Circuit Compilation Methodologies for Quantum Approximate Optimization Algorithm

Mahabubul Alam
Pennsylvania State University
University Park, USA
mxa890@psu.edu

Abdullah Ash- Saki
Pennsylvania State University
University Park, USA
axs1251@psu.edu

Swaroop Ghosh
Pennsylvania State University
University Park, USA
szg212@psu.edu

***Abstract*—The quantum approximate optimization algorithm (QAOA) is a promising quantum-classical hybrid algorithm to solve hard combinatorial optimization problems. The multi-qubit CPHASE gates used in the quantum circuit for QAOA are commutative i.e., the order of the gates can be altered without changing the output state. This re-ordering leads to the execution of more gates in parallel and a smaller number of additional SWAP gates to compile the QAOA-circuit. Consequently, the circuit-depth and cumulative gate-count become lower which is beneficial for circuit execution time and noise resilience. A less number of gates indicates a lower accumulation of gate-errors, and a reduced circuit-depth means less decoherence time for the qubits. However, finding the best-ordered circuit is a difficult problem and does not scale well with circuit size. This paper presents four generic methodologies to optimize QAOA-circuits by exploiting gate re-ordering. We demonstrate a reduction in gate-count by ≈23.0% and circuit-depth by ≈53.0% on average over a conventional approach without incurring any compilation-time penalty. We also present a variation-aware compilation which enhances the compiled circuit success probability by ≈62.7% for the target hardware over the variation unaware approach. A new metric, *Approximation Ratio Gap (ARG)*, is proposed to validate the quality of the compiled QAOA-circuit instances on actual devices. Hardware implementation of a number of QAOA instances shows ≈25.8% improvement in the proposed metric on average over the conventional approach on ibmq_16_melbourne.**

I. Introduction

Quantum computing is one of the most transformative technologies of the present time. Prototypical quantum computers with 5-128 qubits are available or proposed [1]–[4] from industry vendors like IBM, Google, Rigetti, etc. Recently, Google claimed quantum supremacy with a 53-qubit quantum processor to complete a computational task in 200 seconds that might take 10000 years (later rectified to 2.5 days [5]) on the state-of-the-art supercomputers [6]. This is a significant milestone for quantum computing. Apart from the limited number of qubits and connectivity, the near-term devices are plagued with various kind of errors such as gate-error, decoherence, crosstalk, etc. [7]–[11]. Therefore, quantum error correction codes (QECCs) [12]–[17] are necessary for fault-tolerant computation. However, QECCs have prohibitively high physical qubit overhead. Therefore, variational quantum-classical algorithms are being explored to gain the quantum advantage for various problems in physics, chemistry [18]–[22], optimizations [23]–[27], and machine learning [28]–[34]. Quantum Approximate Optimization Algorithm (QAOA) [35]–[37] is at the forefront of these hybrid algorithms which is particularly useful to solve optimization problems and touted as a prime candidate for early demonstration of quantum supremacy [38]. However, the perceived quantum advantage through QAOA may be lost due to the accumulation of gate errors and decoherence [39]–[41]. An optimized circuit can show greater resilience to noise, and enhance the probability of generating the correct quantum state. *This makes QAOA circuit optimization an important problem in the NISQ-era [42].*

The detailed theoretic discussion on QAOA can be found in other literature [26], [27], [35]–[38]. QAOA involves parameter optimization of a multi-level parameterized quantum circuit (PQC). The PQC runs in a quantum-classical hybrid optimization loop to minimize (or maximize) the expectation value of a classical cost function. QAOA performance improves with added levels in the PQC. The total number of levels is referred to as 'p'. However, each level adds additional two parameters (γ, β) to the PQC which may affect the convergence and the speed [27], [43]. These parameter values can be found (without the optimization routines) by exploiting their relationship among similar instances [44] or analytically [45].

The PQC to solve the maximum cut (MaxCut) problem of a 4-node 3-regular graph (Figure 1(a)) with QAOA is shown in Figure 1(b) ('p' = 1). Note that, the PQC has an associated CPHASE operation in every level of the circuit for every edge in the problem graph for the MaxCut problem. CPHASE is a two-qubit unitary parametric quantum gate operating between a control and a target qubit. The black circle and the square enclosing the letter Z and the parameter γ are the control and target qubits of a CPHASE operation in Figure 1(b) and (c). Also, note that two consecutive gates can be executed concurrently if they operate on a different set of qubits. For example, the first two CPHASE operations in the circuit in Figure 1(b) can not be executed concurrently as they share a logical qubit (q2).

The CPHASE operations in a QAOA circuit are commutative [26], [46] i.e., the order of these CPHASE gates can be interchanged without affecting the output state of the quantum circuit. We can use this knowledge to maximize concurrent gate operations by choosing an optimal order of the gates. Figure 1(b) shows the QAOA-MaxCut circuit instance with

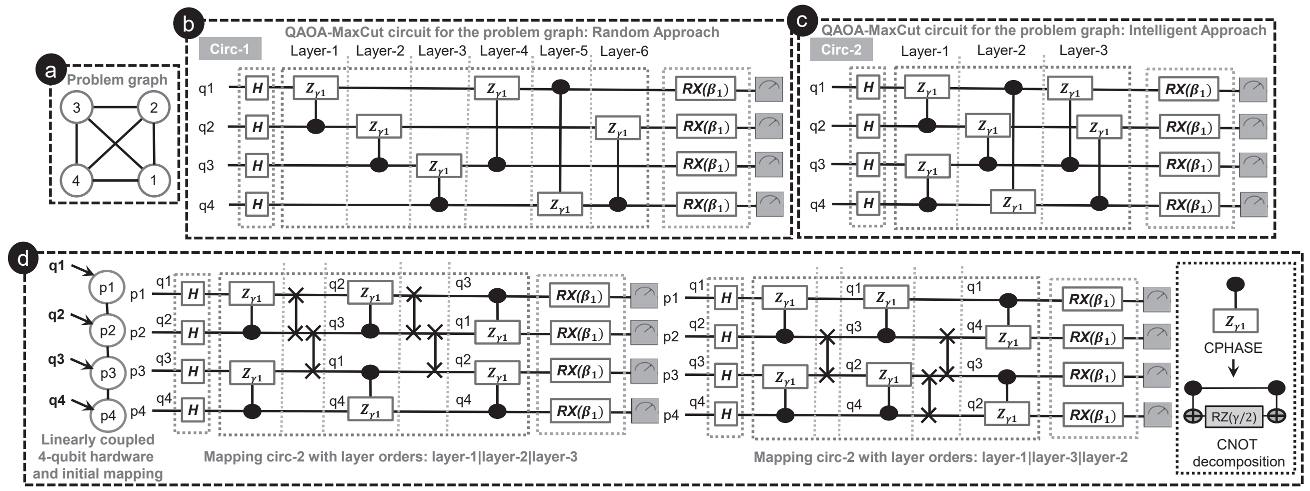

a
Problem graph
b
QAOA-MaxCut circuit for the problem graph: Random Approach
Circ-1
Layer-1 Layer-2 Layer-3 Layer-4 Layer-5 Layer-6
c
QAOA-MaxCut circuit for the problem graph: Intelligent Approach
Circ-2
Layer-1 Layer-2 Layer-3
q1 q2 q3 q4
H
$Z_{\gamma 1}$
$RX(\beta_1)$
d
p1 p2 p3 p4
Linearly coupled 4-qubit hardware and initial mapping
Mapping circ-2 with layer orders: layer-1|layer-2|layer-3
Mapping circ-2 with layer orders: layer-1|layer-3|layer-2
CPHASE
RZ(γ/2)
CNOT decomposition

Fig. 1. (a) A 4-node 3-Regular graph, (b) a randomly constructed QAOA-MaxCut instance (circ-1) of the 4-node graph with p = 1, (c) an optimized circuit (circ-2) for the problem with reduced number of layers, (d) SWAP addition during circuit compilation for a target hardware with different layer orders.

randomly ordered CPHASE operations for the problem graph in Figure 1(a) (circ-1). Figure 1(c) shows an intelligently gate re-ordered circuit (circ-2). Note that, if these circuits are executed in quantum hardware with full qubit-to-qubit connectivity supporting the following basis gates: H, RX, and CPHASE, circ-1 will require 9 time steps while circ-2 will take 6 time steps (including the measurement operations). If every gate takes a similar execution time in the hardware, circ-2 will be 50% faster and will experience less decoherence. Re-ordering these layers of CPHASE gates (e.g. interchanging layer-2 and layer-3 in circ-2) will not provide any reduction in the circuit cumulative execution time.

However, if we consider target hardware with limited connectivity such as the 4 linearly coupled physical qubits (p1, p2, p3, p4) in Figure 1(d), there will be further scope of optimization in circ-2. For such architectures, SWAP gates are added between two layers to meet the hardware constraints [47], [48]. For the initial logical-to-physical qubit assignment shown in Figure 1(d), interchanging the CPHASE layer 2 and 3 (in circ-2) will reduce the additional SWAP operations from 4 to 3. Therefore, the CPHASE gates that are picked for different layers will affect the quality of the compiled circuit for such target architectures.

Conventional compilers can optimize QAOA-circuits using efficient gate scheduling strategies utilizing the commutation properties of the gates [49]. However, incorporating gate-reordering strategies in a compiler is not straight forward. First, the compiler has to check for the commutative gates in the given circuit. The complexity of the problem scales with the size of the quantum circuit. Second, additional constraints need to be added to the compiler optimization heuristic to make use of the commutation properties which can affect the compilation speed. For example, a QAOA-specific compiler developed in [46] to take advantage of the commutation properties reported 70 seconds compilation time for simple 8-qubit circuits. Compiling optimal QAOA-circuits of practical significance (e.g. problems requiring $\approx$100 qubits or more) using such approaches may prove impractical due to the compilation time overhead.

Note that, recent studies claim that various sources of noise flatten the solution space of QAOA [39], [40]. Therefore, finding a mapping with higher reliability (less impacted by noise) is important to extract maximum performance from QAOA. Higher gate-count and depth affect the reliability of the circuit. Moreover, a higher depth quantum circuit increases execution time and reduces the algorithmic speed. Therefore, minimizing the depth/gate-count of the compiled circuit is crucial for QAOA applications. An efficient circuit compiler will choose the order of the commutative gates in QAOA circuits intelligently to maximize the compiled circuit quality in a scalable way (i.e., the methodologies should be applicable for larger problem sizes for powerful quantum hardware with 200-500 qubits). However, relevant methodologies are absent in the literature. In this article, we present scalable heuristics that can be incorporated in conventional compilers to optimize QAOA-circuits utilizing the commutation properties.

We make the following contributions for QAOA circuits compilation. We, (a) present a novel Qubit Allocation and Initial Mapping (QAIM) that chooses an intelligent logical-to-physical qubit mapping based on the problem characteristics and target coupling graph to reduce the need for qubit movement in the subsequent SWAP insertion procedure, (b) propose a greedy heuristic for Instruction Parallelization (IP) to reduce the circuit depth and execution time, (c) propose an Incremental Compilation (IC) technique that reduces the need for added SWAP operations utilizing the dynamic changes in logical-to-physical qubit mapping during the SWAP insertion procedure, (d) propose a Variation-aware Incremental Compilation (VIC) technique that enhances compiled circuit success probability by prioritizing gate operations with higher reliability, (e) present detailed comparative analysis of the proposed methodologies in terms of circuit quality metrics e.g., depth, gate-count, compilation time, success probability, and ARG for a total of 1200 QAOA-MaxCut instances for 3 different qubit architecture with 15 to 36 qubits and, (f) provide directives for their appropriate usage and future developments.

To the best of our knowledge, this is the first work to propose application-specific circuit compilation methodologies for general-purpose compilers. In the remaining paper, we cover the basics of quantum computing and circuit compilation (Sections II and III), discuss our proposed methodologies and their performance (Sections IV, V and VI), summarize prior works and draw conclusions (Sections VII and VIII).

II. Quantum Computing Basics

To keep the article self-contained, we briefly review the basics of quantum computation in this section.

Qubits and Quantum gates: Qubit is analogous to classical bits however, a qubit can be in a superposition state i.e., a combination of 0 and 1 at the same time. Quantum gates such as single qubit (e.g., Pauli-X (σ_x) gate) or multiple qubit (e.g., 2-qubit CNOT gate) gates modulate the state of qubits and thus perform computations.

Gate Error, Decoherence and Crosstalk: Quantum gates are error-prone. Besides, the qubits suffer from decoherence i.e., the qubits spontaneously interact with the environment and lose states. Therefore, the output of a quantum circuit is erroneous. The deeper quantum circuit needs more time to execute and gets affected by decoherence. More gates in the circuit also increase the accumulation of gate error. Thus, lower depth and number of gates in the circuit improves noise resiliency. Parallel gate operations on different qubits can affect each others performance which is known as crosstalk.

Success Probability: The success probability of a gate is the conjugate of the error-rate ($1-$error). The success probability of a circuit is defined as the product of the success probabilities of individual gates [50], [51]. A higher value indicates a higher probability of successful execution of the circuit on actual hardware.

Basis Gates and Coupling Constraints: A practical quantum computer supports a limited number of single and multi-qubit gates known as basis (or native) gates of the hardware. IBM quantum computers offer single-qubit $\{$U1, U2, U3, ID$\}$ and two-qubit CNOT gate as basis gates. However, the quantum circuit may contain non-native gates to the target hardware e.g., the CPHASE gate for IBM quantum computers. Hence, the gates in a quantum circuit need to be decomposed into the basis gates before execution. A CNOT decomposition of a CPHASE gate is shown in Figure 1(d). The native two-qubit gate may or may not be permitted between all the two-qubit pairs in the target hardware. These limitations are also known as coupling constraints. Conventional compilers add necessary SWAP gates to meet these constraints.

QAOA-circuits: Combinatorial optimization problems can be formulated using the Ising spin-glass model [24] which can be directly translated to a Hamiltonian by promoting each of the binary variables to a Pauli-Z operator. Each of the quadratic terms in the Ising model becomes a ZZ-interaction in the Hamiltonian that can be executed using the CPHASE gate in quantum computer [24]. In QAOA, such a Hamiltonian is formed for a given optimization problem and then, it is executed on a quantum system with controlled duration followed by a set of rotation gates. This execution of cost Hamiltonian and rotation gates can be repeated in the multi-level version of QAOA ('p' times for 'p'-level QAOA). Therefore, the cost Hamiltonian in QAOA is a collection of CPHASE gates operating on different qubits e.g., the cost Hamiltonian for the QAOA-MaxCut problem of the 4-node 3-Regular graph (Figure 1(a)) consists of 6 CPHASE gates acting between 4 logical qubits shown in Figure 1(b).

QAOA Optimization Flow and Approximation Ratio: In a QAOA optimization flow, the expectation value of the cost function is calculated by taking its mean over a finite number of samples from the QAOA-circuit output [27]. The parameters of the circuit are iteratively updated to maximize (or minimize) this expectation value. The circuit outputs are sampled a finite number of times with the optimal parameter values. The cost function is evaluated with these samples and the sample producing the highest (/lowest) cost is taken as the approximate solution. In such cases, the approximation ratio (defined as the ratio between the mean cost function value over these samples and the actual maximum function value) quantifies the QAOA performance [26], [27].

III. Background and Motivation

Mapping quantum circuits for target quantum hardware has been proven to be NP-complete [52], [53]. Two disparate approaches are followed to solve the problem. In the first approach, the mapping problem is formulated as a constraint satisfaction problem and later, powerful reasoning engines (such as, SMT solver, ILP solver, etc.) are used to find a solution that meets these constraints [54]–[56]. These approaches can find the global solution for small problems, but they lack scalability [55]. The second approach relies on efficient heuristics that gradually lead towards a solution [47], [57]. These approaches are scalable, however, they may often get stuck in local optima resulting in sub-optimal mapping.

Any qubit mapping procedure has following basic steps: (i) selection of physical qubits (qubit allocation or topology selection), (ii) initial logical-to-physical qubit mapping (initial placement), and (iii) addition of SWAP gates to meet the hardware coupling constraints (using heuristics or constraints solvers). Each of these steps affects the quality of the compiled circuit i.e., depth, gate-count, and reliability [56], [58].

Qubit Allocation: For topology selection, a natural approach is selecting a 'k'-node connected sub-graph from the 'n'-node (n>k) hardware coupling graph with maximum edges to implement a circuit with 'k' qubits. The rationale is that more connectivity will reduce the need for SWAP operations during the qubit mapping procedure. IBM's qiskit compiler uses this optimization technique in one of its many optimization levels [48]. In an interesting perspective to the topology selection problem, Variation aware Qubit Allocation

(VQA) technique selects sub-graph maximizing the cumulative reliability of the links rather than the total number of links [50]. The idea is to maximize the success probabilities of different multi-qubit operations within hardware under variability.

Initial Mapping: The initial mapping problem has been addressed using the reversibility property of quantum circuits [57]. A reverse traversal approach is proposed where a random initial mapping is updated by compiling the original circuit and its reverse iteratively. Between iterations, the final mapping of the previous compilation step is taken as the initial mapping for the current one. A few (3) reverse traversals showed significant performance improvement at the expense of higher compilation time due to repeated compilations [57]. In [53], first, the number of logical qubits coupled with a certain qubit is counted, and then, the algorithm searches for a match with the outdegree of the physical qubit in the coupling graph without considering any temporal information [53]. Two heuristics termed as GreedyE* and GreedyV* are presented in [59]. In GreedyE* policy, program CNOTs, and their control and target qubits are placed in a heaviest edge first order (maximum CNOT operations between two logical qubits). In GreedyV* policy, program qubits are placed on hardware qubits in the heaviest qubit first order (qubit involved in the maximum number of operations).

SWAP Insertion: The majority of the heuristics-guided SWAP insertion algorithms (including IBM's qiskit compiler) partitions the circuit in different layers where each layer consists of gates that can be executed concurrently in the hardware (gates operating on a different set of qubits) [47], [48], [60], and then add necessary SWAP operations between layers to accommodate all the gate operations within each layer. Adding SWAP operations to bring two qubits closer and executing a CNOT operation affects all other subsequent CNOT operations. Therefore, considering many operations at the same time may help in reducing the number of aggregated SWAP operations. In its simplest form, every layer may consist of a single multi-qubit operation [61] where SWAPs are added between layers just to meet the coupling constraint for the target operation. Minimizing the number of SWAPs is often the optimization goal [61] however, an interesting approach is proposed that also considers the reliability of these SWAP operations (VQM) [50]. When multiple paths exist, VQM chooses the path with higher reliability even if the number of SWAPs is higher than the other paths.

Motivating Factors: Note that each qubit may interact with another qubit only once within a level (either 1 CPHASE or no CPHASE) in QAOA-circuits. Hence, heuristics, such as GreedyE*, which prioritizes qubit pair placements with maximum interactions, is not suitable for these circuits. For QAOA-circuits, a more rational approach would be placing a group of qubits closer together that interact with each other. Additionally, the commutation of CPHASE gates can be used to our advantage to maximize parallel gate operations and minimize the number of layers, which will eventually help compilers, such as [47], [48], to generate better quality circuits. To achieve these objectives, we need to address the following questions: (i) how can we perform qubit allocation and initial qubit placement to ensure that qubits are surrounded by other qubits they are coupled to? (ii) what can be an efficient way to parallelize operations so that the existing compilers can come up with better quality compiled circuits?

IV. Proposed Methodologies

A general-purpose compiler (i.e. qiskit compiler from IBM) can be used to compile QAOA circuits with random initial mapping and randomly ordered CPHASE gates. We term this as the NAIVE approach and use it to quantify the performance benefits of four proposed methodologies - QAIM, IP, IC, and VIC - which can be integrated into any conventional compiler. QAIM is an intelligent qubit allocation and initial mapping approach which applies to any circuit compilation. IP, IC, and VIC use a backend compiler to add SWAP operations into the circuit with various target objectives. Each of these methods has exclusive benefits and should be adopted based on the requirements of the target application. The workflow to incorporate QAIM, IP, IC, and VIC in QAOA-circuit compilation is shown in Figure 2. Starting with a QAOA problem instance, target hardware coupling graph and hardware calibration data, QAIM generates an initial logical-to-physical qubit mapping that is passed to the chosen compilation procedure (i.e., IP/IC/VIC) which in turn, uses a backend compiler to generate the hardware compliant circuit. While IP passes a complete circuit description to the backend, IC and VIC send partial circuit descriptions in multiple iterations and stitch the compiled circuits at the end. The details of the individual procedures are discussed in this section.

A. Integrated Qubit Allocation and Initial Mapping (QAIM)

QAIM combines the qubit allocation and initial mapping procedures in a single step and seeks to achieve three objectives: (i) choosing a sub-graph from the hardware coupling graph maximizing the connectivity within the physical qubits (qubit allocation), (ii) minimizing initial distances between the logically neighboring qubits in the sub-graph (initial mapping), and (iii) achieving the first two objectives in a scalable way. QAIM uses efficient heuristics that exploit the following two profiling statistics:

Hardware Profiling: If a physical qubit has many neighbors, the logical qubit mapped to it is less likely to move

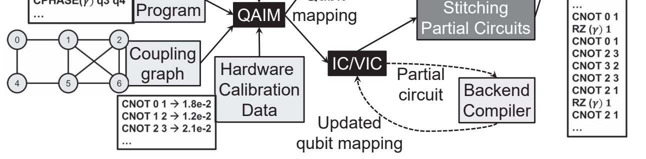
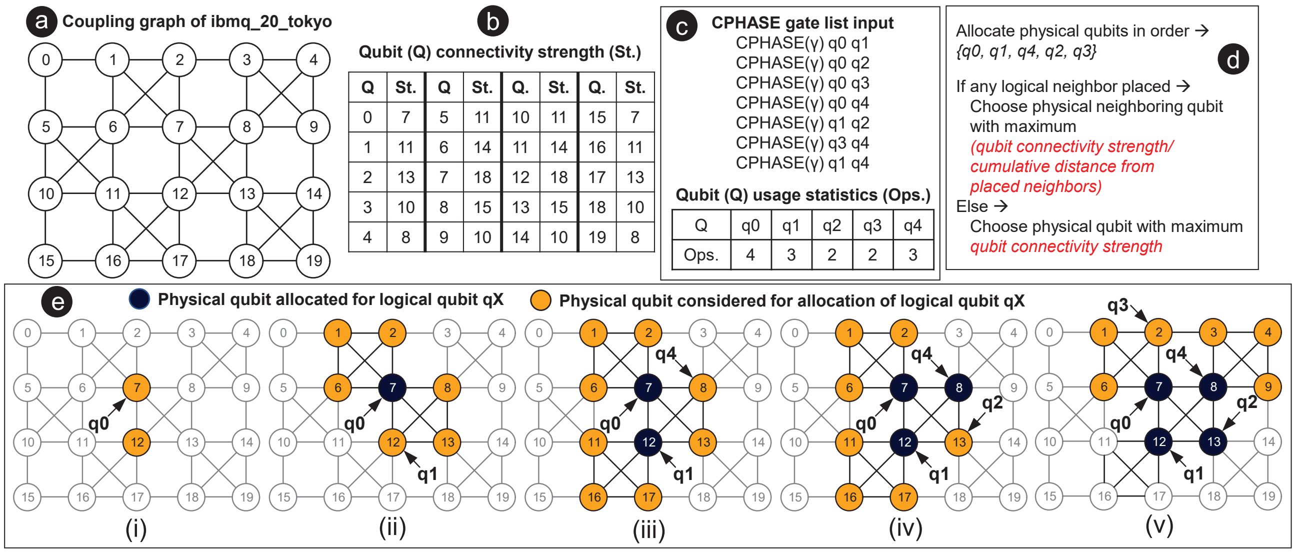

```
...
CPHASE(γ) q0 q1
CPHASE(γ) q1 q2
CPHASE(γ) q0 q4
CPHASE(γ) q0 q3
CPHASE(γ) q3 q4
...
```
```
CNOT 0 1 → 1.8e-2
CNOT 1 2 → 1.2e-2
CNOT 2 3 → 2.1e-2
...
```
```
...
CNOT 0 1
RZ (γ) 1
CNOT 0 1
CNOT 2 3
CNOT 3 2
CNOT 2 3
CNOT 2 1
RZ (γ) 1
CNOT 2 1
...
```

Fig. 2. A generic workflow incorporating the proposed compilation methodologies on top of a traditional compiler backend.


a Coupling graph of ibmq_20_tokyo
e
Physical qubit allocated for logical qubit qX
Physical qubit considered for allocation of logical qubit qX
q0
q1
q2
q3
q4
(i)
(ii)
(iii)
(iv)
(v)

b Qubit (Q) connectivity strength (St.)

Q	St.	Q	St.	Q.	St.	Q.	St.
0	7	5	11	10	11	15	7
1	11	6	14	11	14	16	11
2	13	7	18	12	18	17	13
3	10	8	15	13	15	18	10
4	8	9	10	14	10	19	8

c CPHASE gate list input

CPHASE(γ) q0 q1
CPHASE(γ) q0 q2
CPHASE(γ) q0 q3
CPHASE(γ) q0 q4
CPHASE(γ) q1 q2
CPHASE(γ) q3 q4
CPHASE(γ) q1 q4

Qubit (Q) usage statistics (Ops.)

Q	q0	q1	q2	q3	q4
Ops.	4	3	2	2	3

d Allocate physical qubits in order → *{q0, q1, q4, q2, q3}*

If any logical neighbor placed →
Choose physical neighboring qubit with maximum *(qubit connectivity strength/ cumulative distance from placed neighbors)*
Else →
Choose physical qubit with maximum *qubit connectivity strength*

Fig. 3. (a) Coupling graph of a 20-qubit quantum computer from IBM (ibmq_20_tokyo), (b) connectivity strength metrics of different qubits in ibmq_20_tokyo, (c) a toy QAOA cost Hamiltonian circuit with qubit activity profiles, (d) QAIM decision metric, and (e) qubit allocation and initial mapping for the toy example on ibmq_20_tokyo using QAIM.

during compilation. This is the rationale behind allocating such physical qubits to the heaviest logical qubits in the GreedyV⋆ heuristic [59]. However, it does not consider the expected activities in the neighboring qubits in the succeeding time-steps. If the neighboring physical qubits have sparse connectivity, the logical qubits mapped to them may need to move frequently to meet the coupling constraint. In such scenarios, the heaviest qubit may not need to move that often, however, its neighbors will move back and forth to meet connectivity constraints with other qubits, and thereby, increase the number of added SWAP operations.

To address this issue, we define connectivity strength of a qubit as the summation of its first and second neighboring qubits and create a profile of the available physical qubits based on this metric to assist in the mapping procedure. First neighbors of a qubit are the qubits connected directly to that qubit in the hardware coupling graph. Second neighbors are the unique first neighboring qubits of its first neighbors. For instance, qubit-0 in ibmq_20_tokyo (Figure 3(a)) has two first neighbors (qubit-1 and 5) and 5 second neighbors (qubit-2, 6, 7, 10, and 11). Therefore, the connectivity strength of qubit-0 is 7 (=2+5). The complete hardware profile of ibmq_20_tokyo is shown in Figure 3(b). This profiling can be done once for every hardware and the associated memory can be accessed during compilation. Note that for larger qubit architectures, we may include higher degree neighbors (i.e. third/fourth neighbors) in qubit connectivity strengths calculation.

Program Profiling: The program profile used in QAIM is similar to GreedyV⋆ [59]. For any input QAOA-circuit, we calculate the number of CPHASE operations per logical qubit to create the program profile. A demonstrative example is shown in Figure 3(c).

QAIM Procedure: Starting with the list of CPHASE operations in a QAOA-circuit, target hardware, and program profiling statistics, QAIM adopts following steps:

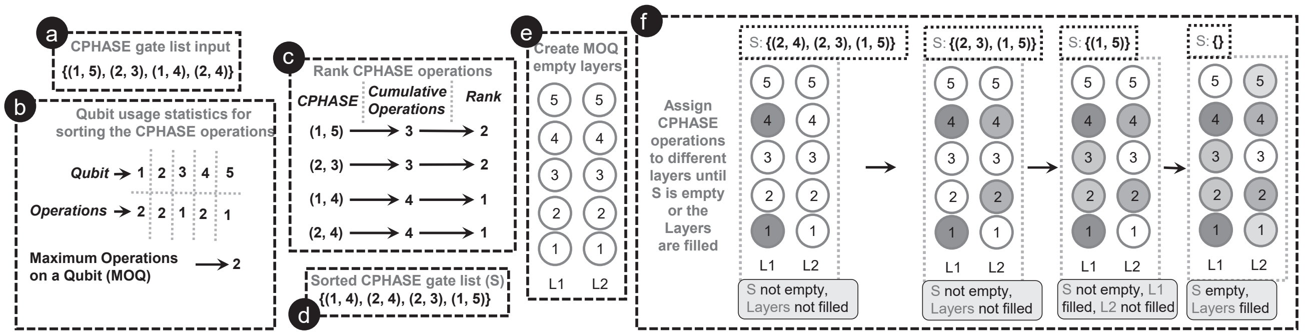
Step–1: The logical qubits are sorted (descending order) in a list based on the number of CPHASE operations per qubit. Physical qubits are allocated to the logical qubits in this order.

Step–2: The first logical qubit is assigned to the physical qubit with the highest connectivity strength. After the assignment, the qubit is removed from the list.

Step–3: For the next logical qubit in the list, we check if any of its logical neighbors (logical qubits that have multi-qubit operations between themselves are referred to as logical neighbors) has been already placed. If none of them has been placed, we pick the unallocated physical qubit with the highest connectivity strength for allocation. If some of its logical neighbors are placed, we find the unallocated physical neighbors of these placed qubits. We pick a qubit from these neighbors maximizing the cost metric - qubit connectivity strength/cumulative distance from the placed neighbors. Here, distance is the shortest path length between the unallocated qubit and a placed neighbor in the hardware coupling graph. Distances between physical qubits can be measured once (using Floyd-Warshall algorithm [57]) and accessed from memory during QAIM. After the assignment, we remove the logical qubit from the list.

Step–4: We repeat *Step–3* until the list is empty.

Example 1: The QAIM procedure for the QAOA-circuit in Figure 3(c) is shown in Figure 3(d) and (e). Logical qubit ‘q0’ has two physical qubit candidate - qubit-7 and qubit-12 (as both have same qubit connectivity strength of 18 each) and qubit-7 is picked randomly in the example (Figure 3(e)(i)). Logical qubit ‘q1’ has 6 possible candidates (as it is a logical neighbor of ‘q0’), all at a distance of 1 from ‘q0’ (Figure 3(e)(ii)). Physical qubit-12 has the highest connectivity



a
CPHASE gate list input
{(1, 5), (2, 3), (1, 4), (2, 4)}
b
Qubit usage statistics for sorting the CPHASE operations
Qubit → 1 2 3 4 5
Operations → 2 2 1 2 1
Maximum Operations on a Qubit (MOQ) → 2
c
Rank CPHASE operations
CPHASE
Cumulative Operations
Rank
(1, 5) → 3 → 2
(2, 3) → 3 → 2
(1, 4) → 4 → 1
(2, 4) → 4 → 1
d
Sorted CPHASE gate list (S)
{(1, 4), (2, 4), (2, 3), (1, 5)}
e
Create MOQ empty layers
L1
L2
f
Assign CPHASE operations to different layers until S is empty or the Layers are filled
S: {(2, 4), (2, 3), (1, 5)}
S not empty, Layers not filled
S: {(2, 3), (1, 5)}
S not empty, Layers not filled
S: {(1, 5)}
S not empty, L1 filled, L2 not filled
S: {}
S empty, Layers filled

Fig. 4. Instruction parallelization heuristics with a demonstrative example- (a) input list of CPHASE gates for a certain QAOA instance, (b) program qubit usage profile, (c) ranking CPHASE operations, (d) sorted CPHASE list based on their ranks, (e) defining empty layers, and (f) assigning CPHASE operations to different layers.

strength/cumulative distance from the placed neighbors (‘q0’). Therefore, qubit-12 is chosen for ‘q1’. The other qubits ‘q4’,‘q2’,‘q3’ are placed to qubit-8, qubit-13, and qubit-2 respectively in a similar fashion.

Two graph data structures are used (one for the hardware coupling graph and another for the program qubit connectivity graph) to search for all nearest physical/logical neighbors during the QAIM procedure using a linear search algorithm. The cost metric is chosen to prioritize unallocated physical qubits in qubit selection procedure which have higher connectivity strengths and are at a closer distance from the already placed logical neighbors of a logical qubit. Note that, the cost metric can be modified (e.g. *weigh* distances based on the number of multi-qubit operations between the logical qubit and its already placed neighbors) to apply QAIM effectively in any arbitrary quantum circuit mapping procedure.

B. Instruction Parallelization (IP)

After allocating physical qubits for the QAOA-circuit logical qubits using QAIM, we can go through the rest of the steps in the compilation procedure (SWAP insertion) following two orthogonal approaches: (i) compile the circuit with randomly ordered CPHASE gate sequences, or (ii) judiciously order the CPHASE gate sequences to extract better performance from the backend compiler. As mentioned before, paralleling instructions in the QAOA-circuit may help to reduce the circuit depth due to more concurrent gate operations and assist compilers (that partitions the circuit into layers of concurrently executable gates) by reducing the number of layers in the circuit [47], [48]. To maximize gate parallelization, we formulate the problem as a binary bin-packing problem and use the first-fit decreasing greedy heuristic for solution [62]. We refer to this approach as IP throughout the paper.

IP Procedure: IP also utilizes the program profile used in QAIM to rank the CPHASE operations (in descending order) in the QAOA-circuit based on a total number of operations on the control qubit and the target qubit. Operations with similar ranks are ordered randomly within themselves. The following steps are adopted to create the CPHASE layers:

$Step-1$: Create MOQ (the maximum number of operations in any qubit in the given QAOA circuit) empty layers of bins (each bin representing a qubit). This is the minimum limit for the number of layers that can be reached using IP in the best case scenario (see Example 2).

$Step-2$: Take the operation from the sorted CPHASE list with the highest rank. If both of the qubits in the CPHASE operation are empty in one of the layers, assign the CPHASE operation to that layer and mark the qubit bins as *occupied*. Remove the CPHASE operation from the list. If operations can not be assigned to any of the layers, move that operation to a separate list of unassigned CPHASE operations.

$Step-3$: Repeat $Step-2$ until the list is empty.

$Step-4$: If unassigned CPHASE operations list is not empty, repeat from $Step-1$ with this list.

Example 2: The input CPHASE gate list and its profiling statistics for a demonstrative example of IP are shown in Figure 4(a) and (b). MOQ for this example is 2 (as qubit 1 and 2 are involved in 2 CPHASEs each). As qubit-1 is involved in 2 CPHASE gates ((1,5), (1,4)), they need to be executed in *at least* 2 different layers (i.e., time-steps). Therefore, the minimum number of required layers for this circuit is (MOQ =) 2. The ranking of the CPHASE operations is shown in Figure 4(c). For example, cumulative operations for (2,3) is 3 as qubit-2 (control) is involved in ‘2’ CPHASE operations, (2,3) and (2,4), qubit-3 (target) is involved in ‘1’ CPHASE operation, (2,3). Thus, cumulative operations for (2,3) is ‘2 + 1 = 3’. The sorted CPHASE operations (based on their ranks) are shown in Figure 4(d) (similar ranked CPHASE operations are ordered randomly). The CPHASE operations in this sorted list are assigned one-by-one to the available qubit bins in the 2 layers (L1 and L2) shown in Figure 4(e). First, CPHASE between (1,4) is assigned to L1. Next, the CPHASE between (2,4) is assigned to L2 as bin 4 is already occupied in L1. CPHASE between (2,3) is assigned to L1 as both bins 2 and 3 are unoccupied in L1. The remaining CPHASE between (1,5) is assigned to L2. The steps in the assignment procedure are shown in Figure 4(f).

The input circuit to the compiler is created by ordering the CPHASE operations based on their layer assignments. For the example, the following CPHASE sequence is given as the input to the compiler (Figure 4(d)): (1,4), (2,3), (2,4), (1,5).

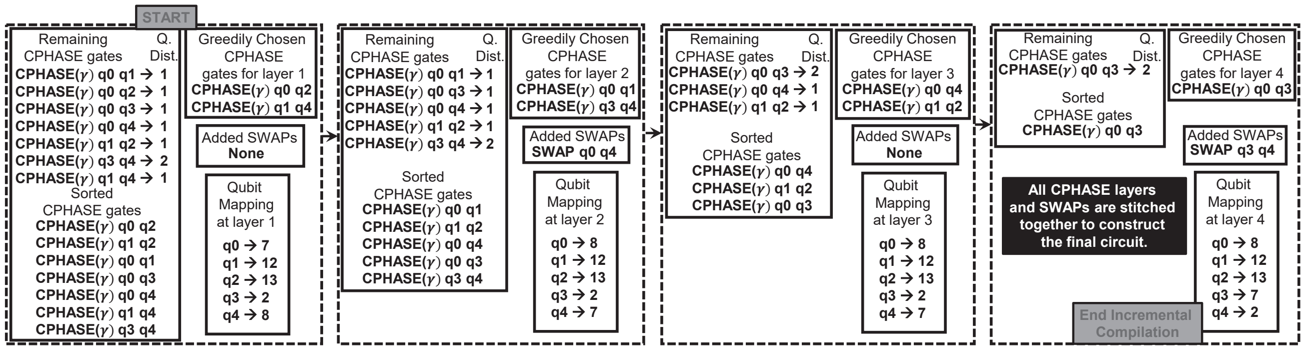


Fig. 5. Demonstration of the incremental compilation procedure for the initial mapping shown in Figure 3.

C. Incremental Compilation (IC)

Conventional compilers [47], [48] add SWAP operations before every layers to meet its coupling constraints. The logical-to-physical qubit mapping changes with every added SWAP operation. Due to these dynamic changes, some of the control and target qubits of the remaining CPHASE operations come closer to each other than the others. If these CPHASE operations are prioritized in the next layer formation, it may reduce the need for qubit movement between consecutive layers. In IP, all the layers are formed at the same time and the complete circuit is passed to the backend for compilation. To take advantage of these dynamic changes, we propose an incremental approach (i.e, IC) to compile QAOA circuits.

IC Procedure: In IC, the circuit layers are formed one-at-a-time and partial circuits are compiled with single CPHASE layers. The partial circuits are stitched at the end to construct the whole circuit. IC adopts the following steps:

$Step-1$: CPHASE operations are sorted in ascending order based on the distance between their control and target qubits for the initial logical-to-physical mapping. Distance between two qubits is defined as the shortest path length between them in the coupling graph. If multiple CPHASE operations have similar distances, they are ordered randomly. This sorted list is used to construct a single CPHASE layer using a greedy approach similar to the one used in IP (CPHASE operations are assigned to a single layer of bins). A partial circuit is compiled with this layer and the initial mapping. The final mapping after SWAP insertion is saved.

$Step-2$: The final mapping is set as the initial mapping. We repeat $Step-1$ until all the CPHASE operations are assigned to different layers and the corresponding partial circuits are compiled.

$Step-3$: We stitch all the compiled partial circuits.

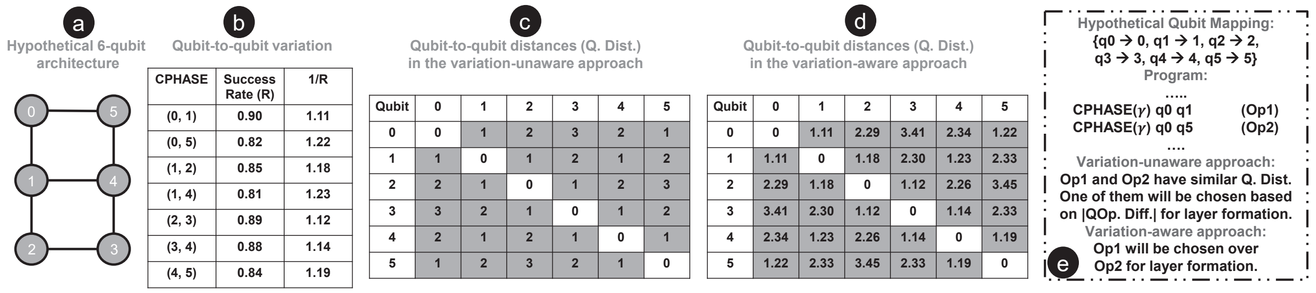
Example 3: A demonstrative example is shown in Figure 5 for the circuit and initial mapping illustrated in Figure 3. For the initial mapping, control and target qubits of all the CPHASE operations are at distance 1 except CPHASE(γ) q3 q4 (referred to as Q. Dist. in Figure 5). CPHASE operations are sorted (ascending order) based on these distances (CPHASE operations with similar Q. Dist. are ordered within themselves randomly). CPHASE(γ) q0 q2 and CPHASE(γ) q1 q4 are chosen using a greedy approach (similar to Figure 4(f)) to construct the first layer of the circuit. During compilation, the backend compiler does not add any SWAP operation for this layer. The second layer is formed in a similar fashion with CPHASE(γ) q0 q1 and CPHASE(γ) q3 q4. However, in this case, the compiler adds a SWAP operation to move q4 closer to q3. This changes the logical-to-physical qubit mapping which is used to re-calculate the Q. Dist. of the remaining CPHASE operations for the next layer. At the end of the procedure, 4 layers are formed and 2 SWAP operations are added to construct a hardware compliant circuit.

D. Variation-aware IC (VIC)

In IC, the distance between two physical qubits connected by an edge in the coupling graph is taken as one. Therefore, for the mapping assignment shown in Figure 6(e) for the hardware in Figure 6(a), the control and the target qubits of both CPHASE operations (Op1 and Op2) are at distance 1. One of them can be picked for the first layer formation. IC will choose one of them randomly. As mentioned before, variability exists within the available native gate operations in practical quantum computers. Hypothetical success rates of different CPHASE operations in the hardware in Figure 6(a) are shown in Figure 6(b). If we consider the reliability of the CPHASE operations, it is apparent that Op1 can be executed more reliably (with 0.90 success rate) than Op2 (success rate - 0.82) for the current mapping.

Taking Variability into Consideration: We can prioritize operations that may be executed with higher reliability during layer formation to maximize the success probability of the circuit. This will push operations that can not be executed reliably for the current mapping to later stages so that they may move towards more reliable paths due to dynamic changes in the logical-to-physical qubit mapping during the SWAP insertion procedure. To incorporate such strategy in IC, distance measurements between various qubits need to reflect the success probabilities of the intended operations. We term this approach as a variation-aware incremental compilation or VIC.

VIC Procedure: In IBM quantum computers, the RZ of the CPHASE gate is implemented virtually. Hence, the success rate of a CPHASE operation depends on the success probability of two consecutive CNOT operations. For a CNOT success rate of 0.9, the CPHASE success rate will be



Qubit-to-qubit variation

CPHASE	Success Rate (R)	1/R
(0, 1)	0.90	1.11
(0, 5)	0.82	1.22
(1, 2)	0.85	1.18
(1, 4)	0.81	1.23
(2, 3)	0.89	1.12
(3, 4)	0.88	1.14
(4, 5)	0.84	1.19

Qubit-to-qubit distances (Q. Dist.) in the variation-unaware approach

Qubit	0	1	2	3	4	5
0	0	1	2	3	2	1
1	1	0	1	2	1	2
2	2	1	0	1	2	3
3	3	2	1	0	1	2
4	2	1	2	1	0	1
5	1	2	3	2	1	0

Qubit-to-qubit distances (Q. Dist.) in the variation-aware approach

Qubit	0	1	2	3	4	5
0	0	1.11	2.29	3.41	2.34	1.22
1	1.11	0	1.18	2.30	1.23	2.33
2	2.29	1.18	0	1.12	2.26	3.45
3	3.41	2.30	1.12	0	1.14	2.33
4	2.34	1.23	2.26	1.14	0	1.19
5	1.22	2.33	3.45	2.33	1.19	0

Hypothetical Qubit Mapping:
{q0 → 0, q1 → 1, q2 → 2,
q3 → 3, q4 → 4, q5 → 5}
Program:
.....
CPHASE(γ) q0 q1 (Op1)
CPHASE(γ) q0 q5 (Op2)
....
Variation-unaware approach:
Op1 and Op2 have similar Q. Dist.
One of them will be chosen based on |QOp. Diff.| for layer formation.
Variation-aware approach:
Op1 will be chosen over Op2 for layer formation.

Fig. 6. (a) A hypothetical 6-qubit quantum computer coupling graph, (b) variations in the native 2-qubit operations reliability in the 6-qubit system, (c) distance measures between the physical qubits (using Floyd Warshall algorithm), (d) distance measures between qubits when edge weights are taken as the inverse of the 2-qubit operation reliability.

approximately 0.9*0.9 or 0.81. To reflect this success rate in distance measurements, we take the inverse of the CPHASE success rate as the distance between neighboring qubits. If a CPHASE success rate between qubit-x and qubit-y is 0.80, the associated edge weight in the coupling graph is taken as 1/0.80 or 1.25. These weighted edges are then used to calculate distances between qubits. The higher the success rate, the lower the distance. The rest of the procedure is similar to IC. The distances between qubits in the hypothetical 6-qubit hardware (using Floyd-Warshall algorithm) in the IC and the VIC procedures are shown in Figure 6(c) and (d) respectively.

V. Performance Evaluation

In this section, we present a quantitative performance analysis of the proposed methodologies individually and with respect to each other in terms of various circuit quality metrics using simulations and experiments on actual hardware. When layers are formed one-by-one in IC/IP, we can choose to populate the layers with gates to the fullest. However, the number of coupling constraints for a single layer scales with the number of gates packed in that layer. We investigate the impact of layer packing density on the compilation performance at the end of this section.

A. Evaluation Metrics

Circuit depth, gate-count, compilation time, and success probability: These metrics are the natural choices to compare various compilation strategies [47], [57]. The length of the critical path in a quantum circuit (the path with the highest number of gate operations) is defined as the circuit depth. The circuit depth is correlated to the circuit execution time on real hardware. A higher-depth circuit is more susceptible to decoherence errors. Gate-count is the total number of native gate operations in the compiled circuit. A lower gate-count generally translates to less accumulation of gate errors. Compilation time is the time taken by the compiler to generate the hardware compliant circuit. A faster compilation is desired for scalability. The circuit success probability metric is useful to quantify performance benefits with variation-aware compilation strategies.

Approximation Ratio Gap: The straight forward approach to judge the quality of the compiled QAOA-circuits would be running the quantum-classical optimization loop with the target hardware and compare the performance in terms of runtime, final solution, etc. It will require too many cloud-based access to the hardware which can be time-intensive with public devices. To circumvent this issue, we propose a new metric, Approximation Ratio Gap (ARG), to compare the performance of compiled QAOA-circuits on actual hardware.

We propose to find optimal QAOA-circuit parameters analytically [45] (or, for small problem size, running the algorithm in simulation) and use these values to compile the QAOA-circuit. We sample the output of the circuit (using a simulator such as qiskit [48]) a finite number of time to calculate the approximation ratio of the given cost function (r_0). Next, we run the circuit on the target hardware and calculate the approximation ratio (r_h) using the same number of samples. We define the percentage difference between these approximation ratios $\{100 * (r_0\text{-}r_h)/r_0\}$ as the approximation ratio gap or ARG. A lower ARG value is desired as it indicates a performance closer to the noiseless scenario.

B. Problem Sets and Compiler Backend

We use qiskit as the backend circuit compiler (run on an Intel Core i-7 processor at 3.41 GHz frequency). We choose 20-qubit ibmq_20_tokyo, 15-qubit ibmq_16_melbourne, and a hypothetical 36-qubit grid (6x6) architectures as the target hardware. Randomly chosen (up to 36-nodes) erdos-renyi random graphs (with varied edge probabilities) and regular graphs (with a varied number of edges/node) are used for the validation purpose inspired from recent works on QAOA [26], [27]. An edge probability of 0.5 means an edge between any two nodes in the graph is 50% likely to be included in a random sample. A randomly chosen n-regular graph has exactly n-edges/node. The exact node-to-node connectivity is different in two randomly chosen n-regular graphs. Graphs with higher edge probabilities or higher number of edges/node are dense graphs that require many CPHASE gates in their corresponding QAOA-MaxCut circuits.

C. QAIM vs. GreedyV* [59]/NAIVE

In this section, we compare QAIM with two other initial logical-to-physical qubit mapping procedures (GreedyV* and NAIVE) to quantify its impact on circuit compilation.

Varying Connectivity: We have varied the edge probability from 0.1 to 0.6 (for erdos-renyi random graphs) and

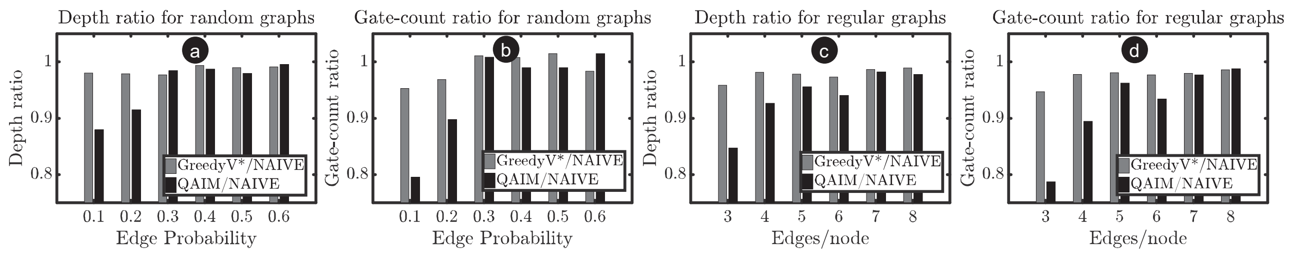


Fig. 7. Comparison among NAIVE, GreedyV⋆ [59], and QAIM in terms of (a) depth and, (b) gate-count ratios for erdos-renyi random graphs; (c) depth and, (d) gate-count for regular graphs (mean of 50 randomly chosen 20-node MaxCut-QAOA instances for each bars, ibmq_20_tokyo target hardware, qiskit backend, and a lower value is better).

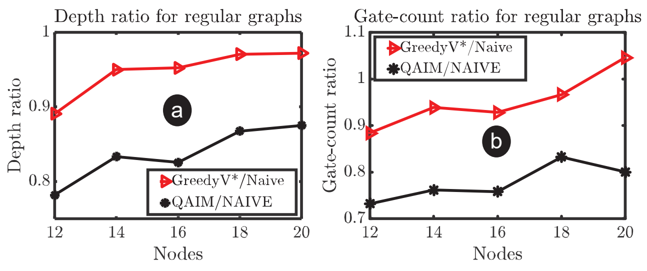


Fig. 8. Comparison among NAIVE, GreedyV⋆ [59], and QAIM in terms of (a) depth and, (b) gate-count ratios for 3-regular graphs with varying problem sizes (mean of 20 randomly chosen MaxCut-QAOA instances for each data points, ibmq_20_tokyo target hardware, qiskit backend, and a lower value is better).

edges/node from 3 to 8 (for regular graphs) and randomly picked 50 20-node graphs in each case. We have compiled the corresponding QAOA-MaxCut circuits (with randomly ordered CPHASE gates at ‘p’ = 1) by selecting the initial mapping using the NAIVE approach, GreedyV⋆ approach, and QAIM. We plot the ratio of the mean depth and gate-counts of the compiled circuits (50 instances per bar) between Greedy⋆ vs. NAIVE, and QAIM vs. NAIVE in Figure 7(a), (b), (c) and, (d). Compilation time was found to be similar.

Note that, for sparse graphs , QAIM performs considerably better than both the NAIVE and GreedyV⋆ approach. For example, for the erdos-renyi random graphs with an edge probability of 0.1, QAIM produces circuits with 12% and 10.3% shorter depth than NAIVE and GreedyV⋆ respectively. The gate-counts are also 20.5% and 16.5% smaller. For the regular graphs with 3 edges/node, QAIM circuit depths are 15.3% and 12.6% shorter than NAIVE and GreedyV⋆ approaches. The gate-counts are also 21.3% and 16.88% smaller. For dense graphs, all these three approaches perform similarly. The reason is the higher number of logical neighbors of a logical qubit for dense graphs than the number of physical neighbors of any hardware qubit. For any qubit placement, some of the logical neighbors will remain far from a logical qubit placed in a physical qubit. Therefore, we do not note any performance improvement with intelligent placements.

Varying Problem Size: Later we vary the problem size by picking 3-regular graphs with the number of nodes varying between 12 to 20 (20 randomly chosen graphs for each node-size) and compile the corresponding QAOA-MaxCut circuits using NAIVE, GreedyV⋆, and QAIM approaches. We plot the ratio of the mean depth and gate-counts of the compiled circuits against the node-size between Greedy⋆ vs. NAIVE, and QAIM vs. NAIVE in Figure 8(a) and (b), respectively.

For smaller problem sizes both GreedyV⋆ and QAIM performed better than NAIVE. The reason is the avoidance of physical qubits with smaller connectivity (e.g. qubit-0 and qubit-15 in ibmq_20_tokyo) in the selection procedure by both GreedyV⋆ and QAIM for smaller problem sizes which translates to a smaller number of SWAP operations during compilation. For the smallest problem size (12), QAIM generated circuits with 21.8% smaller depth and 26.8% smaller gate-counts than the NAIVE approach. Compared to GreedyV⋆, circuit depth and gate-counts were 12.2% and 17.2% smaller.

D. IC/IP vs. QAIM

In this section, we quantify benefits of IP and IC over QAIM-only compilation. We compiled the QAOA-MaxCut instances of randomly chosen erdos-renyi and regular graphs using QAIM (+random CPHASE sequence), IP (+QAIM), and IC (+QAIM). The ratio of the mean depth, gate-count, and compilation time (50 instances per bar) of IP vs. QAIM and IC vs. QAIM are shown in Figure 9. Note that, both IP and IC generated circuits with significantly smaller depths than the QAIM-only approach and the differences were more pronounced for dense graphs. For example, IC generated circuits with 39.3% less depth than QAIM for 3-regular graphs (Figure 9(d)) and it further went down to 68% for 8-regular graphs. On average, IC circuits had 13.2% smaller depth than IP (Figure 9(a) and (d)). A similar trend is observed in the gate-count as well for IC. On average, IC produced circuits with 16.67% and 16.6% smaller gate-counts than QAIM and IP respectively (Figure 9(b) and (e)). However, IP circuits had more or less similar gate-counts to QAIM.

The result is expected as a reduced number of layers translates to smaller depth and compilation time for both IP and IC. IC boosts the performance further by considering dynamic changes in the mapping. A smaller distance between control and target qubits of the CPHASE operations in the subsequent layer helped the compiler to come up with the nearest neighbor compliant mapping with less number of additional SWAP operations in IC. Consequently, the gate-count is found to be smaller in IC compared to IP. However, sorting the remaining CPHASE operations before every layer formation increased the compilation time for IC. On average,

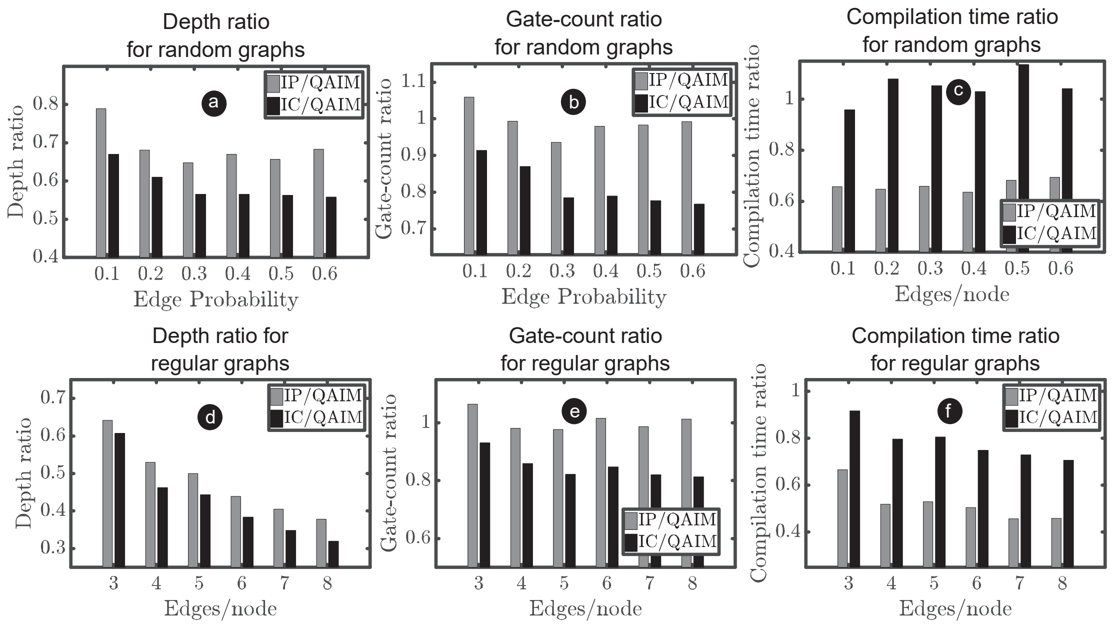


Fig. 9. Comparison among QAIM (+random CPHASE sequences), IP (+QAIM), and IC (+QAIM) in terms of (a) depth and, (b) gate-count, and (c) compilation time ratios - for erdos-renyi random graphs; (d) depth and, (e) gate-count, and (f) compilation time ratios - for regular graphs (mean of 50 randomly chosen 20-node MaxCut-QAOA instances for each bars, ibmq_20_tokyo is the target hardware, qiskit used as the backend compiler, and a lower value is better).

IP compilation time was 37% faster than IC (Figure 9(c) and (f)). Even in the worst cases, IC compilation time remained close to QAIM.

E. VIC vs. IC

VIC adds variation awareness to IC with the goal to increase the success probability of the compiled circuits. To quantify the efficacy of VIC over IC, we picked erdos-renyi random graphs (edge probability of 0.5) and regular graphs (6 edges/node) with varied problem sizes (13, 14, and 15 nodes/graph) and compiled the corresponding QAOA-MaxCut circuits using IC and VIC approach for the 15-qubit *melbourne* architecture (the CNOT error rates used in VIC is shown in Figure 10(a)). The ratio of the mean success probability of the circuits between VIC and IC is shown in Figure 10(b) and (c) (mean of 20 instances). VIC showed an 80% better success probability over IC on average for the random graphs as shown in Figure 10(b) (157% for node size 15). For the regular graphs (Figure 10(c)), we observed a 45.3% better success probability on average (72.2% for node-size 14).

Some of the nodes in a randomly picked erdos-renyi random graph can have a higher number of edges than the others. On the other hand, in the regular graphs, all the nodes have the same number of edges. In our experiments, the number of CPHASE operations in each layer (after parallelization) was higher for the regular graphs than the erdos-renyi ones. The reason behind this is - all the CPHASE operations involving a certain qubit need to be allocated in different layers. Therefore, if a node is connected to other nodes disproportionately, it increases the number of layers in the QAOA-circuit during parallelization. A lower number of CPHASE operations in a layer can be placed in the best pairs of qubits. However, if a layer is fully packed, most of the qubit pairs will be used. This is why the improvement in the success probability is found to be quite smaller for the regular graphs (where layers were heavily packed) compared to the erdos-renyi random graphs (where layers were sparsely packed).

F. Performance Summary

The mean value of the compiled circuit depths, gate-counts, and compilation times (of 600 QAOA 20-node graph MaxCut instances) for the NAIVE, QAIM (+random gate sequences), IP (+QAIM), IC (+QAIM), and VIC (+QAIM) approaches are shown in Figure 11(a) (normalized by the NAIVE approach values). For VIC, the CNOT error-rates for different qubit pairs are picked randomly from a normal distribution (μ = 1.0e-2, σ = 0.5e-2). On average, 45% reduction in the compilation time has been achieved using IP over the NAIVE approach. IC and VIC both provided similar performance in reducing depth (by $\approx$53%), gate-count (by $\approx$23%), and compilation time (by $\approx$15%). Note that, even though VIC and IC show similar performance, VIC offers higher success probability than IC as depicted in Figure 10(b) and (c).

G. Hardware Validation

We ran the quantum-classical optimization loop (L-BFGS-B classical optimizer used from SciPy library [63] with convergence limit set to e^{-6}) to find the QAOA-MaxCut optimal parameter values ('p' = 1) for 20 12-node erdos-renyi random graphs with edge probability of 0.5 and 20 regular graphs with 6 edges/node. We compiled the QAOA-circuits with these optimal parameter values with the proposed strategies for ibmq_16_melbourne architecture. We then sampled the output of every circuit 40960 times from hardware experiments and

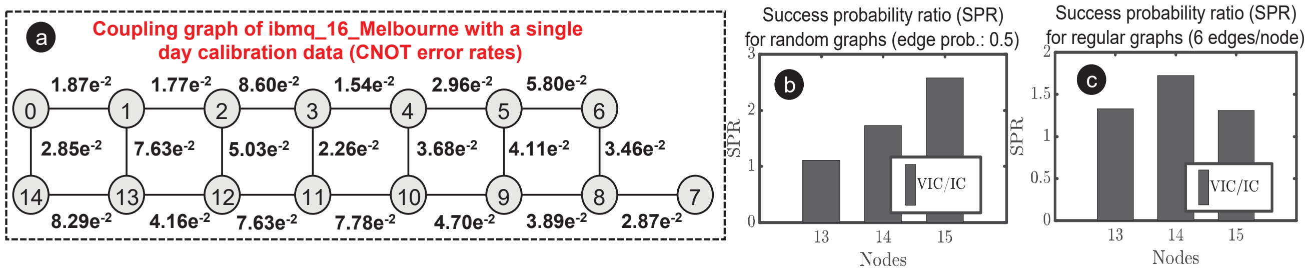


Fig. 10. (a) Coupling graph of ibmq_16_melbourne with reported CNOT operation error rates after calibration on 4/8/2020; comparison between IC(+QAIM) and VIC(+QAIM) compiled circuit success probabilities for (b) erdos-renyi random graphs, and (c) regular graphs (mean of 20 randomly chosen MaxCut-QAOA instances for each bars).

Method	Circuit Depth	Gate-count	Comp. time
NAIVE	1	1	1
QAIM	0.95	0.94	~1
IP	0.54	0.92	0.55
IC	0.47	0.77	0.85
VIC	0.48	0.77	0.86

Average performance over 600 20-node graphs (erdos-renyi + regular)

a

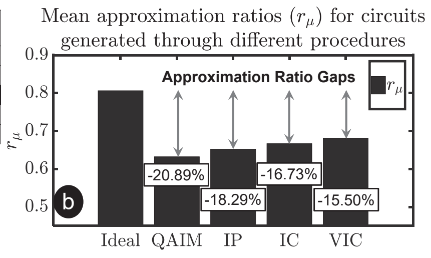


Fig. 11. (a) Performance (normalized by NAIVE) of QAIM, IP, IC, and VIC (ibmq_20_tokyo), (b) mean approximation ratios for various problems in hardware experiments on ibmq_16_melbourne (mean of 20 12-node erdos-renyi random graphs with edge probability of 0.5, and 20 12-node 6-regular graphs - for each bars).

used the results to calculate their corresponding ARG's (using the same set of physical qubits). VIC (+QAIM), IC (+QAIM), and IP (+QAIM) all provided smaller ARG values compared to the QAIM-only circuits (Figure 11(b)). On average, IC provided 8.53% smaller ARG values than IP. VIC provided 7.36% smaller ARG than IC. The results reflect the improvement in the compiled circuits in terms of depth, gate-count, and success probability.

H. Impact of Packing Density

In all the preceding compilation tasks, we tried to pack every layers to the fullest using the greedy approach. To investigate the impact of packing density, we have picked a 36-qubit grid qubit architecture and compiled QAOA-MaxCut circuits of 20 erdos-renyi random graphs (edge probability 0.5) and 20 15-regular graphs (36-nodes) with varying packing limits (maximum allowed number of CPHASE gates per layer is limited) using the IC (+QAIM) approach.

Impact on depth: The mean depth of the compiled circuits is shown in Figure 12(a) against the packing limit. Note that, the depth tends to decrease with an increase in the packing limit at first. However, when it went beyond 11, the qiskit compiler started to produce solutions with poor quality in terms of the circuit depth. In the previous examples, for the 20-node graphs, each layer had at most 10 operations. Therefore, the reported depth was most possibly the lowest that could have been achieved with the qiskit compiler. Compiling the circuits multiple times with different packing limits may help to generate circuits with desired circuit depth.

Impact on gate-count: Gate-count is found to increase with packing limit as shown in Figure 12(b). The increase is rather small for the packing limit between 3 to 11 i.e., 12.7% for the random graphs, and 16.18% for the regular graphs. However, the gate-count increases sharply beyond the packing limit of 11. If circuit depth is of a lesser concern (e.g. target hardware qubits have large enough coherence time), compiling circuits with the minimum packing limit may result in the best performance in terms of gate-counts.

Impact on compilation time: Packing more operations reduces the total number of circuit layers as well as the number of required SWAPs or permutation layers [47]. As the compiler has to satisfy coupling constraints of a fewer number of layers, it may translate to reduced compilation time. Figure 12(c) shows the mean compilation time for the random and regular graph QAOA-MaxCut instances where compilation time consistently reduced with increasing packing limit. If compilation time is of concern, packing the layers to the fullest may provide the best performance. However, this may not hold true for even larger architectures.

VI. Discussion

Comparative analysis: Since QAOA circuit compilation techniques is a new area, we are unable to perform extensive comparative analysis. A related work [46] took 70 sec for a QAOA circuit with only 8 qubits whereas we achieved reasonably good quality solutions for QAOA circuits with 36 qubits within 10s using qiskit compiler backend on a standard desktop machine (i.e., 7X faster for 4.5X bigger problems). For a 8-qubit cyclic hardware architecture, IC generated circuits with 8.51% smaller depth and 12.99% smaller gate-count on average for 50 instances of 8-node erdos-renyi random graphs with exactly 8 edges compared to [46] ('p'=1).

Usage of methodologies: QAIM is a generic qubit allocation and initial mapping procedure that can be incorporated in any QAOA-circuit compiler without adding any significant overhead. Among IP, IC, and VIC - the appropriate choice depends on the application requirements. For example, if reducing the circuit depth with faster compilation time is the prime requirement of the target application - IP will be the prudent choice. However, if reducing gate-count is equally significant, then IC will be a better choice. VIC can provide

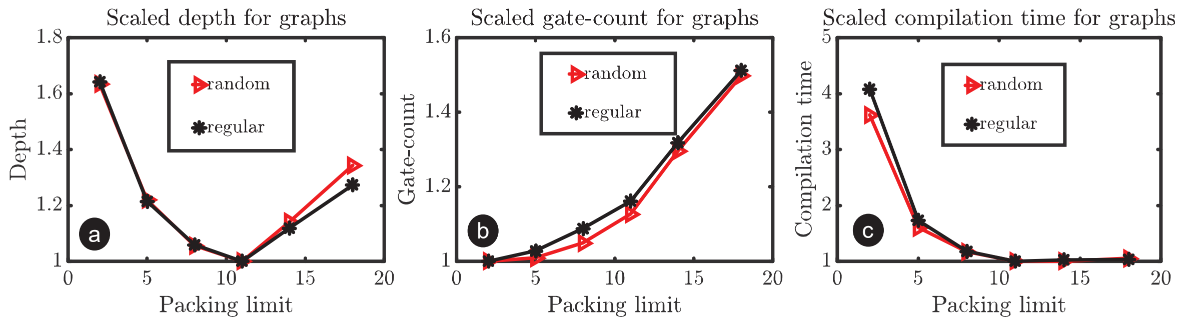


Fig. 12. (a) Depth (scaled by 283), (b) gate-count (scaled by 1428), and (c) compilation time (scaled by 9.48 seconds) against packing limit (maximum allowed CPHASE/layer) in IC(+QAIM) for a 36 qubit grid architecture (20 randomly chosen 36-node graphs for each data point, edge probability 0.5 for the erdos-renyi random graphs, 15 edges/node for the regular graphs, native gates similar to ibmq_16_melbourne, and qiskit used as the compiler backend).

additional benefits by enhancing the compiled circuit success probability when reliable calibration data is available.

Applicability beyond QAOA-MaxCut: The cost Hamiltonian of any arbitrary NP-hard problem can be formulated in the Ising format consisting of ZZ-interactions [24]. Each of these ZZ-interactions can be implemented with a CPHASE gate similar to the QAOA-MaxCut problem. Hence, the proposed compilation methodologies can be applied to other classes of QAOA instances. QAIM can be useful for arbitrary quantum circuits to a varied extent. IP, IC, and VIC can be useful for quantum circuits with large number of commuting operators such as the UCCSD ansatz for VQE [64], [65].

Crosstalk: Excessive gate parallelization may increase crosstalk-errors. Murali et al. [66] demonstrated that only a subset of couplings is highly crosstalk prone and proposed sequentialization of parallel operations on those couplings. For example, in IBM Poughkeepsie, they found only 5 couplings out of 221 to be highly crosstalk prone. A similar optimization step can be added in our flow to sequentialize only the high-crosstalk parallel operations post-compilation when the actual gate pulses are scheduled.

VII. Related Works

A number of compilation-work (not QAOA-specific) discusses QAOA as an example [46], [67], [68]. Machine-level gate pulse optimization has been explored [67] to reduce the execution time (latency) of a QAOA circuit. The compiler checks for gate-commutativity and combines multiple high-level gates to find a machine-level pulse sequence using the GRAPE algorithm. The new pulse will have a shorter duration than the high-level gates. A 2X–10X reduction in execution time has been reported for quantum systems up to 10 qubits. However, the time and memory requirement of the GRAPE algorithm grows exponentially with the problem size (i.e., # of qubits) leading to a high compilation latency.

The high compilation latency of GRAPE is addressed with a 'partial compilation' approach in [68]. It creates sub-circuit blocks with non-parametric (strict) and parametric (flexible) gates. Then, they pre-compute pulse sequences for each of the blocks. In future instances, the pre-computed pulses can be reused to execute a circuit and curtail the compilation time. However, quantum hardware suffers from the temporal variation [69]. Therefore, pre-computed pulses may not give optimal operation fidelity in a future instance.

A more traditional approach is explored in [46] to exploit gate commutation for aggressive QAOA circuit optimization by formulating the mapping problem as a planning problem and using off-the-shelf temporal planners for compilation. The problem with such compilation approach is: the solution is often as good as the plans. To this point, their plans did not incorporate (i) constraints to exploit initial mapping (exploited by QAIM), (ii) constraints for reordering the CPHASE operations involving a certain qubit (exploited by IC), and (iii) constraints to exploit qubit-to-qubit variation (exploited by VIC). Moreover, the solutions lacked scalability which has been tackled by heuristics in our solutions.

Two contemporary works use iterative compilation that re-compiles QAOA-circuits with updated gate-orders until the re-compiled circuits do not exhibit any performance improvement in terms of circuit depth, gate-count or success probability [70], [71]. The procedure is guided by a branch-and-bound optimization heuristic that incurs significant compilation time penalty due to repeated compilations (≈10X-600X reported with qiskit compiler backend for similar problem sets as used in this work).

VIII. Conclusion

We present four novel methodologies to compile QAOA circuits. Our methodologies can be integrated with any conventional compiler to improve the quality of the compiled QAOA-circuits. We validate performance improvement through experiments on real quantum devices from IBM. We demonstrate up to ≈53.0% reduction in circuit-depth, ≈23% reduction in gate-counts and ≈45% reduction in compilation time over a NAIVE approach. We also demonstrate up to ≈25.8% improvement in the approximation ratio gap through practical experiments of selected QAOA-MaxCut problems on ibmq_16_melbourne.

Acknowledgements

The work is supported in parts by National Science Foundation (NSF) (CNS- 1722557, CCF-1718474, DGE-1723687 and DGE-1821766) and seed grants from Penn State Institute for Computational and Data Sciences and Penn State Huck Institute of the Life Sciences.

References

[1] W. Knight, "Ibm raises the bar with a 50-qubit quantum computer, news," 2018.

[2] J. Kelly, "A preview of bristlecone, google's new quantum processor," *Google Research Blog*, vol. 5, 2018.

[3] J. Hsu, "Ces 2018: Intel's 49-qubit chip shoots for quantum supremacy," *IEEE Spectrum Tech Talk*, 2018.

[4] C. Rigetti, "The rigetti 128-qubit chip and what it means for quantum," *Medium*, 2018.

[5] E. Pednault, J. Gunnels, D. Maslov, and J. Gambetta, "On "quantum supremacy"," *IBM Research Blog*, vol. 21, 2019.

[6] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. S. L. Brandao, D. A. Buell, B. Burkett, Y. Chen, Z. Chen, B. Chiaro, R. Collins, W. Courtney, A. Dunsworth, E. Farhi, B. Foxen, A. Fowler, C. Gidney, M. Giustina, R. Graff, K. Guerin, S. Habegger, M. P. Harrigan, M. J. Hartmann, A. Ho, M. Hoffmann, T. Huang, T. S. Humble, S. V. Isakov, E. Jeffrey, Z. Jiang, D. Kafri, K. Kechedzhi, J. Kelly, P. V. Klimov, S. Knysh, A. Korotkov, F. Kostritsa, D. Landhuis, M. Lindmark, E. Lucero, D. Lyakh, S. Mandrà, J. R. McClean, M. McEwen, A. Megrant, X. Mi, K. Michielsen, M. Mohseni, J. Mutus, O. Naaman, M. Neeley, C. Neill, M. Y. Niu, E. Ostby, A. Petukhov, J. C. Platt, C. Quintana, E. G. Rieffel, P. Roushan, N. C. Rubin, D. Sank, K. J. Satzinger, V. Smelyanskiy, K. J. Sung, M. D. Trevithick, A. Vainsencher, B. Villalonga, T. White, Z. J. Yao, P. Yeh, A. Zalcman, H. Adam, and J. M. Martinis, "Quantum supremacy using a programmable superconducting processor," *Nature*, vol. 574, no. 7779, pp. 505–510, 2019.

[7] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, p. 021318, 2019.

[8] K. R. Brown, A. C. Wilson, Y. Colombe, C. Ospelkaus, A. M. Meier, E. Knill, D. Leibfried, and D. J. Wineland, "Single-qubit-gate error below 10- 4 in a trapped ion," *Physical Review A*, vol. 84, no. 3, p. 030303, 2011.

[9] J. Chow, J. M. Gambetta, L. Tornberg, J. Koch, L. S. Bishop, A. A. Houck, B. Johnson, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, "Randomized benchmarking and process tomography for gate errors in a solid-state qubit," *Physical review letters*, vol. 102, no. 9, p. 090502, 2009.

[10] J. Wenner, M. Neeley, R. C. Bialczak, M. Lenander, E. Lucero, A. D. O'Connell, D. Sank, H. Wang, M. Weides, A. N. Cleland, Cleland, and J. M. Martinis, "Wirebond crosstalk and cavity modes in large chip mounts for superconducting qubits," *Superconductor Science and Technology*, vol. 24, no. 6, p. 065001, 2011.

[11] G. Falci, A. D'arrigo, A. Mastellone, and E. Paladino, "Initial decoherence in solid state qubits," *Physical review letters*, vol. 94, no. 16, p. 167002, 2005.

[12] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Physical review A*, vol. 52, no. 4, p. R2493, 1995.

[13] A. M. Steane, "Error correcting codes in quantum theory," *Physical Review Letters*, vol. 77, no. 5, p. 793, 1996.

[14] A. Steane, "Multiple-particle interference and quantum error correction," *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, vol. 452, no. 1954, pp. 2551–2577, 1996.

[15] R. Laflamme, C. Miquel, J. P. Paz, and W. H. Zurek, "Perfect quantum error correcting code," *Physical Review Letters*, vol. 77, no. 1, p. 198, 1996.

[16] C. H. Bennett, D. P. DiVincenzo, J. A. Smolin, and W. K. Wootters, "Mixed-state entanglement and quantum error correction," *Physical Review A*, vol. 54, no. 5, p. 3824, 1996.

[17] A. Ekert and C. Macchiavello, "Error correction in quantum communication," *arXiv preprint quant-ph/9602022*, 1996.

[18] P. J. O'Malley, R. Babbush, I. D. Kivlichan, J. Romero, J. R. McClean, R. Barends, J. Kelly, P. Roushan, A. Tranter, and N. Ding, "Scalable quantum simulation of molecular energies," *Physical Review X*, vol. 6, no. 3, p. 031007, 2016.

[19] 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.

[20] D. Wang, O. Higgott, and S. Brierley, "Accelerated variational quantum eigensolver," *Physical review letters*, vol. 122, no. 14, p. 140504, 2019.

[21] R. M. Parrish, E. G. Hohenstein, P. L. McMahon, and T. J. Martínez, "Quantum computation of electronic transitions using a variational quantum eigensolver," *Physical review letters*, vol. 122, no. 23, p. 230401, 2019.

[22] J. M. Bowman, T. Carrington, and H.-D. Meyer, "Variational quantum approaches for computing vibrational energies of polyatomic molecules," *Molecular Physics*, vol. 106, no. 16-18, pp. 2145–2182, 2008.

[23] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas, "From the quantum approximate optimization algorithm to a quantum alternating operator ansatz," *Algorithms*, vol. 12, no. 2, p. 34, 2019.

[24] G. Nannicini, "Performance of hybrid quantum-classical variational heuristics for combinatorial optimization," *Physical Review E*, vol. 99, no. 1, p. 013304, 2019.

[25] Z. Wang, S. Hadfield, Z. Jiang, and E. G. Rieffel, "Quantum approximate optimization algorithm for maxcut: A fermionic view," *Physical Review A*, vol. 97, no. 2, p. 022304, 2018.

[26] G. E. Crooks, "Performance of the quantum approximate optimization algorithm on the maximum cut problem," *arXiv preprint arXiv:1811.08419*, 2018.

[27] L. Zhou, S.-T. Wang, S. Choi, H. Pichler, and M. D. Lukin, "Quantum approximate optimization algorithm: Performance, mechanism, and implementation on near-term devices," *Physical Review X*, vol. 10, no. 2, p. 021067, 2020.

[28] 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.

[29] S. Lloyd, M. Mohseni, and P. Rebentrost, "Quantum algorithms for supervised and unsupervised machine learning," *arXiv preprint arXiv:1307.0411*, 2013.

[30] V. Dunjko, J. M. Taylor, and H. J. Briegel, "Quantum-enhanced machine learning," *Physical review letters*, vol. 117, no. 13, p. 130501, 2016.

[31] P.-L. Dallaire-Demers and N. Killoran, "Quantum generative adversarial networks," *Physical Review A*, vol. 98, no. 1, p. 012324, 2018.

[32] S. Lloyd and C. Weedbrook, "Quantum generative adversarial learning," *Physical review letters*, vol. 121, no. 4, p. 040502, 2018.

[33] L. Hu, S.-H. Wu, W. Cai, Y. Ma, X. Mu, Y. Xu, H. Wang, Y. Song, D.-L. Deng, C.-L. Zou, and L. Sun, "Quantum generative adversarial learning in a superconducting quantum circuit," *Science advances*, vol. 5, no. 1, p. eaav2761, 2019.

[34] M. Schuld and N. Killoran, "Quantum machine learning in feature hilbert spaces," *Physical review letters*, vol. 122, no. 4, p. 040504, 2019.

[35] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm," *arXiv preprint arXiv:1411.4028*, 2014.

[36] E. Farhi, J. Goldstone, S. Gutmann, and H. Neven, "Quantum algorithms for fixed qubit architectures," *arXiv preprint arXiv:1703.06199*, 2017.

[37] E. Farhi, J. Goldstone, S. Gutmann, and L. Zhou, "The quantum approximate optimization algorithm and the sherrington-kirkpatrick model at infinite size," *arXiv preprint arXiv:1910.08187*, 2019.

[38] E. Farhi and A. W. Harrow, "Quantum supremacy through the quantum approximate optimization algorithm," *arXiv preprint arXiv:1602.07674*.

[39] M. Alam, A. Ash-Saki, and S. Ghosh, "Analysis of quantum approximate optimization algorithm under realistic noise in superconducting qubits," *arXiv preprint arXiv:1907.09631*, 2019.

[40] C. Xue, Z.-Y. Chen, Y.-C. Wu, and G.-P. Guo, "Effects of quantum noise on quantum approximate optimization algorithm," *arXiv preprint arXiv:1909.02196*, 2019.

[41] M. Alam, A. Ash-Saki, and S. Ghosh, "Design-space exploration of quantum approximate optimization algorithm under noise," in *2020 IEEE Custom Integrated Circuits Conference (CICC)*. IEEE, 2020, pp. 1–4.

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

[43] M. Alam, A. Ash-Saki, and S. Ghosh, "Accelerating quantum approximate optimization algorithm using machine learning," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2020, pp. 686–689.

[44] D. Wecker, M. B. Hastings, and M. Troyer, "Training a quantum optimizer," *Physical Review A*, vol. 94, no. 2, p. 022309, 2016.

[45] M. Streif and M. Leib, "Training the quantum approximate optimization algorithm without access to a quantum processing unit," *Quantum Science and Technology*, vol. 5, no. 3, p. 034008, 2020.

[46] D. Venturelli, M. Do, E. Rieffel, and J. Frank, “Compiling quantum circuits to realistic hardware architectures using temporal planners,” *Quantum Science and Technology*, vol. 3, no. 2, p. 025004, 2018.
[47] 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*, 2018.
[48] A. Cross, “The ibm q experience and qiskit open-source quantum computing software,” in *APS Meeting Abstracts*, 2018.
[49] G. G. Guerreschi and J. Park, “Gate scheduling for quantum algorithms,” *ArXiv e-prints*, 2017.
[50] 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.
[51] A. Ash-Saki, M. Alam, and S. Ghosh, “Qure: Qubit re-allocation in noisy intermediate-scale quantum computers,” in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 141.
[52] A. Botea, A. Kishimoto, and R. Marinescu, “On the complexity of quantum circuit compilation,” in *Eleventh Annual Symposium on Combinatorial Search*, 2018.
[53] 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.
[54] P. Murali, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, “Formal constraint-based compilation for noisy intermediate-scale quantum systems,” *Microprocessors and Microsystems*, vol. 66, pp. 102–112, 2019.
[55] R. Wille, L. Burgholzer, and A. Zulehner, “Mapping quantum circuits to ibm qx architectures using the minimal number of swap and h operations,” in *Proceedings of the 56th Annual Design Automation Conference 2019*. ACM, 2019, p. 142.
[56] D. Bhattacharjee, A. A. Saki, M. Alam, A. Chattopadhyay, and S. Ghosh, “Muqut: Multi-constraint quantum circuit mapping on nisq computers,” in *38th IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2019*. Institute of Electrical and Electronics Engineers Inc., 2019, p. 8942132.
[57] 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.
[58] A. Paler, “On the influence of initial qubit placement during nisq circuit compilation,” in *International Workshop on Quantum Technology and Optimization Problems*. Springer, 2019, pp. 207–217.
[59] 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.
[60] A. Zulehner, H. Bauer, and R. Wille, “Evaluating the flexibility of a* for mapping quantum circuits,” in *International Conference on Reversible Computation*. Springer, 2019, pp. 171–190.
[61] 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 *Proceedings of the 46th International Symposium on Computer Architecture*, 2019, pp. 527–540.
[62] G. Dósa and J. Sgall, “First fit bin packing: A tight analysis,” in *30th International Symposium on Theoretical Aspects of Computer Science (STACS 2013)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.
[63] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, and J. Bright, “Scipy 1.0: fundamental algorithms for scientific computing in python,” *Nature methods*, pp. 1–12, 2020.
[64] J. Romero, R. Babbush, J. R. McClean, C. Hempel, P. J. Love, and A. Aspuru-Guzik, “Strategies for quantum computing molecular energies using the unitary coupled cluster ansatz,” *Quantum Science and Technology*, vol. 4, no. 1, p. 014008, 2018.
[65] H. R. Grimsley, D. Claudino, S. E. Economou, E. Barnes, and N. J. Mayhall, “Is the trotterized uccsd ansatz chemically well-defined?” *Journal of Chemical Theory and Computation*, 2019.
[66] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, “Software mitigation of crosstalk on noisy intermediate-scale quantum computers,” in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1001–1016.
[67] 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.
[68] 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.
[69] M. Alam, A. Ash-Saki, and S. Ghosh, “Addressing temporal variations in qubit quality metrics for parameterized quantum circuits,” in *2019 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2019, pp. 1–6.
[70] M. Alam, A. Ash-Saki, and S. Ghosh, “An efficient circuit compilation flow for quantum approximate optimization algorithm,” in *2020 Design Automation Conference (DAC)*, 2020.
[71] ——, “Noise resilient compilation policies for quantum approximate optimization algorithm: Invited talk,” in *2020 International Conference on Computer-Aided Design (ICCAD)*, 2020.