Hindawi
Quantum Engineering
Volume 2023, Article ID 5792902, 10 pages
https://doi.org/10.1155/2023/5792902

Research Article

Quantum SoC and On-Chip Circuit Synthesis in the NISQ Era

Nan Wu,[1,2] Yucheng Hu,[1,2] Fangmin Song,[1,2] and Xiangdong Li[3]

[1]*State Key Laboratory for Novel Software Technology, Jiangsu 210093, China*
[2]*Department of Computer Science and Technology, Nanjing University, Jiangsu 210093, China*
[3]*New York City College of Technology, City University New York, New York, NY 10016, USA*

Correspondence should be addressed to Nan Wu; nwu@nju.edu.cn

Received 22 October 2022; Revised 29 April 2023; Accepted 27 June 2023; Published 12 July 2023

Academic Editor: ShiJie Wei

In recent years, fueled by the breakthroughs in the technology in quantum computation, there has been a rising interest in the noisy intermediate-scale quantum (NISQ) era. In addition to a large number of qubits, the study of error correction and quantum algorithms has made great progress. However, as a primary goal of quantum computation, making practicable quantum computers in the NISQ era still needs further study, mainly focusing on quantum computer organization, architecture, and circuit synthesis. This paper studies the quantum circuit synthesis in a small-scale universal quantum computing device called a "quantum system-on-chip" (QSoC). We analyze the quantum compilation of the hybrid architecture for a small-scaled universal quantum computational device with a specific size (quantum chip). Two kinds of on-chip circuit synthesis algorithms are proposed and discussed.

1. Introduction

In recent years, rapid development has been made in quantum computing science and the quantum information/communication industry, and several milestones in quantum computing devices have been achieved. One remarkable progress is that the number of controllable quantum bits is increased from less than 20 to 20–50 or even to 100. The manipulation of these qubits in a noisy environment is feasible. In 2018, Preskill first used the term "noisy intermediate-scale quantum" (NISQ) [1] for the current quantum era.

Today, a feasible, reliable, scalable, and ultimately universal quantum computational device can meet several rigorous requirements to build a quantum computer in small-scale and noisy environments.

Several researchers focused on the small-scale and integrated realization of quasi-universal quantum computing. The term "on-chip" has also been proposed in some research. In 2012, first, on-chip quantum simulation with a superconducting circuit was demonstrated [2]. In recent years, the on-chip method has been widely used for entangled pair generation [3, 4], quantum control [5], etc. However, there is no research on the integrated architecture for universal quantum computation in the on-chip method, which means all the abovementioned studies are focused on the part of the quantum computing system, not the whole system itself. Our purpose in this paper is to give a tentative study for building a fully functioned quantum computing system on a chip-sized device. So we call this architecture "quantum system-on-chip."

We will mainly focus on the function of quantum system-on-chip that meets the requirements of NISQ quantum computation. In 2000, DiVincenzo proposed five conditions necessary for constructing a quantum computer [6]:

(1) A scalable physical system with well-characterized qubits
(2) The ability to initialize the state of the qubits to a simple fiducial state
(3) Long relevant decoherence times
(4) A "universal" set of quantum gates
(5) A qubit-specific measurement capability.

Although the abovementioned conditions were proposed for experimental quantum computing devices with a very small size (approximately 2–10 qubits), but with additional criteria, they are still necessary to meet the new theoretical and technical requirements in the NISQ era. These new criteria are listed as follows:

(1) Several scalable qubit-based quantum resources (qubits in particular states, entanglement pairs, etc.)
(2) A hybrid "universal" computational architecture (at the "hardware" level)
(3) A set of quantum low-level procedures (at the "software" level) or as "quantum primitive"
(4) A set of noisy suppression and error correction mechanism or algorithms (at both "hardware" and "software" levels).

The hybrid architecture refers to the famous principle of "quantum data + classic control," proposed by Selinger [7]. With the new criteria and studies on a quantum computer system [8], we draw a block diagram showing the scientific (theoretical) and engineering (experimental) roadmap of a quantum computer in the NISQ era (see Figure 1).

The paper also studies a quantum circuit synthesis model that ensembles most parts of the organization of a "quantum system-on-chip." Theoretically, it meets the requirements of DiVincenzo and the new criteria. We mainly focus on quantum circuit synthesis and the related architecture.

As a tentative study of the novel architecture and its synthesis, there are several challenges. First, the small-size is very challenging, although the manipulatable qubits in QSoC.

Here is the outline of this paper: In Section 2, we study the hybrid architecture and organization of quantum computation in QSoC. A reduced quantum instruction set for universal quantum computation is analyzed, and the algorithmic realization of QSoC is discussed. In Section 3, we study the quantum compilation, an equivalence measurement of the quantum algorithm, and the corresponding quantum circuits. Some classic synthesis algorithms and a semiformal method of quantum synthesis are discussed. Quantum gate decomposition and its optimization algorithm for quantum circuit synthesis are studied in Section 4.

2. The Hybrid Architecture of the Quantum System-on-Chip

From a microscopic point of view, quantum computation performs a set of unitary transformations on a given initial set of quantum states, and a quantum computational algorithm at a large scale is a unitary matrix that performs on an initial unitary matrix of the same size. It means that all quantum algorithms have different functions in which the parameters and return values are unitary matrices. The quantum evolutions in a quantum algorithm perform intrinsic actions on the quantum variables. Although a quantum program can be compiled into a sequence of quantum gates to control qubits, from a point of view of reliability and scalability, the sequenced gate instructions cost more physical and logical resources for error correction and may cause more cascading failures in the program implementation.

Here, we give the discussion based on a logical architecture rather than a physical architecture of the QSoC quantum device.

In quantum science, the design of the architecture and organization of a computing device is based on a specified computational model. The traditional architecture of a computational model of a quantum Turing machine (QTM) or quantum random-access machine (QRAM) is mainly an imperative paradigm. Logically, an imperative quantum computational device should have several separate units, each with its own functions. The most common imperative architecture of quantum computing devices is the quantum von Neumann architecture [9]. The quantum architecture contains a quantum memory unit, a control unit, an input/output unit, a quantum arithmetic logic unit (QALU), and a quantum bus system. In the quantum von Neumann architecture, there is no physical QALU and physical quantum control unit; all qubits are stored in a quantum memory unit (like the organization proposed in this paper). However, at the level of software processing, there is a lot of difference between the quantum von Neumann architecture and the proposed QSoC architecture.

We now discuss the different paradigms between classic and quantum computation. In a classic computational algorithm, a series of functions update the variables in a real linear space. In a quantum computational algorithm, a whole function performs the update of all quantum variables in Hilbert's space. This implies that the use of the imperative paradigm in a classic computational model (architecture) is better. For example, the von Neumann architecture of a classic computer is an imperative paradigm. The von Neumann architecture allows the separation of all computational resources into several "units," each of which has its own functions, such as a processing unit, control unit, memory, input, and output mechanism. However, quantum computation is more like a declarative paradigm, which means a quantum computational algorithm always performs an integrated transformation with all intrinsic variables and obtains the final output (i.e., all final states of variables inside an entire unitary matrix). A detailed discussion of the different paradigms is given in [10]. The design of a declarative computational model and declarative architecture is more simplified than that of an imperative architecture. The proposed hybrid architecture of the quantum system-on-chip is based on the imperative paradigm: a quantum memory unit is the unique center of the architecture, storing all qubits and/or quantum computations; quantum communications are performed in the quantum memory, see more discussion below.

From the universal viewpoint of a quantum computer, the essential organization of a hybrid quantum computer can be generalized as "quantum data with classic control." This architectural model mainly contains a classic controller (or classic computing part), a quantum memory unit, a quantum computing unit, and a quantum input-output (I/O) device [10]. If a quantum computing device is designed for a specific use, e.g., quantum communication or quantum random number generation (QRNG), the organization and architecture can be simplified. The central parts of the hybrid

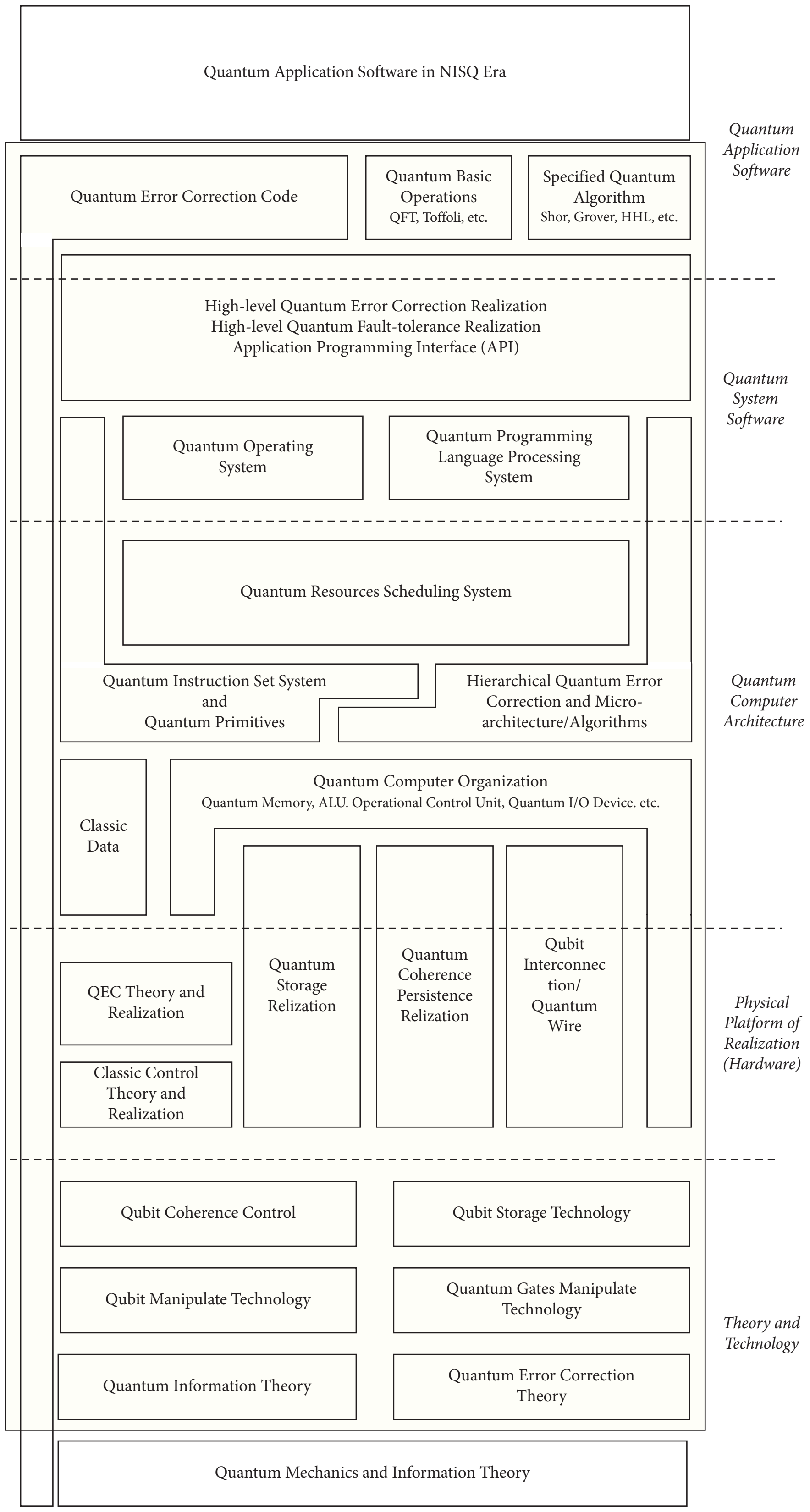
Quantum Application Software in NISQ Era
Quantum Error Correction Code
Quantum Basic Operations
QFT, Toffoli, etc.
Specified Quantum Algorithm
Shor, Grover, HHL, etc.
Quantum Application Software
High-level Quantum Error Correction Realization
High-level Quantum Fault-tolerance Realization
Application Programming Interface (API)
Quantum System Software
Quantum Operating System
Quantum Programming Language Processing System
Quantum Resources Scheduling System
Quantum Instruction Set System and Quantum Primitives
Hierarchical Quantum Error Correction and Micro-architecture/Algorithms
Quantum Computer Architecture
Classic Data
Quantum Computer Organization
Quantum Memory, ALU. Operational Control Unit, Quantum I/O Device. etc.
QEC Theory and Realization
Classic Control Theory and Realization
Quantum Storage Relization
Quantum Coherence Persistence Relization
Qubit Interconnec tion/ Quantum Wire
Physical Platform of Realization (Hardware)
Qubit Coherence Control
Qubit Storage Technology
Qubit Manipulate Technology
Quantum Gates Manipulate Technology
Quantum Information Theory
Quantum Error Correction Theory
Theory and Technology
Quantum Mechanics and Information Theory

Figure 1: Block diagram of a quantum computer system (the parts filled with yellow are in QSoC).

quantum computing device are a quantum memory unit and a quantum computing unit.

In the architecture of QSoC proposed in the paper, the central part of the entire system is the quantum memory unit rather than the quantum computing unit, and we call this "storage-centric" paradigm. Unlike a quantum computer in its early age (so-called"computing-centric" paradigm), quantum memory is a place where quantum states (e.g., quantum variables) are stored and quantum computation is conducted. We have designed several mechanisms, like the decoherence-free subspace and hierarchy QECC, to suppress decoherence in the quantum computing process and qubit states. In the NISQ era, these mechanisms can also be considered in the design of QSoC, see Figure 2 and details in [8, 10].

Now, we give a detailed description and a comparative study of the "storage-centric" and the "computing-centric" paradigms. In the traditional quantum computation model, similar to the classic Turing machine model-based von Neumann architecture, there is a quantum memory subsystem at the hardware level. This traditional quantum memory subsystem plays the complete role of quantum storage, including the original input quantum data and the quantum data generated during compute processing. It is a crucial component of the traditional universal quantum computer. Under this architecture, quantum memories can communicate with the quantum arithmetic and logic unit (QALU) during quantum computing through a quantum state transfer (QST) process, which can be performed with a "quantum wire (use entanglement)" or with other techniques. A QST allows quantum information to be transferred from one physical system to another without disturbing its quantum state. These communication mechanisms allow quantum information to be stored and retrieved in a quantum memory subsystem. However, there are several disadvantages and challenges associated with this QST technique, which are as follows: (1) QST requires the preservation of delicate quantum states over a relatively long distance (due to the separation of the quantum memory subsystem and QALU subsystem. However, quantum states are susceptible to decoherence, which is the loss of quantum coherence due to interactions with the environment. This can limit the distance over which quantum state transfer is possible. (2) Error correction can also be challenging in the QST due to the no-cloning theorem, so quantum error might be accumulating in this architecture. (3) QST often requires the use of entanglement, which is a limited and valuable resource in quantum computing. In addition, the creation and manipulation of entangled states can be resource-intensive and time-consuming. (4) Finally, QST can be a complex process, involving multiple physical systems and control mechanisms. This complexity can increase the likelihood of errors and decrease the overall efficiency of quantum information processing.

In comparison, the paradigm of "storage-centric" allows the transfer of all quantum computing processing into the quantum memory unit, which eliminates the quantum state transfer from the quantum memory subsystem to the QALU. All we need is the resource of qubit in the quantum memory unit, and this actually makes the QALU a coprocessor.

Under the memory architecture (shown in Figure 2) with the "storage-centric" paradigm, quantum memory units tend to have simpler designs than computing-centric systems, because they equalize the logical functionality of computation and storage. The ensemble design of quantum memory units, e.g., decoherence subspace (DFS) control and code teleportation modules, EEU, and SPU, requires less sophisticated control mechanisms to manipulate and measure quantum states in real-time. The second advantage is that the storage-centric paradigm allows error-correcting codes and DFS to protect quantum information from decoherence and other errors, in both storage and computation processing. In contrast, the computing-centric paradigm should use different kinds of error correction to maintain the accuracy of storage and computations. In terms of scalability, storage-centric paradigm can be easier to scale up both quantum storage ability and the useable number of qubits in quantum computation.

The quantum computing instructions in QSoC perform a universal transformation on quantum data (variables) with a given initial or intermediate quantum state. A quantum gate set is a kind of interface between the hardware and system software (quantum compiler). A hardware quantum unitary gate set is a set $\Lambda = \left\{A_j(\theta)\right\}$, where A_j is a single-qubit quantum gate or two-qubit quantum gate, which may contain a continuous internal parameter θ. The set Λ is a discrete set with a continuous variable. The size of the quantum gate set can be small (e.g., IBM's IBMQX4 5-qubit experimental quantum computing device has a gate set that contains only 10 universal gates [11]). The quantum instruction set is designed based on the quantum gate set and the architecture of the quantum computing device. Like a classic instruction set, it can also be designed as a reduced (i.e., reduced instruction set computer, RISC) or complex (complex instruction set computer, CISC) set. A more detailed discussion of quantum instruction sets can be found in [12].

The QSoC from this paper provides a reduced instruction set, which contains five kinds of instructions: pure quantum instructions, quantum initialization, quantum evolution, scheduling, and measurement (finalization).

In a memory unit of QSoC, the quantum bits are organized in a lattice structure shown in Figure 2. Each qubit pool keeps the qubits in several rows and columns; they share an entropy exchange unit (EEU) and a state purification unit (SPU) [13]. The strength of quantum coherence between adjacent qubits can be measured and manipulated by a quantum program, which is realizable in NISQ [14, 15]. Such in-qubit coherence links up all qubits in a pool and their characteristics in a "quantum wire." The disadvantage of the qubit's lattice topology is that it costs superpolynomial SWAP operations in the quantum circuit. However, the EEU can eliminate extraentropy made by SWAP operations by using specific protocols [16].

With the protection designed in the structure and organization of QSoC, all quantum algorithms can be

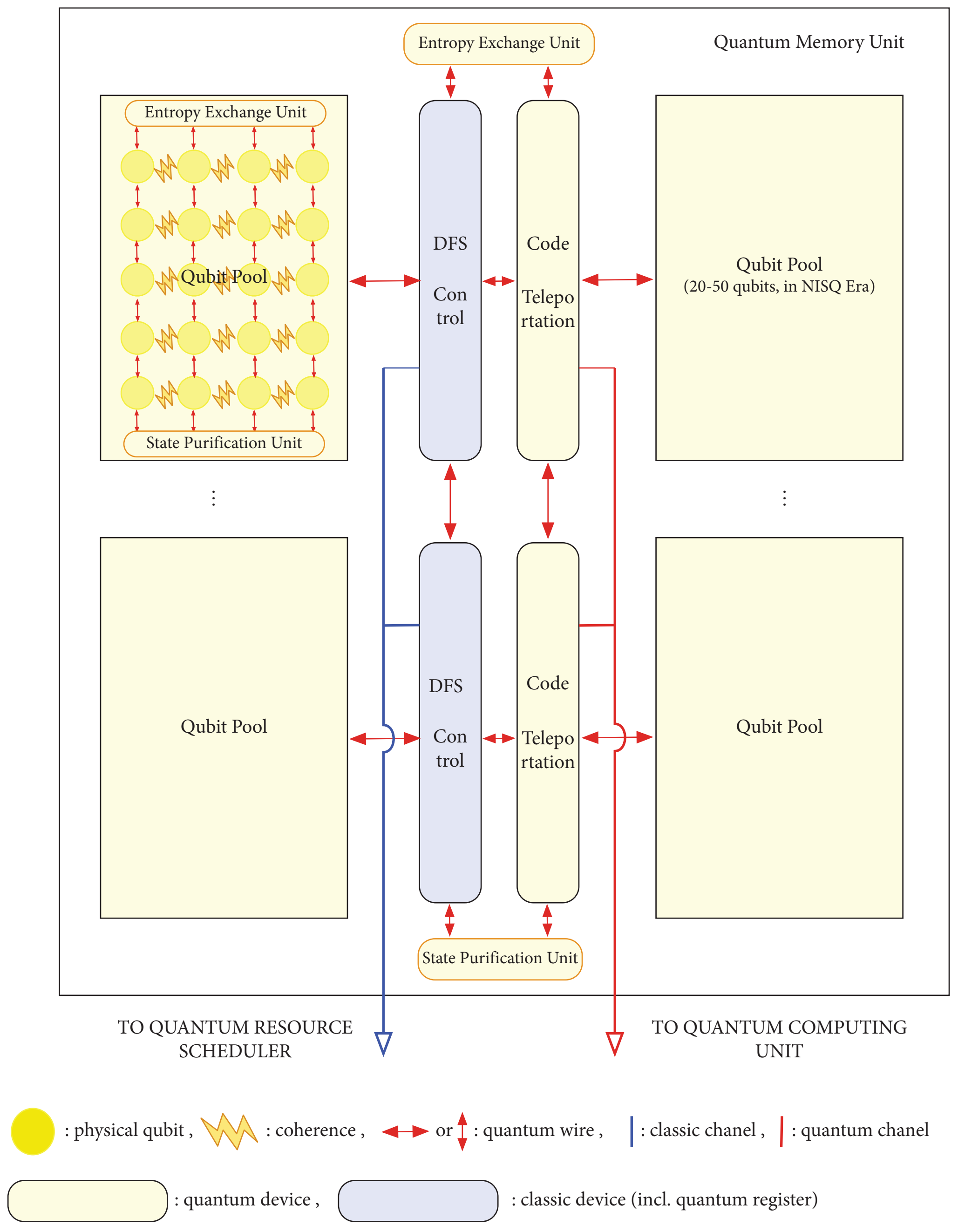

Figure 2: Diagram of a quantum memory unit in QSoC.

programmed and compiled into quantum circuit language and then assembled into quantum gate-based instructions.

In quantum algorithms, the frequently performed procedures can be reused. We call the reusable procedures as quantum primary. The set of quantum primary procedures mainly includes quantum Fourier transform (QFT), quantum phase estimation, fast quantum modular exponentiation, and quantum algorithms for linear system of equations (e.g., HHL algorithm).

3. Quantum Compilation in the Quantum System-on-Chip

3.1. Quantum Compilation in the NISQ Era. Theoretically, quantum computation can provide speed-up (polynomially or even exponentially) in specific algorithms, such as factorization, approximate optimization, and simulation of quantum systems. Building NISQ computers for practical applications is challenging. The main goal of the system software is to transform the formal quantum algorithms into operational sequences of a quantum device, to meet the requirements of NISQ, and to be executed on NISQ equipment at the end. Technically, the limitations of NISQ include the following: (1) the number of qubits, (2) connections (quantum wire) between qubits, (3) quantum gates (specific hardware), and (4) the circuit depth caused by noise. The quantum algorithms are hardware-independent, but quantum programs and applications are hardware-dependent. The quantum compilation can bridge the gap between the requirements of algorithms and programs, which is very important to realize the quantum system on the chip.

In general, the execution of a quantum algorithm is considered to run sequenced instructions in an idealized architecture with perfect and sufficient qubits, quantum gates, and an error-free environment. However, in actual NISQ quantum devices (e.g., IBM, Rigetti Computing [17], Google [18], and Intel [19]), the environment is imperfect. The quantum compilation is used to convert a higher level algorithm into a lower level sequence that can be executed on the NISQ device.

In addition, with the limitation of the quantum gate resource, the compiler can shorten the sequence of computing gate circuits. For example, in an n–qubit quantum Fourier transform (QFT), there is a complex composition of Hadamard gates and controlled rotation gates. The number of gates in the algorithm (QFT) is in $O(n^2)$. The optimized quantum compiler is based on the structure of QFT, so the complied output is more accurate than the one just based on the structure of the deep gate sequence. This optimization can also be applied to quantum Oracle design, noise reduction, and error suppression.

3.2. Evaluation of Quantum Compilation. Several methods can evaluate the efficiency of a quantum compilation system. A norm-based method is discussed below.

In quantum computing theory, if the same quantum transform is simulated on different circuits, the compilation results can be different. The norm can evaluate the quality of quantum compilation and circuit synthesis, that is, to minimize $\|U - U_S\|$ where the operator U is unitary, which describes the quantum transform, and the operator U_S is the output solution of the compiler or synthesis algorithm. An error ϵ is set to approach the transform optimization, i.e., $\|U - U_S\| \leq \epsilon$. In QSoC, the evaluation method is the Hilbert–Schmidt norm [13], due to its low cost of evaluation:

$$\langle U, U_S \rangle_{\mathrm{HS}} = \mathrm{Tr}\left(U^\dagger U_S\right). \tag{1}$$

3.3. Quantum Circuit Synthesis by Heuristic Search. The traditional quantum circuit synthesis algorithm mainly includes cosine-sine decomposition (CSD) [20] and UniversalQ [21].

This section introduces a new algorithm: quantum stimulated annealing circuit synthesis (QSACS), which uses stimulated annealing (SA) and machine learning to optimize quantum compilation and circuit synthesis. QSACS can minimize the objective function (1) based on the search strategy.

Formally, we consider a quantum gate set $\Lambda = \{G_k(\alpha)\}$, where k is a discrete parameter and α is a continuous parameter. The compilation algorithm will output a quantum gate sequence V. In QSoC, compiling unitary transformation U into a sequence of gate V with sequence length L can be described as the following optimization problem:

$$\left(\boldsymbol{\alpha}_{\mathrm{opt}}, \mathbf{k}_{\mathrm{opt}}\right) \coloneqq \underset{(\boldsymbol{\alpha},\mathbf{k})}{\arg\min},\ C\left(U, V_k(\boldsymbol{\alpha})\right), \tag{2}$$

where $V_k(\boldsymbol{\alpha}) = G_{k_L}(\alpha_L) G_{k_{L-1}}(\alpha_{L-1}) \cdots G_{k_1}(\alpha_1)$ is a trainable machine; actually, it is a unitary transformation.

In the architecture of QSoC, the parameters of topology and interconnection are defined. The discrete parameter $\mathbf{k}$ describes the topology of an entire circuit, which is a subspace of the QSoC's circuit space; the continuous parameter $\boldsymbol{\alpha}$ describes the characteristics of each quantum gate, which is the combination of the quantum gates defined in the instruction set. The length L of the sequence V is fixed as a constant in the optimization process, and other parameters may vary in the process of the optimization training procedure. The function $C(U, V_k(\boldsymbol{\alpha}))$ is a cost function to describe the "distance" between the trained and the target unitary transformations. Apparently, for any unitary transformation U and V, there is $0 \leq C(U, V) \leq 1$ and $U = V$ if and only if $C(U, V) = 0$.

With topology-aware optimization, the QSACS method can work properly to obtain the optimal circuit topological mapping from the application quantum algorithms. The advantage of the proposed method contains two parts: the system can use the declarative paradigm of quantum computing as a whole unitary transformation; and the topology-aware method can reduce the computational cost of the quantum part in a QSoC system.

4. Quantum Circuit Synthesis for the Quantum System-on-Chip

In the discussion of Section 3.3, we analyze a quantum circuit synthesis algorithm based on machine learning and simulated annealing (SA). The algorithm defines a cost function and optimizes the target circuit in circuit space by using heuristic search. The algorithm's target circuit (output) will be the optimal circuit with a lower cost. The strategy of this algorithm is if the cost of a synthesis circuit increases, the algorithm will determine whether to accept a structural change based on a specific probability. The acceptance probability decreases exponentially as distance for a change.

As discussed in Section 2 for the QSoC in the hybrid architecture, the classic part of the QSoC maintains a classic version of all quantum transformations, which correspond to the quantum algorithm for execution. The classic part is a coprocessor which assists the quantum part in classical computations in real-time circuit synthesis.

For the quantum part of QSoC, the memory unit is in initial states at the time of circuit synthesis, and all quantum variables are in preparation states. Notice that, in QSoC, there is no quantum processing unit (QPU). The prepared quantum circuit topology is in classic storage (classic memory), similar to von Neumann's stored-program computer. There is no gate-based sequence generated from the circuit synthesis procedure, only a declarative circuit-based topological mapping from the classic part of QSoC to the memory unit in the quantum part of QSoC. The detailed discussion of the quantum circuit synthesis by heuristic search is shown in [22].

We consider that a unitary transformation U and a gate sequence V are performed on a $d-$dimension space with an $n-$qubit, where $d = 2^n$. We define the cost function from (1) and (2):

$$\begin{aligned} C(U,V) &= 1 - \frac{\langle U,V\rangle_{\mathrm{HS}}}{d} \\ &= 1 - \frac{\mathrm{Tr}\left(U^{\dagger}V\right)}{d}, \end{aligned} \tag{3}$$

where $C(U,V) \in [0,1]$ is real, when the term $\mathrm{Tr}(U^{\dagger}V)/d$ is close to 1, and the sequences V and U are equivalent in the global system when the loss function equals to 0. That means the unitary transformation U and gate sequence V are operationally equivalent in the given quantum algorithm. This synthesize is simple and efficient.

A simplified block diagram of the proposed circuit synthesis is shown in Figure 3.

4.1. Method. In Section 3, a probabilistic methodology to optimize the synthesis problem is proposed. A good method may use randomly constructed gate sequences to compute a large amount of data. Half of the data are used to train the algorithm, namely, to optimize the cost function; another half of the dataset is used as a test dataset to evaluate the performance of the training model. The data used for training should cover all possible input branch spaces of a given algorithm. Because the algorithm (compiled gate sequence) is a linear mapping from the data qubit space (in $2^{2n}-$dimension) to a real number (in one $-$dimension), the estimation of the amount of training data to minimize the cost function is 2^{2n}, where n is the number of data qubits. The proposed method uses a unitary matrix with a size of 2^{2n}, and the estimated constraints required to determine the algorithm parameters are 2^{2n}. These data can be processed by using a classic computer in the hybrid model of QSoC.

The structural update discussed in Section 3 refers to the dynamical modification of the discrete parameters, including the use of new quantum gates to randomly replace some of the gates in the generated gate sequence. In the update process, the type and/or position of a given quantum gate may be changed to reoptimize the cost function on a continuous parameter $\boldsymbol{\alpha}$. Due to the restrictions of the structure, reducing the gate sequence is usually not easy. Therefore, a key point of structural optimization is that every change in discrete parameters requires to reoptimize the continuous parameters; this is very important for the optimization efficiency.

We chose the backtracking algorithm to optimize the continuous parameter in which the updates of all single-qubit gates are in operation order, which means that only one single-qubit gate is updated at a time and other quantum gates are kept in quantum memory. After determining an optimal quantum gate (i.e. the quantum gate minimizes the cost function), it moves to the next quantum gate. The algorithm can also change the order of a quantum gate randomly to avoid local minima as possible. We consider the gradient descent method as the continuous parameter optimization of a single quantum gate, because a qubit gate can be described by three real parameters. The gradient descent method can be operated efficiently in a three-dimensional space. Compared with the continuous parameter optimization, the algorithm needs to repeatedly optimize the continuous parameters until the cost function converges [22].

After the iteration (including the update of both discrete and continuous parameters), the cost will be computed. If a structural change lowers the costs, it will be accepted; if costs increase, then it will decide whether to accept the structural change based on a specific probability. This process behaves as simulated annealing (SA), which can effectively solve the problem of locally optimal solutions. The algorithm should find the global optimal solution instead of searching for the local minimum (optimal). Such a synthetic algorithm will have a greater "hit" possibility of finding the optimal solution for the circuit.

Another problem in this method is the length of the generated gate sequence. The simulation on NISQ shows that the upper bound of the length of the gate sequence is nearly eight times of the number of useable physical qubits. If an overlength sequence is adapted, the cost for error correction is also overproportioned. So limiting the length of the gate sequence is important.

After several steps of the iteration, the algorithm checks whether the current gate sequence $V_k(\boldsymbol{\alpha})$ can be reduced. The algorithm will try to find if there is an equivalent $V_k(\boldsymbol{\alpha})$ subsequence that can be substituted as a number of gates with a smaller size. If possible, it will modify the current gate sequence $V_k(\boldsymbol{\alpha})$ accordingly. The length of the gate sequence is shortened, but the cost is not increased. The algorithm to compress the subsequences is consistent with the process of the algorithm, and it is also iterative for both discrete and continuous parameter optimization. The same method can be used recursively since the size of quantum gates is fixed. This compression method may cause a shorter gate sequence than the fixed length L. Regularly, compressing the length of the gate sequence is particularly useful for searching for global optima because the random update of discrete parameters can generate gate sequences containing redundancy. Such gate sequences often reach the local minima of cost after continuous optimization. But if the redundant subsequences are removed, the local minima can be avoided.

The algorithm iterates the process until the cost function converges or the number of iterations reaches the limit; then, the final gate sequence circuit is obtained. For a fixed gate sequence length L, this algorithm can usually obtain an approximate optimization of compilation for the quantum

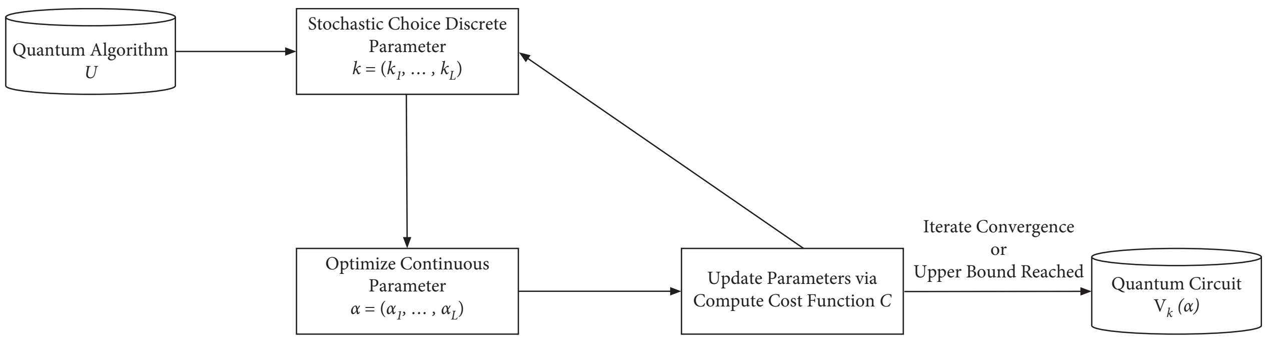
Quantum Algorithm
U
Stochastic Choice Discrete Parameter
$k = (k_1, \ldots, k_L)$
Optimize Continuous Parameter
$\alpha = (\alpha_1, \ldots, \alpha_L)$
Update Parameters via Compute Cost Function C
Iterate Convergence or Upper Bound Reached
Quantum Circuit
$V_k(\alpha)$

FIGURE 3: Diagram of the quantum circuit synthesize method in QSoC.

Input: Input unitary transformation U_n; trainable unitary sequence $V_k(\boldsymbol{\alpha})$; gradient iterate number L; maximum iterate number N; error tolerance $\epsilon' \in (0,1)$; learning rate $\eta > 0$;
Output: Optimal parameter $\boldsymbol{\alpha}_{\mathrm{opt}}$; $C(U, V_k(\boldsymbol{\alpha}_{\mathrm{opt}}))$; $C(U, V_k(\boldsymbol{\alpha}_{\mathrm{opt}})) \le \epsilon'$;

```
(1) $\boldsymbol{\alpha}_{\mathrm{opt}} \longleftarrow 0$; cost $\longleftarrow 1$; $r \longleftarrow 0$
(2) **While** cost $> \epsilon'$ and $r < N$ **do**
(3)   randomly choose the parameter $\boldsymbol{\alpha}^0$
(4)   **for** $i = 1, 2, \ldots, L$ **do**
(5)     compute cost function $C(U, V_k(\boldsymbol{\alpha}^{i-1}))$
(6)     compute gradient $\partial_{\boldsymbol{\alpha}^{i-1}} V_k(\boldsymbol{\alpha}^{i-1})$
(7)     **update** $\boldsymbol{\alpha}^i \leftarrow \boldsymbol{\alpha}^{i-1} - \eta \nabla_{\boldsymbol{\alpha}} C(U, V_k(\boldsymbol{\alpha}^{i-1}))$
(8)   **end**
(9)   compute cost function $C(U, V_k(\boldsymbol{\alpha}^L))$
(10)   **if**cost $> C(U, V_k(\boldsymbol{\alpha}^L))$ **then**
(11)     $\boldsymbol{\alpha}_{\mathrm{opt}} \longleftarrow \boldsymbol{\alpha}^L$
(12)     cost $\longleftarrow C(U, V_k(\boldsymbol{\alpha}^L))$
(13)   **end**
(14)   $r \longleftarrow r + 1$
(15) **end**
(16) **return** $\boldsymbol{\alpha}_{\mathrm{opt}}$, cost
```

ALGORITHM 1: Optimization of continuous parameter based on gradient.

algorithm U, which is sufficient to realize an equivalent circuit generation. If a more precise (or better) compilation is required, more elaborate compilation methods should be considered, such as hierarchical compilation.

4.2. Threshold of Synthesis. The threshold is set in the algorithm to speed up the compilation process. When the algorithm searches for a circuit, it is terminated and generates a circuit when the cost reaches the acceptability threshold ϵ. The threshold is determined by the following three criteria: (1) the new circuit can implement the original quantum algorithm with a given accuracy, (2) the generated unitary gate sequence should do the same task as the original unitary transformation, and (3) the algorithm can be executed within a reasonable time. To meet these criteria, we use the thresholds with different precision in the simulation and introduce a threshold of 10^{-9}.

4.3. Postcompilation Optimization. In the synthesis procedure, the algorithm optimizes the structure of the gate sequence, and the approximate compilation for a gate sequence with length L of the quantum algorithm U is obtained ultimately. Next, the obtained sequence will be optimized by the gradient descent-based machine learning method.

Suppose the gate sequence generated by a compiler is $V_k(\boldsymbol{\alpha}) = G_{k_L}(\alpha_L) G_{k_{L-1}}(\alpha_{L-1}) \cdots G_{k_1}(\alpha_1)$, the gradient of $V_k(\boldsymbol{\alpha})$ can be described as follows:

$$\nabla_{\boldsymbol{\alpha}} V_k(\boldsymbol{\alpha}) = \left(\frac{\partial V_k(\boldsymbol{\alpha})}{\partial \alpha_1}, \cdots, \frac{\partial V_k(\boldsymbol{\alpha})}{\partial \alpha_L} \right). \tag{4}$$

The gradient of one single gate can be described as follows:

$$\left(\frac{\partial V_k(\boldsymbol{\alpha})}{\partial \alpha_l} \right)_{i,j} = \frac{\partial V_k(\boldsymbol{\alpha})_{i,j}}{\partial \alpha_l}, \tag{5}$$

which refers to the gradient of the gate of (i, j)−th matrix element of the given gate sequence $V_k(\boldsymbol{\alpha})$.

For example, we take the rotation gate $R_z(\alpha) = e^{-i\alpha\sigma_z/2}$ according to (5), and the derivate of the parameter α can be obtained:

$$\frac{\partial}{\partial \alpha} R_z(\alpha) = -\frac{i}{2} \sigma_z R_z(\alpha). \tag{6}$$

Here, we only consider the parametrized single-qubit gates, and all are rotation gates. The gradient between them can be easily evaluated.

Both the time and space complexities of the gradient optimization are in polynomials: the primary compute component is to compute the gradient $\partial_{\boldsymbol{\alpha}^{i-1}} V_k(\boldsymbol{\alpha}^{i-1})$, the Kronecker product of a d–dimension unitary matrix $U \in U(d)$ takes $\mathcal{O}(d^2)$ steps, and the normal product of 2 unitary matrices takes $\mathcal{O}(d^3)$ steps. So, to compute the gradient and take optimization in a quantum circuit with depth in D and with the iteration round N, the total time complexity should be $\mathcal{O}(d^3 D^2 N)$ and the space complexity should be $\mathcal{O}(D + d^2)$, due to the fact that the memory for computing can be released between every iteration round.

Algorithm 1 is designed to optimize the continuous parameter via gradient descent method [22].

This algorithm can obtain an ϵ–approximate compilation result of U, where $\epsilon = (d/(d+1))\epsilon', d = 2^n$.

5. Conclusion

Quantum circuit synthesis (quantum compilation) is a fundamental technique in the era of NISQ, and it is a method of generating equivalent circuits for quantum processors. It synthesizes the equivalent circuits based on a given unitary transformation. Various limitations of the QSoC device (such as the limited circuit depth, limited topology, and limited number of quantum gates) restrict the realization of quantum algorithms. Circuit synthesis can be implemented by adapting quantum algorithms to quantum system-on-chip under limited conditions. The development of an efficient and high-fidelity quantum circuit synthesis algorithm has become a focus of QSoC research.

This paper focuses on the quantum circuit synthesis algorithm, which converts a higher level abstract algorithm into a lower level form to be executed on NISQ devices. With this algorithm, we improve the general quantum algorithms for QSoC. The algorithm or the method proposed in the paper adapts a decomposition method based on a heuristic search. When a circuit synthesis problem is formalized, a high-fidelity output circuit can be obtained through iterative optimization on the optimizing target. In the process of algorithm iteration, we use the gradient descent and simulated annealing methods to optimize the cost function.

Compared with the traditional quantum computational architecture with a quantum processing unit (QPU), the quantum memory-centralized architecture proposed in this paper has two advantages. First, all quantum variables are independent of a physical quantum storage unit, making all quantum variables "logical". Second, all procedures that quantum computation executes can be separated into different logical subprocedures. They can be executed in the quantum ALU part, the quantum memory unit, or even in the classic part. However, it adds difficulty to preprocessing because the "whole" logic processing of a set of variables is separated into a number of small processing.

We also study the topology-aware synthesis algorithm. It generates the corresponding gate sequence based on the quantum gate set and topological structure of the QSoC.

For further study, an open quantum system will be modeled in the noisy environment in the QSoC. The synthesis of circuits in an open environment (that is, in an open environment without QECC or with weak error tolerance) is interesting. Furthermore, a study of circuit synthesis-specific instruction sets, which can improve the performance and speed of the circuit synthesis procedure, is worth studying.

Data Availability

No data were used to support this study.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the National Key R&D Program of China (grant no. 2019YFA0308700) and the Chinese National Natural Science Foundation of Innovation Team (grant no. 61321491).

References

[1] J. Preskill, "Quantum computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, 2018.

[2] A. A. Houck, H. E. Türeci, and J. Koch, "On-chip quantum simulation with superconducting circuits," *Nature Physics*, vol. 8, no. 4, pp. 292–299, 2012.

[3] J. W. Silverstone, D. Bonneau, K. Ohira et al., "On-chip quantum interference between silicon photon-pair sources," *Nature Photonics*, vol. 8, no. 2, pp. 104–108, 2014.

[4] M. Kues, C. Reimer, P. Roztocki et al., "On-chip generation of high-dimensional entangled quantum states and their coherent control," *Nature*, vol. 546, pp. 622–626, 2017.

[5] I. Bashir, "A mixed-signal control core for a fully integrated semiconductor quantum computer system-on-chip," in *Proceedings of the ESSCIRC 2019-IEEE 45th European Solid State Circuits Conference (ESSCIRC)*, IEEE, Cracow, Poland, September 2019.

[6] D. P. DiVincenzo, "The physical implementation of quantum computation," *Fortschritte der Physik*, vol. 48, no. 9-11, pp. 771–783, 2000.

[7] D. Carvalho, *Towards Quantum Control*, 2019.

[8] W. Nan and F. Song, "Universal quantum computer: theory, organization and realization" Chinese," *Journal of Computers*, vol. 39, no. 12, pp. 2429–2445, 2016.

[9] M. F. Brandl, "A quantum von Neumann architecture for large-scale quantum computing," 2017.

[10] N. Wu, F. M. Song, and X. Li, "An improved architecture of a realizable quantum computer for quantum programming languages," *Quantum Information and Computation VII*, SPIE, vol. 7342, 2009

[11] J. Morris, F. A. Pollock, and K. Modi, "Non-Markovian memory in IBMQX4," 2019, https://arxiv.org/abs/1902.07980v1.

[12] A. Cross, A. Javadi-Abhari, T. Alexander et al., "OpenQASM 3: a broader and deeper quantum assembly language," *ACM*

Transactions on Quantum Computing, vol. 3, no. 3, pp. 1–50, 2022.

[13] M. A. Nielsen and C. Isaac, "Quantum computation and quantum information," pp. 558-559, 2002.

[14] K. Wu, A. Streltsov, B. Regula, G. Xiang, C. Li, and G. Guo, "Experimental progress on quantum coherence: detection, quantification, and manipulation," *Advanced Quantum Technologies*, vol. 4, no. 9, Article ID 2100040, 2021.

[15] A. Streltsov, G. Adesso, and M. B. Plenio, "*Colloquium*: quantum coherence as a resource," *Reviews of Modern Physics*, vol. 89, no. 4, Article ID 041003, 2017.

[16] M. Sawerwain, J. Wiśniewska, and R. Gielerak, "Switching and swapping of quantum information: entropy and entanglement level," *Entropy*, vol. 23, no. 6, p. 717, 2021.

[17] W. Zeng, B. Johnson, R. Smith et al., "First quantum computers need smart software," *Nature*, vol. 549, no. 7671, pp. 149–151, 2017.

[18] A. Cho, "Google claims quantum computing milestone," *Science*, vol. 365, p. 1364, 2019.

[19] G. G. Guerreschi, J. Hogaboam, F. Baruffa, and N. P. D. Sawaya, "Intel Quantum Simulator: a cloud-ready high-performance simulator of quantum circuits," *Quantum Science and Technology*, vol. 5, no. 3, Article ID 034007, 2020.

[20] G. W. Stewart, "On the perturbation of pseudo-inverses, projections and linear least squares problems," *SIAM Review*, vol. 19, no. 4, pp. 634–662, 1977.

[21] R. Iten, "Introduction to universalqcompiler," 2019, https://arxiv.org/abs/1904.01072.

[22] H. Yucheng, *The Study of Quantum Computation and Quantum Circuit Synthesis Based on Quantum Computing Chips (In Chinese)*, Nanjing University, Nanjing, China, 2021.