On the Co-Design of Quantum Software and Hardware

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

Anbang Wu
University of California
Santa Barbara, USA
anbang@cs.ucsb.edu

Yunong Shi
Amazon Braket
New York, USA
shiyunon@amazon.com

Ali Javadi-Abhari
IBM Quantum
Yorktown Heights, USA
ali.javadi@ibm.com

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

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

ABSTRACT

A quantum computing system naturally consists of two components, the software system and the hardware system. Quantum applications are programmed using the quantum software and then executed on the quantum hardware. However, the performance of existing quantum computing system is still limited. Solving a practical problem that is beyond the capability of classical computers on a quantum computer has not yet been demonstrated. In this review, we point out that the quantum software and hardware systems should be designed collaboratively to fully exploit the potential of quantum computing. We first review three related works, including one hardware-aware quantum compiler optimization, one application-aware quantum hardware architecture design flow, and one co-design approach for the emerging quantum computational chemistry. Then we discuss some potential future directions following the co-design principle.

CCS CONCEPTS

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

KEYWORDS

quantum computing; quantum compiler; superconducting quantum architecture; software-hardware co-design

ACM Reference Format:
Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, and Yuan Xie. 2021. On the Co-Design of Quantum Software and Hardware. In *The Eight Annual ACM International Conference on Nanoscale Computing and Communication (NANOCOM '21), September 7–9, 2021, Virtual Event, Italy.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3477206.3477464

1 INTRODUCTION

Quantum computing has become the new 'race to the moon' pursued with global pride and tremendous investments due to its strong potential in some important applications, including quantum simulation [10], combinatorial optimization [9], machine learning [4], cryptography [29], etc. To build a quantum computing system requires a multi-discipline effort from both theory and engineering. Similar to a classical computer, a quantum computing system can also be divided in the software system and hardware system. Figure 1 shows some key components and their examples in a quantum computing system stack.

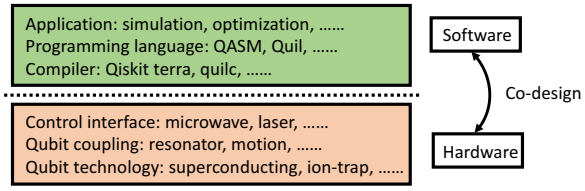

Figure 1: Quantum computing system stack

On the hardware side, we now have several candidate underlying technologies to implement the qubits (e.g., superconducting quantum circuit [8], ion trap [13], photonics [3]). The qubits are coupled together (e.g., by resonators [15] for superconducting qubits, by motion for ion trap qubits [6]) formulate a larger-size quantum system and then controlled by analog signals like microwave or laser. On the software side, we have quantum program languages (e.g., OpenQASM [7], Q# [31], Scaffold [1]) that can describe the quantum algorithms. Recently there are also high-level libraries for different quantum applications, like Qiskit Aqua [2], OpenFermion [22]. then the quantum programs can be compiled and optimized through quantum compilers (e.g., Qiskit Terra [2], ScaffCC [12]).

Nevertheless, state-of-the-art quantum systems are far from being mature. We are still waiting the demonstration of the first practical application that is intractable on classical computers but solvable with a quantum computer. Practical quantum applications require a high volume of resources, including a large number of qubits, many operations, and a long execution time. Yet, existing hardware are still too noisy to maintain many qubits coherently for long time and the operations are also imperfect. This probably requires a shift on the entire quantum computing system stack.

In this review, we argue that software-hardware co-design approach may become a solution to this problem and pave the way towards practical quantum computing. Following the co-design principle, the applications can be made hardware friendly, the hardware can be constructed more efficiently, and the compiler optimizations for the target application onto the target hardware can be more effective. In the rest of this paper, we will discuss three works.

NANOCOM '21, September 7–9, 2021, Virtual Event, Italy

ACM ISBN 978-1-4503-8710-1/21/09.
https://doi.org/10.1145/3477206.3477464

The first work is to map a quantum program to a superconducting quantum architecture effectively and efficiently. The second work, lying in the opposite direction, tailors the superconducting quantum processor architecture to a specific program. Putting these two together, the third work develops a holistic co-design for the quantum computation chemistry, leading to a wide range of benefits across multiple system stacks. Finally, we discuss several potential future research directions under the co-design principle.

2 MAPPING QUANTUM SOFTWARE ONTO QUANTUM HARDWARE ARCHITECTURE

In this section, we introduce a quantum compiler optimization algorithm (proposed in [16]) that can efficiently and effectively map the logical qubits in a quantum program onto the physical qubits in a connectivity-constrained superconducting quantum processor architecture with small mapping overhead and compilation time. This algorithm, SABRE, has been integrated several industry and academia quantum compiler infrastructures, e.g., IBM's Qiskit[2], the Oak Ridge National Lab's qcor compiler [21].

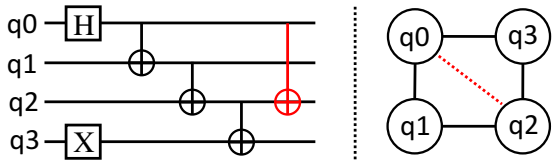

Figure 2: Example of a quantum circuit and a 4-qubit superconducting quantum processor architecture

2.1 Background

To illustrate the qubit mapping problem, we first briefly introduce the quantum program and the related hardware constraint. The quantum bit, also known as qubit, is the basic information processing unit in quantum computing. A classical bit has two deterministic states, '0' and '1'. One qubit also has two basis states, usually denoted as $|0\rangle$ and $|1\rangle$. Different from classical bit, one qubit can be the linear combination of the two basis states, which can be represented by $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. The state vector is (α, β). Moreover, two or more qubits can be entangled. The state of a two-qubit system can be represented by $|\Psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$, whose state vector is $(\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11})$. We can apply quantum gates to manipulate the state of the qubits. The gates can be classified by the numbers of qubits they are applied on. For example, single-qubit gates are applied on one qubit and two-qubit gates are on two qubits.

Quantum circuit is a conventional diagram to represent a quantum program [23]. One example quantum circuit is shown on the left of Figure 2. In quantum computing, the qubit is the basic information processing unit (the quantum analogy of bit in classical computing). In a quantum circuit, each horizontal line represents one qubit. The circuit in Figure 2 has four qubits labeled by $q1$, $q2$, $q3$, and $q4$. In this paper, we assume all gates are decomposed into single-qubit gates (the squares in the circuit in Figure 2) and CNOT gates (one type of gate on two qubits, the vertical lines in Figure 2 circuit). On IBM's superconducting devices, the single-qubit gates and the CNOT gate constitute the gate set that is directly supported on hardware.

After introducing the quantum circuit, we then explain the connectivity constraint of superconducting quantum processor architecture. The qubit connectivity of a superconducting quantum processor can be presented by a graph. One example qubit connectivity graph is shown on the right of Figure 2. Each node in the graph represents one physical qubit. The two-nodes are connected by an edge only when there are physical resonators connection the two corresponding physical qubits of the two nodes. For superconducting quantum processor, to execute a two-qubit gate directly on two physical qubits requires a resonator between them. When fabricating a superconducting quantum processor chip, the resonators can only qubits that are physically nearby due to the wire routing constraints. Therefore, some physical qubit pairs may not be connected and we cannot directly apply two-qubit gates on those physical qubits pairs. For example, in the example in Figure 2, the four physical qubits are connected in a square. The qubit pairs on the edges of the square are connected while the qubit pairs on the two diagonals are not connected. The two-qubit gates cannot be applied on the diagonal physical qubit pairs.

Such a qubit connectivity constraint will make some two-qubit gates in the quantum circuit not executable. For example, in Figure 2, suppose we map the four logical qubits to the example device as shown on the right. It can be noticed that the last two-qubit gate applied on $q0$ and $q2$ are not executable because they are mapped to a pair of physical qubits on the diagonal and there is no resonator connecting them. To address this problem and make the quantum circuit executable on a connectivity-constrained superconducting quantum processor, a quantum compiler needs to transform the quantum circuit and make it hardware compatible. Some SWAP gates will be inserted into the circuit to modify the logical-physical qubit mapping during the program execution. A SWAP gate does nothing logically but just exchange the mapping between two qubits. It should be noticed that one SWAP gate is implemented by three CNOT gates and has a relatively high error rate. Therefore, it is desirable to minimize the number of SWAPs inserted during the compilation. This is known as the qubit mapping problem, which has been proved to be NP-Complete [30].

2.2 SABRE Algorithm: Key Ideas

We propose a **S**WA**P**-based **B**idi**RE**ctional heuristic search algorithm, named SABRE. SABRE is designed to have good scalability to accommodate large size circuits but also maintain a good compilation result with low overhead. The first key design is the SWAP-based search. Some previous work [32] employed a mapping-based search which divides the input circuit into different layers and then search for a mapping in each layer. This will incur a high search complexity because the number of mappings grows exponentially with the number of qubits. Instead of searching for a mapping, SABRE searches candidate SWAPs. For example, in Figure 3, SABRE will divide the circuit into three parts. The front layer contains gates that are ready to execute and their two-qubit dependencies should be resolved first. Some gates right after the front layer are called the near-term gates and the remaining gates are temporarily ignored. SABRE will select from all possible SWAPs associate with at least

one qubit in the front layer (e.g., SWAPs on the red arrows). In this example, we tend to SWAP $q3$ and $q7$ because the two gates in the front layer will become executable and the distance between $q2$ and $q7$ (appear in one near-term gate) is also reduced.

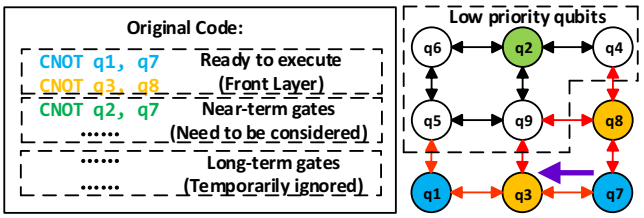

Figure 3: Example of SWAP-Based Heuristic Search

We also designed a sophisticated cost function to evaluate all candidate SWAPs. The cost function is shown in Equation 1. In this equation F is the front layer. E is the set of near-term gates. D is the matrix that records the SWAP distance between any two physical qubits. π is the mapping from logical qubits to physical qubits. W is a parameter to control how much we hope to consider about the near-term gates. Overall, this cost function tends to select a SWAP gate that can minimize the sum of distance between the qubit pairs of the two-qubit gates in the front layer. Also, the cost function will also reduce the distances of qubit pairs in the near-term gates. But it is controlled by the parameter W to ensure that gates in the front layer have a higher priority.

$$\begin{aligned} Cost(SWAP) = max(decay(SWAP.q_1), decay(SWAP.q_2)) \\ *\{\frac{1}{|F|}\sum_{gate\in F} D[\pi(gate.q_1)][\pi(gate.q_2)] \\ +W*\frac{1}{|E|}\sum_{gate\in E} D[\pi(gate.q_1)][\pi(gate.q_2)]\} \end{aligned} \quad (1)$$

In addition, we introduce a *decay* function. This function will record whether the qubit is moved recently. If so, the cost function will be increased and tend not to select SWAPs that involve with recently moved qubits. This feature can help reduce the circuit depth in the post-mapping circuit because the cost function can select non-overlapped SWAPs. For the initial mapping, we propose a reversal search scheme. This scheme starts from a random mapping and then search for SWAPs until reaching the end the input circuit with a final mapping. Then the input circuit is reversed and we search back from the end to the beginning of the input circuit. The final mapping can update the original random mapping since it carries the information from the entire circuit. In practice, we search forward and backward several times and keep the best initial mapping encountered.

SABRE is evaluated with various benchmarks on the IBM 20-qubit chip model [11] compared with the baseline [32]. Experimental results show that SABRE is able to find the optimal mapping for small benchmarks and the number of additional gates is reduced by 91% or even fully eliminated. For larger benchmarks, SABRE can demonstrate exponential speedup against the previous solution (usually 1000× speedup) and still outperform it with around 10% reduction in the number of additional gates on average with the assistance of the high-quality initial mapping generated by our proposed method. In some cases, the baseline cannot even finish execution due to exponential execution time and memory requirement, while SABRE can still work with short execution time and low memory usage. By tuning the decay parameters in our algorithm, SABRE shows the ability to control the generated circuit quality with about 8% variation in generated circuit depth by varying the number of gates. Please kindly refer to [16] for more detailed algorithm design and evaluation results.

3 QUANTUM HARDWARE ARCHITECTURE DESIGN FOR QUANTUM SOFTWARE

In the last section, we discussed how to efficiently map a quantum program onto quantum hardware using a quantum compiler and the modifications are on the software side. In this section, we will explore this mapping problem in the opposite direction and find a qubit connectivity architecture that can be mapped onto with lower overhead. A straightforward solution is to have dense connectivity so that two-qubit gates are directly supported on more physical qubit pairs and the mapping overhead can naturally be reduced. However, trivially increasing the qubit connections may not be efficient because it becomes more difficult to fabricate a complex superconducting quantum processor and one superconducting quantum processor with dense connections usually has lower yield rate. An intrinsic trade-off lies between the processor yield rate and the mapping overhead. Our work [17], leveraging the application-specific design principle, proposes a superconducting quantum processor architecture design flow that can generate architectures with both high yield rate and low mapping overhead for the target quantum program.

3.1 Frequency Collision & Yield Problem

Before introducing the superconducting architecture design flow, we first review some background about superconducting qubits and the frequency collision defect on superconducting quantum processors. In this paper, we focus on the fixed-frequency Josephson-junction-based transmon qubits [14] and all-microwave cross-resonance two-qubit gates [25]. One transmon qubit is an anharmonic oscillator with discrete energy spectrum. In practice, we use the ground state $|0\rangle$ and the first excited state $|1\rangle$ as the computational basis and the qubit frequency is the energy gap between the $|0\rangle$ and $|1\rangle$ states divided by the Planck constant. A typical qubit frequency is around $f = 5GHz$.

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

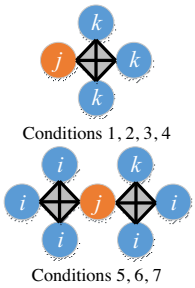

Figure 4: Frequency Collision Conditions ($\delta = -340MHz$)

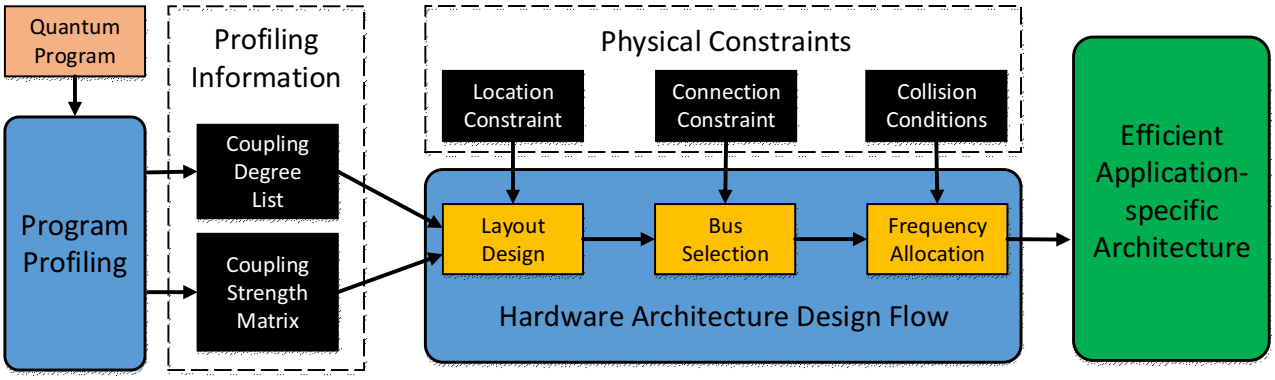

Figure 5: Overview of the proposed efficient superconducting quantum processor architecture design flow

Similar to traditional semiconductor technology, variation is inevitable when fabricating superconducting quantum processors. Suppose the frequency of a physical qubit is designed to f. The actual frequency after fabrication will be $f' = f + n_f$ where n_f satisfied Gaussian distribution $N(0, \sigma)$. The parameter σ is $130MHz-150MHz$ and can be further suppressed to around $14MHz$ with post-processing laser annealing [27]. Since we cannot control the post-fabrication qubit frequencies very precisely, it is possible that the frequencies of two or three connected qubits satisfy some specific conditions. Satisfying these conditions are termed *frequency collision* and cause defects on the device. Figure 4 summaries seven qubit frequency collision conditions in IBM's devices [5, 26]. On the left is a table showing the conditions and thresholds of different collision situations. Condition 1, 2, 3, and 4 involve two connected qubits (j and k). Condition 5, 6, and 7 involve three qubits of which two qubits (k and i) both connect to the other qubit j. The approximate equations and the corresponding thresholds determine whether one frequency collision happens. For example, if qubit j and k are connected and $|f_j - f_k| < 17MHz$, then the first condition is satisfied and frequency collision occurs. Note that the fourth condition has no threshold because it is an inequality rather than an approximate equation. On the right is a graphical illustration, showing the geometric locations of the qubits that may have frequency collisions of different conditions in two subfigures. Each node represents a qubit and the gray square represents that any two of the four surrounding qubits are connected.

3.2 Efficient Architecture Design: Overview

Optimizing the yield rate and reducing the mapping simultaneously are difficult due to the intrinsic trade-off between these two objectives. We are able to overcome this challenge by leveraging the application-specific design principle. By trading in the generalizability of the architecture and only targeting a specific program, it is then possible to realize the two aforementioned objectives simultaneously. We can deploy more hardware resources only on those locations where the performance of on the target program can be increased most.

Our end-to-end design flow proposed in [17] is depicted in Figure 5. It has two major steps, the program profiling and the architecture design. In the program profiling stage, our design flow will extract the logical two-qubit gate information because the two-qubit gate introduces the mapping problem and its underlying hardware support, the qubit connection, causes the frequency collision. We organize the two-qubit gate information in two data structures: the coupling degree list and the coupling strength matrix. The coupling degree list records the number of two-qubits associated with one qubit and the coupling strength matrix records the number of two-qubits between all logical qubit pairs. For example, figure 6 shows the profiling results of two different programs (the indices of rows and columns represent the qubit indices). The number two-qubit gates between different qubit pairs varies significantly and the two-qubit gate patterns are also different for different programs.

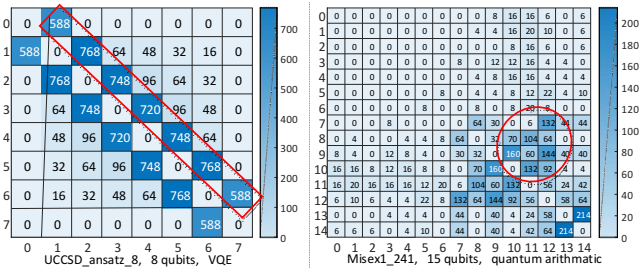

Figure 6: Coupling strength matrices of two programs

The second step is to generate an architecture design based on the profiling information. We decompose the superconducting quantum processor architecture design into three key subroutines, layout design, bus selection, and frequency allocation. Each subroutine targets different hardware components and configurations with profiling results and physical constraints incorporated. In the layout design, we focus on the qubit placement and put the qubits and try to make those qubit pair with more two-qubit gates between them nearby in order to reduce the mapping overhead later. We also assume that physical qubits can only be placed on the nodes of a 2D grid to ensure a modular design. Then, the bus selection subroutine will determine how the physical qubits are connected. We only add qubit connections (also termed as qubit buses) to the locations that are expected to reduce the mapping overhead most according to the profiling information. Finally, the frequency allocation subroutine will allocate frequencies to all placed physical qubits. The

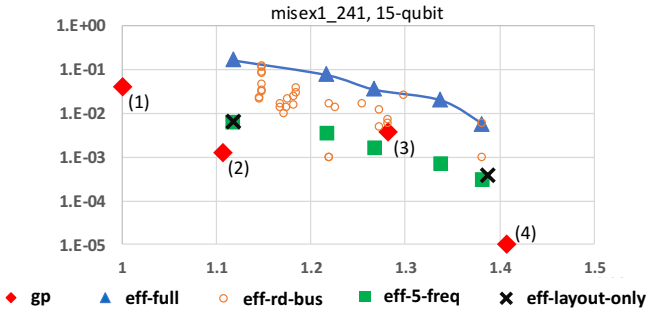

Figure 7: Yield vs performance of the generated architectures for 'misex' program when the fabrication noise strength $\sigma = 30MHz$. X-axis represents the relative performance. Y-axis represents the yield rate.

subroutine will increase the final yield rate by trying to avoid the frequency collision conditions on the generated architecture.

We compare the architectures generated from our design flow with general-purpose regular architectures [5, 11]. Figure 7 shows the yield and performance of the generated architecture for 'misex' program. X-axis represents the relative performance indicated by the post-mapping gate count. Y-axis represents the yield rate simulated by IBM's yield model [5]. The **gp** configurations are four general purpose architectures and the **eff-full** represents the architectures will all optimizations enabled in our design flow. It can be observed that the **eff-full** architectures can provide similar performance with $100\times - 1000\times$ yield rate improvement. We also carefully designed breakdown experiments to study the effect of different stages in the proposed design flow. Please kindly refer to [17] for more design flow details and evaluation results on more benchmarks.

4 SOFTWARE-HARDWARE CO-DESIGN FOR QUANTUM COMPUTATIONAL CHEMISTRY

In the last section, we introduced a design flow that can generate a superconducting quantum architecture for a target quantum program. However, this flow only starts from the low-level gate sequence and only considers the two-qubit gate patterns of a specific program instance. Therefore, the architectures generated can only accommodate the specific input program and may not support other programs of the same category. In our follow-up work [18], we overcome this shortcoming for the domain of computational chemistry and co-design the software and hardware via the high-level domain knowledge instead of the low-level program pattern. The proposed solution is then applicable to the full family of computational chemistry problems sharing a similar structure.

4.1 Variational Quantum Eigensolver, Ansatz, and Pauli Strings

We select computation chemistry as our target because chemistry simulation has many practical usages but large-scale chemistry simulation is intractable on classical computer. Quantum computers are naturally suited to simulate chemistry system []. Variational Quantum Eigensolver (VQE) [24] is the leading quantum algorithm with

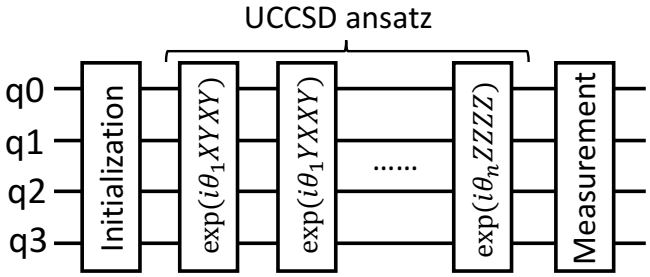

Figure 8: A VQE circuit example with UCCSD ansatz

modest resource requirement and some noise-resilience. Figure 8 shows a VQE circuit example. This circuit has three components, the initialization, the ansatz (parameterized circuit), and the final measurement which can obtain the expectation of an observable. The initialization and the final measurement are shallow and the majority of a VQE circuit is the ansatz. An ansatz is a parameterized quantum circuit. When executing a VQE algorithm, a classical optimizer will tune the parameters in the ansatz to minimize the measured expectation value of an observable. For chemistry simulation, the observable is usually the Hamiltonian (energy operator) of the simulated system and minimizing the energy means that we reach the ground state of the system. We refer [20] for a comprehensive introduction to quantum computational chemistry.

Since the majority of a VQE circuit is the ansatz, it naturally becomes our optimization target. The ansatz design is critical to the performance of a VQE algorithm since it determines how In this paper, we focus on the UCCSD (Unitary Coupled Cluster Singles and Doubles) chemistry-inspired ansatz [24] which the 'standard' ansatz for chemistry simulation. Usually, it can be expected that tuning the parameters in UCCSD can yield a good approximation of the true ground state. However, the size of a UCCSD ansatz is very large with $O(n^4)$ parameters (n is the number qubits). In the quantum circuit, the UCCSD ansatz is turned into a series of circuit blocks and each block is a Pauli string simulation circuit block, which will be introduced later in this section.

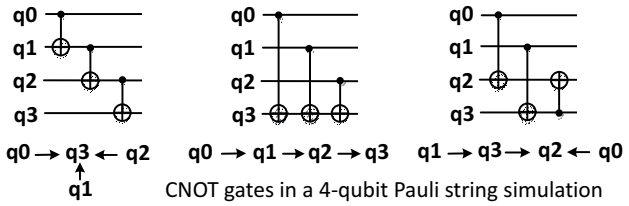

Figure 9: Three different valid CNOT trees

Before introducing the Pauli string simulation circuit, we first define the Pauli string. An n-qubit Pauli string P is an array of n operators (on the n qubits), each of which is one of the three Pauli operators $\{X, Y, Z\}$ or the identity operator I. For example, $P = XYIZ$ is a 4-qubit Pauli string. A Pauli string simulation circuit is to implement $exp(i\theta P)$. In such a circuit block, the two-qubit CNOT gates have a unique pattern. All qubits whose operators are not the identity operator will be connected by the CNOT gates in a

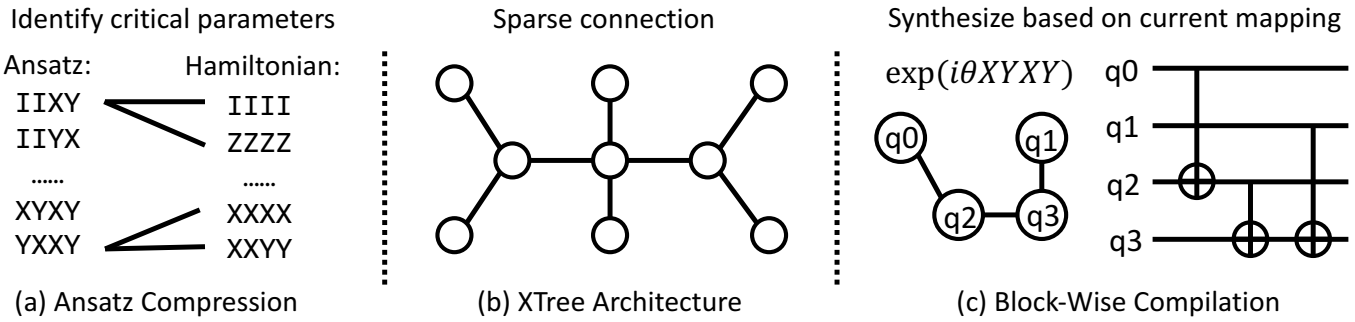

Figure 10: Overview of the Pauli-string-centric software-hardware co-design

tree structure while the exact structure of the CNOT tree can be flexible. For example, Figure 9 shows three valid tree structures of a 4-qubit Pauli string simulation circuit. The circuits are shown on the upper half and the corresponding tree graphs are below the circuits.

4.2 Pauli-String-Centric Co-Designing

It can be observed that the Pauli string is the central building block of VQE chemistry simulation. The ansatz consists of an array of Pauli string simulation circuits. These circuits have a unique CNOT gate pattern that can be leveraged. Figure 10 shows the overview of our co-design for variational quantum chemistry simulation. It comes with three major components.

Ansatz Compression: In order to compress the ansatz and prune some parameters, we need to identify critical parameters that are expected to change the final measured energy most. As shown in Figure 10 (a), we evaluate the importance of a parameter by comparing the Pauli strings associated with this parameter with the Pauli strings from the Hamiltonian of the simulated system. This is possible because the Pauli simulation circuit can be interpreted as a rotation along an axis in a high-dimensional space and we can predict how it can change the projection along an axis where the projection can be considered as a measurement. After selecting the important parameters, we also order them in a hardware-friendly order so that the constructed ansatz can be better mapped to hardware later.

XTree Architecture: As explained in the last section, we hope to reduce the number of qubit connections for a higher yield rate. Since the UCCSD ansatz is composed of a series of Pauli string simulation circuits and the CNOT gates are in a tree structure, we can naturally connect the physical qubits in a tree structure (e.g., Figure 10 (b)). This can be very efficient since the tree structure requires the minimum number of connections to connect all qubits.

Block-Wise Synthesis & Mapping: Finally, our compiler will deploy the VQE circuit onto the XTree architecture. This requires careful optimization algorithm design since a sparse architecture like the XTree usually incurs very high mapping overhead. As shown in Figure 10 (c), the key to our compilation to find the tree structure that fit the current mapping best. For example, the four qubits in Figure 10 (c) are on a XTree architecture. Our compiler will generate the CNOT tree on the right which does not require any SWAP operations.

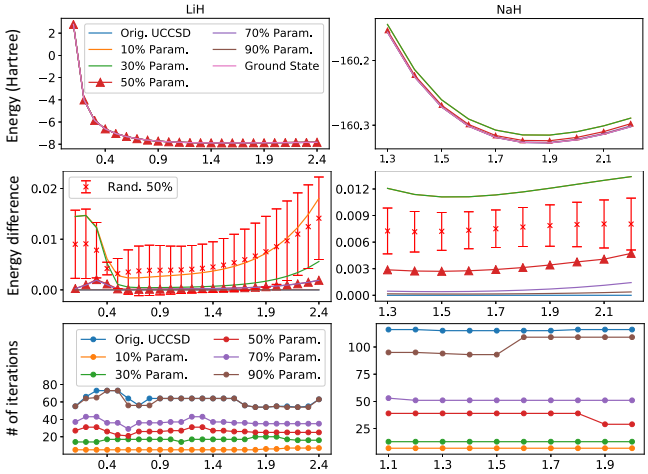

Figure 11: Simulation results of LiH **and** NaH

Figure 11 shows the VQE simulation results of LiH and NaH molecules. We compress the UCCSD ansatz and only keep a portion of the critical parameters. '10%-90% Param.' represent that we keep 10%-90% of the parameters. The 'Ground State' is the theoretical true value. The 'Orig. UCCSD' is the original UCCSD without compression. The simulation accuracy loss is small since the simulated energy difference is very small compared with the absolute simulated energy. Small-size ansatzes are also faster and require fewer iterations to converge. For the hardware and compiler, The XTree architecture with sparse qubit connection has a higher yield rate compared with conventional grid architecture and the mapping overhead can be almost eliminated through our block-wise synthesis and mapping. Please kindly refer to [18] for more co-design details and evaluation results on more molecules of different structures and sizes.

5 CONCLUSION AND FUTURE DIRECTIONS

In this review, we demonstrated that the quantum system performance and efficiency can be significantly improved through co-designing the software and hardware. We introduced some of the previous works on quantum compiler, superconducting quantum processor, and solving quantum computational chemistry aligned

with the co-design principle. In the rest of section, we will discuss some potential research direction from both the hardware technology side and the software application side.

5.1 Co-Design beyond Superconducting Qubits

The reviewed works are mostly on the superconducting quantum computing technology because it is one of the leading technologies in this area and has been adopted by many vend. Meanwhile, there are several other promising technology candidates whose architecture design space is not yet fully explored. For example, for an ion trap quantum computer, one trap cannot maintain many ions without losing good qubit addressability and multiple traps would desirable when scaling up. The number of ions in each trap and the interconnection topology of multiple traps can customized according to the target application.

Going beyond the near-term noisy devices, the co-design for future fault-tolerant quantum computers is also worth to study. Comparing with near-term quantum computing systems, the system of a fault-tolerant quantum computer has one more abstraction layer, the quantum error correction [19], in the middle of the system stack to provide long-living logical qubits and precise logical operations to the quantum programs. The quantum error correction protocols can be co-designed with respect to the underlying hardware or high-level application.

5.2 Co-Design beyond Quantum Chemistry

The application of quantum computing is far beyond the scope of chemistry simulation and there are many other domains. For example, quantum machine learning is another leading candidate application of practical quantum computing. We argue that enabling effective co-design in new domain requires new proper abstractions that can guide the design of software and hardware. For example, our co-design in [18] targeting the quantum chemistry simulation application was carried out through a key concept, Pauli string, which coordinates the design and optimization at different system technology stacks. It is not known what abstractions we should use for software-hardware co-design in other application domains.

One candidate algorithmic target of co-design is the Boolean function because many quantum algorithms [23] involve an oracle that is a subroutine implementing a quantum version of the classical Boolean function. Previous works [28] have studied the compilation of classical oracle as they are abstracted in a Boolean function hardware-independently and application-independently. The compilation of classical oracles can possibly be improved through software-hardware co-design and then benefit a wide range of quantum algorithms.

ACKNOWLEDGMENTS

We thank Dr. Swamit Tannu for the invitation and Dr. Sergi Abadal for the help with editing and publishing. This work was supported in part by NSF 1925717 and 2048144. G. L. was in part funded by NSF QISE-NET fellowship under the award DMR-1747426.

REFERENCES

[1] Ali J Abhari et al. 2012. *Scaffold: Quantum programming language.* Technical Report. PRINCETON UNIV NJ DEPT OF COMPUTER SCIENCE.

[2] MD SAJID ANIS et al. 2021. Qiskit: An Open-source Framework for Quantum Computing. https://doi.org/10.5281/zenodo.2573505

[3] JM Arrazola et al. 2021. Quantum circuits with many photons on a programmable nanophotonic chip. *Nature* 591, 7848 (2021), 54–60.

[4] Jacob Biamonte et al. 2017. Quantum machine learning. *Nature* 549, 7671 (2017), 195–202.

[5] Markus Brink et al. 2018. Device challenges for near term superconducting quantum processors: frequency collisions. In *2018 IEEE International Electron Devices Meeting (IEDM)*. IEEE, 6–1.

[6] Colin D Bruzewicz et al. 2019. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews* 6, 2 (2019), 021314.

[7] Andrew W Cross et al. 2021. OpenQASM 3: A broader and deeper quantum assembly language. *arXiv preprint arXiv:2104.14722* (2021).

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

[9] Edward Farhi et al. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).

[10] Iulia M Georgescu et al. 2014. Quantum simulation. *Reviews of Modern Physics* 86, 1 (2014), 153.

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

[12] Ali JavadiAbhari et al. 2014. ScaffCC: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*. ACM, 1.

[13] David Kielpinski et al. 2002. Architecture for a large-scale ion-trap quantum computer. *Nature* 417, 6890 (2002), 709–711.

[14] Jens Koch et al. 2007. Charge-insensitive qubit design derived from the Cooper pair box. *Physical Review A* 76, 4 (2007), 042319.

[15] Philip Krantz et al. 2019. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews* 6, 2 (2019), 021318.

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

[17] Gushu Li et al. 2020. Towards efficient superconducting quantum processor architecture design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1031–1045.

[18] Gushu Li et al. 2021. Software-Hardware Co-Optimization for Computational Chemistry on Superconducting Quantum Processors. *arXiv preprint arXiv:2105.07127* (2021).

[19] Daniel A Lidar and Todd A Brun. 2013. *Quantum error correction*. Cambridge university press.

[20] Sam McArdle, Suguru Endo, Alán Aspuru-Guzik, Simon C Benjamin, and Xiao Yuan. 2020. Quantum computational chemistry. *Reviews of Modern Physics* 92, 1 (2020), 015003.

[21] Alexander Mccaskey et al. 2021. Extending C++ for Heterogeneous Quantum-Classical Computing. *ACM Transactions on Quantum Computing* 2, 2, Article 6 (July 2021), 36 pages. https://doi.org/10.1145/3462670

[22] Jarrod R McClean et al. 2020. OpenFermion: the electronic structure package for quantum computers. *Quantum Science and Technology* 5, 3 (2020), 034014.

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

[24] Alberto Peruzzo et al. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature communications* 5 (2014), 4213.

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

[26] Sami Rosenblatt et al. 2019. Enablement of near-term quantum processors by architectural yield engineering. *Bulletin of the American Physical Society* (2019).

[27] Sami Rosenblatt et al. 2019. Laser annealing qubits for optimized frequency allocation. US Patent App. 10/340,438.

[28] Vivek V Shende et al. 2006. Synthesis of quantum-logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25, 6 (2006), 1000–1010.

[29] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.

[30] Marcos Yukio Siraichi et al. 2018. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*. ACM, 113–125.

[31] Krysta Svore et al. 2018. Q#: Enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*. ACM, 7.

[32] Alwin Zulehner et al. 2018. Efficient mapping of quantum circuits to the IBM QX architectures. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*. IEEE, 1135–1138.