Relaxed Peephole Optimization: A Novel Compiler Optimization for Quantum Circuits

Ji Liu
North Carolina State University
Raleigh, USA
jliu45@ncsu.edu

Luciano Bello
IBM Research
Yorktown Heights, USA
luciano.bello@ibm.com

Huiyang Zhou
North Carolina State University
Raleigh, USA
hzhou@ncsu.edu

***Abstract*—As in classical computing, compilers play an important role in quantum computing. Quantum processors typically support a limited set of primitive operations or quantum gates and have certain hardware-related limitations. A quantum compiler is responsible for adapting a quantum program to these constraint environments and decomposing quantum gates into a sequence of the primitive ones. During the compilation process, it is also critical for the compiler to optimize the quantum circuits in order to reduce the noise in the computation results. Since the noise is introduced by operations and decoherence, reducing the gate count is the key for improving performance.**

In this paper, we propose a novel quantum compiler optimization, named relaxed peephole optimization (RPO) for quantum computers. RPO leverages the single-qubit state information that can be determined statically by the compiler. We define that a qubit is in a basis state when, at a given point in time, its state is either in the X-, Y-, or Z-basis ($|+\rangle$ / $|-\rangle$, $|L\rangle$ / $|R\rangle$ and $|0\rangle$ / $|1\rangle$). When basis qubits are used as inputs to quantum gates, there exist opportunities for strength reduction, which replaces quantum operations with equivalent but less expensive ones. Compared to the existing peephole optimization for quantum programs, the difference is that our proposed optimization does not require an identical unitary matrix, thereby named 'relaxed' peephole optimization. We also extend our approach to optimize the quantum gates when some input qubits are in known pure states. Both optimizations, namely the *Quantum Basis-state Optimization (QBO)* and the *Quantum Pure-state Optimization (QPO)*, are implemented in the IBM's Qiskit transpiler. Our experimental results show that our proposed optimization pass is fast and effective. The circuits optimized with our compiler optimizations obtain up to 18.0% (11.7% on average) fewer $CNOT$ gates and up to 8.2% (7.1% on average) lower transpilation time than that of the most aggressive optimization level in the Qiskit compiler. When running on real quantum computers, the success rates of 3-qubit quantum phase estimation algorithm improve by $2.30X$ due to the reduced gate counts.

***Index Terms*—quantum computing, peephole optimization**

I. Introduction

Quantum computing shows great potential in chemistry simulation [26], combinatorial optimization [18], cryptography [27], machine learning [8], etc. Recently, Google, IBM, and Intel have announced their quantum computers with 72, 53, and 49 qubits, respectively [20], [21], [25]. These noisy quantum computers are capable of running some quantum algorithms and would be helpful for exploiting the physics of many entangled particles [37]. However, state-of-the-art quantum computers do not have enough qubits to accommodate error correction codes, and the noise in the quantum computers hinders the development of quantum computing [34].

As quantum computers typically support a limited set of basic operations/gates, quantum compilers/transpilers are responsible for decomposing complex quantum gates into the basic ones that the quantum computer supports. Quantum compilers also optimize quantum circuits to reduce the overall gate count or circuit depth. Since the accuracy of the final result can be affected by the system noise, such optimization is extremely important for quantum computers. The existing quantum compilers [3], [39] exploit many optimization techniques. An important one is peephole optimization or operator strength reduction [39]. The peephole optimization is analogous to its homonym in classical computing. The compiler traverses through the quantum circuit to find specific patterns of subcircuits and substitute them with equivalent ones that have less primitive operations or shorter depth. These substitutions keep the semantics of the quantum program as their unitary matrix representations are identical.

In this paper, we propose a new compiler optimization termed relaxed peephole optimization (RPO). It builds upon the fact that some of the qubit states for many quantum gates can be known or derived at compile-time. This presents opportunities for replacing quantum operations with equivalent but less expensive ones. But the difference from the existing peephole optimization is that the unitary matrix of the circuit may change although the circuit functionality remains the same. For example, for a $CNOT$ gate, if the control qubit is in the $|1\rangle$ state, it is functionally equivalent to a NOT gate on the target qubit. After determining the state of the control qubit, our compiler optimization will replace the $CNOT$ gate with a NOT gate. However, the unitary matrices of $CNOT$ and NOT gates are different and the peephole optimization would not be able to take advantage of such opportunities. In other words, our optimization can be viewed as a relaxed type of peephole optimization, which finds functionally equivalent circuits under certain circumstances. In our paper, we derive RPO for a wide range of quantum gates when some of their inputs are in known basis/pure states.

In order to figure out the quantum states for RPO, we propose a quantum state analysis approach. With this analysis, we develop two optimization passes, namely the *Quantum Basis-state Optimization (QBO)* pass and the *Quantum Pure-*

state Optimization (QPO) pass. In our paper, we define $|0\rangle$, $|1\rangle$, $|+\rangle$, $|-\rangle$, $|L\rangle$, and $|R\rangle$ as basis states. The QBO pass identifies these basis states for every qubit during execution and optimizes quantum gates accordingly. The QPO pass determines single-qubit pure states and performs corresponding quantum circuit optimization. We also introduce annotations such that the programmer can guide the compiler optimization. We experimented with several quantum benchmarks on IBM Q quantum simulators and quantum computers. The experiments show that the circuits optimized using our approach have up to 18.0% (11.7% on average) fewer $CNOT$ gates with up to 8.2% (7.1% on average) less transpilation time than that of the most aggressive optimization level in the Qiskit compiler. Since other quantum compilers, e.g., $t\,|ket\rangle$ [39] use a similar gate model to IBM Qiskit, we expect that our proposed idea is applicable to them as well.

The major contributions of this work are listed as follows:

- We propose a new compiler optimization, relaxed peephole optimization (RPO).
- We derive a comprehensive list of circuit optimizations for a wide range of quantum gates when some of their inputs are in known basis or pure states.
- We present a quantum state analysis approach which identifies basis and pure states for each qubit in a quantum circuit. We also introduce annotations to enable users to provide information to facilitate state analysis.
- We implement both QBO and QPO as compiler optimization passes in the IBM Qiskit transpiler.
- We show that our proposed RPO achieves better results than the most aggressive optimization level in Qiskit.

The remainder of the paper is organized as follows. Section II introduces the background of quantum computing and quantum compilers. Sections III and IV describe our findings on optimizing $CNOT$ and $SWAP$ gates with zero states and known pure states, respectively. Section V generalizes the optimization for a broad range of quantum gates. Section VI discusses our compiler scheme to determine the quantum states of each qubit in a quantum circuit. Section VII presents our compiler implementation using IBM Qiskit. Our experimental results are discussed in Section VIII. Finally, Section IX concludes the paper.

II. Background and Related Work

A. Quantum Computing

Qubit (quantum bit) is the basic unit of quantum information. Besides the classical states $|0\rangle$ and $|1\rangle$, a qubit can stay in any superposition state. A superposition state can be represented as $|\psi\rangle = a\,|0\rangle + b\,|1\rangle$ where a and b are complex numbers and $|a|^2+|b|^2 = 1$. When measuring a superposition state $|\psi\rangle$ in the computational basis, the probability of getting $|0\rangle$ and $|1\rangle$ states are $|a|^2$ and $|b|^2$, respectively. The state of a quantum system is represented by a vector in a Hilbert space [33]: a complex vector space with an inner product. The state of multiple qubits can be expressed as the tensor product of each qubit state if they are not entangled: $|\psi_{12}\rangle = |\psi_1\rangle \otimes |\psi_2\rangle = |\psi_1\psi_2\rangle$. Entanglement is an unique property of quantum computing. When two qubits are entangled, their measurement results are correlated and the two-qubit state can not be expressed as the tensor product of individual qubits. For example, the two-qubit Bell state $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is an entangled state. When the measurement outcome of the first qubit is $|0\rangle$, the measurement outcome of the second qubit must be $|0\rangle$.

A pure quantum state can be represented by a single state vector $|\psi\rangle$ in the Hilbert space. A mixed quantum state can not be represented in this way and corresponds to a probability mixture of pure states. A density matrix [9] $\rho = \sum_i P_i\,|\psi_i\rangle\,\langle\psi_i|$ is used to describe the mixed states, where P_i is the probability of the pure state $|\psi_i\rangle$. Generally speaking, a qubit system is in a pure state when the qubits in the system are not entangled with qubits out side the system. When the qubits are entangled with others, they are in the mixed state. For example, the two-qubit bell state is a two-qubit pure state since these two qubits are not entangled with others. The state can be represented by state vector $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$. If we consider each qubit individually, however, since the two qubits are entangled, either qubit is in the mixed state with the density matrix as $\rho = \frac{1}{2}(|0\rangle\,\langle 0| + |1\rangle\,\langle 1|)$. An n-qubit pure state $|\psi\rangle$ can be generated by applying an n-qubit unitary gate U to n qubits, which are all in the $|0\rangle$ state: $|\psi\rangle = U\,|0\rangle^{\otimes n}$. The proof is in Appendix A. A key conclusion from the proof is that when a qubit is not entangled with the others, it is in a single-qubit pure state and this state can be generated by applying a single-qubit gate to a qubit in the $|0\rangle$ state. For simplification purposes, we will refer to single-qubit pure states as *pure state* in the rest of this paper.

A quantum program is essentially a sequence of instructions/gates operating on qubits. There are single-qubit gates such as Identity gate I or id, Hadamard gate H, Phase changing gate S and T, Pauli Gates X, Y, and Z, and two-qubit gates such as controlled-NOT ($CNOT$) gate. The two qubits in the $CNOT$ gate are termed as control and target qubit as illustrated in Figure 1. Instructions, unlike gates, are not necessarily reversible might include classical aspects in the quantum program. Example of such instructions include $RESET$, $MEASURE$, and conditional control. On this work only considers $RESET$ for simplicity. The state-of-the-art quantum computers only support a set of basic gates/instructions. and the quantum compiler needs to decompose the quantum program into the supported primitives. For example, the IBM quantum computers support basis gates including four types of single-qubit gates $u1, u2, u3, id$, and a two-qubit $CNOT$ gate cx [44].

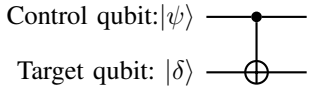


Fig. 1: A $CNOT$ gate with its control and target qubit.

B. Qiskit Framework

Qiskit [3] is an open-source quantum computing software development framework. One element, named Qiskit Aqua,

allows programmers to write codes for quantum algorithms. Another element called Qiskit Terra provides a transpiler, which is responsible for quantum gate decomposition, logical-to-physical qubit mapping, and circuit optimization. The transpiler consists of modular transpiler passes for circuit transformations. The transpiler pass manager schedules the transpiler passes and allows them to communicate. The users can control the pass manager to perform selective optimizations to the circuit. The transpiler has four pre-defined pass managers corresponding to the optimization level from 0 to 3. The higher optimization level, the higher the transpiler effort to optimize the circuit at the cost of transpilation time.

The different optimization levels can be described as follows [3]: Level 0 maps the circuit to the quantum device, with no explicit optimization included. Level 1 maps the circuit and also performs light-weight optimizations such as collapsing adjacent gates. Level 2 and 3 include noise-aware optimizations. Level 2 chooses noise-adaptive layout for the mapping and performs gate-cancellation procedure based on gate commutation relationships. Level 3 extends the passes from level 2 to include re-synthesis of two qubit blocks in the circuit as well as more iterations in the stochastic routing process. The re-synthesis process is performed by the `Collect2qBlocks` and the `ConsolidateBlocks` passes in Qiskit. The `Collect2qBlocks` pass traverses the circuit and collects sequences of gates acting on two qubits. The `ConsolidateBlocks` pass calculates the unitary matrices of two-qubit blocks and re-synthesizes them to more optimized circuits. This transpiler pass is similar to the operator strength reduction or peephole optimization in classical computing. The difference between our proposed optimization and the `ConsolidateBlocks` pass is that we do not preserve the unitary matrix although we replace the circuits with the ones that have the same functionality.

C. Related Work

Prior works have been proposed to optimize quantum circuits at the gate level. Venturelli et al. [46] proposed an automated, architecture-aware software framework aided by constraint programming. Nam et al. [32] developed automated optimization methods using phase polynomials. There are also noise-adaptive compilers [30], [31], [43] which take advantage of the noise characteristics of the target backend to aid the optimization.

Circuit equivalence has been discussed in terms of quantum algorithms [47], quantum compiler optimization [36] and quantum compilation verification [4]. Two circuits are considered as equivalent when their unitary matrix representations are identical [5]. Garcia-Escartin et al. [14] proposed a list of equivalent rules for identifying equivalent circuits. In compiler optimization, the process of substituting sub-circuits with their equivalent ones is termed as peephole optimization.

Peephole optimization [28] identifies small sets of instructions and substitutes them with equivalent sets that have better performance. Prasad et al. [36] proposed a quantum circuit optimization algorithm that relies on peephole optimization.

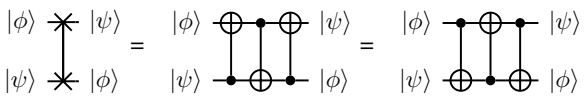


Fig. 2: 3-$CNOT$ decomposition for $SWAP$ gate

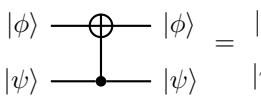
Sivarajah et al. [39] introduced a compiler named $t\,|ket\rangle$ for noisy quantum computers. The peephole optimization in $t\,|ket\rangle$ compiler traverses through the circuit to find long sequences of single-/two-qubit gates and replaces them with the circuits generated by Euler and KAK decomposition [24]. The similar optimization exists in other quantum compilers. For example, the transpiler in Qiskit Terra [3] contains optimization pass `Collect2qBlocks` and `ConsolidateBlocks`, which collaboratively identify and substitute two-qubit blocks with equivalent circuits. The Cirq [1] framework also provides similar optimization for adjacent single- and two-qubit gates.

Hoare logic [19] has been used to optimize quantum circuits. The optimizer removes trivial operations based on postconditions of the subroutines and the triviality conditions. The postconditions can be derived from the hoare triples of quantum subroutines. While our approach shares a few common optimization cases with the prior work, our optimization is more generic as we include optimizations for the quantum gates that are not trivial, e.g., the one shown in Eq. 6. Besides generality, the quantum state analysis discussed in Section VI provides fine-grained analysis for quantum states and enables more quantum gate optimizations. Moreover, since the hoare logic pass requires a classical $Z3$ solver [11] to express the conditions, it significantly increases the transpilation time. In Section VIII, we show that our optimization pass is faster and more effective than the hoare logic pass.

III. Zero State

We use a basic optimization, which only leverages a $|0\rangle$ state, as a stepping stone for more generic optimizations (see Section V). We also introduce some handful notations and concepts.

Consider the following situation: if a $CNOT$ gate with the control qubit being in the $|0\rangle$ state, the $CNOT$ gate has no effect and can be removed or replaced with a wire:

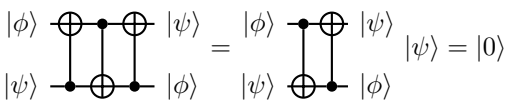


$$if \;\; |\psi\rangle = |0\rangle \qquad (1)$$

The unitary matrices of the $CNOT$ gate and the idle wire are obviously different, and they are not considered as equivalent in the existing compilers. However, they are functionally equivalent when the control qubit is in $|0\rangle$ state.

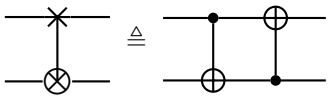
Although this special $CNOT$ example may seem trivial, we can leverage it for optimizing $SWAP$ gates. $SWAP$ gates are often introduced during the logical-to-physical qubit mapping to allow quantum operations on qubits that not physically adjacent. Being symmetric gates, $SWAP$ gates has two possible decompositions as showed in Figure 2.

Many times $SWAP$ gates are introduced in the beginning part of a circuit, when there is a good chance that one of the qubits to be swapped is still in the initial ground state $|0\rangle$. In such cases, we can take advantage of a reduced $SWAP$ gate design, which only swaps any state $|\psi\rangle$ with the state $|0\rangle$:



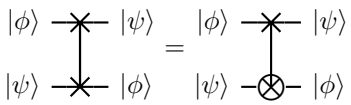
$$|\psi\rangle = |0\rangle \tag{2}$$

Let us introduce a notation for this special 2-$CNOT$-$SWAP$ gate. It is referred to as $SWAPZ$ (swap-zero) in the remaining of the paper:



(3)

In summary, a $SWAP$ gate that involves a $|0\rangle$ state can be replaced for a less expensive $SWAPZ$. This equivalence follows from Equations 2 and 3. The first $CNOT$ gate from the $SWAP$ decomposition can be removed as its control qubit is in $|0\rangle$ state. The resulting $SWAPZ$ gate consists of two $CNOT$ gates, one less compared to the generic $SWAP$ gate:



$$if \;\; |\psi\rangle = |0\rangle \tag{4}$$

IV. $SWAP$ on Pure States

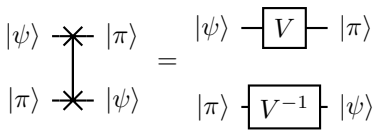
It is possible to extend the explained zero-state optimization of a $SWAP$ gate to any known single-qubit pure state.

A qubit is in a pure state up to a certain point if all the operations on that qubit up to that point do not entangle with other qubits. A pure state can be represented by a single state vector $|\pi\rangle$ in the Hilbert space. Based on the derivation in Appendix A, the single-qubit pure state $|\pi\rangle$ can be obtained by applying a single unitary gate U to the $|0\rangle$ state, $|\pi\rangle = U|0\rangle$. Due to unitarity, we can apply the inverse gate U^{-1} to transform the qubit in the pure state $|\pi\rangle$ back to the state $|0\rangle$.

When a $SWAP$ gate operates on two qubits $|\psi\rangle$ and $|\pi\rangle$ and one of them, e.g., $|\pi\rangle$, is in a known pure state generated with $U|0\rangle$, the $SWAP$ gate can be replaced with a $SWAPZ$ gate. As shown in Equation 5, the inverse gate U^{-1} transforms the pure state $|\pi\rangle$ to state $|0\rangle$ (and back to $|\pi\rangle$ after the $SWAP$). Then, the $SWAP$ gate is to swap state $|0\rangle$ with state $|\psi\rangle$, thereby can be optimized.

Altogether, when a $SWAP$ gate operates on an arbitrary state $|\psi\rangle$ and a known pure state $|\pi\rangle$, we can optimize the $SWAP$ gate into a $SWAPZ$ gate along with two single-qubit gates. This reduces the number of $CNOT$ gates by one and introduces two extra single-qubit gates. In the noisy quantum systems, the gate error of a $CNOT$ gate is usually much higher than the accumulated error of two single-qubit gates. For example, on `ibmq_16_melbourne`, the single-qubit gate error rates range from 10^{-4} to 10^{-3} while the two-qubit $CNOT$ gate's error rate is 10^{-2} [44]. Moreover, the single-qubit gate U and U^{-1} may be canceled or merged during the subsequent compiler optimization passes where multiple single-qubit gates are fused into one single-qubit gate. In those cases, we essentially reduce the number of $CNOT$ gates by one and do not introduce any additional gates.

Consider the special case where both $|\psi\rangle$ and $|\pi\rangle$ are in known pure states. We can always find a single-qubit unitary gate V to transform $|\psi\rangle$ state to $|\pi\rangle$ state, $|\pi\rangle = V|\psi\rangle$. Similarly, the inverse gate V^{-1} transforms the $|\psi\rangle$ state to $|\pi\rangle$ state, as shown in Equation 6. Therefore, we can substitute the $SWAP$ gate with two single-qubit gates V and V^{-1} in this special case.



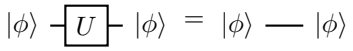
$$if \;\; |\pi\rangle = V|\psi\rangle \tag{6}$$

V. Basis-State and Pure-State Optimization

Sections III and IV showcase that for $CNOT$ and $SWAP$ gates, if some input qubit states can be determined, the gates can be simplified. In this section, we generalize this optimization to more basis and pure states in the context of different types of quantum gates. We term such optimizations as quantum basis-state optimization (QBO) and quantum pure-state optimization (QPO), respectively.

A. Optimizing Single-Qubit Gates

For single-qubit gates, if the input is in a particular basis state, they can be simplified. In particular, if the input state is an eigenstate of the gate and the eigenvalue is 1, i.e., $|\phi\rangle = U|\phi\rangle$ the gate U can be removed, as shown in Equation 7. For example, an X gate with the input $|+\rangle$ has not effect and can be removed since $|+\rangle = X|+\rangle$.



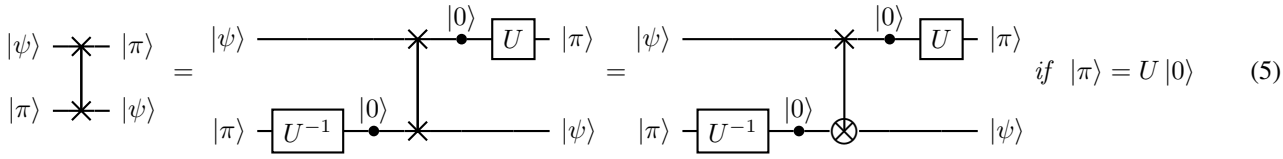
$$, if \;\; |\phi\rangle = U|\phi\rangle \tag{7}$$

Realistically speaking, the opportunity for such single-qubit gate optimization is limited in useful quantum programs. However, it lays the foundation for QBO for multi-qubit gates.

B. Optimizing $CNOT$

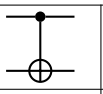
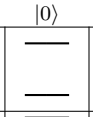
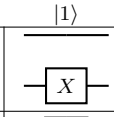
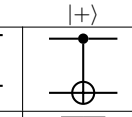
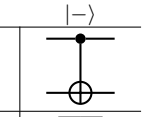
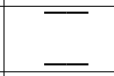
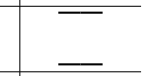
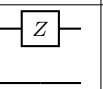
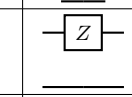
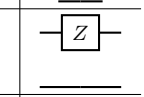
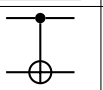
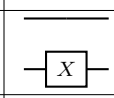
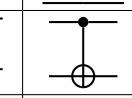
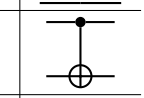
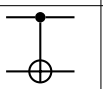
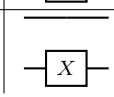
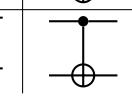
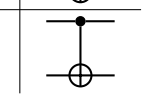
Since the $CNOT$ gate is the essential component to entangle qubits, we first focus on the basis states that have special effect on $CNOT$ gates. For $CNOT$ gates, the observation from Equation 1 can be generalized to other basis besides $|0\rangle$. These basis states include $|0\rangle$, $|1\rangle$, $|+\rangle$, and $|-\rangle$. The $CNOT$ gates can be optimized when their control qubits are in $|0\rangle$ or $|1\rangle$ state or when their target qubits are in $|+\rangle$ or $|-\rangle$ state. We list all the possible combinations and the corresponding optimized circuits in Table I, where $\top$ represents a state, which is not the X- or Z- basis ($|+\rangle$, $|-\rangle$, $|0\rangle$, $|1\rangle$).

When the target qubit is in the $|+\rangle$ state, i.e., $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$, the X gate has no effect as $|+\rangle$ is its eigenstate with eigenvalue of 1, i.e., $|+\rangle = X|+\rangle$. Therefore, the $CNOT$ gate can be



(5)

TABLE I
EQUIVALENCES FOR BASIS-STATES IN 

$\|\psi\rangle$ \ $\|\phi\rangle$	$\top$	$\|0\rangle$	$\|1\rangle$	$\|+\rangle$	$\|-\rangle$
$\top$					
$\|+\rangle$					
$\|-\rangle$					
$\|0\rangle$					
$\|1\rangle$					

replaced with wires no matter what state the control qubit is in. On the other hand, if the target qubit is in the state $|-\rangle$, i.e., $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$, the $CNOT$ gate may be replaced with a wire on the target qubit and a Z gate on the control bit. The derivation is in Appendix B.

Furthermore, the Z gate can be eliminated when the control qubit is known to be in $|0\rangle$ state. Since $|0\rangle$ state is the eigenstate of Pauli matrix Z with eigenvalue equals to 1, applying the Z gate will not change the state: $Z|0\rangle = |0\rangle$.

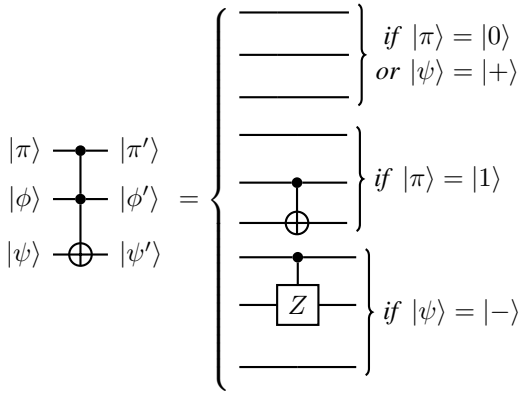
Similarly, we derive the optimized $SWAP$ gate for X- and Z-bases input combinations, which are shown in Appendix E. This optimization can be seen as a particular case of QPO, since a basis-state is also a pure-state. However, implementing QBO separately is more efficient, since it requires fewer changes in the circuit. We also derive similar simplifications to Table I for the controlled-Z gates with inputs in the Z-basis states ($|0\rangle$, $|1\rangle$).

C. Optimizing Multi-Qubit Gates

Besides $CNOT$ gates and $SWAP$ gates, QBO and QPO can be generalized to optimize a broader range of quantum gates.

First, QBO for $CNOT$ gates can be generalized to optimize the multi-controlled NOT gates. The optimization of multi-controlled NOT gates with some known inputs can be derived as follows. 1) If any of the control qubits is in the $|0\rangle$ state, we can remove the multi-controlled NOT gate. 2) If any of the control qubits is in the $|1\rangle$ state, we can remove the control qubit and substitute the gate using a multi-controlled NOT gate with one less control qubit. 3) If the target qubit is in $|+\rangle$ state, we can remove the multi-controlled NOT gate. 4) If the target qubit is in $|-\rangle$ state, we can substitute the gate with a multi-controlled Z gate with the target on any of the previous control qubits. In our study, we find that Toffoli gates are widely used in quantum algorithms. A Toffoli gate is a NOT gate with two controlled qubits. Equation 8 presents the optimizations for the Toffoli gate. The open controlled NOT gates can also be optimized using a similar approach as discussed is in Appendix C.

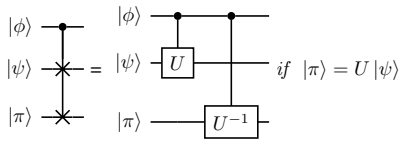
Second, the optimization of multi-controlled NOT gates can be further generalized to multi-controlled unitary gates. When the control qubits are in $|0\rangle$ or $|1\rangle$, the optimization rules are the same. Assume the unitary gate U has eigenstates $|\psi_+\rangle$ and $|\psi_-\rangle$ with eigenvalues of 1 and -1, respectively. We have $|\psi_+\rangle = U|\psi_+\rangle$ and $|\psi_-\rangle = -U|\psi_-\rangle$. When the target qubit is in $|\psi_+\rangle$ and $|\psi_-\rangle$ the optimization rules are the same as the rules for multi-controlled NOT gates with respect to $|+\rangle$ and $|-\rangle$ states.



(8)

Third, QBO/QPO for $SWAP$ gates can also be extended to the Fredkin gate (aka $CSWAP$ gate) and multi-controlled $SWAP$ gates. A Fredkin gate is a controlled $SWAP$ gate with a single control qubit. A Fredkin gate can be decomposed into two $CNOT$ gates and a Toffoli gate. The decomposition is included in Figure 14 in Appendix D. In the same way, multi-controlled $SWAP$ gates can be decomposed into two $CNOT$ gates and a multi-controlled NOT gate. If the control qubit $|\psi\rangle$ is in the $|0\rangle$ state, we can remove the Fredkin gate. If it is in the $|1\rangle$ state, we can substitute the Fredkin gate with a $SWAP$ gate. If any of the target states $|\psi\rangle$ or $|\pi\rangle$ is in a known basis state, we can optimize the first $CNOT$ gate accordingly. If both of the target states $|\psi\rangle$ and $|\pi\rangle$ are in known pure states, following the optimization in Equation 6, we can substitute the Fredkin gate with two controlled U gates, as shown in Equation 9, where gates U and U^{-1} have the relationship $|\pi\rangle = U|\psi\rangle$ and $|\psi\rangle = U^{-1}|\pi\rangle$. Since a Toffoli gate can be implemented with six $CNOT$ gates and eight single-qubit gates [38], the Fredkin gate would need eight $CNOT$ gates and eight single-qubit gates following the decomposition in Figure 14. In comparison, a controlled-U gate can be implemented with at most two $CNOT$ gates and four single-qubit gates [40]. Therefore, our

optimized Fredkin gate in Equation 9 would require at most four $CNOT$ gates and eight single-qubit gates. As a result, our proposed QPO reduces at least four $CNOT$ gates for a Fredkin gate with known pure-state inputs.



(9)

D. Optimizing Qubit Blocks

Our proposed QPO can be further generalized to optimize two-qubit blocks. A sequence of uninterrupted two-qubit gates is considered as a two-qubit block [3]. The Qiskit transpiler optimizes these blocks by calculating the unitary matrices of these blocks and resynthesizing them using the KAK decomposition [23], [24]. Generally speaking, a two-qubit block can decomposed to a circuit consisting of at most three $CNOT$s and eight single-qubit gates [48] as shown in Figure 3. If the two input qubit states $|\psi\rangle$ and $|\pi\rangle$ are in known pure states, the compiler can calculate the output state $|\phi\rangle$ statically. Existing research [29] has proved that any two-qubit state can be prepared by a $CNOT$ gate and four single-qubit gates. It means that we can substitute the two-qubit block with the state preparation circuit shown in Figure 4. As a result, we reduce the number of $CNOT$ gates by two and the number of single-qubit gates by four.

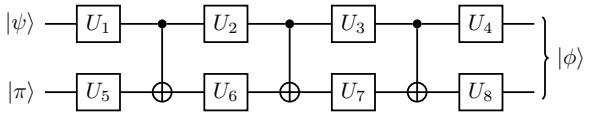


Fig. 3: Universal decomposition of two-qubit block U

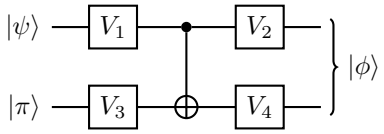


Fig. 4: Universal circuit for preparing quantum state $|\phi\rangle$

The optimization of the two-qubit block can be generalized to n-qubit blocks. If we know the input state and the output state of an n-qubit block, we can use the state preparation circuit to substitute the original circuit. It has been proved that preparing a quantum state can require less $CNOT$ gates than preserving the unitary matrix [35].

VI. Quantum State Analysis

A. Basis-State Analysis

For the purpose of applying QBO, we implemented a state automata, partially shown in Figure 5, to track the basis state of each qubit. The automata consist in six distinguished single-qubit states: $|0\rangle$, $|1\rangle$, $|+\rangle$, $|-\rangle$, $|L\rangle$, and $|R\rangle$. Every single-qubit half- and quarter-turn gate transitions the states in the automata (not all of them reflected in Figure 5). Any other

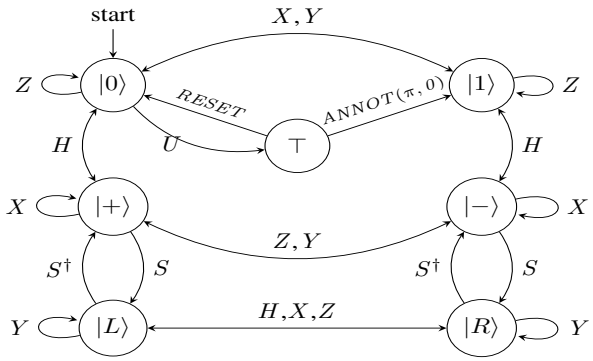


Fig. 5: Partial automata for single-qubit basis-state analysis

gate or operation (except special cases for $SWAP$, $SWAPZ$, and $RESET$), denoted as U, would change the state of the qubit to the *unknown non-basis state* $\top$. That is represented in the automata graph only with the $|0\rangle$ example but the same applies to all the other states.

As quantum processors are initialized in its lowest-energy state known as *ground state*, all the qubits start in state $|0\rangle$. The half- and quarter-turn gates transform basis states into basis states. For example, the Hadamard gate H (a half-turn gate) moves the Z-basis into the X-basis and vice versa. The loop transitions in each node indicate a gate with no effect on that state, i.e. it is the eigenstate with an eigenvalue of 1, as explained in Section V-A.

The instruction $RESET$ turns any state into the zero state. In Figure 5, that is illustrated as the only transition able to downgrade from $\top$ to $|0\rangle$. The annotation instruction $ANNOT$, to be discussed in subsection VI-C, can transit between different states. Essentially, it can be used to change the state from $\top$ to a basis state, as exampled with the edge between $\top$ and $|1\rangle$.

The basis-state analysis also considers the effect of $SWAP$ and $SWAPZ$. When they are encountered, the states of the involved qubits are swapped, including $\top$.

B. Pure-State Analysis

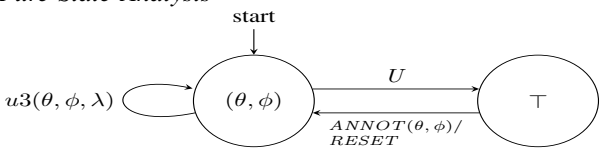


Fig. 6: Automata for pure-state analysis

As QBO, QPO requires a pure-state analysis to track the single-qubit pure states. A single qubit is in pure state when it is not entangled with the other qubits. Any single-qubit pure state can be represented by a state vector with two parameters θ and ϕ, $|\psi(\theta,\phi)\rangle = cos(\frac{\theta}{2})|0\rangle + e^{i\phi}sin(\frac{\theta}{2})|1\rangle$. Knowing the pure state $|\psi(\theta,\phi)\rangle$, we can use a single-qubit gate $u3(\theta,\phi,0)$ to generate this state from $|0\rangle$ state, $|\psi(\theta,\phi)\rangle = u3(\theta,\phi,0)|0\rangle$. Therefore, we choose these two parameters to record the single-qubit pure state information. In our analysis, each qubit associates a tuple (θ, ϕ). When the qubit is not in pure state, the parameters are set to $\top$.

The tuple (θ, ϕ) is updated to track the pure state information for each qubit. Since any single-qubit gate can be expressed by the $u3(\theta, \phi, \lambda)$ gate, when we apply a $u3(\theta, \phi, \lambda)$ gate to a qubit in the pure state (θ_0, ϕ_0), the output state would be a pure state (θ_1, ϕ_1). Calculating the parameters (θ_1, ϕ_1) of the output state is analogous to merging two $u3$ gates, since $|\psi(\theta_1, \phi_1)\rangle = u3(\theta, \phi, \lambda)\,|\psi(\theta_0, \phi_0)\rangle = u3(\theta, \phi, \lambda)\,u3(\theta_0, \phi_0, 0)\,|0\rangle = u3(\theta', \phi', \lambda')\,|0\rangle = u3(\theta_1, \phi_1, 0)\,|0\rangle$. Since the λ' parameter does not change the $|0\rangle$ state and can be ignored, we have $\theta_1 = \theta'$ and $\phi_1 = \phi'$. In our implementation, we leverage the gate merging function in Qiskit to calculate the output state parameters θ_1 and ϕ_1.

When a multi-qubit gate is applied to the qubits in pure state, the output might be in mixed state. Therefore, the resulting states are marked as $\top$ for each implicated qubit. The $RESET$ instruction will reset the qubit back to ground state $(0, 0)$. The $ANNOT(\theta, \phi)$ annotation, to be discussed in subsection VI-C, will transform the qubit to pure state (θ, ϕ). The transitions among these states are illustrated by the automata in Figure 6. Similar to the basis-state analysis, our pure state analysis considers $SWAP$ and $SWAPZ$ gates. When they are encountered, the pure states of the involved qubits are swapped, including $\top$.

Section IV discusses the optimization for a $SWAP$ gate with two known pure states. The optimization needs the unitary gates that transform one pure state to the other. With the two-parameter (θ, ϕ) representation, it is easy to generate such unitary gates. The gate $u3(\theta_2 - \theta_1, \phi_2 - \phi_1)$ transforms the pure state $|\psi(\theta_1, \phi_1)\rangle$ to $|\psi(\theta_2, \phi_2)\rangle$, $|\psi(\theta_2, \phi_2)\rangle = u3(\theta_2 - \theta_1, \phi_2 - \phi_1)\,|\psi(\theta_1, \phi_1)\rangle$.

C. State Annotation

Determining whether a generic quantum state is entangled is an NP-hard problem [17]. In general, it is hard to infer information about the states from a quantum circuit using a classical machine efficiently. However, based on the understanding of the quantum program, the programmer can provide information to facilitate state analysis. For example, in quantum networks for elementary arithmetic operations [45], the network uses reverse computation to unentangle and reuse qubits. The programmers know these qubits are unentangled after reverse computation and they can annotate that these qubits are in particular pure states. Another example is, "clean" ancilla qubits are commonly used in quantum computing. The "clean" ancilla qubits are in $|0\rangle$ state and can be reused after the gate. As shown in Figure 7, after the multi-controlled Toffoli gate, the ancilla qubit remains to be in state $|0\rangle$. To leverage such user-level information, we introduce annotations, which inform the compiler that the qubits are in certain quantum states. For example, in the pure state analysis, we use the annotation $ANNOT(\theta, \phi)$ to indicate a quantum state is in pure state $|\psi(\theta, \phi)\rangle$. The programmer can insert annotations based on the understanding of the quantum program. The annotations can also be inserted by the compiler automatically. For example, when the programmer uses the gate design with

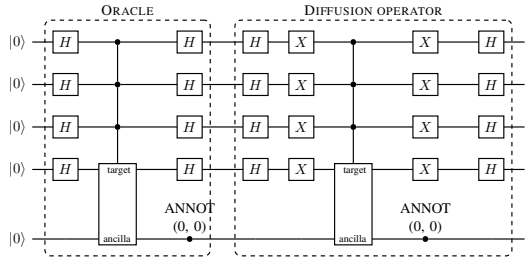


Fig. 7: 4-qubit Grover's algorithm using multi-controlled Z gates with "clean" ancilla qubits and annotations

"clean" ancilla qubits, the compiler may automatically insert annotations $ANNOT(0, 0)$ for such ancilla qubits.

By introducing the annotations, we can avoid the complex quantum state analysis and improve the scalability of our proposed optimization pass.

VII. METHODOLOGY

A. Compiler Implementation

We implemented our QBO and QPO passes on the open-source quantum computing framework Qiskit 0.18 [3] and our implementation is publicly available [1]. Qiskit organizes the transpilation passes in pass managers and it currently includes four pass managers for each level of optimization. The level 3 provides the maximal optimization at the expense of longer transpilation time. As part of our implementation, we extended the pass manager for level 3 to include our optimizations.

Figure 8 outlines the sequence of passes in level 3 and the additions (underlined) that we introduced. The input circuit is run through the `QBO` pass first. The effect of this early optimization cascades in the rest of the pass manager, since any reduction in the gate count will improve the speed and effectiveness of subsequent passes. Additionally, `QBO` checks basis states of the $SWAPZ$ gates in the input circuit, if there are any. If the condition in Equation 4 does not hold for a specific $SWAPZ$ gate, the gate is decomposed into 2 $CNOT$ gates, following the definition from Equation 3. This guarantees that the $SWAPZ$ gates from this point on are semantically equivalent to $SWAP$ gates. After the $SWAP$ gates are inserted during the routing process (line 4) a new pass of `QBO` (line 5) optimizes those inserted $SWAP$ gates. In line 6 and 7, we reuse existing Qiskit functionalities, `Unroller` and `Optimize1qGate` to prepare for the `QPO` pass (line 8). The `Unroller` pass decomposes all the circuit gates into the list of gates defined by the parameter. The variable `basis_gates` is a list of the primitive gates supported by the quantum device. In line 8, the list is extended with the gates $SWAP$ and $SWAPZ$, since `QPO` understands them. The `Optimize1qGates` pass merges the single-qubit gates into a single unitary gate. After `QPO`, the circuit is optimized in a loop until a fixed point is reached. This loop is expensive and we decided to place `QBO` and `QPO` out of it. The loop iterates at least twice in order to find the fixed point. The optimizations in the loop (line 10) do

[1] https://github.com/lucian0/rpo [6]

```
1  QBO()
2  Unroller(basis_gates)
3  <layout selection>
4  <routing process>
5  QBO()
6  Unroller(basis_gates + swap + swapz)
7  Optimize1qGates()
8  QPO()
9  while not <fixed point>{
10   <optimizations>}
```

Fig. 8: Optimization level 3 in Qiskit 0.18 (RPO additions underlined)

not modify the state invariant on the qubits. Therefore, there is no gain running `QBO`/`QPO` more than once.

B. Benchmarks and System Configuration

To evaluate our proposed compiler optimization, we run our experiments upon the following algorithms:

Bernstein-Vazirani algorithm: A blackbox function $f(x)$ is guaranteed to be the dot product between x and a bit string s: $f(x) = x \cdot s$. Given an oracle that implements $f(x)$, the algorithm finds the hidden bit string s with a single evaluation.

Quantum Phase Estimation (QPE) algorithm: QPE estimates the phase of an eigenvector of a unitary matrix. Given a quantum state $|\psi\rangle$ which is the eigenvector of a unitary matrix U, $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, QPE estimates the phase θ.

VQE algorithm: Variational Quantum Eigensolver (VQE) is a hybrid quantum/classical algorithm which finds the eigenvalues of a matrix H. In the VQE algorithm, the circuit for preparing the quantum state is called *anstaz*. We use the VQE program and the hardware-efficient ansatz RY from Qiskit Aqua [3]. In our experiment, we use the VQE algorithm to solve the Max-Cut problem [15].

Quantum Volume: Quantum volume [10] is a metric for characterizing quantum system performance. It is calculated by taking various quantum computer features into account, such as gate error, and connectivity. The quantum volume circuit is randomly generated with a fixed but generic form.

Grover's search algorithm: Given a set X of N elements and a boolean function $f : X \to 0, 1$, Grover's algorithm finds an element x_i in X such that $f(x_i) = 1$.

Since the $RESET$ is currently not supported by the IBMQ quantum hardware, none of the circuits used in our experiments include the $RESET$ instruction. We run our experiments with connectivity maps and noise properties from three different quantum computers, a 15-qubit quantum computer `ibmq_16_melbourne`, a 20-qubit quantum computer `ibmq_almaden`, and a 53-qubit quantum computer `ibmq_rochester`. The connectivity maps of these three quantum computers are shown in Figure 9. We compare our optimization pass with the hoare logic pass [2] implemented in the Qiskit transpiler. For a fair comparison, we append the hoare logic pass to the level 3 pass manager. We also run our experiments on these real quantum computers to evaluate the fidelity rate improvement from our compiler optimization.

Each circuit was transpiled several times to mitigate the effect of corner cases given the non-deterministic nature of Qiskit transpiler. For example, the `StochasticSwap` routing pass included in Qiskit returns significantly different results depending on the random seed and the input circuit. The reported $CNOT$ gate count and the transpilation time are the medians of twenty-five (25) transpilation results. The reported median $CNOT$ gate count is very close to the average $CNOT$ gate count. We use the geometric mean to calculate the average ratio of $CNOT$ gate reduction.

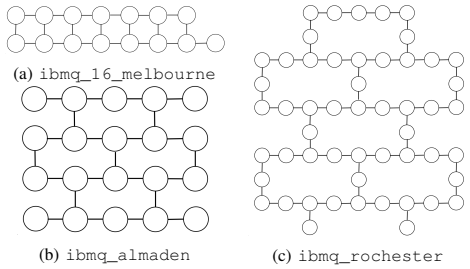


Fig. 9: Connectivity map of three different IBM quantum computers

VIII. Performance

In this section, we first show a case study on the Bernstein-Vazirani algorithm, for which our optimization pass will optimize the boolean oracle with $CNOT$ gates into phase oracle with single-qubit gates. Then, we provide case studies on four widely used algorithms namely the quantum phase estimation (QPE) algorithm, variational quantum eigensolver (VQE) algorithm, quantum volume benchmark, and Grover's algorithm. Subsequently, we show the annotations improve the scalability of our optimization. Next, we study the impact of the backend connectivity on the optimization. In the end, we run the experiments on real quantum computers to show the success rate improvement with our optimization.

A. Bernstein-Vazirani Algorithm

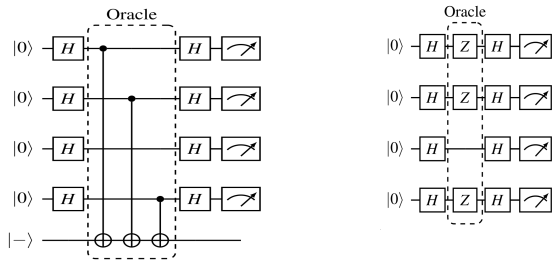
Circuits implementing the Bernstein-Vazirani Algorithm are used extensively for benchmarking in recent quantum computing researches [41]–[43]. QBO has a particular effect on this algorithm implementations that is noteworthy.

The oracle that implements the function $f(x)$ can be represented in two different ways. The boolean oracle method converts an irreversible computation to a reversible one [7]. The other method is to use phase oracles [22], encoding $f(x)$ into phase amplitudes. The phase oracle for the Bernstein-Vazirani algorithm can be done with only Z gates [12]. Figure 10 shows two different implementations of 4-qubit Bernstein-Vazirani algorithm with hidden bit string $s = 1011$.

The first design requires an extra ancilla qubit and $CNOT$ gates. The second design only includes single-qubit gates and it is more feasible for noisy quantum systems. Notice that the ancilla qubit is in the $|-\rangle$ state. Following the discussion in Section V-B, our QBO pass substitutes the $CNOT$ gates with Z gates and the optimized circuit is the same as the design with phase oracle. In other words, QBO converts the design in Figure 10a into Figure 10b.

TABLE II
MEDIAN OF $CNOT$ GATES AND TRANSPILATION TIME OF THREE QUANTUM ALGORITHMS WITH DIFFERENT SIZE (ON `IBMQ_16_MELBOURNE`)

	QPE						VQE						Quantum Volume						Grover					
Metric	$CNOT$ gate count			transpile time(s)			$CNOT$ gate count			transpile time(s)			$CNOT$ gate count			transpile time(s)			$CNOT$ gate count			transpile time(s)		
Optimization	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO
4-qubits	24	21	18	0.29	0.41	0.28	56	51	47	0.43	0.77	0.42	38	28	26	0.39	0.74	0.39	168	159	157	1.80	2.14	1.51
6-qubits	66	62	54	0.81	1.00	0.73	147	141	136	1.53	1.62	1.38	75	75	72	1.73	1.93	1.51	359	345	322	4.61	5.72	4.78
8-qubits	124	117	106	1.34	1.74	1.24	301	289	285	1.95	2.71	1.99	165	158	147	2.92	3.85	3.27	1551	1491	1463	16.6	30.2	13.8
10-qubits	205	197	172	1.88	2.30	1.59	485	470	459	2.77	4.78	2.61	327	313	282	6.39	7.15	5.04	6358	6309	6275	52.4	303.7	45.9
12-qubits	268	261	225	2.77	4.39	3.11	720	699	683	4.74	6.76	4.37	429	424	399	7.56	11.84	7.36	25386	25254	25008	232.5	10271.8	231.8
14-qubits	500	500	451	4.95	10.98	7.04	1142	1136	1136	5.74	11.72	5.69	1505	1491	1479	19.62	25.94	16.27	101020	N.A.	100762	1769.4	N.A.	1828.2



(a) Boolean oracle (b) Phase oracle

Fig. 10: Two different circuits for Bernstein-Vazirani algorithm

We found that our optimization can optimize the costly boolean oracle to a design which has the same cost as the phase oracle. Besides the Bernstein-Vazirani algorithm, the boolean oracles for the Grover's algorithm [13] and general cases [33] can also be optimized by our pass. In comparison, such boolean oracles can't be optimized by the Qiskit compiler or the hoare logic pass.

B. Quantum Algorithms

In this section, we consider four practical quantum algorithms: quantum phase estimation, VQE, quantum volume, and Grover's search algorithm. We compare RPO against the Qiskit compiler with optimization level 3 and the optimization level 3 with hoare logic pass, using the backend properties from `ibmq_16_melbourne`, which has 15 qubits. The median of $CNOT$ gate count and transpilation time are shown in Table II. For all of the circuits, the resulting $CNOT$ gate count of our pass manager is less than or equal to that of level 3. For most of the circuits, the compilation time of our pass manager is shorter than that of level 3, even though we included extra optimization passes. This is due to the early QBO, which cascades its effect to the rest of the passes in the pass manager. Since any reduction in the gate count will improve the subsequent passes, the overall compile time can be reduced. Our RPO pass results in more efficient circuit design and less compile time compared to the hoare logic pass. By checking the optimized circuit, we found that all the gates that are optimized by the hoare logic pass can be captured by our RPO pass. The median of single-qubit gate count and circuit depth are shown in Table VI in Appendix F. As we can see from Table VI, both the single-qubit gate count and circuit depth are improved as a result of our optimization.

For the QPE algorithm, when the logical circuit is decomposed to the basic gates, some of the $CNOT$ gate can be optimized by our compiler pass. Therefore, our optimized circuits have lower $CNOT$ gate count for all different numbers of qubits. Notice that our optimization has a significant impact for the shallow circuits. For the 4-qubit QPE algorithm, our optimization reduced the $CNOT$ gate count by 25%. On average, our optimization leads to 18.0% decrease in the $CNOT$ gate count and 5.5% decrease in the transpilation time for the QPE algorithm.

For the VQE algorithm, we use the hardware-efficient ansatz RY as the circuit design. The hardware-efficient ansatz is concise which limits the possible optimizations. However, when mapped to the physical qubits, the compiler introduces extra $SWAP$ gates. Therefore, it is still possible to optimize the circuit. As the number of qubit increases, the number of $CNOT$ gate optimized by our pass also increases. However, when the qubit count is close to the total number of qubits in the device (for this case 15), all the qubit will quickly fall into the non-basis/pure state $\top$, and our optimization only optimized a small amount of $CNOT$ gates. In the best case, our optimization reduced the $CNOT$ gate count by 16% for the 4-qubit VQE algorithm. Our optimization leads to an average of 5.8% decrease in the $CNOT$ gate count and 7.7% decrease in the transpilation time for VQE algorithm.

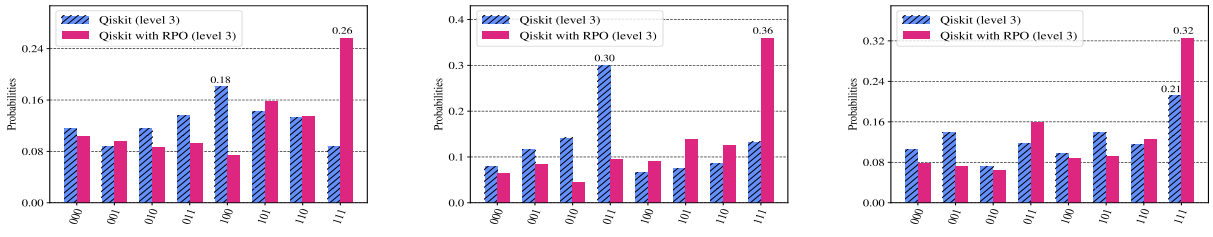
For the quantum volume benchmark, since it is a randomly generated benchmark, the qubits are entangled and it is difficult to analyze the quantum states. Nevertheless, our optimization remains effective.

Since the long compile time may cause compilation failure, we only compile one iteration of the Grover's algorithm. In the 14-qubit case, the hoare logic pass failed due to long compilation time. On average, our optimization reduces the $SWAP$ gate count by 2.4% and the transpilation time by 7.3%.

Across these four benchmarks, our optimization reduces the $CNOT$ gate count by $11.7\%/4.5\%$ and the transpilation time by $7.1\%/40.0\%$ on average compared to Qiskit level 3 and Hoare logic, respectively.

C. Quantum Algorithm with Annotations

In this section, we use the Grover's algorithm to demonstrate that annotations can significantly improve the scalability of our optimization. It is a common practice to use quantum gates with ancilla qubits. Introducing ancilla qubits can significantly reduce the circuit size. For example, when using the multi controlled Toffoli gate without ancilla qubits, the 8-qubit grover's algorithm circuit consists of approximately ~ 1500 $CNOT$ gates. We can use another design which requires six "clean" ancilla qubits. The ancilla qubit design only consists of approximately ~ 400 $CNOT$ gates. Similar to Figure 7, we can add annotations for the "clean" ancilla qubit.



(a) ibmq_16_melbourne (b) ibmq_almaden (c) ibmq_rochester

Fig. 11: Output distribution of QPE algorithm on three different quantum computers

The n-qubit Grover's algorithm requires $O(\sqrt{2^n})$ iterations to maximize the probability amplitude of the correct output. Each iteration consists of an oracle and a diffusion operator. We use the multi-controlled Toffoli gate with ancilla qubits and test the 8-qubit Grover's algorithm with different number of iterations. The experiment results of different optimization passes are shown in Table III. Without annotations, after first few iterations, all the qubits will fall into the non-basis/pure state $\top$. Therefore, $CNOT$ gate count reduced by RPO is ~ 30 regardless of the number of iterations. By introducing annotations, the qubits can transfer from $\top$ back to basis/pure state. For the 2-iteration case, our optimization reduced the $CNOT$ gate count by 10.8%. When the number of iteration is greater than eight, our optimization reduced a constant fraction $\sim 7.4\%$ of the $CNOT$ gate count.

TABLE III
MEDIAN OF $CNOT$ GATES, DEPTH, AND TRANSPILATION TIME OF GROVER'S ALGORITHM WITH DIFFERENT NUMBER OF ITERATIONS

Metric	$CNOT$ gate count			depth			transpile time(s)		
Optimization	level3	RPO	RPO w/ Annot	level3	RPO	RPO w/ Annot	level3	RPO	RPO w/ Annot
2-iteration	653	625	583	626	619	600	7.37	7.15	6.84
4-iteration	1315	1280	1187	1249	1238	1200	15.11	14.88	13.89
6-iteration	1882	1847	1709	1826	1815	1748	21.41	21.78	19.59
8-iteration	2559	2527	2362	2443	2435	2350	25.74	29.20	27.45
10-iteration	3111	3079	2886	3005	2994	2897	37.49	35.65	32.61
12-iteration	3695	3660	3419	3606	3595	3478	45.51	45.02	43.09
14-iteration	4288	4251	3979	4192	4179	4051	48.67	48.12	48.02

D. Different Backend Connectivity

TABLE IV
MEDIAN OF $CNOT$ GATES AND TRANSPILATION TIME OF QPE ALGORITHM ON DIFFERENT QUANTUM COMPUTERS

	QPE ibmq_almaden				QPE ibmq_rochester			
Metric	$CNOT$ gate count		transpile time(s)		$CNOT$ gate count		transpile time(s)	
Optimization	level3	RPO	level3	RPO	level3	RPO	level3	RPO
4-qubits	26	23	0.24	0.23	25	18	0.36	0.34
6-qubits	72	56	0.57	0.53	66	51	0.77	0.72
8-qubits	157	134	0.98	0.97	236	220	2.45	2.42
10-qubits	357	312	1.62	1.54	198	147	1.96	1.87
12-qubits	413	369	2.27	2.16	444	370	3.55	3.44
14-qubits	586	537	3.36	3.25	722	644	6.17	5.77

$SWAP$ gates are introduced when the compiler performs the logical-to-physical mapping. When the backend has limited connectivity, the logical-to-physical mapping will introduce more $SWAP$ gates. Therefore, our optimization pass has a higher chance to optimize the quantum circuit.

We compile the QPE program with connectivity maps from three different quantum computers. These connectivity maps are shown in Figure 9. Among these quantum computers, ibmq_16_melbourne has the best connectivity and ibmq_rochester has the worst. The experiment result of QPE on ibmq_16_melbourne is shown in Table II in the previous section. The results of QPE on ibmq_almaden and ibmq_rochester are shown in Table IV.[2]. From these results, we can see that our optimization is effective on all these quantum computers. Another interesting observation is that, the worse connectivity the quantum computer has, the higher total $CNOT$ gate count, and the more $CNOT$ gates will be optimized by our optimization. The percentage of $CNOT$ gates reduced by our optimization are 18.0%, 15.2%, and 20.6% for ibmq_16_melbourne, ibmq_almaden and ibmq_rochester respectively. The percentage of transpiled time reduced by our optimization are 5.5%, 5.3%, and 4.6%.

E. Experiment on Real Quantum Computers

We ran the 3-qubit QPE algorithm on real quantum computers to highlight the effectiveness of reducing $CNOT$ gates. The output distribution is shown in Figure 11. The correct output should be 111. Since the circuit depth for 3-qubit QPE is shallow, the different quantum computer connectivity doesn't lead to too much difference in the total $CNOT$ count. The gate error and measurement error of different devices have higher impact on the final output distribution. Although the circuit running on ibmq_16_melbourne has the least $CNOT$ count, the fidelity of that circuit is not the best. Without our optimization, we cannot even infer the correct result on both ibmq_16_melbourne and ibmq_almaden. Our optimization reduces the $CNOT$ gate count by 33%, 29%, and 28% and leads to success rate improvements of $2.94X$, $2.69X$, and $1.53X$ on ibmq_16_melbourne, ibmq_almaden, and ibmq_rochester, respectively. The average success rate improvement (geometric mean) is $2.30X$ for 3-qubit QPE algorithm.

IX. CONCLUSIONS

In this paper, we propose a fast and effective compiler optimization named relaxed peephole optimization. Based on this optimization we designed two compiler optimization passes, QBO and QPO, and implemented them in the IBM's Qiskit transpiler. We show that our optimization pass is faster than the most aggressive optimization level in the Qiskit, and the

[2]The abnormal result of 8-qubit QPE algorithm on ibmq_rochester is due to Qiskit transpiler's layout selection function changed the shape of the coupling map subgraph

circuits optimized by our optimization pass also have fewer $CNOT$ gates. Our experiments on the real quantum computers highlight that the reduction in $CNOT$ gate count leads to a significant improvement in the circuit success rate.

ACKNOWLEDGMENT

We thank the anonymous reviewers for their valuable comments. This work is funded in part by NSF grants 1717550 and 1908406.

APPENDIX A
UNITARY U AND PURE STATE $|\psi\rangle$

We prove that an n-qubit pure state $|\psi_0\rangle$ can derived by applying an n-qubit unitary gate U to the n-qubit zero state $|0\rangle^{\otimes n}$: $|\psi_0\rangle = U|0\rangle^{\otimes n}$. Based on the state $|\psi_0\rangle$, we can leverage the Gram-Schmidt process [16] to find a set of vectors $|\psi_0\rangle, |\psi_1\rangle, ..., |\psi_{2^n-1}\rangle$ that forms an orthonormal basis. Then, the unitary gate U can be calculated as $U = |\psi_0\rangle\langle 0|^{\otimes n} + |\psi_1\rangle\langle 0|^{\otimes n-1}\langle 1| + ... + |\psi_{2^n-1}\rangle\langle 1|^{\otimes n}$.

First, we need to prove $|\psi_0\rangle = U|0\rangle^{\otimes n}$. Since the computational basis is an orthonormal basis, we have $\langle 0|0\rangle = 1$ and $\langle 1|0\rangle = 0$. Therefore,

$$U|0\rangle^{\otimes n} = |\psi_0\rangle\langle 0|0\rangle^{\otimes n} + |\psi_1\rangle\langle 0|0\rangle^{\otimes n-1}\langle 1|0\rangle + ... \\ ... + |\psi_{2^n-1}\rangle\langle 1|0\rangle^{\otimes n} = |\psi_0\rangle$$

Second, we need to prove the matrix U is a unitary matrix such that we can find a corresponding quantum gate. Since the set of vectors $\{|\psi_i\rangle\}$ form an orthonormal basis, we have $\sum_i |\psi_i\rangle\langle\psi_i| = I$, here I is the identity matrix. Based on this property, We can prove that $UU^\dagger = |\psi_0\rangle\langle 0|0\rangle^{\otimes n}\langle\psi_0| + |\psi_1\rangle\langle 0|0\rangle^{\otimes n-1}\langle 1|1\rangle\langle\psi_1| + ... + |\psi_{2^n-1}\rangle\langle 1|1\rangle^{\otimes n}\langle\psi_{2^n-1}| = \sum_i |\psi_i\rangle\langle\psi_i| = I$. Similarly, we can prove $U^\dagger U = I$. Therefore, the matrix U is a unitary matrix.

APPENDIX B
$CNOT$ GATE OPTIMIZATION WITH TARGET QUBIT IN $|-\rangle$ STATE

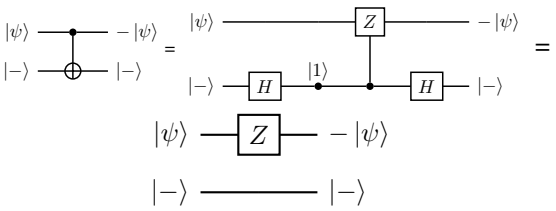


Fig. 12: $CNOT$ gate optimization

As shown in Figure 12, the $CNOT$ gate is equivalent to a controlled-Z gate with two Hadamard gates on the target qubit. When the target qubit of the $CNOT$ gate is in $|-\rangle$, after the Hadamard gate, the target qubit is in $|1\rangle$ state. The controlled-Z gate can be optimized into a Z gate, and the two Hadamard gates will be cancelled out. Therefore, the $CNOT$ gate can be substituted with a Z gate on the control qubit.

APPENDIX C
CLOSED AND OPEN CONTROL

An open control gate is equivalent to a closed control gate with two NOT gates on the control qubit, as shown in Figure 13. In our paper, we discussed the optimization for the closed control gates. For the open control gates, we can use this equivalence to convert them to the closed control gates and then apply our optimization.

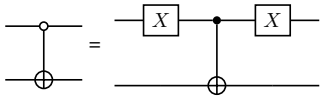


Fig. 13: Open and closed control gate equivalence

APPENDIX D
DECOMPOSITION OF FREDKIN GATE

The decomposition of Fredkin gate is shown in Figure 14

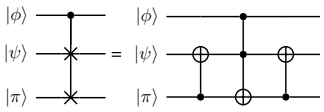


Fig. 14: Decomposition of Fredkin gate

APPENDIX E
SWAP GATE ON BASIS-STATE

Table V shows QBO implementation on the $SWAP$ gate. Take as an example the case where the $SWAP$ gate has the top input in the $|0\rangle$ state and the bottom input in the $|-\rangle$ state. Since $|0\rangle = X|1\rangle$ and $|1\rangle = h|-\rangle$, the $|0\rangle$ state can be obtained by applying H and X gate to the $|-\rangle$ state, $|0\rangle = X|1\rangle = XH|-\rangle$. Similarly, the $|-\rangle$ can be obtained by applying X and H gate to the $|0\rangle$ state.

TABLE V
EQUIVALENCES FOR BASIS-STATES IN

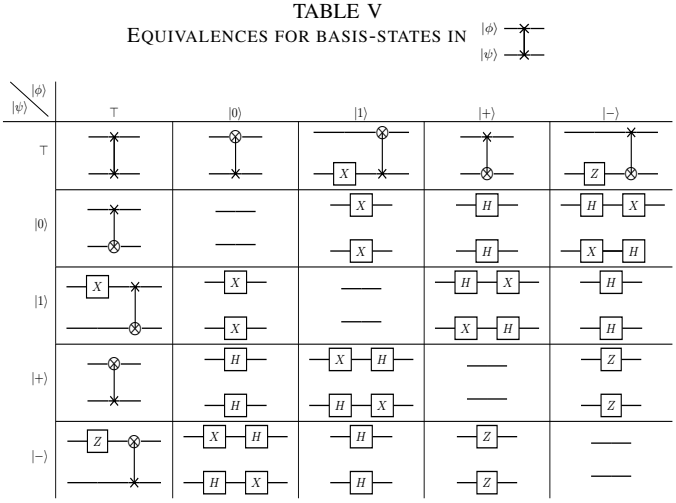


TABLE VI
MEDIAN OF SINGLE-QUBIT GATES AND CIRCUIT DEPTH OF THREE QUANTUM ALGORITHMS WITH DIFFERENT SIZE (ON IBMQ_16_MELBOURNE)

	QPE						VQE						Quantum Volume						Grover					
Metric	single-qubit gate count			depth			single-qubit gate count			depth			single-qubit gate count			depth			single-qubit gate count			depth		
Optimization	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO	level3	hoare	RPO
4-qubits	65	65	59	45	43	38	136	133	129	97	93	89	107	107	103	44	43	43	375	375	368	329	324	318
6-qubits	171	169	154	93	92	87	338	340	327	186	185	181	275	275	270	108	108	103	834	834	798	720	712	658
8-qubits	351	344	327	149	145	134	686	663	647	310	307	300	500	498	479	500	497	493	3666	3666	3502	2963	2955	2874
10-qubits	621	621	591	210	208	196	1061	1056	1053	413	410	406	802	802	758	211	209	193	14674	14666	14455	12219	12181	12012
12-qubits	879	877	832	259	258	243	1591	1582	1570	534	532	526	1199	1199	1163	261	258	244	58424	58242	57874	48315	48219	47899
14-qubits	1211	1211	1161	389	389	368	2268	2340	2263	751	751	751	2903	2910	2889	946	951	931	232290	N.A.	241716	192526	N.A.	192487

APPENDIX F
ADDITIONAL EXPERIMENT RESULTS

The detailed experiment results for QPE, VQE, quantum volume, and Grover's algorithms are included in Table VI.

APPENDIX G
ARTIFACT DESCRIPTION APPENDIX

A. Abstract

Our artifact provides the experiments for all our evaluated benchmarks, along with the experiments to validate the quantum circuit costs.

We also provide the source code for our compiler optimization passes and all of our benchmarks.

B. Artifact Check-list (Meta-Information)

- **Algorithm:** Peephole optimization algorithm
- **Data set:** Benchmarks included in our paper
- **Hardware:** We recommend running the experiments on a 15-qubit IBM Q machine to verify the results
- **Execution:** Run the corresponding python scripts and jupyter notebooks
- **Metrics:** CNOT gates: The number of CNOT gates in the quantum program.
  transpile time: The time for quantum circuit transpilation.
  single-qubit gate count: The number of single-qubit gates in the quantum circuit.
  depth: The depth of the quantum circuit.
  Success rate: The ratio of the count of correct output state over the total count of the trials.
- **Output:** The CNOT gates, transpilation time, single-qubit gate count, and depth will be dumped in a csv file in the results folder. The output distribution of the experiments will be printed in the corresponding jupyter notebooks.
- **Experiments:** We use functions from Qiskit to calculate the number of gates and transpilation time of our circuit. We calculate the success rate based on the output distribution from the IBMQ backends.
- **How much disk space required (approximately)?:** 2GB
- **How much time is needed to prepare workflow (approximately)?:** A couple of minutes.
- **How much time is needed to complete experiments (approximately)?:** 18 hours in total.
- **Publicly available?:** Yes.
- **Code licenses (if publicly available)?:** Apache-2.0 License
- **Archived (provide DOI)?:** 10.5281/zenodo.4281275

C. Description

1) How Delivered: Our source code, benchmarks, and jupyter notebooks for experiments are available on Github: https://github.com/1ucian0/rpo.git

2) Hardware Dependencies: In our paper, we run our experiments on 15-qubit quantum computer ibmq_16_melbourne, a 20-qubit quantum computer ibmq_almaden, and a 53-qubit quantum computer ibmq_rochester. Since some of the quantum computers are not publicly available, in order to reproduce the results, we use the fakebackends from Qiskit to use the actual device configurations, such as coupling maps.

3) Software Dependencies: Python version $\geq 3.5, < 3.9$, Qiskit 0.18.0, Jupyter notebook, matplotlib 3.3, z3-solver, tabulate.

Qiskit requires Ubuntu 16.04 or later, MacOS 10.12.6 or later, or Windows 7 or later.

4) Data Sets: Quantum computing benchmarks mentioned in our paper.

D. Installation

We recommend installing the software in an Anaconda environment with Python version 3.7. After downloading Anaconda, create an environment:

```
$ conda create -n my_env python=3.7
```

Then, activate the environment:

- For Linux or MacOS: `$ source activate my_env`
- For Windows: `$ activate my_env`

You can clone our source code and benchmarks from GitHub:

```
$ git clone https://github.com/1ucian0/rpo.git
```

After cloning the GitHub repository, to install the required software:

```
$ pip install -r requirements.txt
```

For questions regarding Qiskit installation, please refer to: https://qiskit.org/documentation/install.html

After installation, run the unittests to verify the installation:

```
$ python -m unittest discover -v tests
```

E. Experiment Workflow

The experiment results on transpilations (Table II, III, IV, V) can be verified by running the `run_benchmark.py` file with corresponding arguments. For example, the following command is for running the Quantum Phase Estimation (QPE) benchmark on ibmq_16_melbourne backend:

```
$ python run_benchmark.py benchmark/
    qpe_FakeMelbourne.yaml
```

The results will be dumped in a `csv` file `results/qpe_FakeMelbourne.csv` in this case. In general, `python run_benchmark.py benchmark/something.yaml` dumps its result in `results/something.csv`.

The CSV files contain all the raw data of multiple runs. Execute the `table.py` file to generate the table containing medians of multiple runs:

```
$ python table.py  results/qpe_FakeMelbourne.csv
```

To verify the experiments on the real quantum computers (Figure.11), run the corresponding Jupyter notebooks: `QPE_almaden/melbourne/rochester`. The results will be printed in the Jupyter notebook output.

F. Evaluation and Expected Result

The results are dumped after running the corresponding Python script and Jupyter notebook. The Jupyter notebooks contain prior experimental results reported in our paper. The expected results are listed in our paper (Table II,III,IV,V, Figure.11).

G. Experiment Customization

The YAML files and Jupyter notebooks are all customizable to run different benchmarks with different configurations. The parameters of the YAML files are described in `README.md` file.

H. Notes

Some of the experiments ,e.g. QPE_almaden.ipynb, require hardware access to certain IBMQ quantum computers. The user can use publically available machines to verify the result.

The transpilation time for Grover's algorithm is particularly long, to validate the results, we changed the number of experiments to five times.

I. Methodology

Submission, reviewing and badging methodology:

- http://cTuning.org/ae/submission-20190109.html
- http://cTuning.org/ae/reviewing-20190109.html
- https://www.acm.org/publications/policies/artifact-review-badging

References

[1] Cirq: A python framework for creating, editing, and invoking noisy intermediate scale quantum (nisq) circuits. [Online]. Available: https://github.com/quantumlib/Cirq

[2] Qiskit hoare opt pass. [Online]. Available: https://github.com/Qiskit/qiskit-terra/blob/master/qiskit/transpiler/passes/optimization/hoare_opt.py

[3] H. Abraham, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, G. Alexandrowics, E. Arbel, A. Asfaw, C. Azaustre, AzizNgoueya, P. Barkoutsos, G. Barron, L. Bello, Y. Ben-Haim, D. Bevenius, L. S. Bishop, S. Bosch, S. Bravyi, D. Bucher, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Chen, C.-F. Chen, R. Chen, J. M. Chow, C. Claus, C. Clauss, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, S. Dague, T. E. Dandachi, M. Dartiailh, DavideFrr, A. R. Davila, D. Ding, J. Doi, E. Drechsler, Drew, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, P. Eendebak, D. Egger, M. Everitt, P. M. Fernández, A. H. Ferrera, A. Frisch, A. Fuhrer, M. GEORGE, J. Gacon, Gadi, B. G. Gago, J. M. Gambetta, A. Gammanpila, L. Garcia, S. Garion, J. Gomez-Mosquera, S. de la Puente González, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, I. Haide, I. Hamamura, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, H. Imai, T. Imamichi, K. Ishizaki, R. Iten, T. Itoko, A. Javadi-Abhari, Jessica, K. Johns, T. Kachmann, N. Kanazawa, Kang-Bae, A. Karazeev, P. Kassebaum, S. King, Knabberjoe, A. Kovyrshin, V. Krishnan, K. Krsulich, G. Kus, R. LaRose, R. Lambert, J. Latone, S. Lawrence, D. Liu, P. Liu, Y. Maeng, A. Malyshev, J. Marecek, M. Marques, D. Mathews, A. Matsuo, D. T. McClure, C. McGarry, D. McKay, D. McPherson, S. Meesala, M. Mevissen, A. Mezzacapo, R. Midha, Z. Minev, A. Mitchell, N. Moll, M. D. Mooring, R. Morales, N. Moran, P. Murali, J. Müggenburg, D. Nadlinger, G. Nannicini, P. Nation, Y. Naveh, P. Neuweiler, P. Niroula, H. Norlen, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, S. Oud, D. Padilha, H. Paik, S. Perriello, A. Phan, M. Pistoia, A. Pozas-iKerstjens, V. Prutyanov, D. Puzzuoli, J. Pérez, Quintiii, R. Raymond, R. M.-C. Redondo, M. Reuter, J. Rice, D. M. Rodríguez, M. Rossmannek, M. Ryu, T. SAPV, SamFerracin, M. Sandberg, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, J. Schwarm, I. F. Sertage, K. Setia, N. Shammah, Y. Shi, A. Silva, A. Simonetto, N. Singstock, Y. Siraichi, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, SooluThomas, D. Steenken, M. Stypulkoski, J. Suen, H. Takahashi, I. Tavernelli, C. Taylor, P. Taylour, S. Thomas, M. Tillet, M. Tod, E. de la Torre, K. Trabing, M. Treinish, TrishaPe, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. C. Vazquez, D. Vogt-Lee, C. Vuillot, J. Weaver, R. Wieczorek, J. A. Wildstrom, R. Wille, E. Winston, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, D. Yeralin, R. Young, J. Yu, C. Zachow, L. Zdanski, C. Zoufal, Zoufalc, azulehner, bcamorrison, brandhsn, chlorophyll zz, dan1pal, dime10, drholmie, elfrocampeador, faisaldebouni, fanizzamarco, gruu, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, ordmoj, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, welien, willhbang, yang.luh, yelojakit, and yotamvaknin ibm, "Qiskit 0.18.3," Apr. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.3765847

[4] M. Amy, "Towards large-scale functional verification of universal quantum circuits," *arXiv preprint arXiv:1805.06908*, 2018.

[5] M. Amy, "Formal methods in quantum circuit design," 2019.

[6] L. Bello and J. Liu, "Rpo 1.0," Nov. 2020. [Online]. Available: https://doi.org/10.5281/zenodo.4281275

[7] C. H. Bennett, "Logical reversibility of computation," *IBM journal of Research and Development*, vol. 17, no. 6, pp. 525–532, 1973.

[8] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, "Quantum machine learning," *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.

[9] K. Blum, *Density matrix theory and applications*. Springer Science & Business Media, 2012, vol. 64.

[10] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, no. 3, p. 032328, 2019.

[11] L. De Moura and N. Bjørner, "Z3: An efficient smt solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2008, pp. 337–340.

[12] J. Du, M. Shi, X. Zhou, Y. Fan, B. Ye, R. Han, and J. Wu, "Implementation of a quantum algorithm to solve the bernstein-vazirani parity problem without entanglement on an ensemble quantum computer," *Physical Review A*, vol. 64, no. 4, p. 042306, 2001.

[13] C. Figgatt, D. Maslov, K. Landsman, N. M. Linke, S. Debnath, and C. Monroe, "Complete 3-qubit grover search on a programmable quantum computer," *Nature communications*, vol. 8, no. 1, pp. 1–9, 2017.

[14] J. C. Garcia-Escartin and P. Chamorro-Posada, "Equivalent quantum circuits," *arXiv preprint arXiv:1110.2998*, 2011.

[15] M. R. Garey and D. S. Johnson, *Computers and intractability*. freeman San Francisco, 1979, vol. 174.

[16] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2012, vol. 3.

[17] L. Gurvits, "Classical deterministic complexity of edmonds' problem and quantum entanglement," in *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, 2003, pp. 10–19.

[18] K.-H. Han and J.-H. Kim, "Quantum-inspired evolutionary algorithm for a class of combinatorial optimization," *IEEE transactions on evolutionary computation*, vol. 6, no. 6, pp. 580–593, 2002.

[19] T. Häner, T. Hoefler, and M. Troyer, "Using hoare logic for quantum circuit optimization," *arXiv preprint arXiv:1810.00375*, 2018.

[20] J. Hsu, "Ces 2018: Intel's 49-qubit chip shoots for quantum supremacy," Sep 2019. [Online]. Available: https://spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy

[21] J. Kelly, "Preview of bristlecone, google's new quantumprocessor," Sep 2019. [Online]. Available: https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html

[22] D. Kenigsberg, T. Mor, and G. Ratsaby, "Quantum advantage without entanglement." *Quantum Information & Computation*, vol. 6, no. 7, pp. 606–615, 2006.

[23] N. Khaneja, R. Brockett, and S. J. Glaser, "Time optimal control in spin systems," *Physical Review A*, vol. 63, no. 3, p. 032308, 2001.

[24] B. Kraus and J. Cirac, "Optimal creation of entanglement using a two-qubit gate," *Physical Review A*, vol. 63, no. 6, p. 062309, 2001.

[25] F. Lardinois, "Ibm will soon launch a 53-qubit quantum computer," Sep 2019. [Online]. Available: https://techcrunch.com/2019/09/18/ibm-will-soon-launch-a-53-qubit-quantum-computer/

[26] Y. Li, J. Hu, X.-M. Zhang, Z. Song, and M.-H. Yung, "Variational quantum simulation for quantum chemistry," *Advanced Theory and Simulations*, vol. 2, no. 4, p. 1800182, 2019.

[27] V. Mavroeidis, K. Vishi, M. D. Zych, and A. Jøsang, "The impact of quantum computing on present cryptography," *arXiv preprint arXiv:1804.00200*, 2018.

[28] W. M. McKeeman, "Peephole optimization," *Communications of the ACM*, vol. 8, no. 7, pp. 443–444, 1965.

[29] M. Mottonen and J. Vartiainen, "Decompositions of general quantum gates. ch. 7 in trends in quantum computing research," 2006.

[30] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1015–1029.

[31] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," *arXiv preprint arXiv:2001.02826*, 2020.

[32] Y. Nam, N. J. Ross, Y. Su, A. M. Childs, and D. Maslov, "Automated optimization of large quantum circuits with continuous parameters," *npj Quantum Information*, vol. 4, no. 1, pp. 1–12, 2018.

[33] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[34] T. Patel and D. Tiwari, "Disq: a novel quantum output state classification method on ibm quantum computers using openpulse," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[35] M. Plesch and Č. Brukner, "Quantum-state preparation with universal gate decompositions," *Physical Review A*, vol. 83, no. 3, p. 032302, 2011.

[36] A. K. Prasad, V. V. Shende, I. L. Markov, J. P. Hayes, and K. N. Patel, "Data structures and algorithms for simplifying reversible circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 2, no. 4, pp. 277–293, 2006.

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

[38] V. V. Shende and I. L. Markov, "On the cnot-cost of toffoli gates," *arXiv preprint arXiv:0803.2316*, 2008.

[39] S. Sivarajah, S. Dilkes, A. Cowtan, W. S. A. Edgington, and R. Duncan, "t|$ket\rangle$: A retargetable compiler for nisq devices," *arXiv preprint arXiv:2003.10611*, 2020.

[40] G. Song and A. Klappenecker, "Optimal realizations of controlled unitary gates," *arXiv preprint quant-ph/0207157*, 2002.

[41] S. S. Tannu and M. Qureshi, "Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 253–265.

[42] S. S. Tannu and M. K. Qureshi, "Mitigating measurement errors in quantum computers by exploiting state-dependent bias," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 279–290.

[43] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: a case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 987–999.

[44] I. team, "Ibm q 16 melbourne backend specification v2.0.6," 2019, retrieved from https://quantum-computing.ibm.com.

[45] V. Vedral, A. Barenco, and A. Ekert, "Quantum networks for elementary arithmetic operations," *Physical Review A*, vol. 54, no. 1, p. 147, 1996.

[46] D. Venturelli, M. Do, B. O'Gorman, J. Frank, E. Rieffel, K. E. Booth, T. Nguyen, P. Narayan, and S. Nanda, "Quantum circuit compilation: An emerging application for automated reasoning," 2019.

[47] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Checking equivalence of quantum circuits and states," in *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2007, pp. 69–74.

[48] G. Vidal and C. M. Dawson, "Universal quantum circuit for two-qubit transformations with three controlled-not gates," *Physical Review A*, vol. 69, no. 1, p. 010301, 2004.