Towards Effective Qubit Mapping for Noisy Intermediate-Scale Quantum Devices

by

Chi Zhang

Bachelor of Engineering

Xidian University, China

2014

Submitted to the Graduate Faculty of

the School of Computing and Information in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2019

UNIVERSITY OF PITTSBURGH

DEPARTMENT OF COMPUTER SCIENCE

This proposal was presented

by

Chi Zhang

Towards Effective Qubit Mapping for Noisy Intermediate-Scale Quantum Devices

Chi Zhang, PhD

University of Pittsburgh, 2019

Today, quantum devices are comprised of qubits ranging from dozens to hundreds in number. There is a class of problems that cannot be solved using classical computing. Whereas, it's expected that these tasks can be solved by the current and the future quantum devices achieving supremacy over classical computing. However, the current state-of-the-art quantum computing is riddled with qubit mapping problem. CNOT gates, that take two logical qubits as input, can only be mapped to a physical qubit pair, adjacent to each other. But the physical layout of the current quantum hardware is normally irregular. This renders executing quantum programs directly on current quantum devices infeasible. Moreover, the available quantum error correction is not adequate. This leads the state of quantum computing to the Noisy Intermediate-Scale Quantum (NISQ) era, yielding the need of smaller sized quantum circuits to be mapped onto the hardware. This requires thorough investigations into designing solutions to achieve effective mapping strategy between logical and physical qubits.

In this proposal, we are looking into three strategies to partially address the qubit mapping problem namely – (i) a depth-aware SWAP insertion scheme, (ii) GPU-assisted acceleration for mapping generation, and (iii) mapping for co-running quantum programs.

Keywords: Quantum Computing, Emerging languages and compilers.

Table of Contents

1.0 Introduction 1

1.1 Introduction 1

2.0 Background and Related Work 4

2.1 Quantum Computing Basics 4

2.1.1 Qubit 4

2.1.2 Quantum Gates 4

2.1.3 Quantum Circuit 5

2.2 Noisy Intermediate-scale Quantum Hardware 5

2.3 Qubit Mapping and Depth-Awareness 6

3.0 A Depth-Aware SWAP Insertion Scheme (Preliminary) 9

3.1 Proposed Solution 9

3.1.1 Metric 9

3.1.2 Framework Design 10

3.1.3 Circuit Mapping Searcher 11

3.1.4 Optimizations 12

3.1.4.1 Expand More Nodes 12

3.1.4.2 Deeper Search 13

3.2 Evaluation 14

3.2.1 Depth Reduction 16

3.2.2 Gates Count Changes 16

3.2.3 Trade-off between Gate Count and Depth 16

4.0 Parallel Acceleration for Mapping Generation (Future) 19

4.1 Motivation 19

4.2 Problem Statement and Proposed Work 19

5.0 Mapping for Co-Running Quantum Programs (Future) 22

5.1 Motivation 22

5.2 Problem Statement and Proposed Work 22
5.2.1 Sub-graph allocation 22
5.2.2 Swap insertions with/across sub-graphs 23
5.2.3 Proposed Work 24
6.0 Conclusions 25
Bibliography 26

1.0 Introduction

1.1 Introduction

Quantum computing has exhibited its theoretical advantage over classical computing by showing impressive speedup on applications including large integer factoring [16], database search [4], and quantum simulation [12]. It is considered to be a new computational model that may have a subversive impact on the future, and has attracted major interests of a large number of researchers and companies.

With the advent of advanced manufacturing technology, the industry is able to build small-scale quantum computers – Noisy Intermediate-Scale Quantum [13] (NISQ) devices. A NISQ device is equipped with dozens to hundreds of qubits. IBM [8] released its 53-qubit quantum computer in October 2019 and has made it available for commercial use. Google [7] released the 72-qubit *Bristlecone* quantum computer in March 2018. Other companies including Intel [5], Rigetti [15], and IonQ, have released their quantum computing devices with dozens of qubits. The current NISQ technology may not be perfect, but it's a good first step towards the more powerful quantum devices in the future.

In order to map high level quantum programs to NISQ devices, it is important to overcome two obstacles. First, to be able to execute a quantum circuit, it is necessary to map logical qubits to physical qubits with respect to architecture and program coupling constraints. Any quantum program can be implemented using an universal gate set [11] of a small number of elementary gates. For instance, the $\{H,\ CNOT,\ S,\ T\}$ set is an universal gate set, in which the $\{H,\ S,\ T\}$ gates are single qubit gates, the *CNOT* gate is a two-qubit gate. The two-qubit gate must be mapped to two qubits that are physically connected. However, in real quantum architecture, qubits may have limited connection and not every two qubits are connected, as shown in the IBM *QX2* architecture in Fig. 1 (a). For this reason, a quantum circuit is not directly executable on a NISQ device, unless circuit transformation is performed. The common practice is to insert *SWAP* operations to dynamically remap the logical qubit such that the transformed circuit is hardware-compliant for each (set of) two-qubit gate(s).

Second, it is critical that the depth of a quantum circuit be minimized for the NISQ device. A qubit is volatile and error prone. It gradually decays over time and may have phase and bit flip errors. It may completely lose its state after a certain period of time, called *coherence time*. Quantum error correction (QEC) codes can detect error syndromes and fix them. However, QEC needs to use a large number of redundant physical qubits. A realistic QEC circuit may need more than 10,000 physical qubits, which is not possible for today's NISQ device. Without QEC, a program must terminate within a threshold amount of time. The depth of the circuit, which is the number of steps the circuit executes, must be optimized. IBM proposed the metric of *quantum volume* [2] for evaluating the effectiveness of quantum computers which accounts for not only the width of the circuit (the number of qubits), but also the depth, how many steps the circuit can execute.

Transforming the logical circuit into a hardware-compliant one will inevitably result in increased gate count and circuit depth. Most previous work for qubit mapping [9,14,17,19,21, 22] focus on minimizing the number of inserted gates, but not the depth of the transformed circuit. However, even if the gate count is small, it does not necessarily mean the depth of the circuit is small, due to the dependence between different gates. We discover that previous work that aims to minimize number of inserted gate may significantly increase the depth of the circuit . For instance, the Sabre approach by Li *et al.* [9] reduces the gate count by 1.1%, but increases the depth of the 10-qubit *QFT* circuit by over 44.5%. The two studies [10, 18] stress the importance of taking into consideration the variability in the qubit (link) error rates, but they do not directly address the issue of the increased circuit depth. The depth of the circuit, as mentioned above, is critical and determines if a quantum program is executable on a NISQ device with respect to its physical limits.

In this proposal, we first show our preliminary work, which is the first depth-aware qubit mapping scheme for quantum circuits running on arbitrary qubit connectivity hardware. Our depth-aware qubit mapper searches for the mapping that minimizes the transformed circuit depth and keeps the gate count within a reasonable range. Our results show we can reduce the depth of the transformed circuit by up to 30% compared with two best known qubit mappers [9, 19], and in the meantime, have on average less than 3% additional gates over a large set of representative benchmarks. In the rest part of the proposal, we are looking into two future works to partially address the qubit mapping problem namely – (i) GPU-assisted acceleration for mapping generation, and (ii) mapping for co-running quantum programs.

2.0 Background and Related Work

2.1 Quantum Computing Basics

2.1.1 Qubit

A quantum bit or qubit, is the counterpart to classical bit in the realm of quantum computing. Different from a classical bit that represents either '1' or '0', a qubit is in the coherent superposition of both states. It is considered as a two-state quantum system that exhibits the peculiarity of quantum mechanics [11]. An example is the spin of the electron that the two states can be spin up and spin down.

The mathematical form of a qubit is represented as a combination of $|1\rangle$ and $|0\rangle$, which is shown as $|\Phi\rangle = \alpha|1\rangle + \beta|0\rangle$. α and β are the probability amplitudes and are both complex numbers, which have to comply with the constraint: $|\alpha|^2 + |\beta|^2 = 1$. With the increasing of the number of qubits, the total number of states that can be represented grow exponentially. For n qubits, there are at most 2^n states that can be represented at the same time.

2.1.2 Quantum Gates

There are two types of basic quantum gates. One type of basic gates is the single-qubit gate, a unitary quantum operation that can be abstracted as the rotation around the axis of the Bloch sphere [11] which represents the state space of one qubit. A single qubit-gate can be parameterized using two rotation angles around the axes. There are several elementary single-qubit gates including the Hadamard (H) gate, the phase (S) gate, and the $\pi/8$ (T) gate [11]. The other type of basic gates is the multi-qubit gate. However, all complex quantum gates can be decomposed into a sequence of single qubit gates H, S, T, and the two-qubit CNOT gate. Thus we only focus on the two-qubit CNOT gate. The CNOT gate operates on two qubits which are distinguished as a control qubit and a target qubit. If the control qubit is 1, the CNOT gate flips the state of the target qubit, otherwise, the target qubit remains the same.

2.1.3 Quantum Circuit

Quantum circuit is composed of a set of qubits and a sequence of quantum operations on these qubits. There are various ways to describe the quantum circuits. One way is to use the quantum assembly language called OpenQASM [3] released by IBM. Another way is to use the circuit diagram, in which qubits are represented as horizontal lines and quantum operations are the different blocks on those lines. In Fig. 2 (a), we show a simple example of quantum circuit diagram. A single-qubit gate is denoted as a square on the line, and one CNOT gate is represented by a line connecting two qubits and a circle enclosing a plus sign.

2.2 Noisy Intermediate-scale Quantum Hardware

The concept of Noisy Intermediate-scale Quantum (NISQ) device was first proposed by John Preskill [13], that represents the kind of quantum computer with limited number (dozens to hundreds) of qubits and also with limited reliability due to the physical qubits being implemented without quantum error correction. NISQ devices may be able to exhibit quantum supremacy over classical computing on certain computation tasks, but the lack of reliability would make the circuit difficult to scale. The NISQ technology may not change the world immediately, but it's a good first step towards the more powerful quantum devices in the future.

The industry has been deeply involved in the development of NISQ devices. Google, Microsoft, IBM all released their NISQ devices [5, 7, 8], with qubits number ranging from 14 to more than 70. Among these devices, IBM's QX series stand out as are the first quantum chips available to public via cloud access. It has been intensively used by more than 100,000 users since 2017 [19]..

2.3 Qubit Mapping and Depth-Awareness

To enable the execution of a quantum circuit, the logical qubits in the circuit must be mapped to the physical qubit on the target hardware. When applying a CNOT gate, the two qubits connected by the CNOT gate need to be physically connected to each other. Due to the irregular physical qubit layout of existing devices, it is generally considered impossible to find an initial mapping that makes the entire circuit CNOT-compliant. The common practice is to insert SWAP operations to remap the logical qubits. A swap operation exchanges the states of the two input qubits of interest. As shown in Fig. 3, a SWAP operation is implemented using 3 CNOT gates for architecture with bi-directional links, or 3 CNOT gates plus 4 Hadamard gates for architecture with single-direction links, where a bi-directional link means both ends of the link can be the control or target qubit, while single-direction link means only one end of it can be the control qubit.

IBM's Qiskit uses a stochastic method to insert SWAPs [14] operations but often results in significant increase in the number of inserted gates and depth. Existing works [9,17,22] are more efficient than IBM's Qiskit mapper. They use efficient heuristics to find the mapping rather than a stochastic method. However, the main objective of these methods is to reduce the gate count. It makes sense to minimize the gate count, but it is more important to focus on the depth of circuit, as in the NISQ era the depth is equivalent to the estimated execution time. Reducing the depth of the circuit can reduce the likelihood of the circuit failing at an early stage.

We show an motivation example in Fig. 2. The hardware model is shown in Fig. 1 (a). It has five qubits and the connectivity is the same as the IBM QX2 architecture except that the links are all bidirectional. There are 5 physical qubits: Q_1 to Q_5 and six bi-directional edges. One CNOT gate can only be applied on one of these edges.

In the example, the initial mapping between logical qubits (denoted by lower case q) and physical qubits (denoted by the upper case Q) is shown next to each qubit (line), which is $\{\ \{q_1 \rightarrow Q_1\}, \{q_2 \rightarrow Q_2\}, \{q_3 \rightarrow Q_3\}, \{q_4 \rightarrow Q_4\}, \{q_5 \rightarrow Q_5\}\ \}$. With this initial mapping, it starts scheduling gates one by one until it encounters a (set of) CNOT gate(s) which cannot be scheduled due to physical constraints. We show the interaction of logical qubits in Fig.

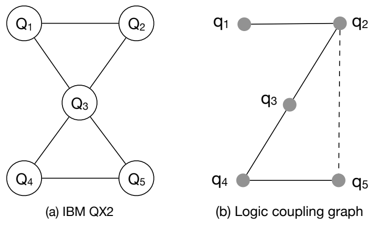


Figure 1: (a) The connectivity structure of IBM QX2, (b) The coupling graph for logical qubits in the motivation example in Fig. 2

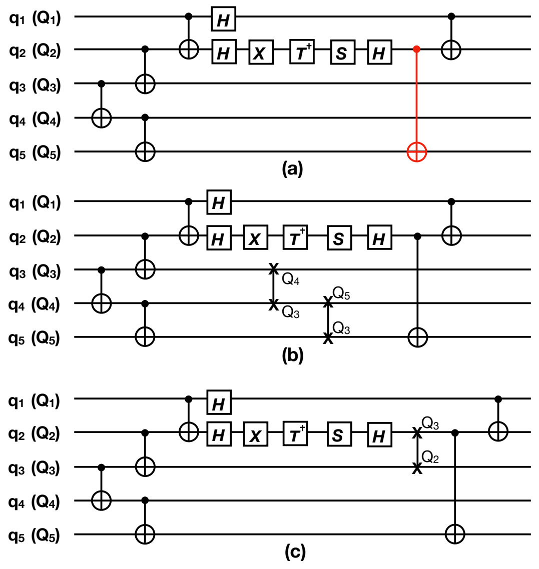


Figure 2: Motivation Example: (a) the original logical circuit; (b) uses 2 swaps but the depth of the circuit is not increased; (c) only uses 1 swap but the depth of the circuit has been increased

1(b) such that two logical qubits are connected if there is a CNOT operation between them. When we encounter the gate "CNOT q_2, q_5" (marked red in the circuit diagram in Fig. 2 and as the dotted line in the logical coupling graph Fig. 1), the scheduling has to terminate since this translates into "CNOT Q_2, Q_5" on the hardware, while no physical link exists between Q_2 and Q_5. Necessary *SWAP* operations are needed. When applying a SWAP operation, the two input physical qubits will exchange their states. Fig. 2 (b) and (c) provide two options for transforming the circuit. Fig. 2 (b) inserts 2 SWAPs (*SWAP* Q_3, Q_4 and *SWAP* Q_3, Q_5) such that "CNOT q_2, q_5" becomes "CNOT Q_2, Q_3", however the two SWAPs can run in parallel with existing single qubit gates in the circuit, without having to increase the depth of the circuit. Fig. 2 (c) inserts only 1 SWAP (*SWAP* Q_2, Q_3) such that "CNOT q_2, q_5" becomes "CNOT Q_3, Q_5", but it can not overlap with existing single-qubit gates in the circuit and will only increase the depth of the circuit by 3 (assuming we use 3 gates to implement the SWAP operation and each elementary gate takes 1 cycle in this example).

In this example, the best two known approaches by Zulehner *et al.* [22] and Li *et al.* [9] will both choose to insert 1 SWAP since they only optimize the number of gates inserted into the circuit (or the depth of the inserted gates), but not the depth of the entire transformed circuit. This example stresses the importance of depth-awareness in SWAP insertion schemes and motivates our work.

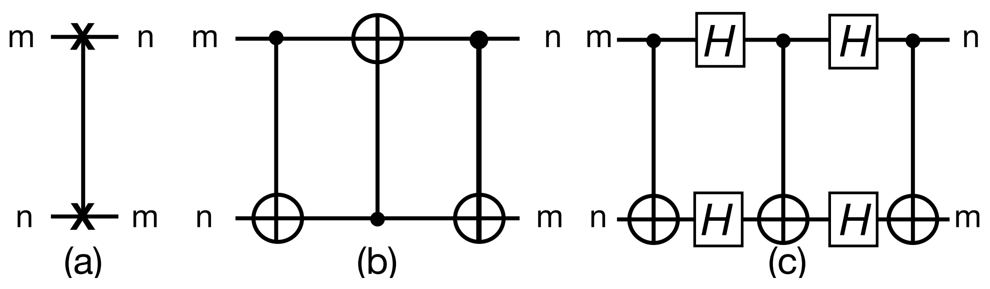


Figure 3: Implementation of a SWAP operation

3.0 A Depth-Aware SWAP Insertion Scheme (Preliminary)

3.1 Proposed Solution

3.1.1 Metric

As our work is a depth-aware SWAP insertion scheme, we first precisely define the metric for characterizing the depth of a circuit. In order to fully explain the metric, we need to introduce the concepts of *dependency graph* and *critical path*.

The dependency graph represents the precedence relation between quantum gates in a logical quantum circuit. The definition is below:

Definition 1. *Dependency Graph : The dependency graph of a quantum circuit C with a set of gates Ψ is a Directed Acyclic Graph $G_\psi = (\Psi, E_\psi)$, $E_\psi \subseteq \psi \times \psi$. A directed edge from node ψ_1 to node ψ_2 exists if and only if the output of gate ψ_1 is (part of) the input of gate ψ_2 in the quantum circuit C.*

The critical path is referred to as the longest path in the dependency graph. And the definition is below:

Definition 2. *Critical Path : Given a dependency graph $G_\psi = (\Psi, E_\psi)$ of a quantum circuit. The critical path is $CP = Max(Path(\psi_1, \psi_2))$ s.t. $\psi_1, \psi_2 \in E_\psi$ and $\psi_1 \neq \psi_2$*

The depth is characterizing the number of execution steps of a quantum circuit, which is tantamount to the critical path length of the circuit. The longest path in the dependence graph describes the minimal number of steps the circuit needs in order for every gate's data dependence be resolved. In Algorithm 1, we show how we calculate the critical path.

We first sort the nodes in the directed acyclic graph in topological order. Then we process the nodes in that order. For each node, we check the earliest start time for each of its predecessors, and add it by the latency of that predecessor, then we choose the maximum and use it as the earliest start time of this node. The maximum of all nodes' earliest start time added by their latency is the critical path length.

We use the critical path length as the metric for ranking different swap insertion options.

3.1.2 Framework Design

With the metric precisely explained in previous section, now we continue to explain the work flow of our framework and the intuitions behind it.

Before delving into the details of this framework, we need to define the *layer* and the *coupling graph.*

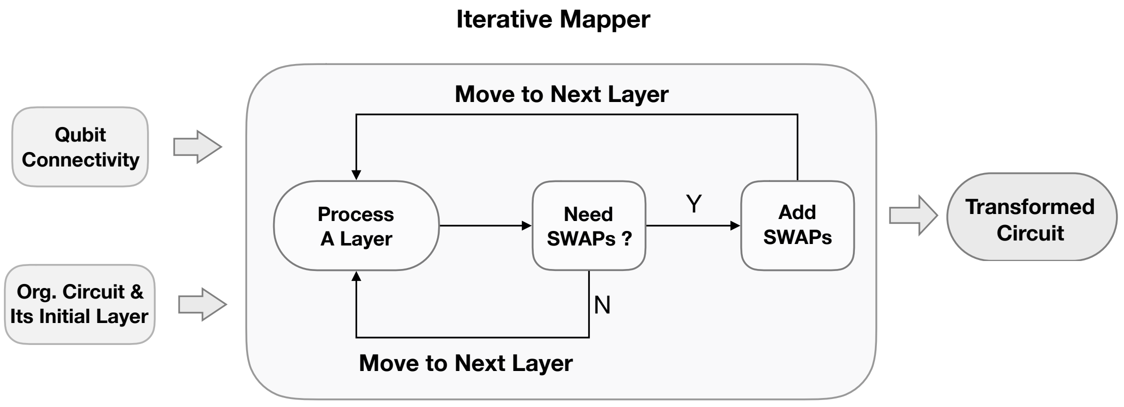


Figure 4: The Qubit Mapping Framework

Definition 3. *Coupling Graph : The coupling graph of a quantum architecture X with a set of physical qubits Q is a directed graph $G = (Q, E)$, $E \subseteq Q \times Q$. The edge $E_x = (Q_1, Q_2) \in E$ if and only if a $CNOT$ gate can be applied to Q_1 and Q_2 in X with Q_1 being the control qubit and Q_2 being the target qubit.*

We can divide the set of quantum gates in a circuit into layers, so that all gates in the same layer can be executed concurrently. The formal definition of a layer is:

Definition 4. *Layer : A quantum circuit C can be divided into layers $L = l_1, l_2, l_3, ..., l_m$, while $\bigcup_{i=1}^{m} l_i = C$ and $\bigcap_{i=1}^{m} l_i = \emptyset$. The set of gates at layer l_i can run concurrently and act on distinct sets of qubits.*

To divide a circuit into layers, we group the gates that have the same *earliest start time* (defined in Algorithm 1) into the same layer. The order of the layers is thus determined by the order of the *earliest start times*.

We use an iterative process to find the mapping. Our framework is depicted in Fig. 4. And this iterative process is explained as below. We start the framework by taking the input of the coupling graph (also denoted as *Qubit Connectivity*) and the original circuit's initial layer.

We process the circuit layer by layer. Given a layer, we perform the following steps.

- We check the layer to see if it is hardware-compliant based on the coupling graph and the qubit mapping before current layer is scheduled.
- If YES, we move on to next layer.
- If NO, we invoke our mapping searcher to search for (the set of) swaps that are necessary to solve the current layer. We consider depth-awareness during the selection of the set of swap gates – the resulted mapping of which generates the smallest critical path length (described in Section 3.1.3). After we find a hardware-compliant mapping, we move to the next layer.

After all layers are processed, the mapping terminates.

3.1.3 Circuit Mapping Searcher

Here we describe the specific mapping searcher we use to overcome the coupling constraint for a given layer.

We build our method upon the *A-star* algorithm for finding valid mappings that minimize the number of only the inserted SWAP gates [22]. We extend it by changing the ranking metric and allowing it to search for feasible mappings that do not necessarily have the smallest SWAP gate counts. It will help us search in a way that minimizes the depth while not significantly increasing the gate count.

We rank the swap options by the increase in the critical path length. Since it is an iterative process that handles the gates layer by layer, it is tempting to consider only minimizing the depth of the already processed circuit when deciding which swaps to use.

But the example in Fig. 5 shows that not only the processed circuit, but also the remaining circuit can help overlap the SWAPs with existing gates in the circuit without affecting the critical path. As shown in Fig. 5, for the CNOT gate (in red), there is no

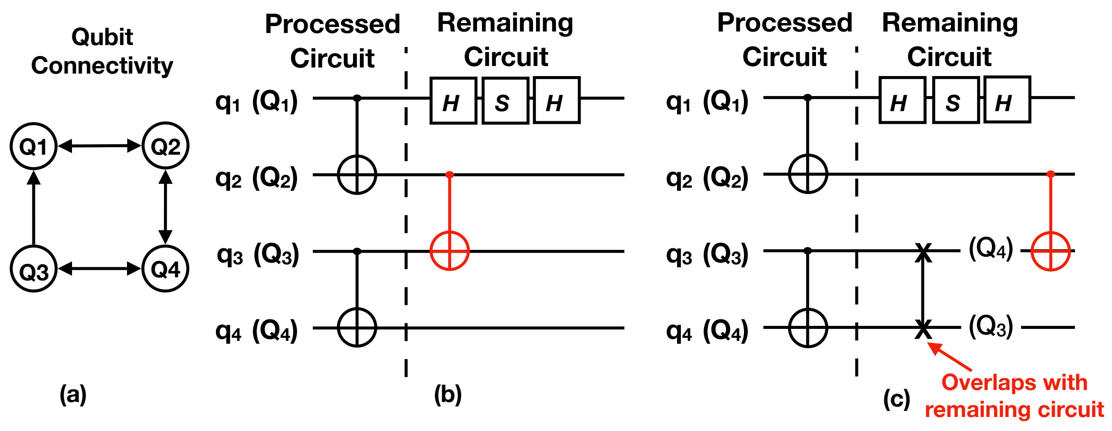


Figure 5: (a) Layout of an example architecture with 4 physical qubits (b) Example of a quantum circuit, the dashed line separates the processed circuit and the remaining circuit (c) Inserted SWAP overlaps with remaining circuit instead of existing processed circuit

way it can overlap the necessary SWAPs with the processed circuit (dubbed as the circuit before the dashed line). But when we look after the dashed line, the three single-qubit gates can overlap with inserted SWAP. And this renders less impact to the depth of the resulting circuit, compared to if we insert the SWAP on Q_1 and Q_2.

Based on this intuition, we design our scheme of choosing the SWAP candidate as in Fig. 6. For each of the hardware-compliant remapping candidates that we acquire from the *A-star* searcher, we calculate the critical path after merging the candidate (set of) swap(s) with both the processed circuit and the not-processed circuit. We choose the mapping that yields the shortest critical path.

3.1.4 Optimizations

We use two ways to optimize our proposed solution. One is to expand more nodes during the *A-star* search, and another one is to search into deeper levels.

3.1.4.1 Expand More Nodes In the search process for *A-star*, the normal routine is to expand the one node of least cost at each step. Here, we can expand more than one node

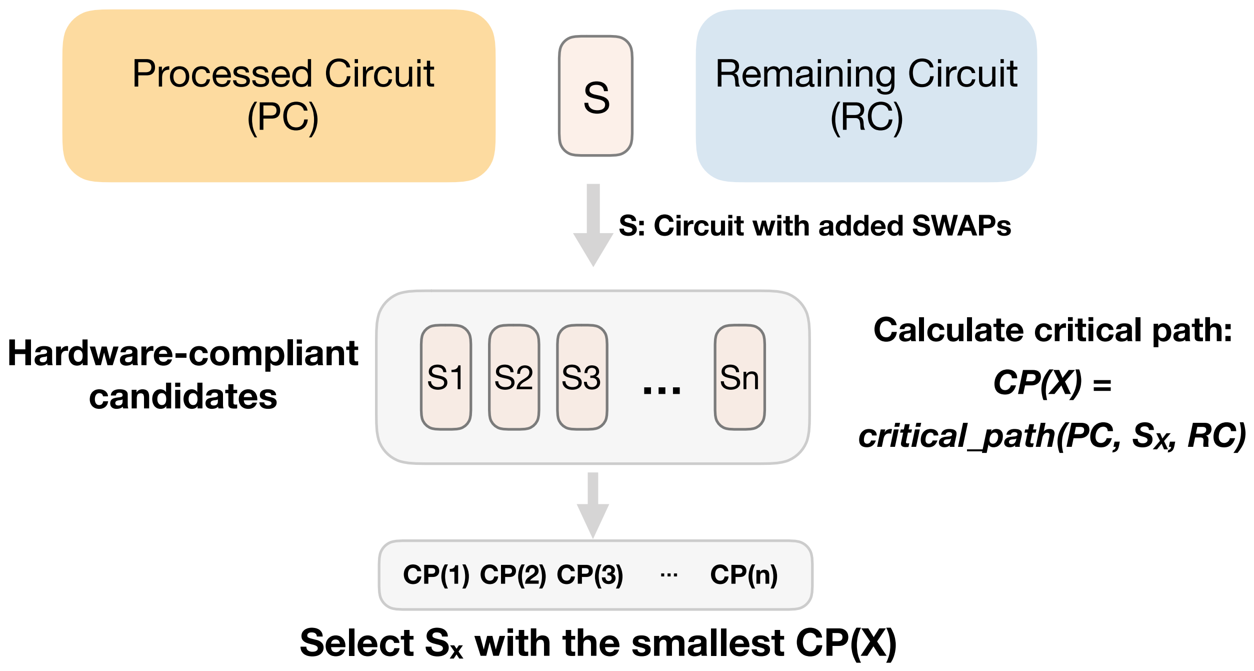

Processed Circuit (PC)
S
Remaining Circuit (RC)
S: Circuit with added SWAPs
Hardware-compliant candidates
S1 S2 S3 ... Sn
Calculate critical path: CP(X) = critical_path(PC, Sx, RC)
CP(1) CP(2) CP(3) ··· CP(n)
Select Sx with the smallest CP(X)

Figure 6: Choose SWAP Candidates

at each step and increase the search space. The number of nodes that can be expanded at a time can go from 1 to larger number.

3.1.4.2 Deeper Search We increase the depth of the A-star search tree. In normal case, the search process ends when it finds the first node that minimizes the number of SWAPs, which is reflected as a certain level of the A-star tree. To this end, the second optimization that we applied here is to continue the search into a deeper level of the A-star tree. We can specify and tune the parameter of the deeper search.

By tuning these parameters, there are more possible nodes added into our search space. With a larger search space, we have a larger possibility to jump out of one local optima and go to the global optima.

3.2 Evaluation

Table 1: Summary of Experiment results

Benchmark		**Total Gate #**			**Depth**	**Depth-delta**			**Improvement**	
name	n	Zulehner	Sabre	**DPS**	Original	Zulehner	Sabre	**DPS**	Min	Max
4gt5_75	5	131	122	119	47	44	44	29	1.52	1.52
mini-alu_167	5	435	396	432	162	131	125	119	1.05	1.10
mod10_171	5	361	328	298	139	117	89	39	2.28	3
alu-v2_30	6	804	717	795	285	261	241	201	1.20	1.30
decod24-enable_126	6	533	476	509	190	187	150	141	1.06	1.33
mod5adder_127	6	849	780	858	302	256	256	222	1.15	1.15
4mod5-bdd_287	7	94	94	94	41	18	23	17	1.06	1.35
alu-bdd_288	7	126	117	135	48	36	36	30	1.2	1.2
majority_239	7	915	780	885	344	265	194	182	1.06	1.46
rd53_130	7	1619	1508	1619	569	529	482	384	1.26	1.38
rd53_135	7	419	410	422	159	116	112	109	1.03	1.06
rd53_138	8	186	183	174	56	37	40	21	1.76	1.90
cm82a_208	8	899	944	1007	337	219	295	213	1.03	1.38
qft_10	10	266	263	281	63	47	96	44	1.07	2.18
rd73_140	10	347	329	338	92	84	79	67	1.18	1.25
dc1_220	11	2868	2685	3129	1038	820	697	681	1.02	1.20
wim_266	11	1505	1415	1511	514	431	450	311	1.39	1.45
z4_268	11	4453	4477	4972	1644	1162	1492	1076	1.08	1.39
cycle10_2_110	12	9143	8666	10115	3386	2467	2640	2421	1.02	1.09
sym9_146	12	493	454	472	127	118	138	86	1.37	1.60
adr4_197	13	5299	5017	5530	1839	1439	1599	1210	1.19	1.32
rd53_311	13	467	413	446	124	138	157	87	1.59	1.80
cnt3-5_179	16	325	238	286	61	79	59	43	1.37	1.84

We compare the total gate count generated. For depth, we compare the increased depth for each benchmark, denoted as "Depth-delta" here. The improvement represents the ratio of a baseline's depth-delta divided by DPS's depth-delta. Min/Max represents the improvement over the best/worst baseline.

In this section, we evaluate our **dep**th-aware **s**wap insertion scheme (denoted as DPS) and compare it with the two state-of-the-art qubit mappers. The experiment setup is listed below:

- **Benchmarks**: We use the quantum circuits from RevLib [20], IBM Qiskit [14], and ScaffCC [6].
- **Hardware Model**: We use IBM's 20-qubit Q20 Tokyo architecture, which was used in [9]'s work. The qubit connectivity graph is shown in Fig. 7.
- **Evaluation Platform**: The mapping experiments are conducted on a Intel 2.4 GHz

Core i5 machine, with 8 GB 1600 MHz DDR3 memory. The operating system is MacOS Mojave. We use IBM's Qiskit [14] to evaluate the depth of the transformed circuit.

- **Baselines**: We compare our work with two best know qubit mapping solutions, the work by Zulehner and others [22] (denoted as *Zulehner*), the Sabre qubit mapper from [9] (denoted as *Sabre*), and IBM's stochastic mapper in Qiskit. Since IBM's Qiskit mapper is significantly worse in terms of gate count and depth than all other mappers we evaluate, as also evidenced in the work by Zulehner *et al.* [22], we do not present Qiskit results.

- **Metrics**: We are comparing the depth and gate count of the transformed circuit circuits for all different strategies.

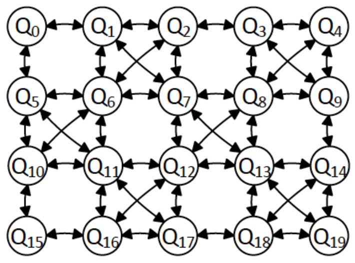


Figure 7: IBM Q20 Tokyo Physical Layout [9]

Table. 1 shows a summary experimental results. For gate count, we compare the total gate count generated in the transformed circuit. For depth, we compare the increased depth for each benchmark, denoted as "Depth-delta" in Table 1. The improvement columns provides the ratio between one of the two baseline's *depth-delta* and our *depth-delta*. We use the term *minimum improvement* to denote the improvement over the best of the two baselines, and the term *maximum improvement* to denote the improvement over the worse of the two baselines.

We discuss our findings from the following three aspects: depth reduction, gate count change, and the trade-off between gate count and depth.

3.2.1 Depth Reduction

For depth reduction, as shown in Table 1, our proposed solution outperforms the two baselines *Zulehner* and *Sabre*. Comparing *depth-delta*, the added depth of the circuit, our approach outperforms the better of the two baselines by more than 20% and up to 3X. For five out of the twenty-three benchmarks, our improvement on *depth-delta* is less than 20% compared with the better of the two baselines. However, for these cases, our approach still achieves considerable improvement over the worse of the two baselines. In these cases, it is possible that one of the two baselines happen to achieve very good depth in the transformed circuit and there is not much potential to improve. But our approach is still able to find a good mapping for these benchmarks and the performance is on par with the better of the two baselines.

3.2.2 Gates Count Changes

The primary goal of our depth-aware qubit mapper is to minimize the depth of the circuit. However, we discover that our qubit mapper can sometimes reduce the gate count. We discover that four out of the twenty three (17%) benchmarks, our qubit mapper yields the smallest number of gates among all three versions of qubit mappers. For 57% of these benchmarks, our method is ranked among top-2 of the three qubit mappers in terms of gate count. For the benchmarks where our method yields the largest gate count, the increased gate count percentage is negligible. On average, our depth-aware qubit mapper adds 3% gate count. From the experiment results, we can see that our solution does not greatly increase the number of gates while reducing the depth of the circuit.

3.2.3 Trade-off between Gate Count and Depth

While all previous works focus on reducing the total gate count (and the depth among the inserted gates themselves) after qubit mapping transformation, it is crucial to think about the trade-off between the resulted gate count and depth. Sometimes the choice made during the search process that favors the reduced gate count, might adversely affect the critical path. In Table 1, the *Sabre* mapper reduces the number of gates for 10-qubit QFT by 1.1% compared with *Zulehner*'s mapper, but increases the depth by 44.5%. For the *sym_9_246* benchmark, *Sabre* reduces the gate count by 3.8% compared with our approach, but increases the depth by 25.5%. Therefore a small reduction in the gate count may not be worthwhile if it increases the circuit depth significantly.

Algorithm 1: Calculate the Critical Path of a Circuit

```
Input  : The circuit's dependency graph $G(V, E)$
Output: The critical path $CP$

earliest_start = {};
CP = 0;
for $n \in V$ in topological order do
    temp = 0;
    for $p \in V$'s predecessors do
        if temp < earliest_start[p] + latency[p] then
            temp = earliest_start[p] + latency[p];
        end
    end
    earliest_start[n] = temp;
    if CP < temp + latency[n] then
        CP = temp;
    end
end
return CP ;
```

4.0 Parallel Acceleration for Mapping Generation (Future)

4.1 Motivation

Qubit Mapping problem inherently has very large search space due to its nature of being a NP-Complete problem [1]. Current works focus on using heuristics to speed up the searching process, but still suffer from long execution time for large benchmarks [9, 22]. Our existing work that focuses on depth-aware SWAP insertions is based on searching in the decision tree of expanding mappings. In this case, it is natural to think of an efficient way to accelerate the searching process. With the support of parallel processing unit like GPU, it is possible to greatly reduce the time of certain phases of the searching.

4.2 Problem Statement and Proposed Work

Here we are to formulate this case as a parallel-accelerated mapping generation problem. To enable the parallel acceleration for this program, we need to locate the patterns inside the program that can benefit from the parallel programming paradigm.

- Searching based on the heuristics and inherent cost in a decision tree, would actually result in massive computation on the combined cost of each node in the tree. This pattern can be parallelized using GPU.
- When performing our depth-aware swap insertion scheme, there are massive repeated tasks on calculating the resulting critical path of each mapping candidate. Due to the large amount of repetitive work, it's deemed obvious to utilize GPU to accelerate this process.

For these two cases, we propose our tasks as below:

1. Accelerate the searching in decision tree.
2. Accelerate the calculation of critical path of hardware-compliant mapping candidate.

The two cases can be further explained in the two figures below.

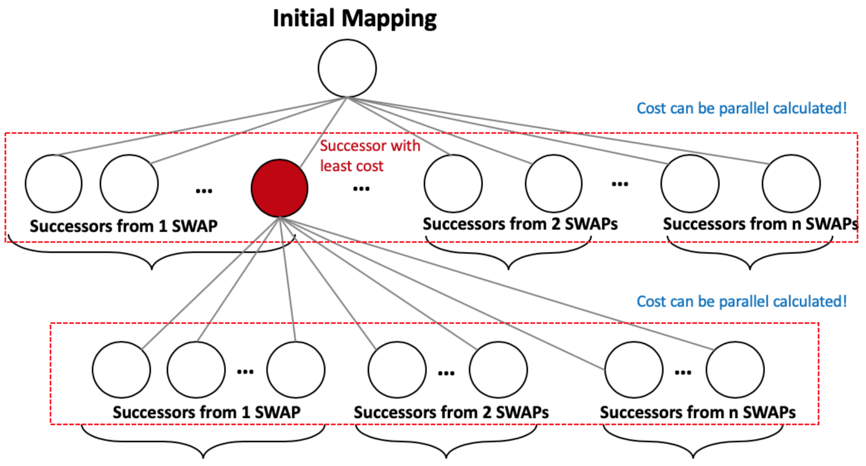
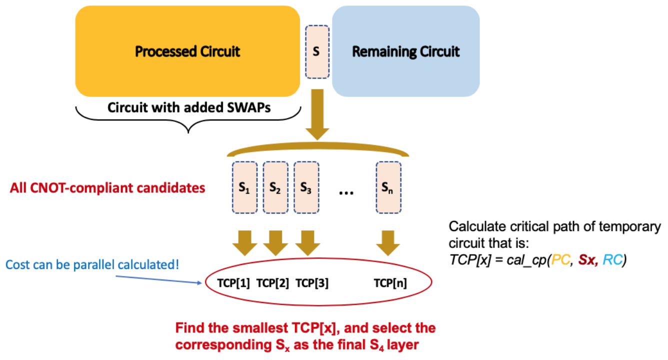


Figure 8: Patterns that can be accelerated during the searching in decision tree



Processed Circuit
S
Remaining Circuit
Circuit with added SWAPs
All CNOT-compliant candidates
S_1 S_2 S_3 ... S_n
Calculate critical path of temporary circuit that is:
TCP[x] = cal_cp(PC, Sx, RC)
Cost can be parallel calculated!
TCP[1] TCP[2] TCP[3] TCP[n]
Find the smallest TCP[x], and select the corresponding S_x as the final S_4 layer

Figure 9: Patterns that can be accelerated during the calculation of critical path of hardware-compliant mapping candidates

What we can get after these two proposed work is a faster framework that targets the depth-minimal mappings. While at the same time, we can utilize the speedup to increase the search space, as we can

- Search much more deeper
- Expand more nodes each level in the *A*-*star* tree.

These sure can produce better performance than current framework, and we are hoping to see more good results yielded in the end this work.

5.0 Mapping for Co-Running Quantum Programs (Future)

5.1 Motivation

Nowadays, the quantum devices in NISQ-era are equipped with qubits ranging from dozens to hundreds in number. When running quantum programs, it is very common to see the entire set of physical qubits under utilized. Many current quantum circuits use very limited number of qubits, from 5 qubits to 20 qubits. In the near future, more and more quantum devices with more than 50 qubits will be introduced. This brings about the opportunity for executing multiple quantum programs on a single quantum device.

5.2 Problem Statement and Proposed Work

Qubit mapping problem prevails when it comes to applying quantum programs onto the quantum devices, no matter if we are applying one circuit or multiple circuits. The proposed work aim to cope with the two problems below:

5.2.1 Sub-graph allocation

The dependency relationships among the qubits of one circuit would directly affect how the mapping is initially made. The gates dependency graph generated from each circuit already has its layout restriction. We call the dependency graph that needs to fit in the physical layout as the sub-graph. When applying multiple circuits onto the quantum device, how to allocate the sub-graphs for each circuit should be dealt with carefully. It can greatly affect how we utilize the resources. It can be treated as an sub-graph isomorphism problem, which has been proven as NP-complete, but comes with many effective algorithms that can be helpful for our situation here.

For example in the Fig. 10, we are applying three circuits $C1$, $C2$, $C3$ onto the quantum

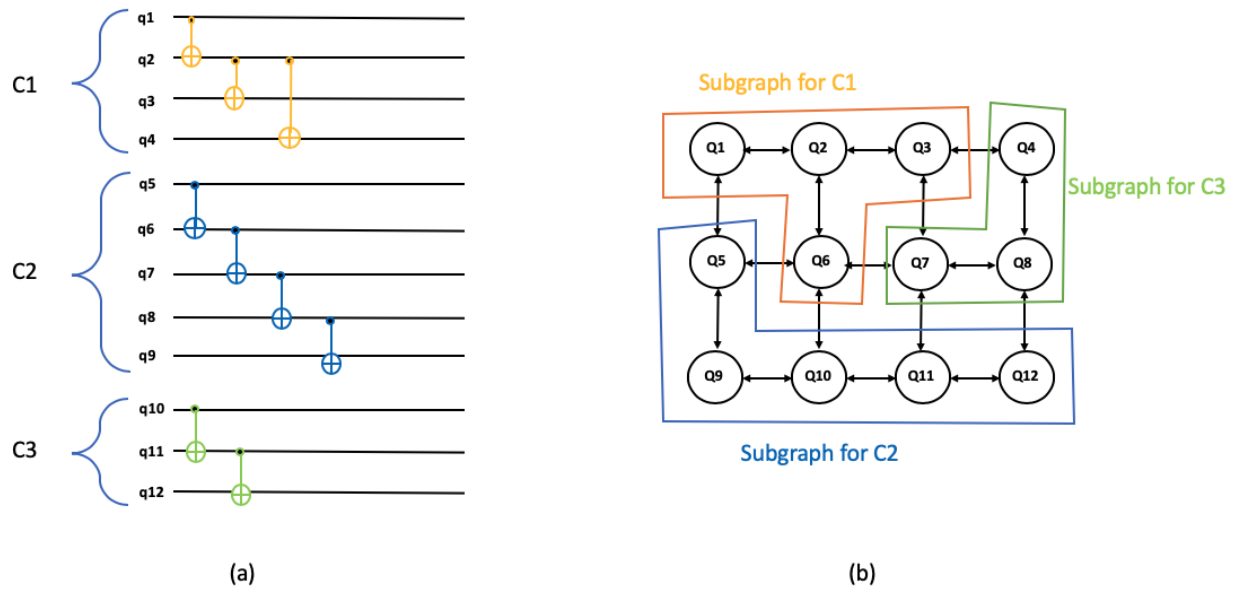


Figure 10: (a) Three independent circuits (b) The physical device we are applying these three circuits on. And how we can divide it into three sub-graphs for these three circuits.

device whose physical layout is shown in (b). And we show that based on the dependency graph of these circuits, we found a way to allocate these sub-graphs onto this device with no conflict. But if we allocate $Q1$, $Q5$, $Q6$, $Q7$, $Q8$ for $C2$, then the remaining physical qubits cannot accommodate for $C1$ and $C3$ at the same time. This shows that the effective sub-graph allocation here is of much importance here.

5.2.2 Swap insertions with/across sub-graphs

There is no surprise that qubit mapping still remains a problem for multi-circuit execution. And the cases can be different - as the inserted swaps can be within one sub-graph or across several sub-graphs. How to effectively coordinate the swaps among different quantum circuit becomes a problem here that can be researched.

5.2.3 Proposed Work

We will first transfer this problem into a graph problem. Then, we will try to utilize some graph algorithms to see if we can improve this problem. What we are aiming is to have multiple circuits mapped to one device, with the highest utilization rate while keeping the gate count and depth minimal. A lot of design decisions are still awaiting to be confirmed in the future. Here we try to define this problem and work out some directions we can go.

6.0 Conclusions

The physical layout of contemporary quantum devices imposes limitations for mapping a high level quantum program to the hardware. It is critical to develop an efficient qubit mapper in the NISQ era. Existing studies aim to reduce the gate count but are oblivious to the depth of the transformed circuit. This proposal first presents our design of the first depth-aware swap insertion scheme. Experiment results show that our proposed solution generates hardware-compliant circuits with reduced depth compared with state-of-the-art mapping schemes, with negligible overhead of increased gate count. Then we propose two future work that aim to (i) assist the searching of mapping using GPU acceleration and (ii) cope with the qubit mapping for multi-circuit execution scenarios on NISQ devices.

Bibliography

[1] Adi Botea, Akihiro Kishimoto, and Radu Marinescu. On the complexity of quantum circuit compilation. In *Eleventh Annual Symposium on Combinatorial Search*, 2018.

[2] Andrew W. Cross, Lev S. Bishop, Sarah Sheldon, Paul D. Nation, and Jay M. Gambetta. Validating quantum computers using randomized model circuits. *Physical Review A*, 100(3), Sep 2019.

[3] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

[4] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 212–219, New York, NY, USA, 1996. ACM.

[5] Jeremy Hsu. Intel's 49-Qubit Chip Shoots for Quantum Supremacy. https://spectrum.ieee.org/tech-talk/computing/hardware/intels-49qubit-chip-aims-for-quantum-supremacy, 2018.

[6] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. Scaffcc: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, page 1. ACM, 2014.

[7] Julian Kelly. A Preview of Bristlecone, Google's New Quantum Processor. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, 2018.

[8] Will Knight. IBM Raises the Bar with a 50-Qubit Quantum Computer. `https://www.technologyreview.com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer`, 2017.

[9] Gushu Li, Yufei Ding, and Yuan 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*, pages 1001–1014. ACM, 2019.

[10] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret 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*, ASPLOS '19, pages 1015–1029, New York, NY, USA, 2019. ACM.

[11] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[12] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J. Love, Alán Aspuru-Guzik, and Jeremy L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. In *Nature Communications*, volume 5, page 4213, 2014.

[13] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[14] QISKit: Open Source Quantum Information Science Kit. `https://https://qiskit.org/`.

[15] Rigetti. `https://www.rigetti.com/`.

[16] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[17] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintão Pereira. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, pages 113–125. ACM, 2018.

[18] Swamit S. Tannu and Moinuddin 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*, ASPLOS '19, pages 987–999, New York, NY, USA, 2019. ACM.

[19] Robert Wille, Lukas Burgholzer, and Alwin 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*, page 142. ACM, 2019.

[20] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W Dueck, and Rolf Drechsler. Revlib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, pages 220–225. IEEE, 2008.

[21] Alwin Zulehner, Stefan Gasser, and Robert Wille. Exact global reordering for nearest neighbor quantum circuits using A$*$. In *International Conference on Reversible Computation*, pages 185–201. Springer, 2017.

[22] Alwin Zulehner, Alexandru Paler, and Robert Wille. Efficient mapping of quantum circuits to the ibm qx architectures. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1135–1138. IEEE, 2018.