–67.
[42] Rigetti C, Devoret M. Fully microwave-tunable universal gates in superconducting qubits with linear couplings and fixed transition frequencies. Phys Rev B 2010;81:134507.
[43] Chow JM, Córcoles AD, Gambetta JM, Rigetti C, Johnson BR, Smolin JA, et al. Simple all-microwave entangling gate for fixed-frequency superconducting qubits. Phys Rev Lett 2011;107:080502.
[44] Li G, Shi Y, Javadi-Abhari A. Software-hardware co-optimization for computational chemistry on superconducting quantum processors. In: Proceedings of the 48th annual international symposium on computer architecture. IEEE Press; 2021, p. 832–45.

[45] Goldberg DE. Genetic algorithms in search, optimization and machine learning. 1st ed.. USA: Addison-Wesley Longman Publishing Co., Inc.; 1989.
[46] Bollobas B. Modern graph theory. 1st ed.. USA: Springer New York, NY; 2001.
[47] Lamura MDP, Hidayat T, Ammarullah MI, Bayuseno AP, Jamari J. Study of contact mechanics between two brass solids in various diameter ratios and friction coefficient. Proc Instit Mech Eng Part J: J Eng Tribol 2023;14657503221144810.
[48] Danny Pratama Lamura M, Imam Ammarullah M, Hidayat T, Izzur Maula M, Jamari J, Bayuseno AP. Diameter ratio and friction coefficient effect on equivalent plastic strain (PEEQ) during contact between two brass solids. Cog Eng 2023;10(1):2218691.
[49] Ammarullah MI, Santoso G, Sugiharto S, Supriyono T, Wibowo DB, Kurdi O, et al. Minimizing risk of failure from ceramic-on-ceramic total hip prosthesis by selecting ceramic materials based on tresca stress. Sustainability 2022;14(20).
[50] Ammarullah MI, Afif IY, Maula MI, Winarni TI, Tauviqirrahman M, Akbar I, et al. Tresca stress simulation of metal-on-metal total hip arthroplasty during normal walking activity. Materials 2021;14(24).
[51] Ammarullah MI, Santoso G, Sugiharto S, Supriyono T, Kurdi O, Tauviqirrahman M, et al. Tresca stress study of CoCrMo-on-CoCrMo bearings based on body mass index using 2D computational model. J Tribol 2022;33(2):31–8.
[52] Tauviqirrahman M, Ammarullah MI, Jamari J, Saputra E, Winarni TI, Kurniawan FD, et al. Analysis of contact pressure in a 3D model of dual-mobility hip joint prosthesis under a gait cycle. Sci Rep 2023;13(1):3564.
[53] Ammarullah MI, Hartono R, Supriyono T, Santoso G, Sugiharto S, Permana MS. Polycrystalline diamond as a potential material for the hard-on-hard bearing of total hip prosthesis: Von mises stress analysis. Biomedicines 2023;11(3).
[54] Salaha ZFM, Ammarullah MI, Abdullah NNAA, Aziz AUA, Gan H-S, Abdullah AH, et al. Biomechanical effects of the porous structure of gyroid and Voronoi hip implants: A finite element analysis using an experimentally validated model. Materials 2023;16(9).
[55] Jamari J, Ammarullah MI, Santoso G, Sugiharto S, Supriyono T, Permana MS, et al. Adopted walking condition for computational simulation approach on bearing of hip joint prosthesis: Review over the past 30 years. Heliyon 2022.
[56] Tauviqirrahman M, Jamari J, Susilowati S, Pujiastuti C, Setiyana B, Pasaribu AH, et al. Performance comparison of Newtonian and non-Newtonian fluid on a heterogeneous slip/no-slip journal bearing system based on CFD-FSI method. Fluids 2022;7(7).
[57] Khor CY, Mujeebu MA, Abdullah MZ, Ani FC. Finite volume based CFD simulation of pressurized flip-chip underfill encapsulation process. Microelectron Reliabil 2010;50(1):98–105.
[58] Liu MB, Liu GR, Lam KY. Computer simulation of flip-chip underfill encapsulation process using meshfree particle method. Int J Comput Eng Sci 2012;4(02):405–8.
[59] Dai P, Wang M, Wang X. A simulation platform for reconfigurable processor. J Comput Inform Syst 2013;9(4):1659–68.
[60] Yuan B, Wang W, Liu F, He H, Shan Z. Comparison of lumped oscillator model and energy participation ratio methods in designing two-dimensional superconducting quantum chips. Entropy 2022;24(6).
[61] He H, Wang W, Liu F, Yuan B, Shan Z. Suppressing the dielectric loss in superconducting qubits through useful geometry design. Entropy 2022;24(7).
[62] Sun Y, Ding J, Xia X, Wang X, wen Xu J, Song S, et al. Fabrication of airbridges with gradient exposure. Appl Phys Lett 2022;121(7):074001.
[63] Li A, Stein S, Krishnamoorthy S, Ang J. QASMBench: A low-level quantum benchmark suite for NISQ evaluation and simulation. ACM Trans Quantum Comput 2022.
[64] Wille R, Burgholzer L, Zulehner A. Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations. In: 2019 56th ACM/IEEE design automation conference. 2019, p. 1–6.
[65] IBM. The IBM quantum heavy hex lattice. 2022, https://research.ibm.com/blog/heavy-hex-lattice. [Accessed 16 July 2022].
[66] Chamberland C, Zhu G, Yoder TJ, Hertzberg JB, Cross AW. Topological and subsystem codes on low-degree graphs with flag qubits. Phys Rev X 2020;10:011022.
[67] IBM. IBM qiskit transpiler. 2022, https://qiskit.org/documentation/stable/0.34/apidoc/transpiler.html. [Accessed 16 July 2022].