Graph Neural Network Assisted Quantum Compilation for Qubit Allocation

Travis LeCompte
Louisiana State University
Baton Rouge, LA, USA
tlecom3@lsu.edu

Fang Qi
Tulane University
New Orleans, LA, USA
fqi2@tulane.edu

Xu Yuan
University of Louisiana at Lafayette
Lafayette, LA, USA
xu.yuan@louisiana.edu

Nian-Feng Tzeng
University of Louisiana at Lafayette
Lafayette, LA, USA
nianfeng.tzeng@louisiana.edu

M. Hassan Najafi
University of Louisiana at Lafayette
Lafayette, LA, USA
najafi@louisiana.edu

Lu Peng
Tulane University
New Orleans, LA, USA
lpeng3@tulane.edu

ABSTRACT

Quantum computers in the current noisy intermediate-scale quantum (NISQ) era face two major limitations - size and error vulnerability. Although quantum error correction (QEC) methods exist, they are not applicable at the current size of computers, requiring thousands of qubits, while NISQ systems have nearly one hundred at most. One common approach to improve reliability is to adjust the compilation process to create a more reliable final circuit, where the two most critical compilation decisions are the qubit allocation and qubit routing problems. We focus on solving the qubit allocation problem and identifying initial layouts that result in a reduction of error. To identify these layouts, we combine reinforcement learning with a graph neural network (GNN)-based Q-network to process the mesh topology of the quantum computer, known as the backend, and make mapping decisions, creating a Graph Neural Network Assisted Quantum Compilation (GNAQC) strategy. We train the architecture using a set of four backends and six circuits and find that GNAQC improves output fidelity by roughly 12.7% over pre-existing allocation methods.

CCS CONCEPTS

• **Computer systems organization** → **Architectures**; *Quantum Computing*.

KEYWORDS

Quantum Compilation, Graph Neural Networks, Qubit Allocation, Fidelity

ACM Reference Format:
Travis LeCompte, Fang Qi, Xu Yuan, Nian-Feng Tzeng, M. Hassan Najafi, and Lu Peng. 2023. Graph Neural Network Assisted Quantum Compilation for Qubit Allocation. In *Proceedings of the Great Lakes Symposium on VLSI 2023 (GLSVLSI '23), June 5–7, 2023, Knoxville, TN, USA.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3583781.3590300

GLSVLSI '23, June 5–7, 2023, Knoxville, TN, USA.

ACM ISBN 979-8-4007-0125-2/23/06…$15.00
https://doi.org/10.1145/3583781.3590300

1 INTRODUCTION

Modern quantum computers are classified as noisy intermediate-scale quantum (NISQ) devices. These NISQ devices are named as such due to their limitations on both the number of qubits (or quantum bits) available and the reliability of these qubits and their operations. Most NISQ devices contain from ten to one hundred noisy qubits, though many systems are smaller on the smaller end of this range, containing only 5-32 qubits. Due to the relatively high error rates in quantum computers, many computations are unlikely to complete without some error. Researchers have put forth great effort to both make the algorithms resilient and reduce the vulnerability of the physical machines.

Most approaches increase the reliability of quantum circuits during execution rather than completely removing errors. It is common to modify the circuit during compilation to choose more reliable configurations when applying the circuit to a physical backend. Using different qubits, physical connections, and operations, can significantly impact the outcome of the circuit, as each configuration may exhibit a very different error profile. These error rates can vary due to different environmental conditions. Since there are many possibilities when applying a circuit to a backend, it is computationally difficult to identify the best possible configuration. However, many pursuits have found success with a variety of methods [1, 4–6].

Our work aims to improve upon existing qubit allocation approaches, as our investigation shows there are considerable performance improvements to be made. To solve the qubit allocation problem, we incorporate graph neural networks (GNNs) to aid in processing the inherent graph representation of the superconducting quantum backend, creating a Graph Neural Network Assisted Compilation strategy (GNAQC). We choose to use GNNs due to their intrinsic graph processing capabilities. We combine this GNN processing of the backend with feedforward networks for processing input circuits to create a total system for providing suggested layouts as solutions to the qubit allocation problem. We implement GNAQC using Qiskit and TensorFlow and evaluate its performance on two different IBM backend configurations and six different quantum circuits. We find that GNAQC generally outperforms the other layout methods with some variation across the backends and circuits, increasing relative fidelity by approximately 12.7%. We also find that GNAQC is more consistent at choosing better layouts, providing a more reliable allocation method.

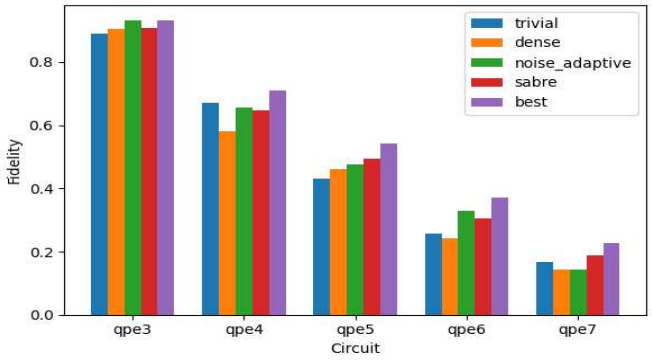

Figure 1: Fidelity of Qiskit's four qubit allocation methods on the 3-to-7-qubit Quantum Phase Estimation (QPE) algorithm after execution on *ibm_nairobi*.

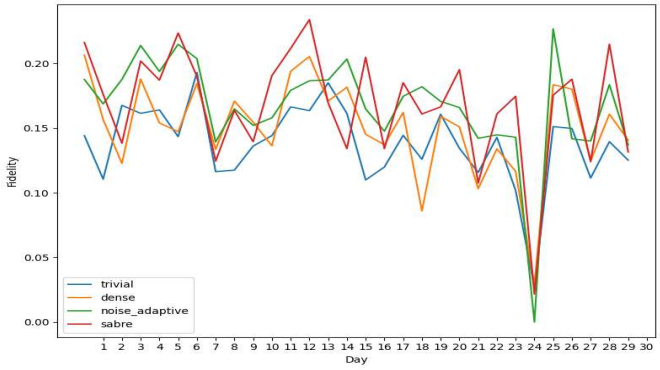

Figure 2: Fidelity of 7-qubit QPE when compiling with Qiskit's four allocation methods across one month of backend configurations for *ibm_nairobi*.

Our contributions can be summarized as follows:

- We demonstrate the limitations of existing layout methods.
- We provide GNAQC, a new solution to the qubit allocation problem built on GNNs with feedforward networks.
- We test GNAQC on two physical backends of 7 and 27 qubits using six different benchmarks, finding that GNAQC can consistently provide better or comparable initial layouts to pre-existing methods.
- We demonstrate that GNAQC reduces the error of quantum circuits by providing more reliable layouts, yielding a 12.7% relative increase in fidelity.

2 MOTIVATION

We utilize IBM's Qiskit API [3] to investigate the current performance of qubit allocation methods. Qiskit natively contains four different allocation methods: trivial, dense, noise-adaptive [6], and sabre [5]. The four methods address the mapping problem using very different approaches. Specifically, the trivial layout simply maps the virtual qubits $(q_1, q_2 \ldots q_n)$, in order, to the physical qubits $(0, 1 \ldots N)$. The dense layout identifies highly connected sub-graphs of the mesh and places qubits in these areas. The noise-adaptive layout is the first to rely on the most recent backend configuration data, aiming to utilize the most reliable two-qubit connections available. The sabre method utilizes an iterative process to fully route the circuit to find the final layout, then reversing the circuit using the previous final layout as a proposed initial layout. This process is repeated to minimize the number of required operations.

We tested the four layout methods on IBM's 7-qubit *ibm_nairobi* backend using 3-qubit to 7-qubit quantum phase estimation (QPE) circuits. We only test up to a maximum of seven qubits as access to larger machines is limited. To evaluate their effects, we first run a trial of each circuit using Qiskit's simulator with no error involved to attain a flawless theoretical outcome that we use as the ground truth for every circuit. While the measurements of the qubits are probabilistic in nature, we execute all trials with 10000 shots to minimize the random influence. We then execute the six test circuits on the *ibm_nairobi* backend using each layout method during compilation, again using 10000 shots. All other compilation settings were kept default, including the routing methods. Next, we compared the resulting output distribution with the ground truth distribution by computing the fidelity between them. The fidelity acts as a similarity metric between the ground-truth state and the real output state. A higher fidelity (bound [0,1]) indicates a higher similarity between states.

For ease of computing fidelity F, we rely on the Hellinger distance formula described below:

$$F = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{N} (\sqrt{p_i^{GT}} - \sqrt{p_i^{T}})^2} \tag{1}$$

Here, N is the total number of observed outputs, p_i^{GT} is the probability of output i for the ground truth distribution, and p_i^T is the probability of output i for the test distribution. The results of these trials are shown in Fig. 1. In order to provide a metric for comparison, we decided to execute and evaluate the error of every possible layout and examine the effects. Note that this is only feasible as we are working with a small number of qubits, as the total number of layouts grows extremely quickly with an increase in qubits. We display the exact maximum fidelity achieved as *best* in Fig. 1. As shown, we find that no method is consistently close to optimal. When looking at all 5 circuit sizes, we see situations where the trivial, noise-adaptive, and sabre methods are the best of the four options.

To provide more insight into the differences between layout methods, we evaluated every layout on one month of daily calibration data for *ibm_nairobi*, as shown in Fig. 2. This allows us to see how frequently each allocation method performs best or worst. As expected, the best allocation method is frequently either the noise-adaptive or sabre layout methods. However, the accuracy improvements are inconsistent, and we frequently see changes between which is best over time. Occasionally, they are even outperformed by the dense or trivial layouts. In total, these experiments demonstrate two main points: **1)** the choice of initial layout can have a considerable impact on circuit fidelity, and **2)** existing methods are inconsistent at choosing effective layouts. There is room

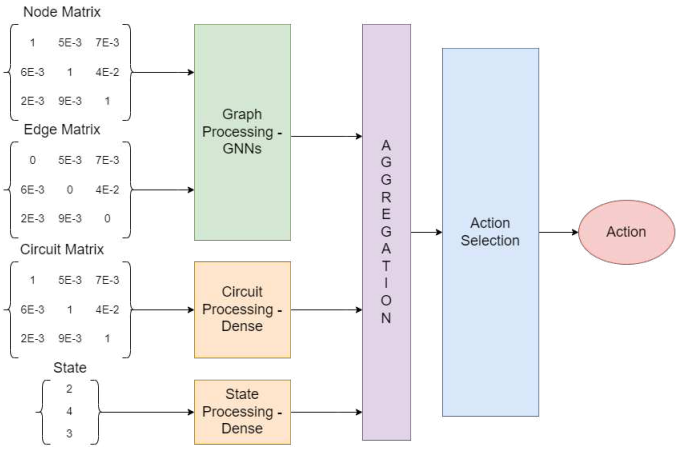

Figure 3: High-level architecture of the GNAQC Q-network.

for improvement when selecting layouts to reduce vulnerability to error.

3 ARCHITECTURE AND DATA REPRESENTATION

To improve the performance of current layout methods, we look to use graph neural networks as the quantum backends are naturally represented in a graph form. We combine GNNs with additional feedforward layers to predict optimal layouts given the backend error properties and an input circuit. The following subsections discuss our network architecture in detail, including two main areas: the backend graph input representation and processing and the circuit input representation and processing. Details of the current state vector and output actions are found in Section 4. The overall architecture is shown in Fig. 3. The dense unit in the circuit processing region contain two dense layers of $N * F$ nodes where F is the number of node features. Similarly, the GNN layers are each composed on N^2 nodes. The dense layer for processing the input state is composed of N nodes. The dense section is composed of two layers of $N^2 + N * (F + 1)$ and N^2 nodes respectively, giving our total N^2 outputs for all possible placement actions.

3.1 Backend Representation and Processing

To prepare the backend for the GNN layers, we must construct both a node and edge matrix (which replaces the adjacency matrix in standard GNN). For the node matrix X, we collect several properties from each node and arrange the matrix where each row holds the properties of an individual node. The total set of properties that we collect is found in Table 1, totaling 14 different error rates and gate lengths. The final size of the node matrix is thus $N \times 14$, where N is the total number of physical qubits in the backend. We access these properties using Qiskit's IBMQ provider API. The set of single-qubit gate data we collect varies depending on the basis set of gates, though all of the backends we test contain the same basis set. We then normalize the matrix by row to accelerate convergence.

The edge matrix E takes the same form as a weighted adjacency matrix, where $E_{i,j}$ equals the CNOT error between qubit i and qubit j. Although it is not required that the CNOT error be symmetrical on all hardware implementations, we found that, for the backends we tested, the error rates were always symmetrical. We then normalize the edge matrix in a doubly-stochastic manner, following the design of [2] to ensure that both the rows and columns of E sum to 1 to again aid in convergence. Given that the edge matrix is a variation of the adjacency matrix, its final dimensions are $N \times N$.

Table 1: List of all node features collected from physical backends.

Feature	Description
E_{ID}	Identity gate error
L_{ID}	Identity gate length
E_{RZ}	RZ gate error
L_{RZ}	RZ gate length
E_{SX}	SX gate error
L_{SX}	SX gate length
E_X	X gate error
L_X	X gate length
T_1	Relaxation time
T_2	Dephasing time
F	Qubit Frequency
E_M	Measurement error
P_{01}	Probability measure 0 as 1
P_{10}	Probability measure 1 as 0

These two matrices are then fed into the network, specifically into two stacked GNN layers. Together these layers generate a new representation of the graph, which is then passed through a flattening layer to reshape the representation in preparation for concatenation with the processed circuit matrix. The GNN layers perform an edge-aware version of the forward computation described in Equation 2:

$$X^{(k)} = \sigma(\tilde{E}X^{(k-1)}W) \tag{2}$$

3.2 Circuit Representation and Processing

To provide the circuit information to the prediction network, we first prepare a matrix containing hand-picked features to capture the behavior of the circuit. After testing a variety of different combinations, our final decision of circuit features is shown in Table 2. We believe that capturing the single-qubit operations each qubit, the measurement status of each qubit, the count of CNOT operations and a set of CNOT partners for each qubit is sufficient for most basic circuits. We provide results in Section 6 that show the influence of different numbers of CNOT partners from each qubit, though by default we only use the first CNOT partner. Currently, this representation would likely fail to represent more complex circuits involving mid-execution measurement and reset, though these operations do not occur in any of our test circuits.

It is important to note that we do not use the original logical circuit to prepare these representations, as they may change through the steps of the compilation process before preparing a layout. The most important changes that can occur are decomposing multi-qubit operations and sub-circuits and mapping to basis gates, as

Table 2: List of all circuit features collected from test circuits. These features are collected for every qubit in the circuit. * Note that the number of CNOT operations we collect for look-ahead is variable. We test look-ahead counts from 1 to 5. Default look-ahead is 1.

Feature	Description
N_{ID}	Number of involved identity operations
N_{RZ}	Number of involved RZ operations
N_{SX}	Number of involved SX operations
N_X	Number of involved X operations
N_{CNOT}	Number of involved CNOT operations
M	Measurement status
T_i	ith target qubit for CNOT operations*

these can greatly change the view of which operations the circuit performs. Instead, we acquire the intermediate circuit during the compilation process at the point where qubit mapping normally occurs, after these other operations. This allows us to represent the circuit as accurately as possible for choosing a layout.

4 REINFORCEMENT LEARNING SETUP

4.1 Actions

When mapping the qubits, the available actions are simply one placement action for each ($logical$, $physical$) qubit pair. This placement action represents assigning the logical qubit to the associated physical qubit for the initial layout. To account for circuits with fewer logical qubits than the available physical qubits, we extend the logical qubits with ancilla qubits to equal the number of physical qubits. In total, this results in N^2_{phys} actions. This also characterizes the total number of outcomes resulting from the final dense layer in Fig. 3. Following an Epsilon-Greedy policy, with $\epsilon = 0.05$, we select the action with the maximum predicted value with probability $1 - \epsilon$ and a random action with probability ϵ to drive our training decisions.

4.2 Environment

To define the environment, we first represent the state of the physical hardware and the circuit as described in Section 3 These inputs are then complemented with a vector containing the current mapping of qubits, specifically mapping from $physical \rightarrow logical$ qubits. This captures the current state of the layout, specifically a snapshot of the current layout at a given time during compilation. The vector is initialized to all zero values, indicating no qubits have been placed, and gradually fills with non-zero values as placement actions are taken each iteration. Together, the matrices and state vector capture the problem as well as the current intermediate solution.

4.3 Rewards

When providing rewards, we first consider the placement of ancilla qubits. As these qubits are not important to the execution of the circuit, placing the qubits provides no reward. Similarly, when attempting to place a qubit that has already been assigned to a physical qubit, no reward is given. In contrast, placing a previously

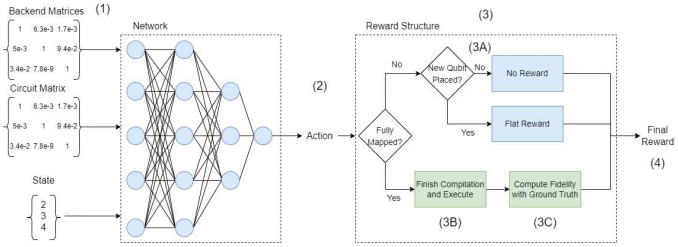

Figure 4: GNAQC training process. (1) Data input. (2) Chosen action. (3) Reward breakdown. (4) Output reward for training.

unplaced logical qubit provides a constant reward to encourage prioritization.

The most interesting case is the reward given when completing the mapping of all logical qubits. In this case, we first execute the circuit on the simulator using the error profile of the backend. We choose to use the simulator as we do not have dedicated access to a physical backend for training. We then compare the output distribution to an error free output distribution that acts as our ground truth. This error-free distribution is obtained by executing the circuit on a simulator with no error simulation. This is effectively a theoretically perfect outcome for the circuit. To provide a tangible value, we compute the Hellinger fidelity between the two distributions, as shown in Equation 1. The more similar the output distributions are, the closer this value approaches 1. This is then scaled by 100 and provided as the final reward. This guides GNAQC to target configurations that are most similar to the error-free distribution.

4.4 Training

The full training process is shown in Fig. 4. First, the processed edge, node, and circuit matrices are fed to the prediction network in step (1). The network outputs a suggested action to take, namely a qubit placement, in step (2). The reward for this action is calculated in step (3), where the value for the reward depends on the result of the action. If the action results in a fully-mapped circuit, we finish compilation (routing and final optimization) and simulate the final circuit in step (3B) using Qiskit's Aer simulator. The simulator is prepared with a noise model built on the error properties of the collected backend under test. In step (3B), we collect the output counts from the simulator and compute the fidelity with the ground truth distribution. If the action did not result in a fully-mapped circuit, we instead give either a reward of 0 if the qubit was already placed or 10 if the qubit is newly placed. We use this reward for the update process following the typical Q-learning update rule in step (4).

5 DATA COLLECTION AND EXPERIMENTATION

Throughout our experimentation, we rely on a set of various test circuits at different sizes executed on several different physical backends. We focus on a set of six different circuits as mentioned previously in Section 2 the Deutsch-Jozsa (DJ) algorithm, the Bernstein-Vazirani (BV) algorithm, Simon's algorithm, the quantum Fourier transform (QFT), the quantum phase estimation (QPE) algorithm,

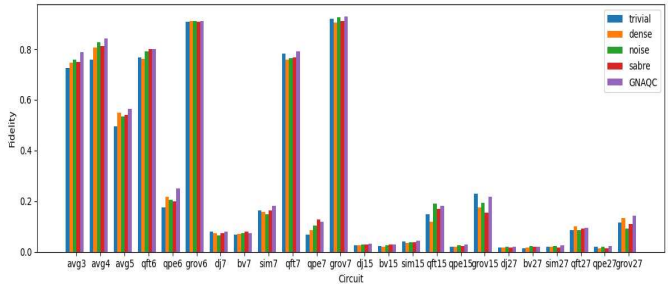

Figure 5: Fidelity for all 5 layout methods, on all six benchmarks, at various qubit sizes.

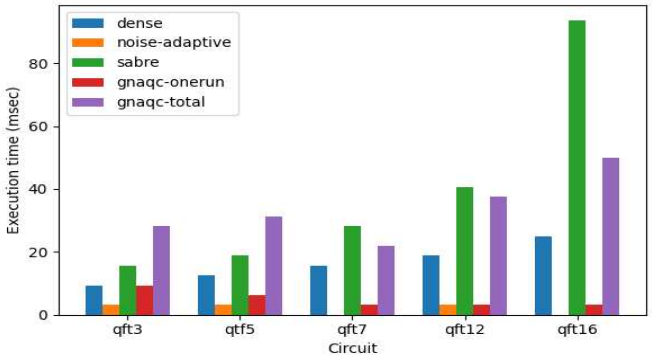

Figure 6: Compilation time for each of the layout methods. Trivial not included due to 0 execution time.

and Grover's search algorithm. We prepare these circuits using 3 to 27 qubits. We believe that two qubits are simply too trivial, and we are limited to backends with 7 or 27 maximum qubits.

For the backends, we collected calibrations for *ibm_nairobi*, a 7-qubit backend, and *ibm_algiers*, a 27-qubit backend. We selected these two as a representative sample of the available 7-qubit and 27-qubit machines that were available for experimentation. We chose these sizes because they provide insight into both larger and smaller scale quantum computers. We specifically collected the archived daily calibrations from January 1st, 2022 through the end of May 2022. The backends vary in topology, with *ibm_nairobi* having an I shape and *ibm_algiers* having an adjusted square shape.

6 RESULTS

To evaluate the general performance of GNAQC, we predict layouts for each test circuit using the most recent calibrations for the target quantum machine. The circuits are then executed on the physical backends to evaluate how GNAQC performs relative to the four pre-existing methods within Qiskit. Our main metric is the fidelity of the output compared to an error-free execution. These results are shown in Fig. 5.

It can be observed that GNAQC generally outperforms the pre-existing layouts for each benchmark at different algorithm sizes. The GNAQC layouts consistently perform better on smaller algorithms like DJ, BV, and Simon. We believe this is due to the length of each algorithm, where shorter circuits are more influenced by the initial position of qubits, while longer algorithms are more influenced by the routing methods. There is reduced improvement on the larger algorithms. In the worse cases, GNAQC performs comparably to the best alternative layout method. On average, we see a relative improvement in fidelity of approximately 12.7%.

When examining by circuit size, the largest improvement is found in smaller circuit sizes. We see the most variation in behavior among the layouts at 3-5 qubits, with more consistent performance among all five methods at larger sizes. We identify two main reasons for this variation in behavior. First, as the depth of the circuit increases due to the increased number of qubits, the fidelity decreases drastically. This results in less room for the layouts to vary as the fidelity is simply so low. Second, we believe that this has to do with the percentage of qubits used on the backend and the topology of the machine itself. When using all of the qubits on the machine, more SWAPs will likely need to be added to allow the circuit to function regardless of the initial position of qubits. At smaller sizes, the number of added SWAPs may vary greatly based on the initial position of qubits.

To evaluate the cost of GNAQC, we measure the execution time of each layout method for QFT as shown in Fig. 6. GNAQC-onerun is the time to perform a single inference stage, while GNAQC-total is the total time to place all qubits (one inference iteration per qubit). It is clear that GNAQC performs in-line with the other methods, and scales better to larger qubit sizes than the sabre method.

7 CONCLUSION

We have proposed GNAQC, a new GNN-based neural network architecture for improving the reliability of superconducting quantum circuits by identifying more resilient layouts. We compare the proposed layouts with the pre-existing layout methods contained within Qiskit and find a mean 12.7% relative increase in fidelity across both backends configurations with six different circuits. In the future, we believe we could achieve even greater results by expanding the work to include a routing method using recurrent GNNs or experimenting with different feature representations.

REFERENCES

[1] Will Finigan, Michael Cubeddu, Thomas Lively, Johannes Flick, and Prineha Narang. 2018. Qubit allocation for noisy intermediate-scale quantum computers. *arXiv preprint arXiv:1810.08291* (2018).
[2] Liyu Gong and Qiang Cheng. 2019. Exploiting edge features for graph neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9211–9219.
[3] IBM. 2016. Open-Source Quantum Development. Retrieved Retrieved on 12-16-2022 from https://qiskit.org/
[4] Travis LeCompte, Fang Qi, and Lu Peng. 2020. Robust Cache-Aware Quantum Processor Layout. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 276–287.
[5] Gushu Li, Yufei Ding, and Yuan Xie. 2019. Tackling the qubit mapping problem for NISQ-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. 1001–1014.
[6] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. 2019. 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*. 1015–1029.