Automatic Architecture Design for Distributed Quantum Computing

Ting-Yu Luo (骆挺宇)[1], Yu-Zhen Zheng(郑宇真)[2,3], Xiang Fu(付祥)[2]*, and Yu-Xin Deng(邓玉欣)[1]†

[1]Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, 200062, China

[2]Institute for Quantum Information & State Key Laboratory of High Performance Computing, College of Computer, National University of Defense Technology, Changsha, 410073, China

[3]Tianjin Institute of Advanced Technology, Tianjin, 300459, China

Abstract

In distributed quantum computing (DQC), quantum hardware design mainly focuses on providing as many as possible high-quality inter-chip connections. Meanwhile, quantum software tries best to reduce the required number of remote quantum gates between chips. However, this "hardware first, software follows" methodology may not fully exploit the potential of DQC. Inspired by classical software-hardware co-design, this paper explores the design space of application-specific DQC architectures. More specifically, we propose AutoArch, an automated quantum chip network (QCN) structure design tool. With qubits grouping followed by a customized QCN design, AutoArch can generate a near-optimal DQC architecture suitable for target quantum algorithms. Experimental results show that the DQC architecture generated by AutoArch can outperform other general QCN architectures when executing target quantum algorithms.

Keywords: Distributed quantum computing, quantum architecture, quantum circuit partitioning

PACS: 03.67.Lx, 85.25.-j

1. Introduction

Quantum computing [1] is promising to accelerate solving certain problems which are hard for classical computers, such as quantum simulation [2,3] and factoring [4]. In the current quantum computer design flow for circuit-model-based [5] quantum computers, quantum hardware takes a leading role when collaborating with quantum software to implement a quantum computer. In other words, quantum software is tailored to match the requirements and constraints exposed by quantum hardware. For example, although quantum algorithms can be designed without considering the hardware primitive gate set and qubit connectivity, the quantum compiler [6] is responsible for making quantum programs executable on target hardware by performing transformations, such as quantum gate decomposition [7] and qubit mapping and routing [8,9]. To be more radical, some work even crafts the quantum algorithm to make it suitable for the the target quantum chip, trying to maximize the computational capability of the hardware [10]. However, relatively

*Corresponding author. E-mail: xiangfu@quanta.org.cn

†Corresponding author. E-mail: yxdeng@sei.ecnu.edu.cn

little attention has been paid to the efficiency of the quantum (micro)architecture when executing given quantum algorithms.

Integrating a larger number of physical qubits is a key to increase the computational power of quantum computing systems. Solid-state quantum computing systems, especially those based on superconducting qubits, have achieved a steady growth in the number of qubits in a single system, from tens of qubits to hundreds of qubits [10–19]. The scale of individual superconducting chips is limited by fabrication and control challenges. As the number of qubits increases, the amount of elements in a single superconducting quantum chip grows proportionally, which in turn can significantly lower the superconducting chip fabrication yield rate. Also, the number of wire used to transmit control signals also grows proportionally to the number of qubits, which can eventually exceed the capacity of dilution refrigerators holding these quantum chips. As a result, the number of superconducting qubits integrated into a single chip is limited.

To tackle this challenge, an alternative approach is to investigate distributed quantum computing (DQC). In DQC, a larger system is composed of multiple quantum chips interconnected via remote connections, which forms a quantum chip network (QCN). In DQC architectures, remote multi-qubit gates can be implemented by consuming Einstein-Podolsky-Roson (EPR) [20] pairs created on the remote connection between chips. Remote gates enable entangling qubits on different chips in the system, providing a method to realize large-scale quantum computing.

Preparing a remote EPR pair could consume a longer time than local quantum gates, and the created remote EPR pair normally has a fidelity that is significantly lower than local EPR pairs [21–24]. Consequently, remote quantum gates are costly, both in time and fidelity. To improve the fidelity of executing a quantum program on DQC architecture, it is crucial to minimize the usage of EPR pairs during execution.

Previous research on DQC architecture can be roughly classified into two categories: (1) hardware architecture construction [25–28], and (2) software techniques to support executing quantum programs on DQC architecture [24,28–36]. In these works, hardware design mainly focuses on providing an architecture with a number of high-quality connection among chips, and quantum software tries best to reduce the number of EPR pairs consumed by mapping quantum algorithms targeting these chips. However, this methodology may not fully exploit the potential of DQC based on superconducting quantum processors in executing quantum algorithms.

We observe that various quantum algorithms require consuming different numbers of EPR pairs when executing on different DQC architectures. This fact hints that, given a potential quantum application in the near term, if we can customize the DQC architecture on which the algorithm executes, we may further reduce the required remote EPR pairs, and hence enhance the fidelity of the final result.

Guided by this insight, we adopt a software-hardware co-design methodology for DQC and exploit application-specific DQC architectural design space to enhance the fidelity of near-term applications. Addressing the challenges in revealing related features of quantum applications to DQC architecture design and exploiting the large design space, we propose an automatic DQC architecture design framework *AutoArch*. It can generate efficient DQC architecture designs for given potential quantum programs in the near-term.

The main contributions of this paper are as follows:

1. We explore customized design space for DQC architectures tailored to the characteristics of quantum programs. Furthermore, we incorporate the actual constraints inherent in each quantum chip into our consideration, aiming to achieve more efficient design outcomes.

2. We incorporate innovative algorithmic primitives to automatically produce a spectrum of application-

specific architecture designs under certain constraints of various quantum chips.

3. Comprehensive experiments show that the benchmark quantum algorithms can be executed with better efficiency on the customized DQC architectures generated by our design framework compared to other DQC architectures. The customized DQC architecture can lead to an average reduction of 50.66% (maximum 92.6%) in the estimated EPR pair consumption, and an average reduction of 42.65% (maximum 87.93%) in the number of additional local SWAP gates.

The paper is structured as follows. Section 2 introduces the basics of DQC, quantum circuit partitioning, and quantum compilation. Our solution *AutoArch* is introduced in Section 3 and evaluated in Section 4. Related works are summarized in Section 5 and we finally conclude this paper in Section 6.

2. Background

DQC deals with complex computational problems by splitting out the computational workload among multiple quantum devices to lighten the burden on individual quantum chips [37]. We begin with two fundamental infrastructures of DQC. The first is the quantum chip network, and the second is remote quantum communication. Following that, we discuss two patterns of remote quantum communication.

Quantum Chip Network Multiple independent quantum chips, interconnected through special physical connections, form a quantum chip network that overcomes the limitations of the number of qubits within a single quantum chip. This physical inter-chip connection is typically a specially designed superconducting coaxial cable [21,23] for superconducting quantum hardware. Based on the remote physical connections within the quantum chip network, an entangled state can be generated between two qubits located in two different quantum chips. This entangled state serves as a quantum communication channel that can be used for remote communication, allowing for the transmission of quantum information between quantum chips.

Remote Quantum Communication Similar to classical distributed computing, remote quantum communication is the foundation of DQC, yet it also presents a significant challenge. Unlike classical systems, quantum data cannot be easily transferred across quantum chips due to the constraints imposed by the quantum no-cloning theorem [38]. The solution to this dilemma is to utilize inter-chip quantum entanglement. When two qubits are entangled in the state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, they form an EPR pair. If two qubits of an EPR pair are distributed in two different quantum chips, a remote EPR pair is formed. A remote EPR pair is a crucial quantum communication resource that facilitates the transfer of quantum data between quantum chips. In this paper, physical qubits capable of establishing remote EPR pairs are referred to as *communication qubits*. In contrast, physical qubits within a quantum chip designed to store program information are known as *data qubits*. In the short term, the number of communication qubits available on a single quantum chip to participate in the creation of EPR pairs is limited.

Remote Quantum Communication Patterns Indeed, EPR pairs have led to the development of two communication patterns, named *Cat-Comm* and *TP-Comm* [24]. The *Cat-Comm* pattern is constructed using cat-entangler and cat-disentangler, while the *TP-Comm* pattern is founded on the quantum teleportation [39]. Specifically, executing a remote CNOT gate on two adjacent quantum chips using the *Cat-Comm* communication pattern consumes one EPR pair, but a remote SWAP gate requires up to three EPR pairs [30]. In contrast, performing a general remote two-qubit gate on two adjacent quantum chips using *TP-Comm* pattern consumes two EPR pairs. This demonstrates that the number of EPR pairs required to execute remote quantum gates in an immature quantum chip network is not only influenced by

the remote communication pattern employed but also closely related to the distance between the quantum chips that host these qubits.

Quantum circuit partitioning was proposed [24,29,32–36] to reduce required remote quantum gates by grouping together qubits with more interconnections. Quantum circuit partitioning typically involves dividing the virtual qubits used in a quantum program into k partition blocks, each block containing several virtual quits. The objective of quantum circuit partitioning is to minimize the number of required *remote quantum gates*, i.e., two-qubit gates applied on those qubits that lie in different blocks. In contrast, the gates acting on the qubits within the same partition block are termed *local quantum gates*.

NISQ devices are subject to hardware constraints, such as limited topology connectivity and native physical gate sets, preventing high-level quantum circuits from being directly executable. For instance, two-qubit gates can only be executed on the nearest neighbouring physical qubits. Consequently, movement operations such as SWAP gates need to be inserted, so to enable the execution of quantum gates that cannot be directly performed on adjacent physical qubits. For this purpose, quantum compilation techniques [7–9,40–45], including gate decomposition, qubit mapping and routing, and gate scheduling, have been devised to make high-level quantum circuits compatible with hardware constraints. During quantum compilation, qubit mapping and routing are two crucial steps. **Qubit mapping** aims to find an optimal qubit initial placement that can minimize the number of additional SWAP gates required for implementing all two-qubit gates [8,44]. **Qubit routing** involves continuously inserting movement operations like SWAP gates into the input quantum circuit to relocate the virtual qubits that are mapped onto non-nearest neighbouring physical qubits [9,40]. This step ensures that all quantum gates in the circuit satisfy the constraints of hardware topology connectivity.

3. DQC Architecture Design

3.1. Overview of AutoArch

Given current constraints on DQC architectures, there is not a universal DQC architecture that can guarantee the high-quality execution of various quantum programs. The purpose of the *AutoArch* framework is to customize a DQC architecture targeting particular quantum programs so that the overhead of local SWAP gates and remote EPR pairs can be minimized in qubit mapping and routing. As shown in Fig. 1, *AutoArch* consists of two key phases: constrained quantum circuit partitioning and interconnection design of QCNs.

The first phase of *AutoArch* aims to curtail the generation of remote quanutm gates. To this end, we propose a Tabu-Search-based [46,47] constrained quantum circuit partitioning algorithm (*CPA*). It can optimally partition quantum circuits into k parts, where k represents the number of quantum chips required for execution in a distributed scenario. The partitioning process is dominated by the physical data qubit constraints of quantum chips to ensure the generation of partitioning results that satisfy the aforementioned constraints. *CPA* ends with outputting a set of partitioned **quantum circuit blocks** and corresponding **remote connection intensity matrix**, which will guide subsequent steps in *AutoArch*.

In the near-term, it is more practical for a single quantum chip to connect to a subset of other chips instead of all other chips in the network. The second phase of *AutoArch* carefully designs the interconnections between chips while considering the remote connection capacity of each quantum chip. This is done by **quantum chip allocation** and **physical remote connection selection**. The result of

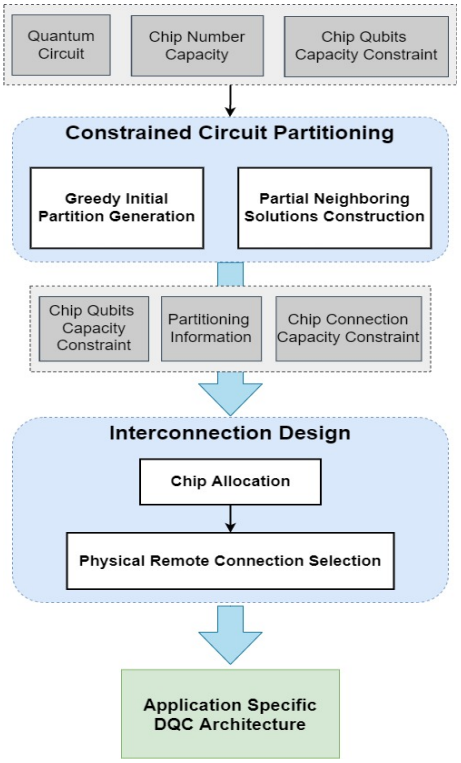

Fig. 1. Overview of the DQC Architecture Design Flow. Two blue circular frames represent two core modules. The gray rectangular frames denote the input information of the modules as well as the quantum circuit partition information. The green rectangular frame represents the final output of the DQC architecture information.

CPA leads to a straightforward quantum chip allocation process, while the complexity of selecting inter-chip connections can be exponential. Addressing this problem, we propose a heuristic algorithm to search for the optimal connections between quantum chips, which automates the design of DQC architectures.

3.2. An Example

We use a simple example (cf. Fig. 2) to illustrate the working process of *AutoArch*. Consider the quantum circuit in Fig. 2(a), which has eight virtual qubits denoted by $q_0, \ldots, q_7$, and all of them are initially set to $|0\rangle$. Then some single-qubit gates and two-qubit gates are applied. Measurement operations are omitted and not shown in the diagram.

Fig. 2(b) displays four available quantum chips denoted by $Chip_0, \ldots, Chip_3$, with their physical data qubit capacities being 2, 2, 2, and 3, respectively. Moreover, these quantum chips can be connected to at most two other quantum chips. After the constrained circuit partitioning, four partition blocks denoted by $Block_0, \ldots, Block_3$ are obtained as given in Fig. 2(c), and their sizes all comply with the constraints of the aforementioned quantum chips. Additionally, a remote connection intensity matrix can be derived, which is the symmetric matrix displayed in Fig. 2(d).

Fig. 2(e) to Fig. 2(g) show the process of interconnecting quantum chips. Considering the capacities of quantum chips and the sizes of partitioned blocks, it is intuitive to identify the mapping relationship between them: a partition block corresponds to the quantum chip with the same subscript. After that, we decide the order of interconnections between chips based on the number of associated remote quantum gates between partition blocks revealed by the remote connection density matrix. Given that

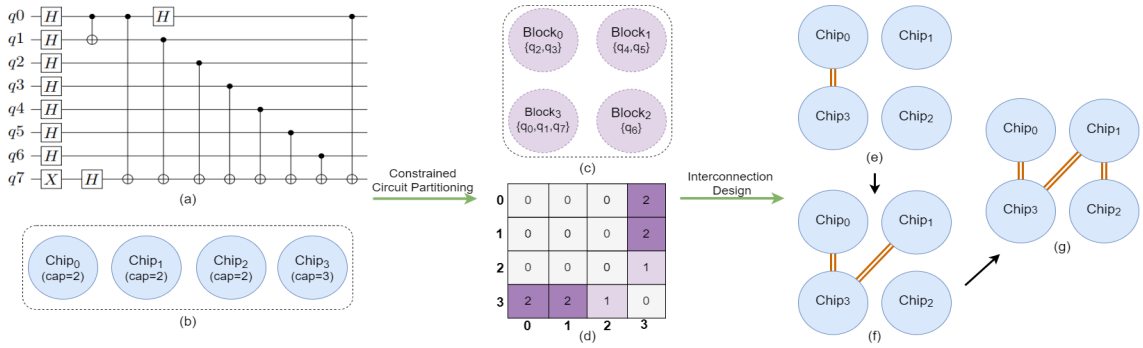

Fig. 2. Example of the working details of *AutoArch*.

$(Block_0, Block_3)$ and $(Block_1, Block_3)$ are each associated with two remote quantum gates, we sequentially connect $Chip_0$ and $Chip_1$ to $Chip_3$, preventing $Chip_3$ from establishing new interconnections with other chips. Next, we consider $Chip_2$, whose feasible connection targets are limited to $Chip_0$ and $Chip_1$. Regardless of which one is chosen for connection, the connection distance between $Chip_0$ and $Chip_3$ remains consistent. Finally, we decide to connect $Chip_0$ with $Chip_2$, thereby constructing a new QCN shown in Fig. 2(g).

3.3. Constrained Quantum Circuit Partitioning

In this section, we first formulate the constrained quantum circuit partitioning problem for DQC. Following that, we explore how to minimize the number of remote quantum gates with limited physical qubit capacity.

3.3.1. Problem Formulation and Analysis

Adopting reasonable partitioning strategies to partition quantum circuits is a common method for reducing the number of remote quantum gates. However, in the short term, the limited availability of physical qubits on quantum chips imposes new constraints on quantum circuit partitioning. The input to the constrained quantum circuit partitioning problem comprises the following:

1. A quantum circuit with a set of virtual qubits (Q), and a collection of CNOT gates (G). Each element $g = \text{CNOT}(q_i, q_j) \in G$ is a CNOT gate applied on a pair of qubits $q_i, q_j \in Q$.

2. A partition block relation graph $R = (B, C)$. Each element $b \in B$ is a partition block with a set of virtual qubits, and the element $c = (q_i, q_j) \in C$ stands for a remote CNOT gate applied on two remote qubits $q_i \in b_m$ and $q_2 \in b_n$ where $b_m \neq b_n$.

For all $q \in Q$ and $b \in B$, we introduce a binary variable $x_{q,b} \in \{0, 1\}$, representing whether the virtual qubit q is assigned to the partition block b. To guarantee that each virtual qubit is exactly allocated to one partition block, the following constraint should be satisfied:

$$\forall q \in Q, \qquad \sum_{b \in B} x_{q,b} = 1. \tag{1}$$

Let s_b be the maximum number of virtual qubits that block b can accommodate, then the following constraint holds:

$$\forall b \in B, \qquad \sum_{q \in Q} x_{q,b} \leq s_b. \tag{2}$$

Let $n(q_1, q_2) \in \mathbb{N}$ be the number of CNOT gates acting on virtual qubits q_1 and q_2. Additionally, we define a binary variable $c_{q_1,q_2,b_1,b_2} \in \{0, 1\}$, which is one when q_1 and q_2 are located in two different partition blocks $b_1 \in B$ and $b_2 \in B$, respectively.

Therefore, the number of remote CNOT gates generated by the circuit partitioning algorithm is denoted as

$$N_{remote\_CNOT} = \sum_{q_1 \neq q_2} \sum_{b_1 \neq b_2} n(q_1, q_2) \cdot c_{q_1,q_2,b_1,b_2} \cdot x_{q_1,b_1} \cdot x_{q_2,b_2}. \tag{3}$$

Overall, the constrained quantum circuit partitioning problem can be formulated as the following:

$$\begin{aligned} &\min \sum_{q_1 \neq q_2} \sum_{b_1 \neq b_2} n(q_1, q_2) \cdot c_{q_1,q_2,b_1,b_2} \cdot x_{q_1,b_1} \cdot x_{q_2,b_2} \\ &\text{s.t. } \forall q \in Q, \sum_{b \in B} x_{q,b} = 1, \quad \text{and} \quad \forall b \in B, \sum_{q \in Q} x_{q,b} \leq s_b. \end{aligned}$$

After formulating the constrained quantum circuit partitioning problem, we observe that its model shares the same structure as the quadratic assignment problem [48]. Since quadratic assignment problem is already known to be NP-hard [49], the constrained quantum circuit partitioning problem is also NP-hard. Therefore, it is impossible to find the optimal solution to the constrained quantum circuit partitioning problem in polynomial time. Inspired by existing metaheuristic algorithms used to tackle NP-hard problems, we propose a partitioning algorithm based on Tabu Search to solve the constrained quantum circuit partitioning problem.

3.3.2. Constrained Partitioning Algorithm

In this section, we will briefly explain the fundamental principles of the Tabu Search algorithm [46,47] and introduce our *CPA* algorithm based on this method. Specifically, we will focus on two key design features *of CPA*: a greedy strategy for initial partition generation and the partial neighboring solution set construction.

Tabu Search is a metaheuristic algorithm widely used for solving combinatorial optimization problems [50]. It starts by generating an initial solution, which can be either random or tailored to the problem (step 1). Next, this algorithm generates a list of neighboring solutions by making local variations to the current solution, such as swapping the positions of two elements (step 2). The objective function is then evaluated for each neighboring solution to indicate its quality (step 3). During the search process, the algorithm records solutions that have already been explored to avoid getting trapped in local optima. These recorded solutions are called "tabu" and are temporarily disallowed from being used in future searches. Based on the values of the objective function and the content of the tabu list, the algorithm chooses an appropriate neighboring solution as the direction for the next search step (step 4). Steps 1 to 4 constitute one iteration of the Tabu Search algorithm. After multiple iterations, the algorithm terminates either when the value of the objective function no longer changes beyond a threshold or when a given maximum number of iterations have been reached.

The detailed workflow of *CPA* is shown in Fig. 3. It terminates with a partitioning result with several blocks of different qubits. *CPA* can ensure that the number of qubits in every block does not exceed the number of data qubits on the quantum chip which the block will be assigned to. A key step used in *CPA* is to get the neighboring solution from the current solution. The neighboring solution is generated by changing the location of one qubit or two qubits. There are two cases: (i) the virtual qubit is moved from

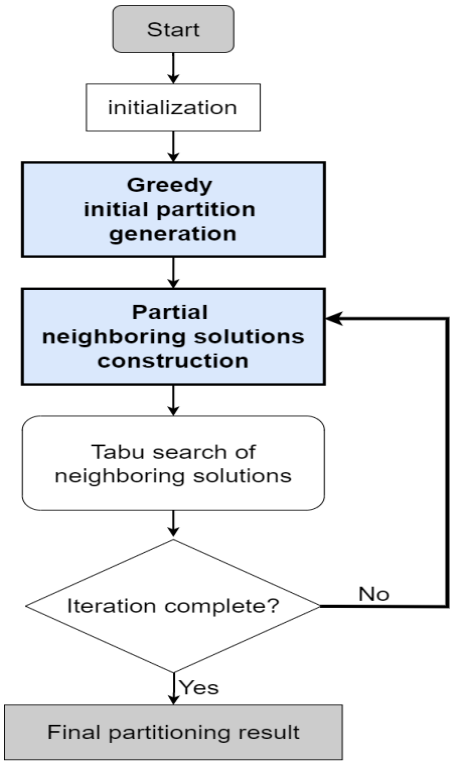

Fig. 3. Workflow of the *CPA* algorithm.

one block to another block with spare capacity, and (ii) the positions of two virtual qubits belonging to two distinct blocks are exchanged.

Greedy Strategy for Initial Partition Generation Since the quality of the initial partition can significantly affect the required number of Tabu iterations to find a good enough solution, a well-designed initial partition is desired. We develop a greedy strategy in *CPA* to generate an initial partition. In quantum algorithms, a group of qubits may have more interactions via CNOT gates with each other than other qubits. Hence, this group of qubits can be regarded as a cluster. This fact inspires us to design a greedy strategy to generate the initial partition in the following way:

1. We sort qubits according to the number of CNOT gates acting on them, from the highest to the lowest. Let S denote all virtual qubits that have not been assigned to any blocks, and B represent all blocks with each block corresponding to a target quantum chip. At the beginning, S includes all virtual qubits, and B includes all empty partition blocks. We start by selecting the block with the maximum capacity in B as the current target block b.

2. We process the qubit (denoted as q) currently with the most CNOT connections with other qubits. Our goal is to place q and its connected qubits within the same block. If the current target block b does not have enough space to accommodate qubit q and all its connected qubits, a connected qubit with a higher-ranking is placed in b. When b is full, a new empty block with the currently maximum capacity in B is selected as the new target block b. After the placement, these qubits are removed from S.

3. We repeat this process until set S is empty, indicating that the initial partition has been successfully generated.

Partial Neighboring Solution Set Construction The Tabu Search algorithm typically evaluates all neighboring solutions of the current solution during each iteration and selects the most optimal one to serve as the starting point for the subsequent iteration. For example, if we want to divide a quantum circuit into k parts, with each part having size m, there are $\binom{k}{2}$ partition block pairs. For each partition block pair, there are $\binom{m}{1}\binom{m}{1}$ swap scenarios. Therefore, the total number of neighboring solutions will be $N_{sol} = \binom{k}{2}\binom{m}{1}\binom{m}{1} = \frac{k(k-1)m^2}{2}$. However, as the number of virtual qubits used in the quantum circuit increases, the number of neighboring solutions explored in each iteration grows quadratically, leading to a sharp increase in the computational cost to find the optimal partition under constraints. Considering the interconnectivity of virtual qubits within existing quantum circuits is not immediately complex. Thus, it is possible that the current partition solution has numerous neighboring solutions with the same count of remote CNOT gates. To accelerate the convergence rate of the partitioning algorithm, a parameter known as the neighborhood solutions ratio δ is introduced when traversing the neighboring solutions of the current partition. This parameter is used to randomly select a subset of δN_{sol} neighboring partition solutions to form the neighborhood set.

After obtaining the partitioning result of a quantum circuit, we can get the quantum chip allocation relationship (ϕ) and the remote connection intensity matrix ($\boldsymbol{M}$). For ease of description, we set the index of the partition block in ϕ to be the same as that of the corresponding quantum chip.

3.4. Interconnection Design of QCNs

Once the circuit partition result is obtained, there is a straightforward way to construct a QCN, of which the structure shares the same topology as the partition blocks. However, there are two reasons why we do not simply adopt this structure. First, the interconnections between quantum chips is limited, which can be far fewer than those required by the partitioning result. Second, more interconnections between quantum chips can reduce the number of data qubits used to perform quantum computation, which in turn reduces the utility rate of the quantum chip. In this case, the required number of quantum chips used to perform the same quantum algorithm can be increased. Hence, it is important to design an interconnection scheme which uses as few as possible interconnections while not reducing the fidelity of executing quantum algorithms on the this QCN. This task forms the second phase of *AutoArch*.

The challenge of QCNs interconnection design lies in the large size of the design space. For k quantum chips, there are a total of $N_p = \binom{k}{2}$ distinct chip pairs. Each of these pairs can be connected or disconnected, resulting in 2^{N_p} different cases.

In this paper, we aim to simplify the interconnection design problem by making the assumption that each quantum chip can only be connected to a limited number of other quantum chips. This assumption is based on the fact that until a QCN reaches a significant scale, there is a finite pool of qubits available on any given chip for remote communication. Given this constraint, our primary goal in designing the interconnections of QCNs is to minimize the incidence of cross-chip remote quantum communication when executing remote quantum gates on the network. Lowering the frequency of such cross-chip communications results in a reduced consumption of EPR pairs. Despite the imposed constraints on interconnection design, the design space for the interconnections of QCNs remains exponential, $O(\exp(k))$, where k is the number of quantum chips. Nonetheless, it is possible for us to develop a heuristic algorithm to guide this design process effectively.

Considering the constraints and optimization objectives of the interconnection design phase, we assume that:

- Since more remote physical links reduce available data qubits, we set a limit on the number of remote links to make the design easier.
- We set a limit on how many other quantum chips each chip can connect to, which means there is a finite number of interconnections. Once we decide which chips are linked, we need not further to worry about the exact number of remote physical links between them. Knowing how many qubits each chip has for remote communication, we can figure out how many remote physical links are used between them.

Algorithm 1 explains how the interconnection design algorithm adds connections between quantum chips. Before discussing the details of this algorithm, we first introduce some new variables and concepts used by the algorithm:

- A quantum chip is marked as *connected* if and only if there is an interconnection between this chip and any other chip. Otherwise, it is *unconnected*. For example, in Algorithm 1, countChipCxn($\boldsymbol{I}, i$) = 0 means that the chip c_i is *unconnected*, and countChipCxn($\boldsymbol{I}, i$) > 0 indicates that the chip c_i is *connected*, where $\boldsymbol{I}$ represents the QCN interconnection matrix. Details of the method countChipCxn() can be found in Algorithm 2.
- The quantum chip distance matrix $\boldsymbol{D}$, of which the element $\boldsymbol{D}_{i,j}$ represents the shortest distance between quantum chips c_i and c_j. Two chips with a direct interconnection have distance 1. The full distance matrix $\boldsymbol{D}$ can be easily calculated based on the current topology of quantum chips using the Floyd-Warshall algorithm [51] (See Algorithm 3 for more details).
- The sorted remote connection intensity information I_s, where each element is a triplet (i, j, *degree*). Here i and j represent the indices of the quantum chips c_i and c_j corresponding to the partition blocks $\phi(i)$ and $\phi(j)$, respectively. The *degree* signifies the number of remote CNOT gates existing between the partition blocks $\phi(i)$ and $\phi(j)$. And I_s can be obtained by using the method SortPartitionBlockPairs(), which can be found in Algorithm 4.

The interconnection design algorithm adds connections between quantum chips iteratively. In each iteration, one interconnection is added to the QCN, once again, in a greedy way. Different partition block pairs can have different numbers of remote connections between the partition blocks in this pair. The remote connection intensity between two partition blocks can be used to sort all N_p partition block pairs. During each iteration of the algorithm, a pair of blocks (quantum chips) with the highest degree (the number of remote CNOT) is selected. Based on the connectivity of both quantum chips (c_i, c_j), there are three distinct scenarios, each corresponding to a strategy to add interconnections:

- **Both chips are unconnected:** An interconnection is directly added between these two chips. Then we use the method connectChip() to update the QCN interconnection matrix $\boldsymbol{I}$ and update $\boldsymbol{D}$ using the method updateDistanceMatrix() (Algorithm 3). The list of connected quantum chip pairs V is updated accordingly, as shown in line 7 of Algorithm 1.
- **Only one chip is unconnected:** If one of the quantum chips in the pair is unconnected, an interconnection needs to be added to incorporate this unconnected quantum chip into the QCN. To determine the optimal interconnection, a heuristic function is designed to evaluate all possible interconnections that could be established with the unconnected chip. Quantum chips that have

Algorithm 1: Physical Remote Connection Selection Algorithm

Input:
- k: number of quantum chips;
- S_{rc} (size: k): maximum number of interconnections supported by each chip;
- $\boldsymbol{M}$ (size: $k \times k$): partition block remote connection intensity matrix, which is symmetric.

Output: $\boldsymbol{I}$ (size: $k \times k$): QCN interconnection matrix, which is symmetric.

begin
  Initialization: $\boldsymbol{I} \leftarrow \mathbf{0}_{k\times k}$; $\boldsymbol{D} \leftarrow \mathbf{0}_{k\times k}$; $V \leftarrow \emptyset$; $I_s \leftarrow$ SortPartitionBlockPairs($\boldsymbol{M}$);
  for (i, j) in I_s **do**
    $Cost \leftarrow [\,]$;
    if ($\exists r$ for countChipCxn($\boldsymbol{I}, r$) $< S_{rc}[r]$) **then**
      if countChipCxn($\boldsymbol{I}, i$) $= 0$ ***and*** countChipCxn($\boldsymbol{I}, j$) $= 0$ **then**
        $V \leftarrow V \cup \{(i,\ j)\}$; $\boldsymbol{I} \leftarrow$ connectChip($\boldsymbol{I},\ i,\ j$); $\boldsymbol{D} \leftarrow$ updateDistanceMatrix($\boldsymbol{D}, i, j$);
      else if countChipCxn($\boldsymbol{I}, i$) $= 0$ ***or*** countChipCxn($\boldsymbol{I}, j$) $= 0$ **then**
        $cur \leftarrow$ countChipCxn($\boldsymbol{I}, i$) $= 0$? $i : j$;
        for ($tmp \in \{cand \mid$ countChipCxn($\boldsymbol{I}, cand$) $< S_{rc}[cand]$ and $cand \neq cur\}$) **do**
          $\boldsymbol{D'} \leftarrow$ updateDistanceMatrix($\boldsymbol{D},\ cur,\ tmp$); $V' \leftarrow V \cup \{(cur,\ tmp)\}$;
          $Cost[tmp] = H(\boldsymbol{M}, V', \boldsymbol{D'})$;
        end
        $chosen \leftarrow c$ where $Cost[c] = \min(Cost)$; $V \leftarrow V \cup \{(cur,\ chosen)\}$;
        $\boldsymbol{I} \leftarrow$ connectChip($\boldsymbol{I},\ cur,\ chosen$); $\boldsymbol{D} \leftarrow$ updateDistanceMatrix($\boldsymbol{D},\ cur,\ chosen$);
      else
        for $cur \in \{tmp \mid tmp \in \{i, j\}$ and $countChipCxn(\boldsymbol{I}, tmp) < S_{rc}[tmp]\}$ **do**
          for ($tmp \in \{cand \mid$ countChipCxn($\boldsymbol{I}, cand$) $< S_{rc}[cand]$ and $cand \neq cur\}$) **do**
            $V' \leftarrow V \cup \{(cur,\ tmp)\}$; $\boldsymbol{D'} \leftarrow$ updateDistanceMatrix($\boldsymbol{D},\ cur,\ tmp$);
            $Cost[tmp] = H(\boldsymbol{M}, V', \boldsymbol{D'})$;
          end
        end
        $chosen \leftarrow c$ where $Cost[c] = \min(Cost)$;
        if $Cost[chosen] > H(\boldsymbol{M}, V, \boldsymbol{D})$ **then**
          $V \leftarrow V \cup \{(cur,\ tmp)\}$; $\boldsymbol{I} \leftarrow$ connectChip($\boldsymbol{I},\ cur,\ chosen$);
          $\boldsymbol{D} \leftarrow$ updateDistanceMatrix($\boldsymbol{D},\ cur,\ chosen$);
        end
      end
    else
      break;
    end
  end
  return $\boldsymbol{I}$;
end

Algorithm 2: Count Quantum Chip Interconnections

Input:
- $\boldsymbol{I}$ (size: $k \times k$): partition block connection intensity matrix, which is symmetric;
- i: the index of quantum chip c_i.

Output: num_ic: The number of interconnections connected to quantum chip c_i.

```
begin
    num_ic = 0;
    for j ← 0 to k − 1 do
        num_ic += I[i][j];
    end
    return num_ic;
end
```

Algorithm 3: Update Quantum Chip Distance Matrix

Input:
- $\boldsymbol{D}$ (size: $k \times k$): quantum chip distance matrix, which is symmetric;
- i: the index of quantum chip c_i;
- j: the index of quantum chip c_j.

Output: $\boldsymbol{D}_{updated}$ (size: $k \times k$): the updated quantum chip distance matrix, which is symmetric.

```
begin
    D[i][j] = 1; D[j][i] = 1;
    D_updated ← FloydWarshall(D);
    return D_updated;
end
```

Algorithm 4: Sorting Partition Block Pairs

Input:
- $\boldsymbol{M}$ (size: $k \times k$): partition block connection intensity matrix, which is symmetric.

Output: I_s: sorted partition block pair remote connection intensity result.

```
begin
    I_tmp ← ∅;
    for i ← 0 to k − 1 do
        for j ← 0 to i − 1 do
            I_tmp ← I_tmp + {(i, j, M[i][j])};
        end
    end
    I_s ← sort I_tmp using the 3rd value in the tuple as the key;
    return I_s;
end
```

already reached their interconnection capacity constraints are excluded from consideration. Subsequently, the one with the minimal cost is selected from the remaining candidate interconnections. This strategic approach ensures that the integration of the new chip is done to optimize the overall connectivity and efficiency of QCN while adhering to the established constraints. Afterward, V, $\boldsymbol{I}$, and $\boldsymbol{D}$ will be updated. Further details of the heuristic function are provided in equation (4), with the corresponding information in lines 9 to 15 of Algorithm 1.

- **Both chips are connected:** When both quantum chips in a pair are connected, the algorithm requires the selection of two quantum chips from those previously utilized but have not yet reached their interconnection capacity constraints. Similarly, it employs the heuristic function to evaluate all possible interconnections. It chooses the interconnection with the smallest evaluation and also yields a positive gain to establish a link between the two quantum chips that are not yet linked. Then V, $\boldsymbol{I}$, and $\boldsymbol{D}$ will be updated, and the details are shown in lines 17 to 27 of Algorithm 1.

The algorithm goes through a series of steps to update the topology of the QCN. Once all partition block pairs have been visited, the search process terminates. The final output is a structured topology that defines the interconnections within the QCN.

The heuristic cost function mentioned above is formally defined as follows:

$$H(\boldsymbol{M}, V, \boldsymbol{D}) = \sum_{(i,j)\in V} \boldsymbol{M}[i][j] * (2 * \boldsymbol{D}[i][j] - 1) \tag{4}$$

Here, $\boldsymbol{M}$ represents the partition block remote connection intensity matrix, V represents the list of connected quantum chip pairs, and $\boldsymbol{D}$ represents the quantum chip distance matrix. In a QCN, when a remote CNOT gate is located on adjacent quantum chips, by default we use the *Cat-Comm* pattern, which consumes one EPR pair. When a remote CNOT gate spans multiple quantum chips, we use a fusion of the *TP-Comm* pattern and the *Cat-Comm* pattern. For example, for a remote CNOT gate $g_{rm}(q_1, q_2)$, where virtual qubits q_1 and q_2 are allocated to quantum chips c_1 and c_2 respectively, with the shortest distance of two between c_1 and c_2, executing this remote CNOT gate requires consuming three EPR pairs. This is because we first need to use the *TP-Comm* pattern to propagate the state of q_1 from c_1 to the intermediate quantum chip between c_1 and c_2, which consumes one EPR pair. Then, we implement $g_{rm}(q_1, q_2)$ using the *Cat-Comm* pattern, and finally consume one EPR pair using the *TP-Comm* pattern to transfer the state of q_1 back to c_1. In summary, we can use the number $2 * \boldsymbol{D}[i][j] - 1$ to estimate the number of consumed EPR pairs when implementing a remote CNOT gate in a QCN.

3.5. Time and Space Complexities Analysis

We assume N used virtual qubits and E two-qubit gates are in the input quantum program. And we assume that the number of quantum chips required is M, the maximum qubit scale of the quantum chip is m, and the total number of iterations is N_{iter}. In the *CPA* algorithm, the greedy generation of the initial partition first requires traversing each two-qubit gate in the quantum program to obtain the usage frequency of each virtual qubit. This process has a time complexity of $O(E)$. Next, the qubits need to be sorted in $O(N \log N)$. Therefore, the time complexity of producing the initial partition is $O(E + N \log N)$. The space complexity is $O(N)$ because it requires storing the initial partition. During the iterative search process of the partitioning algorithm, each iteration requires traversing $\frac{\delta M(M-1)m^2}{2}$ neighboring partition solutions. Since $N \leq mM$, the time complexity of the partitioning algorithm is $O(E+N \log N+\frac{\delta N_{iter} M(M-1)m^2}{2}) \approx O(E+XN+C_1 m^2 M^2)$, where $C_1 = \frac{\delta N_{iter}}{2}$ and $X = logN - C_1 m$. And

the space complexity is $O(C_2 m^2 N M^2)$, where $C_2 = \frac{\delta}{2}$, so as to store neighboring solutions information. For the physical remote connection selection algorithm, its time complexity is $O(M^2)$. Its space complexity is $O(M^2)$ as it needs to store the shortest path in $O(M^2)$, the distance matrix in $O(M^2)$, and the evaluation of each candidate quantum chip that can be connected to the current quantum chip in $O(M)$.

In summary, since the *CPA* algorithm and the physical remote connection selection algorithm are executed sequentially, the time complexity of the *AutoArch* framework is $O(E + XN + C_1 m^2 M^2)$, and the space complexity is $O(C_2 m^2 N M^2)$.

4. Evaluation

To validate our proposed application-specific architecture design framework, we conduct experiments across various benchmarks.

4.1. Experiment Setup

Benchmarks: We select eleven benchmark programs from QASMBench [52] and MQTBench [53], which cover several typical application domains (such as simulation and arithmetic) with various sizes (from 64 to 130 qubits).

Device Topology: For our experimental evaluation, we have chosen the topology of the IBM quantum processor named *ibmq_guadalupe* [54] (connectivity shown in Fig. 4(a)) to construct QCNs. Besides, each processor is assumed to support at most three inter-chip connections. Qubits Q_0, Q_6, and Q_9 are selected as boundary data qubits, with each connected to two communication qubits.

Metrics: We employ two metrics to evaluate the efficiency of a DQC architecture: the number of EPR pairs consumed and the number of local SWAP insertions when executing quantum algorithms. When a quantum program is compiled targeting a specific DQC architecture, lower resulted EPR pair consumption and local SWAP gate insertions indicate higher quality of the DQC architecture. In our experiment, we repeat the SABRE algorithm for each benchmark 10 times, and use the geometric mean to calculate the average ratio of the EPR pairs and the additional local SWAP gates between the proposed architecture and the baseline.

Compilation Configuration: In SABRE, The front layer (F) represents the set of independent two-qubit gates being processed in each iteration of the SABRE algorithm. The extended layer (ES) is the collection of two-qubit gates considered at deeper levels in the circuit. The weight of ES signifies the impact of gates in ES when evaluating the objective function used by SABRE. The size and weight of ES are configured as 0.5 and $10 \times |F|$, respectively.

4.2. Experiment Design

To evaluate the performance of *AutoArch*, three experimental configurations are created, which address the following questions:

1. When employing *CPA* to process quantum circuits, does carefully designed initial partition accelerate the process of finding a solution close to the optimum?

2. Can the Partial Neighbor Solution strategy introduced in the *CPA* aid in finding a solution that is closer to the optimum?

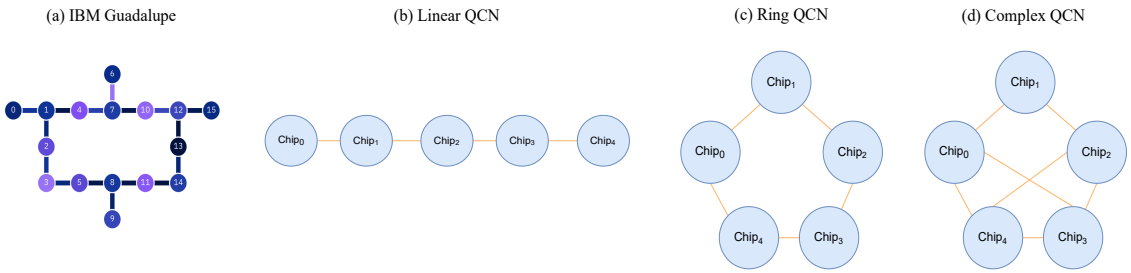

Fig. 4. The superconducting quantum processor used to build QCNs as well as QCNs with different architectures. (a) The device topologies of *ibmq_guadalupe* quantum processor. Nodes represent individual physical qubits, while edges depict the connectivity between pairs of qubits. (b) A QCN consisting of five quantum chips with linear architecture. (c) A QCN consisting of five quantum chips with ring architecture. (d) A QCN consisting of five quantum chips with more complex architecture.

3. Can *AutoArch* generate a corresponding QCN on which quantum algorithms execute with a better performance?

Addressing the first question, we use two types of initial partitions for the *CPA*, randomly generated or carefully designed. We then compare their effectiveness by couting the number of iterations of *CPA* and the quantity of generated remote CNOT gates.

Addressing the second question, we examine the impact of the neighborhood solution ratio parameter δ on the quantum circuit partitioning results of the partitioning algorithm. The impact is assessed using two metrics: the number of generated remote CNOT gates and the number of search iterations required to achieve the minimum number of remote CNOT gates. The value of δ determines the scale of neighboring solutions accessed by the algorithm at each iteration. A higher value of δ implies that the partitioning algorithm accesses larger-scale neighboring solutions. When $\delta = 1$, the partitioning algorithm accesses all neighboring solutions of the current solution at each iteration. In this experiment, we set δ to be 0.25, 0.5, 0.75, and 1, respectively.

Addressing the third question, four different distributed architectures are chosen as the compilation target for compiling quantum algorithms using SABRE [40]. Among them, three architectures are shown in Fig. 4(b) - 4(d), and the fourth architecture is a customized architecture generated by *AutoArch*. Every node in these architectures is an *ibm_guadalupe* chip. For the sake of fairness, each chip in all architectures is assumed to support at most three remote connections. The complex QCN is constructed so as to support as many interconnections between chips as possible in an architecture with five chips.

4.3. The Efficiency of Deliberately Designed Initial Partition

Fig. 5 shows the iterative search of the *CPA* on eleven benchmarks. We set the maximum iteration search count to 1000. The X-axis represents the distribution of iteration counts. Due to the varying characteristics of each benchmark, the number of iterations required by the partitioning algorithm to obtain the optimal partitioning result differs. The Y-axis represents the number of remote CNOT gates produced by the *CPA*. The blue solid line represents the trajectory of the algorithm when it utilizes a randomly initialized quantum circuit partition as the starting solution. Conversely, the green solid line depicts the trajectory of the algorithm when it employs a deliberately designed initial quantum circuit partition as the starting solution. The lower number of remote CNOT gates indicates the better partitioning of quantum circuits. Table 1 shows the minimum number of remote CNOT gates obtained from processing

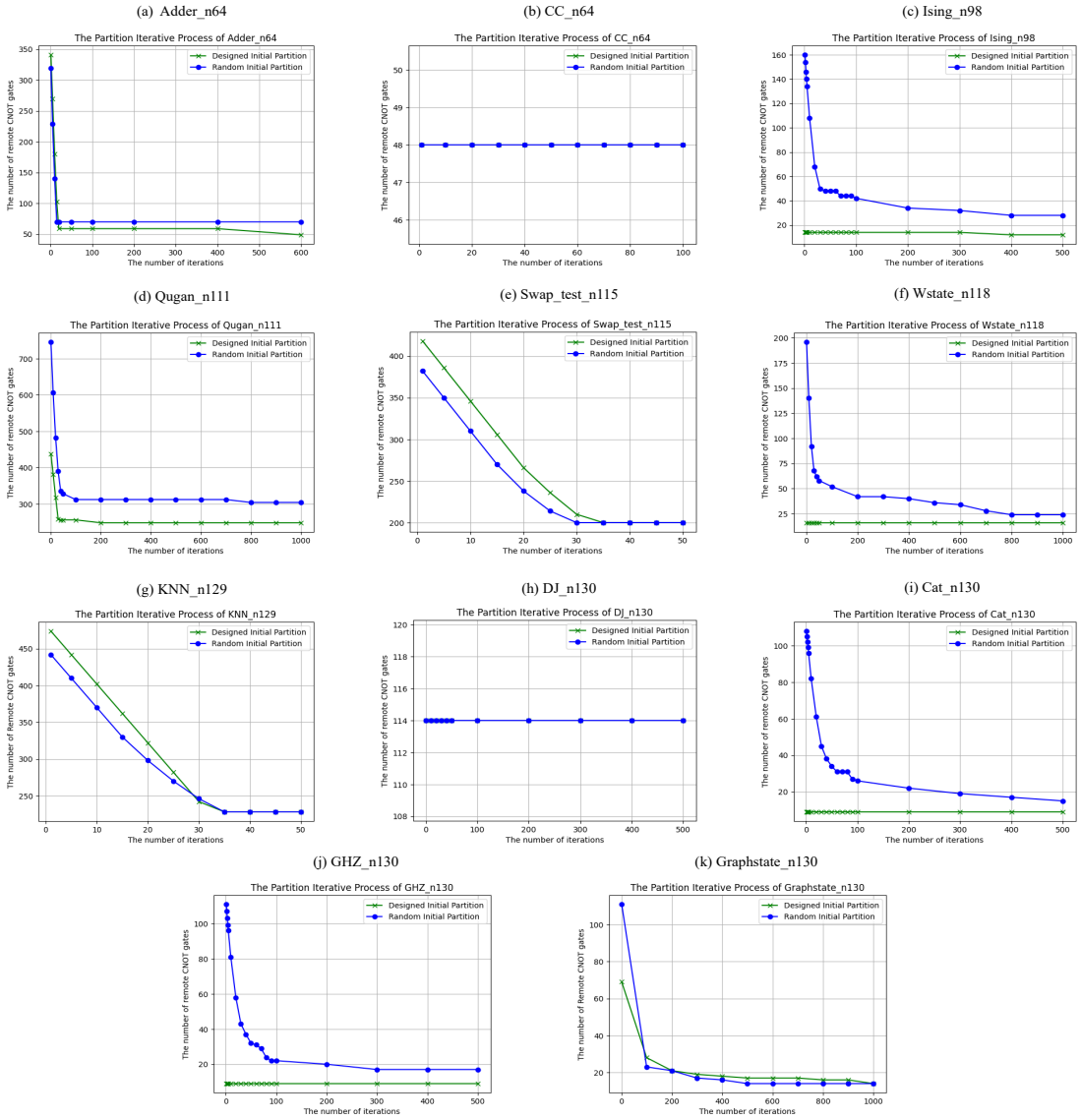

Fig. 5. The performance comparison of partitioning algorithm using random and designed initial partition strategies. (a)-(k) show the iterative processing of eleven quantum algorithms using random and designed initial partitions within the partitioning algorithm.

each benchmark by the *CPA* with random or designed initial partition (with 1000 iterations), as well as the minimum iteration search count required to find the minimum number of remote CNOT gates.

For *Adder_n64* (Fig. 5(a)), *Ising_n98* (Fig. 5(c)), *Qugan_n111* (Fig. 5(d)), *Wstate_n118* (Fig. 5(f)), *Cat_n130* (Fig. 5(i)), and *GHZ_n130* (Fig. 5(j)), we can observe that when our algorithm uses a deliberately designed initial partition as the starting point, it converges to the optimal partitioning result more quickly and finds a better quantum circuit partitioning result compared to using a random initial partition. As shown in Table 1, the number of remote CNOT gates is reduced by 50% in the best case. And the partitioning algorithm with $Init\_Partition_{designed}$ performs the iterative search 855× faster than the partitioning algorithm with $Init\_Partition_{random}$. For *CC_n64* (Fig. 5(b)), *Swap_test_n115* (Fig. 5(e)), *KNN_n129* (Fig. 5(g)) and *Graphstate_n130* (Fig. 5(k)), although the partitioning algorithm using a deliberately designed initial partition does not significantly accelerate the search for the optimal result, the optimal results obtained are the same.

Table 1. Under a maximum iteration count of 1000, the minimum number of remote CNOT gates resulted from processing the benchmark by the *CPA* algorithm, as well as the iteration count required to achieve this minimum number of remote CNOT gates.

Benchmark	#Qubits	$Init\_Partition_{random}$		$Init\_Partition_{designed}$		Comparison	
		$remote\_CNOT_{total}$	$iter\_num_{min}$	$remote\_CNOT_{total}$	$iter\_num_{min}$	$\Delta remote\_CNOT_{total}$	$i_{random}/i_{designed}$
Adder_n64	64	70	15	49	553	**30%**	0.02
CC_n64	64	48	1	48	1	0.00%	1.00
Ising_n98	98	24	724	12	321	**50.00%**	2.25
Qugan_n111	111	304	760	248	168	**18.42%**	4.52
Swap_test_n115	115	200	29	200	33	0.00%	0.87
Wstate_n118	118	24	775	16	1	**33.33%**	775
KNN_n129	129	228	35	228	33	0.00%	1.06
DJ_n130	130	114	1	114	1	0.00%	1.00
Cat_n130	130	13	718	9	1	**30.77%**	718
GHZ_n130	130	14	885	9	1	**35.71%**	885
Graphstate_n130	130	14	403	14	988	0.00%	0.41

$\Delta remote\_CNOT_{total}$ is the percentage change in the number of totally generated remote CNOT gates: $\Delta remote\_CNOT_{total}$ = 1 - $remote\_CNOT_{total}(designed)/remote\_CNOT_{total}(random)$. $i_{random}/i_{designed}$ is the ratio between the minimum iteration count required to achieve the minimum number of remote CNOT gates of the partitioning algorithm with $Init\_Partition_{designed}$ and $Init\_Partition_{random}$: $i_{random}/i_{designed} = iter\_num_{min}(designed)/iter\_num_{min}(random)$.

To summarize, for many quantum programs, employing our custom initial partition strategy as a part of partitioning algorithm enables it to acquire better results swiftly. However, due to the diverse characteristics of different quantum programs, we cannot develop a universally applicable strategy that consistently outperforms others across all quantum programs. In other words, different quantum programs may require specific initial partition strategies to achieve efficient partitioning. The initial partition strategy proposed in this paper represents just one exploration in this direction.

4.4. The Effectiveness of the Partial Neighbor Solution Strategy

Fig. 6 shows the iterative search process of the *CPA* handling eleven benchmarks, with the partial neighbor resolution parameter δ set to 0.25, 0.5, 0.75, and 1 respectively, and the maximum iteration search count set to 1000. The X-axis represents the distribution of iteration counts. The Y-axis represents the number of remote CNOT gates produced by the *CPA*. The solid lines of different colors represent the iterative search trajectories of the *CPA* when it utilizes designed initial partitions as the initial solution. Table 2 shows the minimum number of remote CNOT gates obtained from processing each benchmark by the *CPA* with designed initial partition (with 1000 iterations), as well as the minimum iteration search count required to find the minimum number of remote CNOT gates.

Table 2. The default iteration count of the *CPA* algorithm is set to 1000. When the neighborhood solution ratio parameter δ is set to 0.25, 0.5, 0.75, and 1.0, respectively, it determines the minimum number of remote CNOT gates obtained from processing the benchmarks, as well as the time cost.

Benchmark	#Qubits	$\delta = 0.25$		$\delta = 0.5$		$\delta = 0.75$		$\delta = 1.0$		$t_{\delta=1}/t_{\delta_{opt}}$
		$remote\_CNOT_{total}$	$runtime(s)$	$remote\_CNOT_{total}$	$runtime(s)$	$remote\_CNOT_{total}$	$runtime(s)$	$remote\_CNOT_{total}$	$runtime(s)$	
Adder_n64	64	60	228.14	71	452.86	37	679.64	37	904.94	1.33
CC_n64	64	48	224.81	48	449.34	48	673.83	48	914.28	4.07
Ising_n98	98	14	249.32	14	495.52	14	747.29	14	994.07	3.99
Qugan_n111	111	248	300.98	248	598.29	248	896.88	248	1201.49	3.99
Swap_test_n111	115	200	281.60	200	560.54	200	838.67	200	1116.92	3.97
Wstate_n118	118	16	264.54	16	525.26	16	789.90	16	1055.46	3.99
KNN_n129	129	228	294.86	228	586.96	228	882.96	228	1181.63	4.01
DJ_n130	130	114	280.78	114	566.22	114	842.14	114	1128.37	4.02
Cat_n130	130	9	272.76	9	544.02	9	817.51	9	1089.64	3.99
GHZ_n130	130	9	271.97	9	545.38	9	816.95	9	1089.01	4.00
Graphstate_n130	130	12	274.68	10	549.91	11	824.17	10	1099.11	2.00
Geometric mean										3.40

$t_{\delta=1}/t_{\delta_{opt}}$ is the ratio between the iteration search time of partitioning algorithm with $\delta = 1$ and δ_{opt}, where $\delta_{opt}(\delta_{opt} \neq 1)$ represents the value of δ corresponding to the minimum number of remote CNOT gates found during the algorithm search: $t_{\delta=1}/t_{\delta_{opt}} = runtime(\delta = 1)/runtime(\delta_{opt})$.

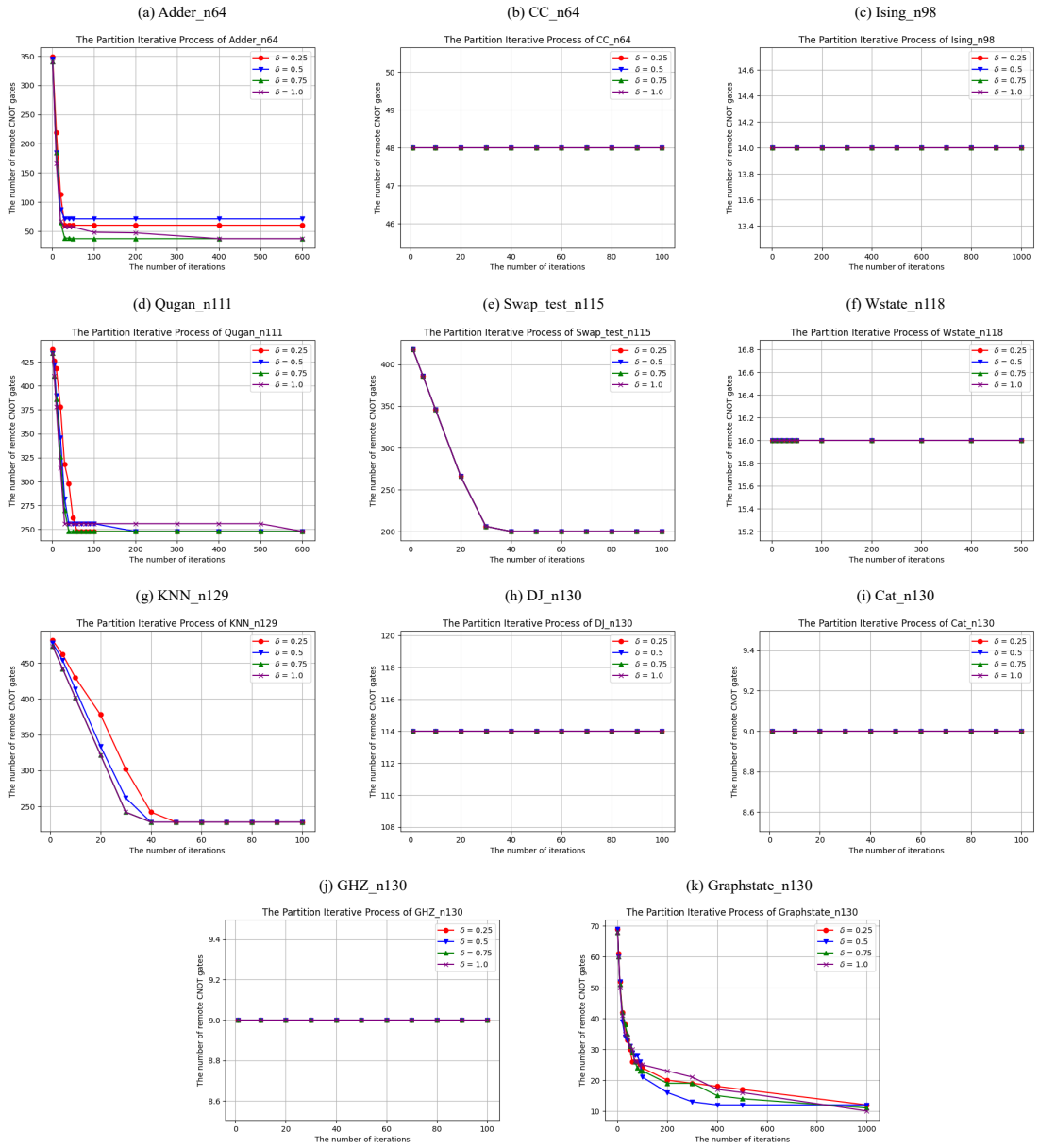

Fig. 6. The performance comparison of the partitioning algorithm with the partial neighbor resolution ratio parameter δ configured to different values. (a)-(k) show the iterative processing of eleven quantum algorithms using the partitioning algorithm with different δ values.

Table 3. Number of estimated usage of EPR pairs and additional local swap gates of SABRE on four different architectures (linear, ring, complex, and customized) of QCNs.

Benchmark	#Qubits	QCN infrastructure	*SABRE+QCN(linear)*		*SABRE+QCN(ring)*		*SABRE+QCN(complex)*		*SABRE+QCN(customized)*		customized v.s. linear		customized v.s. ring		customized v.s. complex	
			EPR_{use}	$SWAP_{add}$	EPR_{use}	$SWAP_{add}$	EPR_{use}	$SWAP_{add}$	EPR_{use}	$SWAP_{add}$	ΔEPR_{use}	$\Delta SWAP_{add}$	ΔEPR_{use}	$\Delta SWAP_{add}$	ΔEPR_{use}	$\Delta SWAP_{add}$
Adder_n64	64	4*Guadalupe(16)	90.0	394.1	78.2	259.9	106.0	255.1	62.5	241.9	**30.56%**	**38.62%**	**20.08%**	**6.93%**	**41.04%**	**5.17%**
CC_n64	64	4*Guadalupe(16)	24.8	127.3	25.6	117.2	29.0	91.4	14.7	110.2	**40.73%**	**13.43%**	**42.58%**	**5.97%**	**49.31%**	-20.57%
Cat_n65	65	5*Guadalupe(16)	48.9	190.3	50.7	182.6	44.7	133.5	6.4	51.2	**86.91%**	**73.1%**	**87.38%**	**71.96%**	**85.68%**	**61.65%**
Ising_n66	66	5*Guadalupe(16)	34.0	111.3	33.2	99.3	35.8	76.9	11.8	35.6	**65.29%**	**68.01%**	**64.46%**	**64.15%**	**67.04%**	**53.71%**
KNN_n67	67	5*Guadalupe(16)	66.2	234.8	61.9	192.3	63.3	152.3	49.9	176.7	**24.62%**	**24.74%**	**19.39%**	**8.11%**	**21.17%**	-16.02%
qugan_n71	71	5*Guadalupe(16)	120.6	433.9	106.8	362.5	130.1	318.0	98.2	275.3	**18.57%**	**36.55%**	**8.05%**	**24.06%**	**24.52%**	**13.43%**
Wstate_n76	76	5*Guadalupe(16)	76.2	307.2	82.4	298.5	75.0	219.7	16.2	96.1	**78.74%**	**68.72%**	**80.34%**	**67.81%**	**78.4%**	**56.26%**
GHZ_n78	78	5*Guadalupe(16)	70.6	275.8	53.3	201.4	57.6	173.1	8.6	64.1	**87.82%**	**76.76%**	**83.86%**	**68.17%**	**85.07%**	**62.97%**
Ising_n98	98	7*Guadalupe(16)	66.4	242.5	63.2	207.7	60.0	141.3	16.2	55.5	**75.6%**	**77.11%**	**74.37%**	**73.28%**	**73.0%**	**60.72%**
qugan_n111	111	7*Guadalupe(16)	231.3	882.9	207.7	700.5	229.5	629.4	94.1	375.9	**59.32%**	**57.42%**	**54.69%**	**46.34%**	**59.0%**	**40.28%**
SWAP_test_n115	115	8*Guadalupe(16)	163.1	614.1	131.2	457.3	149.6	372.7	115.1	388.1	**29.43%**	**36.8%**	**12.27%**	**15.13%**	**23.06%**	-4.13%
Wstate_n118	118	8*Guadalupe(16)	250.4	974.3	187.1	684.5	180.3	542.9	27.7	172.3	**88.94%**	**82.32%**	**85.2%**	**74.83%**	**84.64%**	**68.26%**
GHZ_n127	127	8*Guadalupe(16)	170.9	676.7	158.8	603.6	131.7	403.0	13.1	96.8	**92.33%**	**85.7%**	**91.75%**	**83.96%**	**90.05%**	**75.98%**
KNN_n129	129	9*Guadalupe(16)	258.6	904.9	184.2	648.1	190.5	481.3	125.0	421.9	**51.66%**	**53.38%**	**32.14%**	**34.9%**	**34.38%**	**12.34%**
DJ_n130	130	9*Guadalupe(16)	485.7	1913.8	381.1	1469.9	410.9	1225.7	327.2	1078.9	**32.63%**	**43.63%**	**14.14%**	**26.6%**	**20.37%**	**11.98%**
Cat_n130	130	9*Guadalupe(16)	200.2	777.3	172.0	655.9	149.7	430.7	15.5	104.9	**92.26%**	**86.5%**	**90.99%**	**84.01%**	**89.65%**	**75.64%**
GHZ_n130	130	9*Guadalupe(16)	225.8	870.8	184.7	682.0	145.5	417.8	16.7	105.1	**92.6%**	**87.93%**	**90.96%**	**84.59%**	**88.52%**	**74.84%**
Graphstate_n130	130	9*Guadalupe(16)	273.1	1163.6	131.5	529.6	101.2	289.0	50.1	165.0	**81.66%**	**85.82%**	**61.9%**	**68.84%**	**50.49%**	**42.91%**

ΔEPR_{use} is the percentage change in estimated EPR pair overhead, and $\Delta SWAP_{add}$ is the percentage change in additional local SWAP gates. In the case of customized v.s. linear, $\Delta EPR_{use} = 1 - ERP_{use}(customized)/EPR_{use}(linear)$, and $\Delta SWAP_{add} = 1 - SWAP_{add}(customized)/SWAP_{add}(linear)$.

In Fig. 6 and Table 2, we use $\delta = 1$ as the comparative baseline. For some benchmarks, when δ is less than 1, the optimal obtained partitioning results in fewer remote CNOT gates. In other words, the search speed is faster and the partitioning results are better. Therefore, it can be seen that using the partial neighbor solution strategy in the *CPA* is effective. As the value of δ increases, the search time overhead of algorithm also increases. As demonstrated in Table 2, when δ is 0.25, 0.5, or 0.75, the partitioning algorithm achieves the same or even superior partitioning results with a time overhead averaging 3.4× faster compared to the case of δ being 1.

4.5. Overall Improvement

Table 3 shows the number of additional local SWAP gates and the estimated overhead of EPR pairs for quantum circuits compiled by the SABRE algorithm across all benchmarks. As indicated in the *QCN infrastructure* column, benchmarks of varying qubit scales require varying numbers of individual quantum processors. The column EPR_{use} shows the estimated overhead of EPR pairs when a compiled quantum program is executed on a QCN, and column $SWAP_{add}$ represents the total number of additional local SWAP gates. The ΔEPR_{use} under columns *customized v.s. linear*, *customized v.s. ring* and *customized v.s. complex* is the change percentage of estimated EPR pair overhead, and $\Delta SWAP_{add}$ is the change percentage of additional local SWAP gates. For example, in the *customized v.s. linear* column, $\Delta EPR_{use} = 1 - ERP_{use}(customized)/EPR_{use}(linear)$, and $\Delta SWAP_{add} = 1 - SWAP_{add}(customized)/SWAP_{add}(linear)$.

Since the SABRE algorithm incorporates re-synthesis, we use it to transpile each benchmark ten times and get the average value. For all of the benchmarks, the estimated EPR pairs overhead of *SABRE+QCN(customized)* is lower compared to *SABRE+QCN(linear)*, *SABRE+QCN(ring)* and *SABRE +QCN(complex)*. Furthermore, for most benchmarks, the additional local SWAP gates are also lower than the other three scenarios. When comparing *SABRE+QCN(customized)* and *SABRE+QCN(linear)*, the geometric mean of ΔEPR_{use} and $\Delta SWAP_{add}$ are 55.91% (with a maximum improvement of 92.6%) and 55.04% (with a maximum improvement of 87.93%). When comparing *SABRE+QCN(customized)* and *SABRE+QCN(ring)*, the geometric mean of ΔEPR_{use} and $\Delta SWAP_{add}$ are 44.46% (with a maximum improvement of 91.75%) and 37.84% (with a maximum improvement of 84.59%). In instances where enhancements are observed upon comparing *SABRE+QCN(customized)* and *SABRE+QCN(complex)*, the

geometric mean of ΔEPR_{use} and $\Delta SWAP_{add}$ are 52.29% (with a maximum improvement of 90.05%) and 37.26% (with a maximum improvement of 75.98%).

For most benchmarks, when compiled on DQC architectures designed by *AutoArch* with SABRE, there are fewer estimated EPR pairs overhead and fewer inserted local SWAP gates compared to other common DQC architectures. This is because the initial qubit placement obtained by SABRE, which takes into account the partition block allocation information provided by *AutoArch*, can ensure that the occurrence of remote quantum gates is minimized as much as possible. At the same time, the DQC architectures customized by *AutoArch* can reduce cross-chip remote quantum communication in some degree. For *CC_n64*, *KNN_n67* and *SWAP_test_n115*, although *SABRE+QCN(customized)* introduces a greater number of local SWAP gates, the benefits of reducing the number of EPR pairs far outweigh the increased overhead from the additional local SWAP gates (the overhead of remote CNOT and SWAP gates far surpasses that of local SWAP gates). For instance, in the case of *SWAP_test_n115*, despite the inclusion of 15.4 extra local SWAP gates, there is a significant reduction of 34.5 in the estimated usage of EPR pairs.

5. Related Work and Discussion

Most existing works [25–28] focus on optimizing various factors within general DQC architectures. Smith *et al.* [25] consider the optimization of frequency collision in the DQC hardware design. Ang *et al.* [26] focus on the design of DQC systems by integrating various factors such as internode links, entanglement distillation, and local architecture. Lin *et al.* [27] design customized DQC architectures for quantum error correction codes. Zhang *et al.* [28] propose a mechanism for program concurrency for DQC systems. In summary, these studies do not consider the design optimizations at the architectural level of DQC systems. However, there is substantial design space for customized DQC architectures tailored for quantum algorithm execution.

This paper primarily explores the design space of specific DQC architectures for quantum algorithms. Although the customized DQC architectures designed by our framework are able to significantly enhance the execution quality of quantum algorithms, there is still much room left for potential improvements. Our framework mainly focuses on the connections at the quantum chip level during the design of DQC architecture. Some factors have not been considered, such as the selection of qubits to serve as endpoints for remote physical connections between quantum chips, the customization of the number of communication qubits for configuring remote physical connections, and frequency collision, etc. If these factors are collectively taken into account in the design process of DQC architectures, it will further enhance the actual execution effect of quantum algorithms.

6. Conclusion

The efficiency of executing quantum programs on DQC is greatly affected by their architectures. In this paper, we explore an application-specific DQC architecture design framework to achieve lower overhead of EPR pairs and fewer additional local SWAP gates. To achieve this, we have proposed an efficient architecture design framework. The characteristic information of quantum programs is first extracted through constrained quantum circuit partitioning. Then, a heuristic algorithm is developed to design the interconnections of a QCN based on the above characteristic information. Experiments

indicate that in the quantum program compilation results obtained by SABRE based on the customized DQC architecture, the estimated EPR pair overhead and the number of additional local SWAP gates are significantly lower than those of the other three general-purpose DQC architectures.

Acknowledgment

Deng was supported by the National Key R&D Program of China under Grant No. 2023YFA1009403, the National Natural Science Foundation of China under Grant No. 62072176, and the "Digital Silk Road" Shanghai International Joint Lab of Trustworthy Intelligent Software under Grant No. 22510750100.

Reference

[1] Nielsen M A and Chuang I L 2010 *Quantum computation and quantum information* (Cambridge university press)

[2] Lloyd S 1996 *Science* **273** 1073–1078

[3] Xin T, Wang B X, Li K R, Kong X Y, Wei S J, Wang T, Ruan D and Long G L 2018 *Chin. Phys. B* **27** 020308

[4] Shor P W 1994 Algorithms for quantum computation: discrete logarithms and factoring *Proceedings of 35th Annual Symposium on Foundations of Computer Science* pp 124–134

[5] Deutsch D E 1989 *Proceedings of the royal society of London. A. mathematical and physical sciences* **425** 73–90

[6] Svore K M, Aho A V, Cross A W, Chuang I and Markov I L 2006 *Computer* 74–83

[7] Barenco A, Bennett C H, Cleve R, DiVincenzo D P, Margolus N, Shor P, Sleator T, Smolin J A and Weinfurter H 1995 *Phys. Rev. A* **52** 3457

[8] Fowler A G, Devitt S J and Hollenberg L C 2004 *arXiv:0402196v1 [quan-ph]*

[9] Siraichi M Y, Santos V F d, Collange C and Pereira F M Q 2018 *Proceedings of the 2018 International Symposium on Code Generation and Optimization* 113–125

[10] Kim Y, Eddins A, Anand S, Wei K X, Van Den Berg E, Rosenblatt S, Nayfeh H, Wu Y, Zaletel M, Temme K and Kandala A 2023 *Nature* **618** 500–505

[11] Kelly J, Barends R, Fowler A, Megrant A, Jeffrey E, White T, Sank D, Mutus J, Campbell B, Chen Y and Chen Z 2015 *Nature* **519** 66

[12] McKay D C, Hincks I, Pritchett E J, Carroll M, Govia L C and Merkel S T 2023 *arXiv:2311.05933v1 [quant-ph]*

[13] Castelvecchi D 2023 *Nature* **624** 238–238

[14] Song C, Xu K, Li H K, Zhang Y R, Zhang X, Liu W X, Guo Q J, Wang Z, Ren W H, Hao J, Feng H, Fan H, Zheng D N, Wang D W and Zhu S Y 2019 *Science* **365** 574–577

[15] Wu Y L, Bao W S, Cao S R, Chen F S, Chen M C, Chen X W, Chung T H, Deng H, Du Y J, Fan D J, Gong M, Guo C, Guo C, Guo S J, Han L C, Hong L Y, Huang H L, Huo Y H, Li L P, Li N, Li S *et al.* 2021 *Phys. Rev. Lett.* **127** 180501

[16] Zhu Q, Cao S, Chen F, Chen M C, Chen X W, Chung T H, Deng H, Du Y J, Fan D J, Gong M, Guo C, Guo C, Shao-Jun G, Han L C, Hong L Y, Huang H L, Huo Y H, Li L P, Li N, Li S W, Li Y, Liang F T *et al.* 2022 *Science bulletin* **67** 240–245

[17] Shi Y H, Liu Y, Zhang Y R, Xiang Z, Huang K, Liu T, Wang Y Y, Zhang J C, Deng C L, Liang G H, Mei Z Y, Li H, Li T M, Ma W G, Liu H T, Chen C T, Liu T, Tian Y, Song X, Zhao S P, Xu K, Zheng D, Nori F and Fan H 2023 *Phys. Rev. Lett.* **131** 080401

[18] Xu H Z, Zhuang W F, Wang Z A, Huang K X, Shi Y H, Ma W G, Li T M, Chen C T, Xu K, Feng Y L *et al.* 2024 *Chin. Phys. B* **33** 050302

[19] BAQIS Large-scale quantum cloud computing cluster released with upgraded capability URL http://en.baqis.ac.cn/news/detail/?cid=2005

[20] Einstein A, Podolsky B and Rosen N 1935 *Phys. Rev.* **47** 777

[21] Zhong Y, Chang H S, Bienfait A, Dumur É, Chou M H, Conner C R, Grebel J, Povey R G, Yan H, Schuster D I and Cleland A N 2021 *Nature* **590** 571–575

[22] Yan H, Zhong Y, Chang H S, Bienfait A, Chou M H, Conner C R, Dumur É, Grebel J, Povey R G and Cleland A N 2022 *Phys. Rev. Lett.* **128** 080504

[23] Niu J, Zhang L, Liu Y, Qiu J, Huang W, Huang J, Jia H, Liu J, Tao Z, Wei W, Zhou Y X, Zou W J, Chen Y Z, Deng X W, Deng X H, Hu C K, Hu L, Li J, Tan D, Xu Y, Yan F, Yan T X Y, Liu S, Zhong Y P, Cleland A N and Yu D P 2023 *Nature Electronics* **6** 235–241

[24] Wu A B, Zhang H Z, Li G G, Shabani A, Xie Y and Ding Y F 2022 *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)* 1027–1041

[25] Smith K N, Ravi G S, Baker J M and Chong F T 2022 *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)* 1092–1109

[26] Ang J, Carini G, Chen Y, Chuang I, DeMarco M A, Economou S E, Eickbusch A, Faraon A, Fu K M, Girvin S M *et al.* 2022 *arXiv:2212.06167v1 [quant-ph]*

[27] Lin S F, Viszlai J, Smith K N, Ravi G S, Yuan C, Chong F T and Brown B J 2024 *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* 216–231

[28] Zhang H Z, Yin K Y, Wu A B, Shapourian H, Shabani A and Ding Y F 2024 *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* 699–714

[29] Baker J M, Duckering C, Hoover A and Chong F T 2020 *Proceedings of the 17th ACM International Conference on Computing Frontiers* 98–107

[30] Ferrari D, Cacciapuoti A S, Amoretti M and Caleffi M 2021 *IEEE Transactions on Quantum Engineering* **2** 1–20

[31] Wu A B, Ding Y F and Li A 2023 *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture* 479–493

[32] Nikahd E, Mohammadzadeh N, Sedighi M and Zamani M S 2021 *Physica Scripta* **96** 035102

[33] Dadkhah D, Zomorodi M, Hosseini S E, Plawiak P and Zhou X 2022 *IEEE Access* **10** 70329–70341

[34] Andres-Martinez P and Heunen C 2019 *Phys. Rev. A* **100** 032308

[35] Daei O, Navi K and Zomorodi-Moghadam M 2020 *International Journal of Theoretical Physics* **59** 3804–3820

[36] Davarzani Z, Zomorodi-Moghadam M, Houshmand M and Nouri-Baygi M 2020 *Quantum Information Processing* **19** 1–18

[37] Cuomo D, Caleffi M and Cacciapuoti A S 2020 *IET Quantum Communication* **1** 3–8

[38] Wootters W K and Zurek W H 1982 *Nature* **299** 802–803

[39] Bennett C H, Brassard G, Crépeau C, Jozsa R, Peres A and Wootters W K 1993 *Phys. Rev. Lett.* **70** 1895

[40] Li G S, Ding Y F and Xie Y 2019 *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* 1001–1014

[41] Zulehner A, Paler A and Wille R 2018 *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **38** 1226–1236

[42] Murali P, Baker J M, Javadi-Abhari A, Chong F T and Martonosi M 2019 *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems* 1015–1029

[43] Tannu S S and Qureshi M K 2019 *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* 987–999

[44] Lao L L and Browne D E 2022 *Proceedings of the 49th Annual International Symposium on Computer Architecture* 351–365

[45] Deng H W, Zhang Y and Li Q X 2020 *2020 57th ACM/IEEE Design Automation Conference (DAC)* 1–6

[46] Glover F 1989 *ORSA Journal on computing* **1** 190–206

[47] Glover F 1990 *ORSA Journal on computing* **2** 4–32

[48] Lawler E L 1963 *Management science* **9** 586–599

[49] Sahni S and Gonzalez T 1976 *Journal of the ACM (JACM)* **23** 555–565

[50] Graham R L 1995 *Handbook of combinatorics* (Elsevier)

[51] Floyd R W 1962 *Communications of the ACM* **5** 345–345

[52] Li A, Stein S, Krishnamoorthy S and Ang J 2023 *ACM Transactions on Quantum Computing* **4** 1–26

[53] Quetschlich N, Burgholzer L and Wille R 2023 *Quantum* **7** 1062

[54] IBM Ibm quantum experience devices URL https://quantum-computing.ibm.com/