A Pulse Generation Framework with Augmented Program-aware Basis Gates and Criticality Analysis

Yanhao Chen, Yuwei Jin, Fei Hua, Ari Hayes, Ang Li[y], Yunong Shi[z], Eddy Z. Zhang
Rutgers University, [y]Pacific Northwest National Laboratory, [z]Amazon Braket

2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA) | 978-1-6654-7652-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/HPCA56546.2023.10070990

Abstract—Near-term intermediate-scale quantum (NISQ) devices are subject to considerable noise and short coherence time. Consequently, it is critical to minimize circuit execution latency and improve fidelity. Traditionally, each basis gate of a transpiled circuit is decoded into a fixed episode of the device control pulses. Recent studies investigate the merged pulse generation method for customized gates through quantum optimal control (QOC). In this work, we propose PAQOC, a novel QOC framework that can (i) exploit an augmented program-aware (APA) basis gate set for the tradeoff between compilation time and circuit performance, (ii) prune the search space based on a criticality-centric analytical model and experiment observations we learned from 150 benchmarks. Evaluations using seventeen applications show that PAQOC can achieve an average 54% reduction of the circuit latency, on average 43% reduction in compilation overhead, and a 1.27 improvement in fidelity. PAQOC is available on GitHub[1].

I. Introduction

Quantum Computing has garnered considerable attention due to its potential for enormous computation acceleration. Quantum algorithms are promising techniques for solving intractable computational problems, such as cryptography [44], machine learning [7], database search [23], and others [4], [27], [40]. Google, IBM, Intel, and Rigetti have built a variety of quantum computers with qubit counts ranging from 5 to 127 [8], [18], [24], [28].

A quantum program is expressed using a gate-level intermediate representation (IR) where a logical circuit consists of basis gates. To execute it on quantum hardware, it first needs to be translated into physical circuits and then be compiled into machine-control pulses as shown in Fig. 1. Machine-control pulses for generating a single arbitrary gate (unitary) using quantum optimal control (QOC) [19] have been extensively studied [2], [15], [29], [31], [38], [47]. In this paper, we focus on circuit-level pulse generation. The approaches for circuit-level pulse generation can be categorized into two types.

The first type draws on the fact that every circuit is built upon a small set of basis gates [37]. It generates machine-control pulses for each basis gate and stores them in a table. Then the compiler looks up the table for pulse-generation of the entire circuit [2], [3], [34]. We refer to this approach as the fixed-gate approach as it only generates pulses for a fixed set of universal basis gates. The advantage of this approach is its low compilation overhead. The disadvantage, however, is that it usually results in suboptimal circuit performance [9]. This approach is shown in Fig. 1-left.

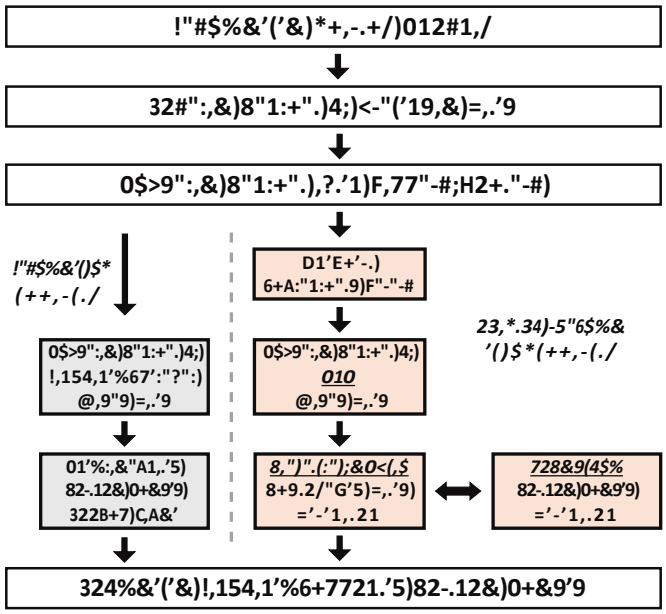


Fig. 1. Two different approaches to compile quantum programs to low-level control pulses. The left one synthesizes high-level quantum circuits using only fixed hardware-specific basis gates [43]. Our approach on the right side explores APA-basis gates in circuits and generates criticality-aware customized gates.

The second type does not view a circuit as a composition of universal basis gates. Instead, it views a circuit as a composition of customized gates. Each customized gate is a group (or a sequence) of consecutive basis gates, and pulses are generated for each customized gate. The benefit of this approach is that it can significantly reduce the latency. The latency of the pulses generated by QOC does not necessarily increase with the number of gates in the group as that in the fixed-gate approach [43]. For instance, generating control pulses for a sequence of two gates is better than generating pulses for each and stitching them, as shown in Fig. 2. The latency reduction benefit has been reported in previous studies [9], [20], [21], [43]. The disadvantage of this approach, however, is that it has a high compilation overhead [9], [20], [31]. We refer to this approach as the customized-gate approach.

Today's quantum devices continue to be plagued by short

[1]https://github.com/ruadapt/paqoc

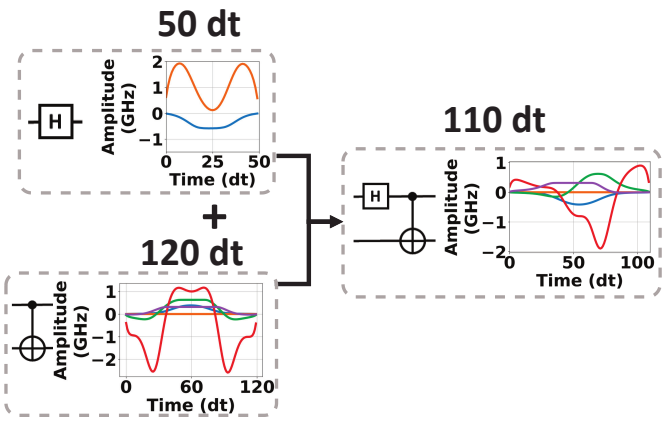


Fig. 2. Pulse generation for a group of two gates (consolidated into a single unitary) is better than that separately for each gate: Hadamard (H) and CX (CNOT) gates. The latency for the joint unitary is 110 dt (dt is time unit), while is 170 dt for the separate case.

coherence time. That means quantum machines can only support a short execution of programs before failing. Latency reduction can significantly mitigate the decoherence problem [1], [9], [20], [30], [37], [43]. In this paper, we focus on the customized-gate approach for latency reduction purposes, with respect to a given circuit fidelity budget. Our version of the customized-gate approach is outlined in Fig. 1-right.

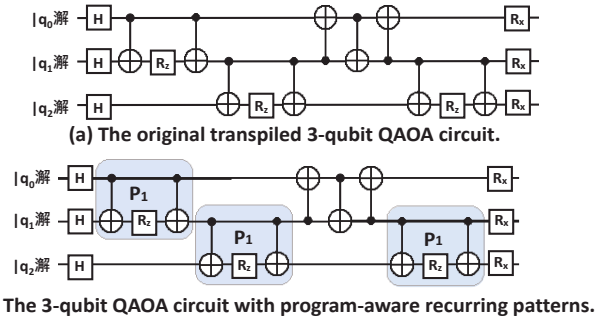
In previous studies focusing on the customized gate approach, Cheng et al. [9] exploit the similarity between customized gates and optimize the compilation time by using a pre-compiled gate as an initial guess to a new similar gate. Gokhale et al. [20] tackle variational algorithms by performing partial online and partial offline pulse generation based on hyperparameter optimization. Shi et al. [43] apply commutativity-aware instruction aggregation to further improve pulse generation efficiency.

Compared with prior studies, our main contribution is two-fold. (1) We propose an augmented program-aware basis (APA-basis) gate approach. We discover that there are often recurring patterns in a circuit. A recurring pattern is a subcircuit that frequently appears in a circuit (or different circuits). An example is shown in Fig. 3. We can replace each frequent subcircuit as an APA-basis gate to simplify the circuit. Then we can perform gate grouping and pulse generation. It reduces the compilation overhead significantly and, in the meantime, still yields the same or better pulses compared with the state-of-the-art [9]. Gokale et al. [21] also use augmented basis gates, but in a hardware-aware way, while ours does it in a program-aware way.

(2) We propose a criticality-based search model for achieving desired customized gate groups. During the search, we need to try grouping different sequences of consecutive gates before determining the best combination. The search space is large. Considering a dependence graph of n gates that is linear, if we allow any number of consecutive gates to be grouped, there are 2^n number of ways to group the gates. The reasoning is that we can break down the sequence at any point in between two consecutive gates, and there are n locations (omitting the last one), and we have a binary choice: splitting it or not.

Hence the search space is exponential if enumerating all gate grouping choices. We develop an analytical model based on the observation of over 150 benchmarks and based on the criticality of gates in the circuit. It only allows a monotonic decrease of the circuit latency at each gate grouping step. A very simple example is shown in Fig. 4.

We also try to reduce the compilation overhead of pulse generation for quantum circuits, but in a different way compared with prior approaches. We allow a less restricted search space. AccQoc [9] exploits the similarity between different gate sequences to reduce the compilation overhead. However, due to the similarity comparison, it also has to restrict the maximum depth of a sequence of gates to a fixed number, usually 3 to 5. We do not restrict the depth of the sequence of gates to be merged. Our overhead is also small due to the analytical model we used in Section III. By doing this, a relatively less restricted search space gives us a better chance to find improved circuit pulses with respect to a given fidelity requirement.



(a) The original transpiled 3-qubit QAOA circuit.

(b) The 3-qubit QAOA circuit with program-aware recurring patterns.

Fig. 3. The QAOA-maxcut circuit after simplification. Recurring sub-circuits are identified as APA-basis gates.

Besides these two main contributions, we also have multiple other contributions: (3) We designed a graph mining model for automatically searching frequent subcircuits that can be turned into APA-basis gates, (4) We provide a tuning knob for the size of APA-basis gate, which allows users to exploit the recurring pattern in a circuit to some extent, since if too many gates are grouped in advance, then the search sub-space is significantly reduced in the criticality-based search, and (5) We can decouple the compilation into the online component and the offline component, where the offline component can detect APA-basis gate, even for parameterized circuit. The online component can generate pulses only for the grouped gates on the fly. Gokale et al. [20], [21] also divide a circuit into blocks, but based on which blocks have parameters, not on whether a sub-circuit block frequently appears.

Overall, we propose a compilation framework that generates pulses of an entire circuit. We name it as PAQOC, a Program-Aware QOC-based optimal control pulse generation framework, as it uses program-aware augmented basis gate and

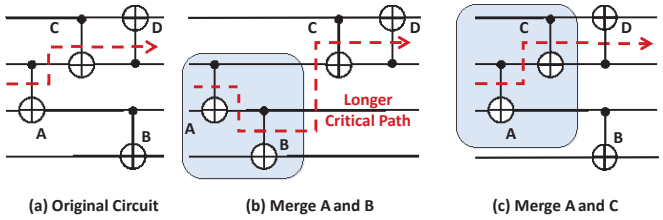


Fig. 4. Different ways to form a 3-qubit customized gate: (a) the original circuit, (b) merging A and B may elongate the critical path as it creates false dependence between B and C, and (c) merging A and C does not. The scheme in (c) is better than that in (b) as it does not increase the critical path. Here we assume the merged gate latency is smaller than the sum of the individual gates A and B. Our criticality analysis model is more complex than what is shown in this example, as shown in Section V-A.

criticality analysis information, as opposed to using hardware-aware basis gate and other information. PAQOC can achieve up to 2.17 speedup in circuit latency, with an average 1.75 compilation speedup and with 1.27X improvement for fidelity compared to the state-of-the-art.

The rest of the paper is organized as follows: Section II and III give a background of our work and preliminary observations. Section IV gives an overview of PAQOC. Section V describes PAQOC implementation about its key components. Section VI presents our comprehensive experiment evaluation. We relate previous works in Section VII and conclude our work in Section VIII.

II. Background

In this section, we provide a brief introduction to gate-level programming, quantum optimal control (QOC), and our preliminary observations about customized gates.

A. Gate Level Programming

In quantum computing, an important IR is the gate-level IR. A quantum algorithm is built upon a universal set of basis gates. The basis gates in the universal set can be supported by different hardware vendors, and the low-level details are hidden from the programmers. A universal gate set [5] typically consists of one-qubit and two-qubit gates (or three-qubit gates). For instance, X, SX, CX, ID, and RZ gates are used for IBM-Q devices [3], [34] and RX, RZ, and CZ are used for Rigetti devices [36], [46]. A programmer can define a customized gate based on universal basis gates in QASM [11], [12]. However, during transpilation, a quantum circuit is always compiled into machine-dependent basis gates.

A quantum gate performs a unitary transformation on a (set of) qubit(s). A quantum circuit can be represented as a single unitary matrix by manipulating the unitaries of the gates in the circuit. The unitary matrix of a quantum circuit can be obtained in the following ways: for gates operating on disjoint qubits, their unitary can be obtained by tensor product, and for gates operating on the same qubits, their unitary can be obtained by matrix product [37].

B. Pulse Level Model

The quantum states of quantum hardware are manipulated by external physical operations, which are system-specific control fields with a unique and time-dependent quantity called the Hamiltonian matrix [39], [48], [51]. The Hamiltonian determines the evolution path of the quantum states. A gate is translated to a state transfer, which can be interpreted as a time-evolving process of changing the control field values at individual time steps. This translation from a gate to the time-evolving control of machine-level pulses is called pulse generation.

The combined Hamiltonian for the system can be represented as follows:

$$H(t) = H_0 + \sum_k {}_k(t) H_k \quad (1)$$

where H_0 is the drift Hamiltonian and H_k are the time-dependent control Hamiltonians [2]. The ${}_k(t)$ are time-varying amplitude functions for the specific control fields.

The technique for determining the control fields of an arbitrary state transfer is called quantum optimal control (QOC) [14], [19], [48], which relies on a gradient-based approach to optimize the control fields and improve the fidelity and latency of the generated pulses. We use the tool GRadient Ascent Pulse Engineering (GRAPE) [31] to generate pulses for a single unitary. An example of generated pulses by GRAPE is shown in Fig. 2.

Although there are extensive studies on how to generate pulses for a single unitary, only a few studies [9], [20] focus on pulse generation for a whole circuit. On one hand, a circuit can be represented as a single unitary. However, the overhead grows exponentially with the number of qubits. For example, generating pulses for a ten-qubit unitary takes over a day even using GPU-accelerated QOC [31]. Hence, a circuit is usually further decomposed into multiple unitaries, each of which can be handled by GRAPE [20], [31]. On the other hand, the generated control pulses must be optimized. In this paper, we focus on circuit-level pulse generation.

C. Fidelity

While generating pulses, we need to ensure the fidelity of the compiled circuit. The QOC tools such as GRAPE requires as input a given Unitary U and an error term such that $\| U - H(t) \| $.

The latency of generated pulses is correlated with . The higher it is, the lower the latency is. This is, however, the metric for one grouped gate instead of the entire circuit. To ensure that we have a metric for the entire circuit, we use the metric estimated success rate (ESP) such that it is a product of the success rate of each customized gate:

$$ESP = \prod_{i=1:::n} (1 - \| U_i - H_i(t) \|) \quad (2)$$

n is the number of gates after gate grouping. We ensure our circuit produces at least the same ESP as the baseline approach for comparison. In a lot of cases, our ESP is even better. Under this constraint, we further reduce the circuit latency.

We also utilize Qutip [26] pulse simulation for the entire circuit if possible. Our circuit fidelity is also always better or the same based on the simulation results. We can only perform pulse simulation for a few benchmarks since the simulation takes a significant amount of time.

We do not run pulse experiments on real machines. Although the Hamiltonian form [50] can take into account the error terms, it is prohibitive to perform detailed calibration for real machines to collect exact error terms [35]. However, the technique developed in our paper still has its value. Once the error terms are determined, we only have to update Equation (1) and apply the same method. Our work along the same line of research [9], [21], [43] is useful for real machines when the calibration problem is tackled.

III. Key Insights and Observations

Before going into the implementation details, we talk about our key insights and observations that lead to the design of the PAQOC framework. The first one is the APA-basis gate identification, and the second one is the correlation between gate latency and gate sizes.

A. APA-basis Gate Identification

We discovered that there are often recurring subcircuits that frequently in quantum circuits. We can extract these subcircuits and turn them into augmented program-aware (APA) basis gates to simplify the original circuit.

Programmers can annotate the frequent sub-circuits, but it adds a programming burden. It may not be trivial since the physical circuits are different from the logical circuit that programmers wrote. The physical circuit usually includes circuit transformation to adapt to a given topology.

To automatically detect frequent sub-circuits, we first construct a labeled directed graph that encloses information from the original physical circuit. We then exploit the subgraph mining model [16] to find common sub-circuits.

In our directed graph, each node represents a quantum gate. Each edge represents that the two quantum gates share one qubit. There is a direction based on the dependence relationship of two gates. Each node is labeled with the name of the quantum operator. For rotation operations such as the phase shift gate R_z, the rotation degrees are included in the gate label symbolically to handle parameterized circuits. Each edge is labeled to indicate whether the control qubit or target qubit is shared between the two gates if at least one of the two gates is the two-qubit gate. For example, in Fig. 5(c), the edge label between the leading CX and R_z gates is "2-1" indicating that the sharing qubit is the CX gate's target qubit and the R_z gate's first qubit.

Two subcircuits that are identical imply the number of nodes, the node connectivity, and the labeling information in two sub-graphs is all equivalent. The example in Fig. 5 shows how to convert a physical quantum circuit to a labeled graph for graph mining and how to disambiguate similar but not identical sub-circuits using our proposed edge labels.

In practice, different frequent subcircuits may be overlapping. For example, the two kinds of frequent subcircuits in Fig. 5(c) are overlapped. We consider which frequent subcircuits to use based on its coverage of the circuit, i.e., how many original gates in total are covered by them.

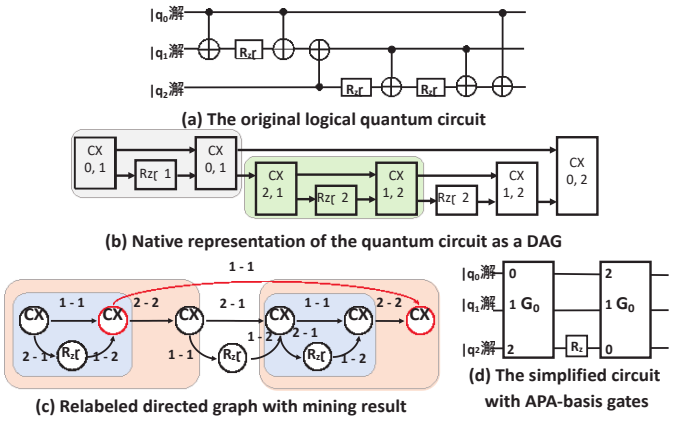


Fig. 5. Converting a physical quantum circuit to a labeled directed graph. Each node is labeled with the name of the quantum operation and rotation angle if needed. Each edge implies how two quantum gates are connected. The two circuit blocks in (b) are not identical despite they look similar. Our method can correctly identify it due to the control-target labeling of edges in our method.

B. Correlations between Gate Latency and Gate Size

We also discover the correlation between merged gate latency and gate sizes which can be useful for pruning the search space using an analytical model.

We run experiments on 150 real-world benchmarks from RevLib and ScaffCC [3], [22], [25], [49]. We extract subcircuits from them. Each subcircuit is a maximum consecutive sequence of 1-qubit, 2-qubit, or 3-qubit gates sharing the same qubit(s). We compare two values: (1) Latency of the generated pulse for the subcircuit as a group, and (2) The summation of the latencies for each gate in the group. We show them as Y-axis and X-axis values in Fig. 6.

As can be seen, the merged gate latency is always smaller than the summation of individual gate latencies. The dashed line shows when X-axis and Y-axis values are equivalent. All points fall below the dashed line. This means that if we evaluate the local effect of universal basis gate merging, it is always beneficial. In hindsight, it echoes the claims in previous studies [9], [43] for supporting consecutive gate grouping, but it is the first time being validated by extensive experiments.

We make the following observations and assumptions based on the information of these 150 benchmarks. We let $N_Q(X)$ represent the number of qubits in the gate sequence X, $L(M)$ represent the latency of the merged gate sequence M.

Observation 1: For two quantum gates X and Y that $N_Q(X) = N_Q(Y)$ and their corresponding merged customized gate $\overline{XY}$, we usually have $L(\overline{XY}) \leq L(X) + L(Y)$.

We also observe that the latency for the majority of the merge gates with a bigger qubit count, it is larger than that with a smaller qubit count. For instance, according to Fig. 6,

most two-qubit merged gate sequences have a latency larger than those of single-qubit gates. This holds for the comparison between two- and three-qubit gates.

Observation 2: For two gate sequences X and Y, we observe that $L(X) \ L(Y)$ if $jN_Q(X)j \ jN_Q(Y)j$, for most cases.

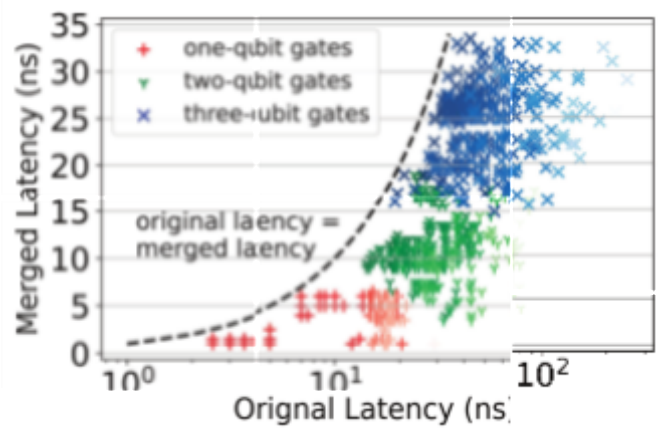


Fig. 6. Comparing the original and the merged latency of subcircuits from real benchmarks with up-to three qubits. The more transparent of the marker, the more gates in the subcircuit. The black dotted curve represents X and Y values being the same.

We leverage observations 1 and 2 during our compilation optimization step in Section V-A. For a lot of cases, we can estimate whether merging a gate sequence is beneficial or not, without having to actually generating pulses first. We use criticality information together with observations 1 and 2. Therefore we can reduce compilation overhead significantly without sacrificing the generated pulse quality.

IV. PAQOC Overview

The core of PAQOC is an iterative combinatorial search engine applied to a (pre-)processed circuit. The frequent subcircuits mining component of PAQOC allows us to extract frequent subcircuits and then convert them into APA-basis gates. The criticality-aware customized gates generator works hand in hand with the control pulse generator to iteratively update the decomposition of the circuit until the circuit-level pulses cannot be further improved (or with respect to a target goal). Fig. 7 provides a high-level overview of PAQOC.

PAQOC takes as input the following: a physical quantum circuit, Hamiltonian-level pulse information of a given hardware, and the maximal number of qubits allowed in a customized gate. It outputs a new circuit consisting of customized gates with corresponding gate grouping. It also outputs the generated pulses.

a) Frequent Subcircuits Miner: PAQOC extracts frequent subcircuits from the input physical circuit. The PAQOC frequent subcircuits miner is used for finding such frequent subcircuits. With the identified frequent subcircuits, PAQOC replaces each with an APA-basis gate, thus reducing the number of gates compared with the original circuit.

b) Criticality-Aware Customized Gates Generator: After the simplified circuit is generated, it generates a set of candidate customized gate grouping choices and then ranks them. The ranking is performed by our analytical model in Section V. The top-ranked gate sequences are merged at each iteration. PAQOC ensures that each step of merging only decreases the overall latency.

c) Control Pulses Generator: This component works together with the Customized Gates Generator. During the ranking step, we need to check whether a candidate customized gate is beneficial by trying to generate pulses to get the actual latency. We only need to generate actual pulses for some cases, as aforementioned. For other scenarios, we leverage the observations in Section III.

V. PAQOC Implementation

We now describe the implementation of PAQOC. Since we have already described how to identify the frequent subcircuits. We focus on the customized gates generator and the QOC-based control pulses generator.

A. Customized Gates Generator

In this component, PAQOC constructs customized gates as units for pulse generation. It takes an input circuit as a physical circuit. It generates customized gates as groups of consecutive basis gates. At each iteration, PAQOC chooses and ranks the gate sequences as merging candidates. It assigns each candidate a score and chooses the top candidate(s) with the highest score(s). It then merges the gates in the selected candidate sets and updates the circuit. These steps repeat until no more gate-merging can improve the latency of the entire circuit.

In the following of this section, we discuss our criticality-aware search space prune strategies and ranking heuristics. In Algorithm 1, we show the pseudo-code of the entire process.

1) Search Space Prune Strategies: Any sequence of consecutive gates in the quantum circuit can form a customized gate. It leads to a large search space for determining which customized gates to construct. It is prohibitive to evaluate each of these candidate groupings. In PAQOC, we introduce criticality-aware pruning strategies that effectively reduce the search space while not hurting the circuit-level pulse efficiency.

First, we perform a hierarchical grouping search. At each level of the search, we start with considering the candidates of the two-gate grouping. The multi-level search could enable the merging of multiple gates. An example of only allowing two-gate grouping is shown in Fig. 8 (b).

The search space can be pruned by pre-processing based on Observation 1. The gates that share the same qubit(s), if merged, are typically beneficial. For instance, the gates A, B, and C in Fig. 8 can be pre-processed to $(\overline{ABC})$ as a merged gate. After pre-processing, the search space is reduced.

The search space can be further pruned by performing a criticality-aware analysis. It is important that we only perform gate-merging that leads to a shorter critical path. We categorize the gates into two types: (1) The gates on the critical path and

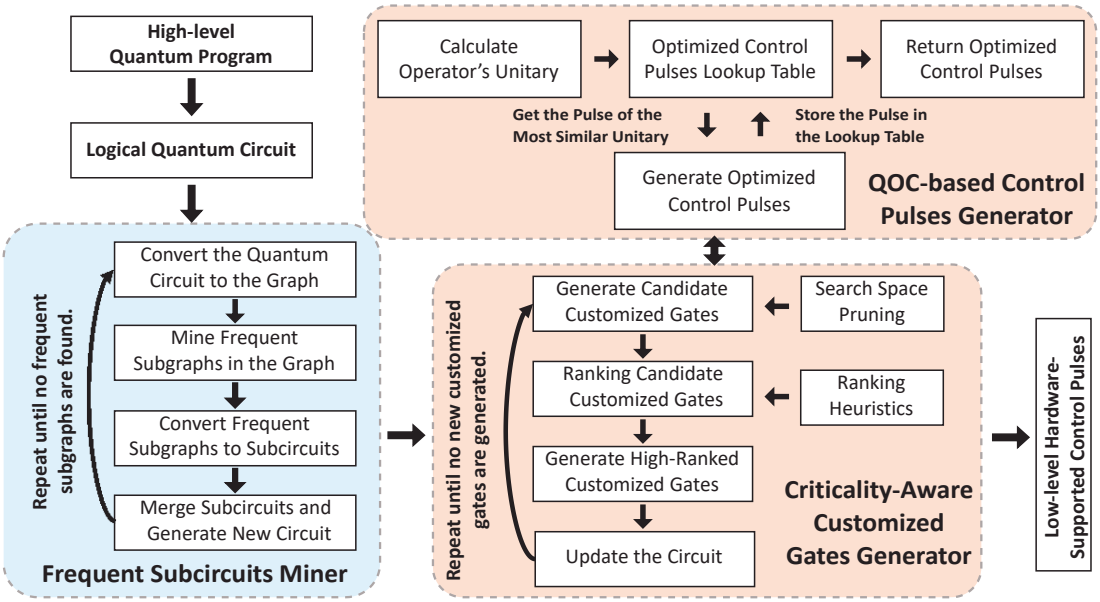


Fig. 7. Overview of our PAQOC pulse generation framework.

Algorithm 1 Generate customized gates for a quantum circuit

```
Require: Original Quantum Circuit(QC)
Require: Control Pulses Generator(PG)
Require: Maximum Allowed Subcircuit Size(maxN)
Require: Number of Generated Customized Gates at Each Iteration(topK)
Ensure: A latency optimized quantum circuit
while True do
    C = preprocess_prune(QC.get_two_gate_candidates(maxN))
    if C is empty then return
    all_scores = []
    for gates in C do
        gate = get_customized_gates(QC, gates)
        if N_Q(gate) > maxN then
            continue
        score = get_ranking(gate, PG)
        all_scores.append([score, gate])
    if all_scores is empty then
        break
    sort all_scores in the decreasing order
    for i in range(topK) do
        customized_gate = all_scores[i].second
        if customized_gate is no longer valid then
            continue
        PG.calculate_optimal_control_pulse(customized_gate)
        Update QC, replace gates by customized_gate
```

(2) the gates not on the critical path. There are three scenarios when it comes to merging gates

Case I: Both gates are on the critical path.

Case II: Only one gate of the two is on the critical path.

Case III: Neither gate is on the critical path.

First, for Case III, merging two gates that are not on the critical path does not affect the overall latency of the circuit as a whole. It, at most, reduces the length of the non-critical paths but does not reduce the length of the critical path. In Fig. 9-(d), we show such a case where gate A and B are on the critical path, while C and C' are not. Even though merging C and C' will be faster for only two gates C and C', it is actually

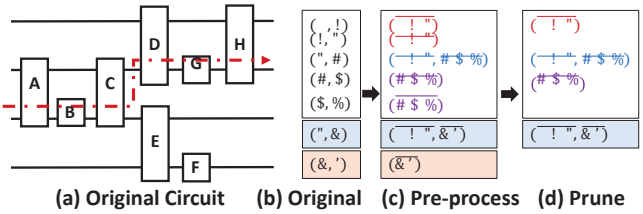


Fig. 8. Candidate space search: (a) Original circuit; (b) All candidates for merging two gates; (c) Pruned candidate set by pre-processing; (d) Further pruned candidate sets via criticality-analysis. The red dashed line shows the critical path before the gate merging.

not beneficial to the entire circuit. It may even adversely affect the circuit due to the created false dependencies. Hence, in our pruning, gate-merging that only involves non-critical gates is not considered. In Fig. 8, candidate (E; F) is pruned since neither E nor F is on the critical path.

Next, after pruning the candidates not involving critical gates, we only have to consider Case I and Case II. We only rank the candidates involving at least one critical gate.

Before describing Case I and Case II, we first describe our notations. We use $\overline{XY}$ to represent the customized gate constructed by merging gate X and gate Y. We let L(X) be the latency of gate X. We let CP(X) represent the longest path from gate X to the end of the circuit.

We discuss case II first. For Case II, one gate is on the critical path, and the other is not. Assuming in this set of two gates, A is the one on the critical path, C is its successor, and C is not on the critical path. C must not depend on any successor of A, otherwise merging A and C will be an invalid choice for ranking. Merging A and C could cause false dependence that may or may not elongate the critical path. To model this,

we assume B is the immediate successor of A on the critical path. An illustration of this case is shown in Fig. 9 (c). Now the circuit latency for comparison is the three:

$L(A) + L(B) + CP(B)$ // the not-merging case
$L(\overline{AC}) + L(B) + CP(B)$ // the merging case with one possible critical path
$L(\overline{AC}) + CP(C)$ // the merging case with another possible critical path

To find the maximum of the last two items, we only need to compare $L(B) + CP(B)$ with $L(C)$. Since A and B are on the critical path, it implies that $L(B) + CP(B) > CP(C)$. Then we compare the first item and the second item to see if merging is beneficial. We need to perform the merging of A and C to get $L(\overline{AC})$ in order to test if it will be beneficial.

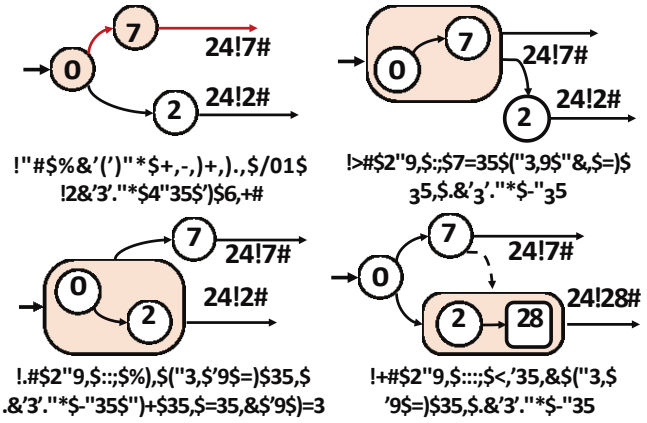


Fig. 9. A simplified example shows how the critical path changes by merging different gates. Each node in the DAG is a quantum gate. The (dotted) edge represents the (potential) dependence between gates.

For Case I where we only merge two gates on the critical path, assuming the two gates are A and B. Assuming A's immediate successor that is not on the critical path is C. The updated critical path of the entire circuit is the maximum of the following (also illustrated in Fig. 9-b):

$L(A) + L(B) + CP(B)$
$L(\overline{AB}) + CP(B)$
$L(\overline{AB}) + L(C) + CP(C)$

Now it is not immediately clear whether the merging is beneficial. The delta compared with the original critical path is either $L(\overline{AB}) \quad L(A) \quad L(B)$ or $L(\overline{AB}) \quad L(A) + L(C) \quad L(B) + CP(C) \quad CP(B)$. As $L(\overline{AB})$ is the common term, to compare these two, we only need the information on the latency of the current updated circuit. Again we can use observation 2, if $\overline{AB}$ uses more qubits than the maximum of A or B, the $L(\overline{AB})$ must be the dominant factor. We approximate $\overline{AB}$ with the average gate latency of that size to check whether merging is beneficial. If $\overline{AB}$ uses the same number of qubits as the maximum of A or B, we use observation 1 to compare the first item and the second item. In either case, we do not have to generate pulses for merging A and B to test if the merging is beneficial.

2) Ranking Heuristic: For each merging option, we give it a score that is the delta between the original whole-circuit latency and the updated circuit latency, assuming the merging takes effect. We choose the top-k (disjoint) options and merge gates correspondingly. The parameter k controls the number of newly generated customized gates at each iteration. With a larger k, more customized gates being generated in one iteration, this may result in a less optimized final latency, as each merging might change the critical path of the entire circuit again.

B. Optimal Control Pulses Generation

To generate pulses for a customized gate, we use quantum optimal control (QOC) similar to that in previous studies [9], [20]. It calculates the minimum duration of the control pulses of a customized gate by binary search. GRAPE [31] is the state-of-the-art tool for quantum optimal control. We use GRAPE to achieve this.

PAQOC uses a lookup table to store previously generated control pulses for different customized gates. For customized gates that frequently appear in the circuit, we only need to generate the optimal control pulse once. Moreover, for the same customized gate with permuted qubits, it will also be detected in PAQOC. For a customized gate that has a similar unitary to that of a previously generated customized gate, we use the previous gate's pulses as the initial guess to GRAPE, as GRAPE requires an initial guess of the pulse sequence, and a closer pulse sequence will reduce the convergence time and speedup pulse generation. This is similar to AccQOC [9].

C. Putting Everything Together: Exploiting the Tradeoff

When using PAQOC, users can specify the maximal size of the APA-basis gates. Moreover, users can specify whether the frequent subcircuits miner component and the customized gates generator component be enabled for exploring the trade-off between the compilation overhead and the circuit latency.

With the use of the frequent subcircuits miner and a specified size of the APA-basis gate, it simplifies the circuit by converting frequent subcircuits into APA-basis gates. This will reduce the search space for the later customized gate grouping thus achieving less compilation overhead.

Note that we can disable the customized gates generator completely to construct a simplified circuit with only APA- (and potentially universal) basis gates. Our APA-basis gate sets are chosen in a way that it will guarantee not to increase the critical path based on our observations in Section III-B.

VI. Evaluation

In this section, we present the evaluation results of our proposed PAQOC framework. We begin with a broad analysis of 17 benchmarks in terms of circuit latency and circuit compilation overhead. Then, we select five representative benchmarks to demonstrate the frequent subcircuits miner's performance and discuss the tradeoff between circuit latency and compilation overhead.

TABLE I
OVERVIEW OF APPLICATION BENCHMARKS

Name	Description	# qubits	1q-gate number	2q-gate number
mod5d2_64	Toffoli network	16	28	25
rd32_270	Bit adder	5	48	36
decod24-v1_41	Binary decoder	5	47	38
4gt10-v1_81	4 greater than 10	5	82	66
cnt3-5_179	Ternary counter	16	90	85
hwb4_49	Hidden weighted bit	5	126	107
ham7_104	Hamming code	16	171	149
majority_239	Majority function	16	345	267
bv	Bernstein Vezirani	21	43	20
adder	Cuccaro Adder	18	160	107
qft	QFT	16	16	120
qaoa	QAOA	10	65	90
supre	Supremacy	25	245	100
simon	SImon's algorithm	6	14	16
qpe	QPE	9	28	33
dnn	Deep neural network	8	192	1008
bb84	Crypto. proto	8	27	0

a) Benchmarks: We select seventeen representative benchmarks from existing quantum programs libraries, including RevLib [49], ScaffCC [25] and Qiskit [3] that are used in previous studies [9], [52], [54]. These benchmarks cover important quantum applications, including quantum bit decoder, Toffoli network synthesis, and quantum fourier transformation. The selected benchmarks have a range of qubit numbers from 5 to 25 and have up to 1200 gates. These input quantum circuits are built upon universal basis gates. We provide the information of these benchmarks in Table I.

b) Baseline: We use the state-of-the-art approach AccQOC [9] as our baseline. AccQOC divides the quantum circuit into small fixed-size subcircuits. In particular, AccQOC allows a maximum of two qubits in each customized gate, and each customized gate must have a fixed depth. To ensure a fair comparison, we use an extended version of AccQOC that supports customized gates with a maximum of three qubits. We use two variants of the extended AccQOC in which the depth of each fixed-size subcircuit is set to three and five. We refer to the two versions as accqoc_n3d3 for three-qubit group size and a depth of five and accqoc_n3d5 for three-qubit group size and a depth of five, respectively.

c) Platform: We perform simulation based on the superconducting architecture with XY interaction. We use a 5x5 grid topology with Sabre [32] qubit routing and mapping heuristic. We apply the same setup that has been used in the baseline [9], and other optimal control pulse generation studies [20], [43]. We assume that the control fields of all two-qubit interactions and single-qubit rotation are identical. The control field limitation of the XY interaction is set to $_{max}$ = 0.02GHz and the single qubit rotation control field is set to 5_{max}.

We limit the maximum qubit number of any APA-basis gate or customized gate, maxN, to 3 in this experiment. In practice, PAQOC can accept any number of maxN. The maximum number of the additional APA-basis gates allowed in the circuit, M, is also taken as an input to PAQOC. M is the size of the set of APA-basis gates that are different from the universal basis gates. By using different values of M, we can explore the tradeoff between the circuit latency and the compilation overhead.

In particular, we provide three versions of our PAQOC framework with a different numbers of APA-basis gates (M) allowed to merge in the circuit:

1) paqoc(M = 0) It assumes the gate count in each frequent subcircuit is one and thus does not add any additional APA-basis gate. This can be considered as a special version of PAQOC with only the customized gates generator being enabled.
2) paqoc(M = inf) sets no limit on M, thus considers all frequent subcircuits found by the frequent subcircuits miner as APA-basis gates (any gate sequence that appears more than twice).
3) paqoc(M = tuned) tunes the value of M, it selects the smallest M makes the APA-basis gates the majority in the circuits [2].

d) Control Pulses Generator Setup: We use GRAPE [31] to generate the optimal control pulses for APA-basis gates and customized gates. In GRAPE, the total time of the control pulses of the quantum gate is specified as a static parameter total_time. GRAPE requires the target unitary matrix of the quantum gate, together with the gate fidelity as the input parameters. We set the fidelity as high as possible such that it ensures that the whole circuit ESP is no worse than that of the baseline. GRAPE allows the user to select different optimization methods for evaluating the gradient, and we choose ADAM as our optimization method for generating the control pulses. We simulate the control pulse under the assumption of using a transmon superconducting architecture as the underlying hardware, which has been used in previous works [20], [43] and our baseline [9].

A. Impact on Overall Latency

In Fig. 10, we show the overall circuit latency reduction of PAQOC compared with two AccQOC methods normalized to the baseline accqoc_n3d3. The latency of paqoc(M=0) achieves the most significant reduction with an average of 54% latency while paqoc(M=inf) achieves a minor improvement with an average of 40% reduction. The improvement of paqoc(M=inf) is smaller than paqoc(M=0) because paqoc(M=inf) uses the APA-basis gates and the usage of APA-basis gates excludes specific ways to group the consecutive gates. However, paqoc(M=inf) reduces the compilation time, as will be discussed in the next section.

Further, this set of experiments also demonstrates that depth-limited customized gates generation yields less desired performance. One of the issues is that a fixed depth may not fit all benchmarks. For the two AccQOC methods, accqoc_n3d5 with depth 5 for most of the time give a smaller latency than accqoc_n3d3 with depth three as it merges more gates in a customized gate. However, for qaoa and supre, accqoc_n3d3

[2]The majority here means the total count of APA-basis gate use is larger than the total count of original basis gate use in the circuit.

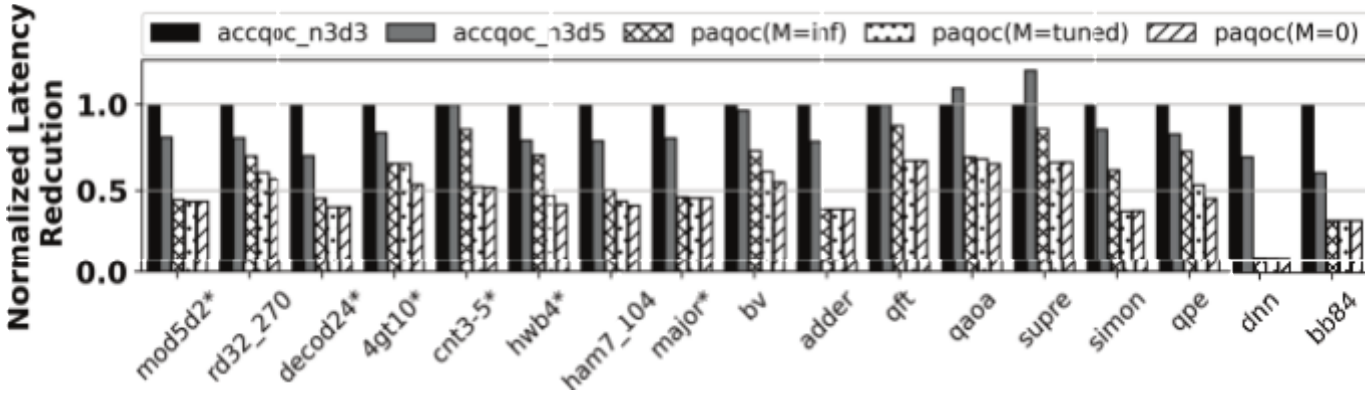


Fig. 10. The normalized circuit latency reduction for 17 benchmarks.

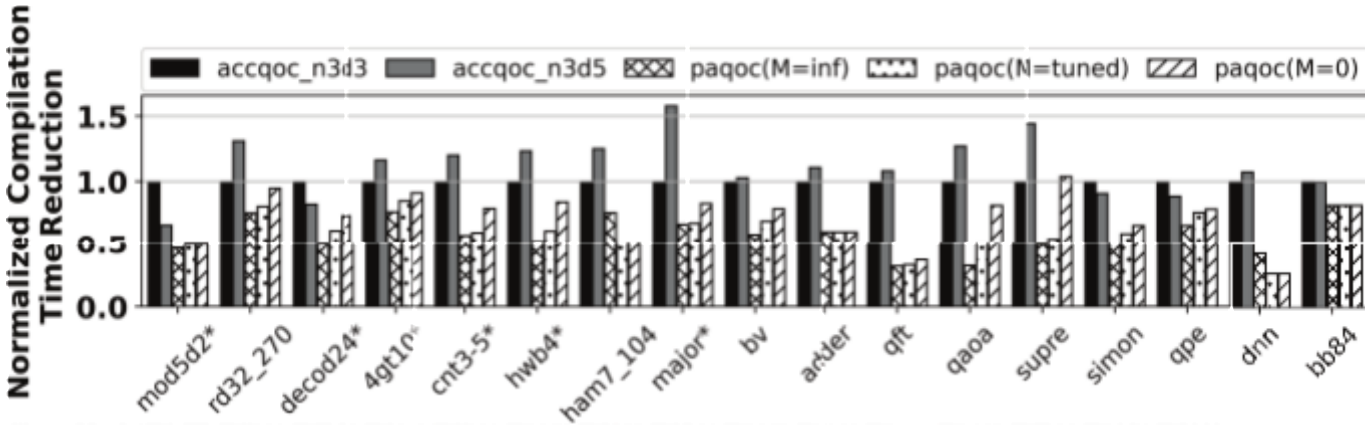


Fig. 11. The normalized circuit compilation time reduction for 17 benchmarks.

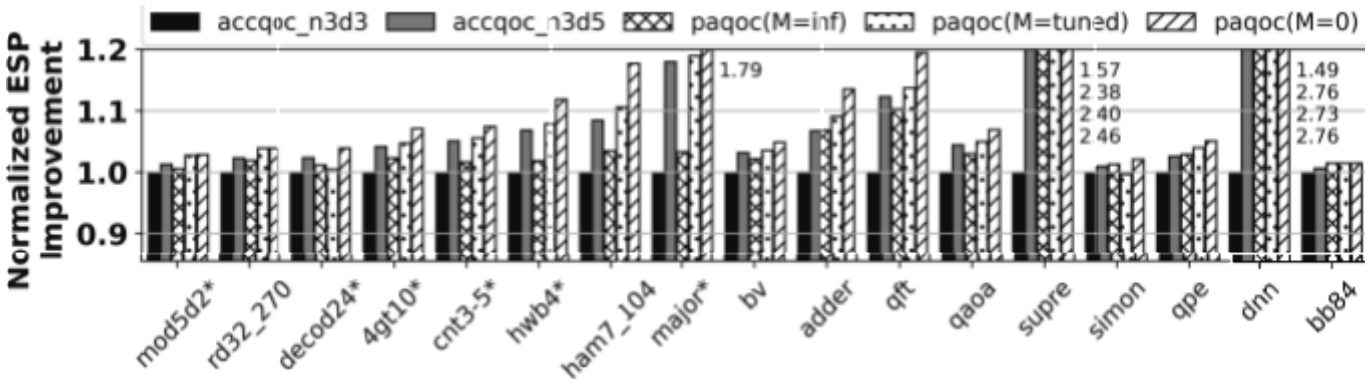


Fig. 12. The normalized ESP improvement for 17 benchmarks.

has a smaller latency as the depth of a frequent subcircuit 3, and by setting depth to 3, it happens to find the frequent subcircuit which implies the desired gate merging. For instance, in Fig. 13, we show a part of the qaoa benchmark and show how two methods partition into customized gates. As we can see, accqoc_n3d3 divides the circuit into two CPHASE gates which benefits the performance, but accqoc_n3d5 does not. With PAQOC, we can automatically detect the CPHASE gates using our frequent subcircuits miner without setting the depth parameter, which will be discussed in detail in Section VI-F.

B. Impact on Circuit Compilation Time

The comparison of the overall circuit compilation time is shown in Fig. 11, which is normalized to accqoc_n3d3. The most time-consuming part of PAQOC is the step of generating optimized control pulses, which contributes to an on-average 95% of the overall compilation time. For those APA-basis gates, the corresponding optimized control pulses only need to be calculated once and can reduce the overall compilation

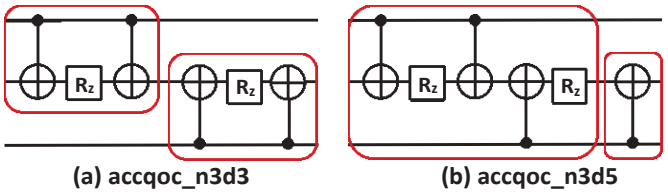


Fig. 13. Different gate grouping results from depth-limited AccQOC approaches: (a) Depth-3 limit happens to discover the CPHASE gate pattern, and (b) Depth-5 however does not.

overhead. Thus paqoc(M=inf) has a smaller compilation overhead than paqoc(M=0) and has a much smaller compilation overhead compared with the two accqoc baselines. In the meantime, it achieves much better circuit latency than AccQOC.

Moreover, the trade-off between the circuit latency reduction and the compilation time reduction is well-considered in paqoc(M=tuned). Using paqoc(M=tuned) can achieve a relatively better latency than paqoc(M=inf) and a faster compilation time than paqoc(M=0). In the meantime, its latency is comparable to that of paqoc(M=inf). It takes advantage of both the frequent sub-circuits miner and the critical-aware customized gates generator.

C. Impact on Circuit ESP

In Fig. 12, we show the improvement [3] of the circuit ESP of PAQOC compared with the AccQOC methods normalized to the baseline accqoc_n3d3. The circuit ESP is the product of the success rate of each customized gate using the fidelity reported by GRAPE as described in Eq. 2. For all benchmarks, paqoc(M=0) obtains the best ESP with an average 27% improvement compared with the baseline, and paqoc(M=tuned) is not bad either while maintaining a small compilation overhead.

D. Scalability of PAQOC

In Fig. 14, we show the scalability of paqoc(M=inf) with respect to the number of APA-basis gates in the transpiled circuit. The blue dashed linear regression line is calculated based on the results of all benchmarks. The compilation time of PAQOC scales well (almost linearly) with respect to the number of gates in the circuit, it takes less than an hour to compile a circuit with around 2,000 gates. paqoc shows a similar trend as generating the optimized control pulses dominates the compilation overhead. Moreover, the frequent subcircuit miner simplifies the circuits and further reduces the compilation overhead.

Note that the maximal compilation time of PAQOC for almost 1,200 gates is < 25 minutes (majority_239). Every 25 minutes PAQOC can make a recompilation, which is much less frequent than the daily calibration frequency of today's quantum computers. If the error terms in Hamiltonian can be calibrated quickly, our method will be readily deployable on the machines to account for real errors.

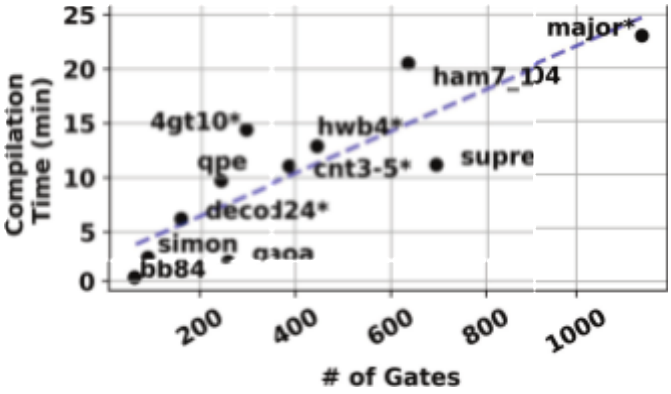


Fig. 14. paqoc(M=inf) circuit compilation time

E. Fidelity Improvements Shown by Pulse Simulation

Table II shows fidelity improvement using PAQOC, compared with AccQOC. We use Qutip [26] pulse simulator to calculate the fidelity of the generated pulse circuit. For all benchmarks, our method run with the best fidelity. The fidelity improvements come from the shorter control pulses generated by PAQOC and our criticality-aware customized gates generation method.

TABLE II
THE QUALITY OF EXECUTION, LARGER IS BETTER.

	accqoc n3d3	accqoc n3d5	paqoc (M=0)	paqoc (M=tuned)	paqoc (M=inf)
4gt10*	29.75%	18.09%	29.49%	35.24%	34.85%
decod24	8.73%	6.03%	35.32%	30.93%	29.13%
hwb4_49	7.32%	14.59%	16.54%	7.20%	4.57%
rd32_270	20.99%	2.16%	35.34%	36.31%	8.29%
bb84	2.25%	2.24%	7.12%	4.04%	2.84%
simon	6.07%	7.61%	9.31%	10.11%	7.47%

F. APA-Basis Gates Generation

PAQOC's frequent subcircuits miner generates the APA-basis gate set, which can be further used to simplify the original circuits. We show the frequent subcircuits found by PAQOC. Due to the space limit, we only show the frequent subcircuit for five benchmarks. These five benchmarks are Bernstein Vezirani [6] (bv), Cuccaro Adder [13] (adder), Quantum Fourier Transform [10] (qft), Quantum Approximation Optimization Algorithm [17] (qaoa) and the Supremacy algorithm used in [4] (supre). We only showcase the most frequent and second most frequent subcircuits in Table III[4]

For bv, the major component of the circuit is its oracle which consists of a linear sequence of CX gates and the DAG of the CX gates follows a linear pattern [6]. As today's

[3]The value of those out-of-scale bars are listed next to the bar in the order of the labels in the legend

[4]Due to limited space, the Toffoli gate represents the combination of multiple single- and two-qubit universal gates.

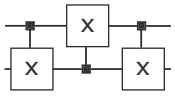
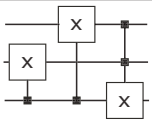
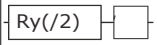
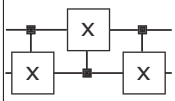
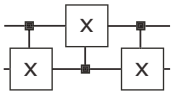
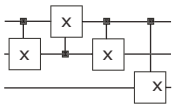
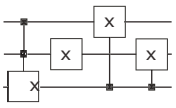
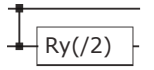
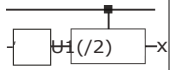
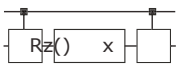
	!"	#$$%&	'()&%*+,-	./0	.#1#
!"#$%&'()*(+$% ,*-./'.*/$					
0%"..*''(+.(	12	3	43	51	67
,("+8%9"#$% &'()*(+$% ,*-./'.*/$					
0%"..*''(+.(	1:	3	43	:7	17

TABLE III
THE MOST AND THE SECOND MOST FREQUENT SUBCIRCUITS FOUND BY PAQOC.

superconducting architectures have sparse connectivity, SWAP gates are inserted [3], [32], [55]. At the low-level representation of the circuit, PAQOC finds that three concatenated CX gates form the most frequent subcircuit, which happens to be a SWAP gate in the high-level representation of the circuit. Aside from that, PAQOC finds the second most frequently occurring pattern is the three concatenated CX gates on the same two qubits plus one CX gate with a target on a third qubit. This is because SWAPs are performed to enable the original CX gates in bv in the circuit.

For adder, PAQOC also finds the most frequent and second most frequent subcircuits. We discovered that the most frequent subcircuit matches the majority gate (MAJ), and the second most frequent subcircuit matches a part of the "unmajority and add" gate (UMA) [13], according to the literature on the quantum adder algorithm. In hindsight, UMA and MAJ are the main components of a one-bit adder, which are the building blocks of a general adder. PAQOC is able to find them automatically, which further demonstrates the effectiveness of PAQOC in finding frequent subcircuits as the basis of the APA-gate set.

For supre, it starts and ends with a Hadamard (H) gate on each qubit and arranges nearest neighbor controlled-Z (CZ) gates in a repeated pattern [4]. One-qubit $\sqrt{X}$, $\sqrt{Y}$ and T gates are randomly interspersed between CZ gates. Thus the frequent subcircuits mined in supre depend on the input circuit.

For qft, the circuit has a pattern of a leading H gate followed by a controlled-U1 (CU1) gate between each pair of qubits. As QFT requires more communication between qubits, SWAP gates are added. Again, PAQOC detects SWAP gates as the most frequent pattern, and an H gate applying on the target qubit of a CU1 gate as the second most frequent subcircuit.

For qaoa, the circuit has a pattern of the CPHASE gate [17]. The CPHASE gate is usually a non-native gate for most hardware [3], [42]. One possible decomposition of a CPHASE gate consists of two CX gates and one R_z rotation gate between [1]. Again, PAQOC is able to extract the CPHASE gate as a very frequent subcircuit automatically.

VII. RELATED WORK

Quantum computing's traditional gate-based workflow has been extensively studied. Techniques for optimizing the compiler front-end and the hardware mapping problem have been proposed [32], [35], [45], [52], [53].

Recent research has discovered that even with GPU acceleration, using QOC to generate the optimal control pulses for circuits with a large number of qubits still has a significant overhead [9], [20], [31]. Gokhale et al. [20] applies QOC for a specific type of quantum algorithms that have an iterative nature, such as VQE [33], [41] and QAOA [17]. These programs are executed iteratively while between different iterations, the parameters of these variational gates (rotation angles) are determined by the previous iteration and the remaining non-parameterized gates are unchanged. To reduce the compilation time, the authors use precomputation to obtain the hyperparameters for QOC calculation. This strategy, however, does not work for non-variational quantum algorithms in general.

Shi et al. [43] proposed CLS that increases flexibility in pulse generation with the help of commutativity. Their work is orthogonal to ours. Our framework can also take commutativity into account and further improve circuit performance, which is left as future work.

A more recent study proposed by Cheng et al. [9] called AccQOC reduces the compilation overhead. AccQOC divides the circuit into small, fixed-size subcircuits. They keep the subcircuit size to two qubits and create a pulse database for sub-circuits where pulses have already been generated. They use a similarity graph based on the distance between different subcircuits of fixed sizes, and also not in the database. Then, the MST of the similarity graph is used to determine the construction order of the corresponding control pulses for these subcircuits to further reduce the compilation time.

Gokhale et al. [21] use an augmented basis gate set that adds flexibility and exploits low-level optimizations for quantum programs. Their augmented basis gates are based on specific

hardware, not based on quantum applications. And they do not consider merging gates to improve circuit performance. Rather they use the fixed-gate approach, just with more hardware basis gates.

Compared with these recent works, our proposed PAQOC framework provides a better exploration of different ways to construct customized gates. Additionally, by extracting program-aware basis gates, PAQOC is able to further reduce the compilation overhead while in meantime improving circuit pulse performance.

VIII. Conclusion

In this paper, we propose PAQOC, a pulse optimization framework that transforms general physical circuits into low-level control pulses using Quantum Optimal Control (QOC) technique. PAQOC extracts frequent subcircuits in the quantum circuit to build an APA basis gate set. Moreover, PAQOC constructs a lightweight criticality-aware customized gate grouping strategy to reduce the pulse generation overhead and improve circuit performance. PAQOC consists of three components: a frequent subcircuits miner, a customized gates generator, and a pulse database. Compared with the state-of-the baseline, PAQOC achieves up to 2.17X speedup in latency and an average of 1.75X compilation overhead.

IX. Acknowledgement

This work is supported by Rutgers Research Council grant and NSF-CCF-2129872. This work is also supported by the U.S. Department of Energy, Office of Science, National Quantum Information Science Research Centers, Co-design Center for Quantum Advantage (C2QA) under contract number DE-SC0012704. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of our sponsors.

References

[1] M. Alam, A. Ash-Saki, and S. Ghosh, "Circuit compilation methodologies for quantum approximate optimization algorithm," in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2020, pp. 215–228.

[2] T. Alexander, N. Kanazawa, D. J. Egger, L. Capelluto, C. J. Wood, A. Javadi-Abhari, and D. C. McKay, "Qiskit pulse: programming quantum computers through the cloud with pulses," Quantum Science and Technology, vol. 5, no. 4, p. 044006, 2020.

[3] M. S. ANIS, H. Abraham, AduOffei, R. Agarwal, G. Agliardi, M. Aharoni, I. Y. Akhalwaya, G. Aleksandrowicz, T. Alexander, M. Amy, S. Anagolum, E. Arbel, A. Asfaw, A. Athalye, A. Avkhadiev, C. Azaustre, P. BHOLE, A. Banerjee, S. Banerjee, W. Bang, A. Bansal, P. Barkoutsos, A. Barnawal, G. Barron, G. S. Barron, L. Bello, Y. Ben-Haim, M. C. Bennett, D. Bevenius, D. Bhatnagar, A. Bhobe, P. Bianchini, L. S. Bishop, C. Blank, S. Bolos, S. Bopardikar, S. Bosch, S. Brandhofer, Brandon, S. Bravyi, N. Bronn, Bryce-Fuller, D. Bucher, A. Burov, F. Cabrera, P. Calpin, L. Capelluto, J. Carballo, G. Carrascal, A. Carriker, I. Carvalho, A. Chen, C.-F. Chen, E. Chen, J. C. Chen, R. Chen, F. Chevallier, K. Chinda, R. Cholarajan, J. M. Chow, S. Churchill, CisterMoke, C. Claus, C. Clauss, C. Clothier, R. Cocking, R. Cocuzzo, J. Connor, F. Correa, A. J. Cross, A. W. Cross, S. Cross, J. Cruz-Benito, C. Culver, A. D. Córcoles-Gonzales, N. D, S. Dague, T. E. Dandachi, A. N. Dangwal, J. Daniel, M. Daniels, M. Dartiailh, A. R. Davila, F. Debouni, A. Dekusar, A. Deshmukh, M. Deshpande, D. Ding, J. Doi, E. M. Dow, E. Drechsler, E. Dumitrescu, K. Dumon, I. Duran, K. EL-Safty, E. Eastman, G. Eberle, A. Ebrahimi, P. Eendebak, D. Egger, ElePT, Emilio, A. Espiricueta, M. Everitt, D. Facoetti, Farida, P. M. Fernández, S. Ferracin, D. Ferrari, A. H. Ferrera, R. Fouilland, A. Frisch, A. Fuhrer, B. Fuller, M. GEORGE, J. Gacon, B. G. Gago, C. Gambella, J. M. Gambetta, A. Gammanpila, L. Garcia, T. Garg, S. Garion, J. Garrison, T. Gates, L. Gil, A. Gilliam, A. Giridharan, J. Gomez-Mosquera, Gonzalo, S. de la Puente Gonzaléz, J. Gorzinski, I. Gould, D. Greenberg, D. Grinko, W. Guan, J. A. Gunnels, H. Gupta, N. Gupta, J. M. Gunther, M. Haglund, I. Haide, I. Hamamura, O. C. Hamido, F. Harkins, K. Hartman, A. Hasan, V. Havlicek, J. Hellmers, Ł. Herok, S. Hillmich, H. Horii, C. Howington, S. Hu, W. Hu, J. Huang, R. Huisman, H. Imai, T. Imamichi, K. Ishizaki, Ishwor, R. Iten, T. Itoko, A. Ivrii, A. Javadi, A. Javadi-Abhari, W. Javed, Q. Jianhua, M. Jivrajani, K. Johns, S. Johnstun, Jonathan-Shoemaker, JosDenmark, JoshDumo, J. Judge, T. Kachmann, A. Kale, N. Kanazawa, J. Kane, Kang-Bae, A. Kapila, A. Karazeev, P. Kassebaum, J. Kelso, S. Kelso, V. Khanderao, S. King, Y. Kobayashi, Kovi11Day, A. Kovyrshin, R. Krishnakumar, V. Krishnan, K. Krsulich, P. Kumkar, G. Kus, R. LaRose, E. Lacal, R. Lambert, H. Landa, J. Lapeyre, J. Latone, S. Lawrence, C. Lee, G. Li, J. Lishman, D. Liu, P. Liu, Y. Maeng, S. Maheshkar, K. Majmudar, A. Malyshev, M. E. Mandouh, J. Manela, Manjula, J. Marecek, M. Mar-ques, K. Marwaha, D. Maslov, P. Maszota, D. Mathews, A. Matsuo, F. Mazhandu, D. McClure, M. McElaney, C. McGarry, D. McKay, D. McPherson, S. Meesala, D. Meirom, C. Mendell, T. Metcalfe, M. Mevissen, A. Meyer, A. Mezzacapo, R. Midha, D. Miller, Z. Minev, A. Mitchell, N. Moll, A. Montanez, G. Monteiro, M. D. Mooring, R. Morales, N. Moran, D. Morcuende, S. Mostafa, M. Motta, R. Mo-yard, P. Murali, J. Muggenburg, T. NEMOZ, D. Nadlinger, K. Nakan-ishi, G. Nannicini, P. Nation, E. Navarro, Y. Naveh, S. W. Neagle, P. Neuweiler, A. Ngoueya, J. Nicander, Nick-Singstock, P. Niroula, H. Norlen, NuoWenLei, L. J. O'Riordan, O. Ogunbayo, P. Ollitrault, T. Onodera, R. Otaolea, S. Oud, D. Padilha, H. Paik, S. Pal, Y. Pang, A. Panigrahi, V. R. Pascuzzi, S. Perriello, E. Peterson, A. Phan, F. Piro, M. Pistoia, C. Piveteau, J. Plewa, P. Pocreau, A. Pozas-Kerstjens, R. Pracht, M. Prokop, V. Prutyanov, S. Puri, D. Puzzuoli, J. Perez, Quant02, Quintiii, I. R, R. I. Rahman, A. Raja, R. Rajeev, N. Ramagiri, A. Rao, R. Raymond, O. Reardon-Smith, R. M.-C. Redondo, M. Reuter, J. Rice, M. Riedemann, Rietesh, D. Risinger, M. L. Rocca, D. M. Rodrıguéz, RohithKarur, B. Rosand, M. Rossmannek, M. Ryu, T. SAPV, N. R. C. Sa, A. Saha, A. Ash-Saki, S. Sanand, M. Sand-berg, H. Sandesara, R. Sapra, H. Sargsyan, A. Sarkar, N. Sathaye, B. Schmitt, C. Schnabel, Z. Schoenfeld, T. L. Scholten, E. Schoute, M. Schulterbrandt, J. Schwarm, J. Seaward, Sergi, I. F. Sertage, K. Setia, F. Shah, N. Shammah, R. Sharma, Y. Shi, J. Shoemaker, A. Silva, A. Simonetto, D. Singh, P. Singh, P. Singkanipa, Y. Siraichi, Siri, J. Sistos, I. Sitdikov, S. Sivarajah, M. B. Sletfjerding, J. A. Smolin, M. Soeken, I. O. Sokolov, I. Sokolov, V. P. Soloviev, SooluThomas, Starfish, D. Steenken, M. Stypulkoski, A. Suau, S. Sun, K. J. Sung, M. Suwama, O. Słowik, H. Takahashi, T. Takawale, I. Tavernelli, C. Tay-lor, P. Taylour, S. Thomas, K. Tian, M. Tillet, M. Tod, M. Tomasik, C. Tornow, E. de la Torre, J. L. S. Toural, K. Trabing, M. Trein-ish, D. Trenev, TrishaPe, F. Truger, G. Tsilimigkounakis, D. Tulsi, W. Turner, Y. Vaknin, C. R. Valcarce, F. Varchon, A. Vartak, A. C. Vazquez, P. Vijaywargiya, V. Villar, B. Vishnu, D. Vogt-Lee, C. Vuillot, J. Weaver, J. Weidenfeller, R. Wieczorek, J. A. Wildstrom, J. Wilson, E. Winston, WinterSoldier, J. J. Woehr, S. Woerner, R. Woo, C. J. Wood, R. Wood, S. Wood, J. Wootton, M. Wright, L. Xing, J. YU, B. Yang, D. Yeralin, R. Yonekura, D. Yonge-Mallo, R. Yoshida, R. Young, J. Yu, L. Yu, C. Zachow, L. Zdanski, H. Zhang, C. Zoufal, aed-dins ibm, alexzhang13, b63, bartek bartlomiej, bcamorrison, brandhsn, charmerDark, deeplokhande, dekel.meirom, dime10, dlasecki, ehchen, fanizzamarco, fs1132429, gadial, galeinston, georgezhou20, georgios ts, gruu, hhorii, hykavitha, itoko, jessica angel7, jezerjojo14, jliu45, jscott2, klinvill, krutik2966, ma5x, michelle4654, msuwama, ntgiwsvp, ordmoj, sagar pahwa, pritamsinha2304, ryancocuzzo, saswati qiskit, septembrr, sethmerkel, shaashwat, sternparky, strickroman, tigerjack, tsura crisaldo, vadebayo49, welien, willhbang, wmurphy collabstar, yang.luh, and M. Čepulkovskis, "Qiskit: An open-source framework for quantum computing," 2021.

[4] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell et al., "Quantum supremacy using a programmable superconducting processor," Nature, vol. 574, no. 7779, pp. 505–510, 2019.

[5] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," Physical review A, vol. 52, no. 5, p. 3457,

1995.
[6] E. Bernstein and U. Vazirani, "Quantum complexity theory," SIAM Journal on computing, vol. 26, no. 5, pp. 1411–1473, 1997.
[7] 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.
[8] S. Boixo, S. V. Isakov, V. N. Smelyanskiy, R. Babbush, N. Ding, Z. Jiang, M. J. Bremner, J. M. Martinis, and H. Neven, "Characterizing quantum supremacy in near-term devices," Nature Physics, vol. 14, no. 6, pp. 595–600, 2018.
[9] J. Cheng, H. Deng, and X. Qian, "Accqoc: Accelerating quantum optimal control based pulse generation," in 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2020, pp. 543–555.
[10] D. Coppersmith, "An approximate fourier transform useful in quantum factoring," arXiv preprint quant-ph/0201067, 2002.
[11] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," arXiv preprint arXiv:1707.03429, 2017.
[12] A. W. Cross, A. Javadi-Abhari, T. Alexander, N. de Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, J. Smolin, J. M. Gambetta, and B. R. Johnson, "Openqasm 3: A broader and deeper quantum assembly language," arXiv preprint arXiv:2104.14722, 2021.
[13] S. A. Cuccaro, T. G. Draper, S. A. Kutin, and D. P. Moulton, "A new quantum ripple-carry addition circuit," arXiv preprint quant-ph/0410184, 2004.
[14] D. d'Alessandro, Introduction to quantum control and dynamics. Chapman and hall/CRC, 2021.
[15] P. De Fouquieres, S. Schirmer, S. Glaser, and I. Kuprov, "Second order gradient ascent pulse engineering," Journal of Magnetic Resonance, vol. 212, no. 2, pp. 412–417, 2011.
[16] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," Proceedings of the VLDB Endowment, vol. 7, no. 7, pp. 517–528, 2014.
[17] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," arXiv preprint arXiv:1411.4028, 2014.
[18] E. Farhi and A. W. Harrow, "Quantum supremacy through the quantum approximate optimization algorithm," arXiv preprint arXiv:1602.07674, 2016.
[19] S. J. Glaser, U. Boscain, T. Calarco, C. P. Koch, W. Kockenberger, R. Kosloff, I. Kuprov, B. Luy, S. Schirmer, T. Schulte-Herbrüggen et al., "Training schrödinger's cat: quantum optimal control," The European Physical Journal D, vol. 69, no. 12, pp. 1–24, 2015.
[20] P. Gokhale, Y. Ding, T. Propson, C. Winkler, N. Leung, Y. Shi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Partial compilation of variational algorithms for noisy intermediate-scale quantum machines," in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 266–278.
[21] P. Gokhale, A. Javadi-Abhari, N. Earnest, Y. Shi, and F. T. Chong, "Optimized quantum compilation for near-term algorithms with openpulse," in 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2020, pp. 186–200.
[22] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, and B. Valiron, "Quipper: a scalable quantum programming language," in Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation, 2013, pp. 333–342.
[23] L. K. Grover, "A fast quantum mechanical algorithm for database search," in Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, 1996, pp. 212–219.
[24] J. Hsu, "Intels 49-qubit chip shoots for quantum supremacy," 2018.
[25] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "Scaffcc: a framework for compilation and analysis of quantum computing programs," in Proceedings of the 11th ACM Conference on Computing Frontiers, 2014, pp. 1–10.
[26] J. R. Johansson, P. D. Nation, and F. Nori, "Qutip: An open-source python framework for the dynamics of open quantum systems," Computer Physics Communications, vol. 183, no. 8, pp. 1760–1772, 2012.
[27] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," Nature, vol. 549, no. 7671, pp. 242–246, 2017.
[28] W. Knight, "Ibm raises the bar with a 50-qubit quantum computer," Sighted at MIT Review Technology, 2017.
[29] V. F. Krotov, Global Methods in Optimal Control Theory. Boston, MA: Birkhäuser Boston, 1993, pp. 74–121. [Online]. Available: https://doi.org/10.1007/978-1-4612-0349-0_3
[30] L. Lao, P. Murali, M. Martonosi, and D. Browne, "Designing calibration and expressivity-efficient instruction sets for quantum computing," in 2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2021, pp. 846–859.
[31] N. Leung, M. Abdelhafez, J. Koch, and D. Schuster, "Speedup for quantum optimal control from automatic differentiation based on graphics processing units," Physical Review A, vol. 95, no. 4, p. 042318, 2017.
[32] G. Li, Y. Ding, and Y. Xie, "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, 2019, pp. 1001–1014.
[33] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms," New Journal of Physics, vol. 18, no. 2, p. 023023, 2016.
[34] D. C. McKay, T. Alexander, L. Bello, M. J. Biercuk, L. Bishop, J. Chen, J. M. Chow, A. D. Corcoles, D. Egger, S. Filipp et al., "Qiskit backend specifications for openqasm and openpulse experiments," arXiv preprint arXiv:1809.03452, 2018.
[35] 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.
[36] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, "Full-stack, real-system quantum computer studies: Architectural comparisons and design insights," in 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2019, pp. 527–540.
[37] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.
[38] A. Omran, H. Levine, A. Keesling, G. Semeghini, T. T. Wang, S. Ebadi, H. Bernien, A. S. Zibrov, H. Pichler, S. Choi et al., "Generation and manipulation of schrodinger cat states in rydberg atom arrays," Science, vol. 365, no. 6453, pp. 570–574, 2019.
[39] A. P. Peirce, M. A. Dahleh, and H. Rabitz, "Optimal control of quantum-mechanical systems: Existence, numerical approximation, and applications," Physical Review A, vol. 37, no. 12, p. 4950, 1988.
[40] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," Nature communications, vol. 5, p. 4213, 2014.
[41] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'brien, "A variational eigenvalue solver on a photonic quantum processor," Nature communications, vol. 5, no. 1, pp. 1–7, 2014.
[42] Rigetti. (2020) Rigettiqpu. [Online]. Available: http://docs.rigetti.com/en/1.9/qpu.html
[43] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," in Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, 2019, pp. 1031–1044.
[44] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," SIAM review, vol. 41, no. 2, pp. 303–332, 1999.
[45] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, "Qubit allocation," in Proceedings of the 2018 International Symposium on Code Generation and Optimization, 2018, pp. 113–125.
[46] R. S. Smith, M. J. Curtis, and W. J. Zeng, "A practical quantum instruction set architecture," arXiv preprint arXiv:1608.03355, 2016.
[47] C. Song, K. Xu, H. Li, Y.-R. Zhang, X. Zhang, W. Liu, Q. Guo, Z. Wang, W. Ren, J. Hao et al., "Generation of multicomponent atomic schrödinger cat states of up to 20 qubits," Science, vol. 365, no. 6453, pp. 574–577, 2019.
[48] J. Werschnik and E. Gross, "Quantum optimal control theory," Journal of Physics B: Atomic, Molecular and Optical Physics, vol. 40, no. 18, p. R175, 2007.
[49] R. Wille, D. Große, L. Teuber, G. W. Dueck, and R. Drechsler, "Revlib: An online resource for reversible functions and reversible circuits," in 38th International Symposium on Multiple Valued Logic (ismvl 2008). IEEE, 2008, pp. 220–225.

[50] L. Xie, J. Zhai, Z. Zhang, J. Allcock, S. Zhang, and Y.-C. Zheng, "Suppressing zz crosstalk of quantum computers through pulse and scheduling co-optimization," in Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS 2022. New York, NY, USA: Association for Computing Machinery, 2022, p. 499–513. [Online]. Available: https://doi.org/10.1145/3503222.3507761
[51] J. You and F. Nori, "Quantum information processing with superconducting qubits in a microwave field," Physical Review B, vol. 68, no. 6, p. 064509, 2003.
[52] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, 2021, pp. 360–374.
[53] A. Zulehner, A. Paler, and R. Wille, "Efficient mapping of quantum circuits to the ibm qx architectures," in 2018 Design, Automation Test in Europe Conference Exhibition (DATE), 2018, pp. 1135–1138.
[54] A. Zulehner, A. Paler, and R. Wille, "An efficient methodology for mapping quantum circuits to the ibm qx architectures," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 38, no. 7, pp. 1226–1236, 2018.
[55] A. Zulehner and R. Wille, "Compiling su (4) quantum circuits to ibm qx architectures," in Proceedings of the 24th Asia and South Pacific Design Automation Conference, 2019, pp. 185–190.