Quantum Circuit Resizing via Serial Execution

Movahhed Sadeghi, Soheil Khadirsharbiyani, Mahmut Taylan Kandemir

Pennsylvania State University
CSE Department
{mus883,szk921,mtk2}@psu.edu

Abstract. Quantum systems with limited physical qubits cannot execute quantum circuits with more logical qubits than physically-available ones, leading to compile-time errors. As it is unrealistic to expect quantum systems to provide sufficient qubits in the near future, there is a pressing need to explore strategies to execute large circuits on small systems, as current systems are comparatively small in comparison to the needs of the existing and emerging quantum algorithms/circuits. In this work, we analyze quantum programs to identify qubits that can be reused mid-program to execute the circuit with fewer qubits; this process is termed as resizing or serialization. Based on our analysis, we then propose a compiler-driven approach that selects the most beneficial qubits for circuit resizing, and provide proof of work for the algorithm. The results with our proposed circuit resizing indicate that it can i) execute large circuits that cannot originally fit into small number of physical qubits in current quantum systems, ii) significantly improve PST (Probability of Successful Trial) by 2.1X, and iii) and 53% reduction in circuit execution time when both the original and our serialized programs can fit into the target quantum hardware.

Keywords: Quantum Computing, Reliability, Serial Execution, NISQ, Compiler

1 Introduction

Quantum computing is positioned to address the growing need for enhanced computational capacity in solving today's highly complex problems such as machine learning, cryptography, and computational chemistry. In recent years, various vendors have built quantum hardware to benefit from quantum phenomena and execute quantum algorithms on quantum systems. However, the practical applications of quantum computing are limited today due to, primarily, low qubit counts and high errors.

Noisy Intermediate Scale Quantum Computers (NISQ) [10] have been designed to execute small-to-medium circuits on current hardware. While several works targeting NISQ machines have been introduced to reduce different types of errors [18,21,24,33,36], getting reliable outputs from large circuits is still an unsolved problem. Furthermore, a difficult aspect of quantum computing is the problem of operating a 'big' quantum circuit with many 'logical qubits' on 'small' quantum devices with a limited number of 'physical qubits'. While in the classical computing domain there exist lots of works focusing on recycling/reusing memory locations [30, 34], till recently, there had been no mechanism to solve this problem for quantum circuits in general. However, quantum vendors [14] have recently started to introduce the *Middle Reset* (MR) and *Middle*

Measurement (MM) gates, which can potentially be utilized to *resize* quantum circuits (i.e., minimize the number of qubits needed) via 'serial execution'. By utilizing MR/MM, theoretically, famous quantum algorithms like Bernstein-Vazirani [3] with any qubit count can be executed using only 2 qubits [16]. Additionally, when running on 2 qubits, no SWAP operations is needed since the 2 physical qubits to which the program is assigned would normally have a connection between them, eliminating the gate error due to SWAP operations in the process. This approach can be highly effective in improving the system's reliability. However, in order to employ MM gates, we need to verify that MM gates are operated in *isolation* from other qubits, meaning that measurement on one qubit should not affect other entangled qubits.

In this paper, first, we perform an analysis to identify the qubits that are most suitable for circuit resizing. Our results reveal that, in most quantum programs, there exist qubits that can be 'reused' mid-program to *serially execute* the circuit employing 'fewer qubits'. We further verify that MM gates are operated in isolation, and propose different techniques to isolate them if the vendor does not implement them that way. Motivated by these, we design, implement and evaluate a *compiler-based approach* that i) identifies the qubits that can be most beneficial for serialized circuit execution; ii) selects those qubits to reuse at each step of execution for size minimization of the circuit[1]; and iii) minimizes the Middle Measurement (MM) delays due to impractical implementation of shots to improve the circuit reliability. Furthermore, since our approach intends to execute a given circuit in a serialized fashion, the 'crosstalk errors' can also be optimized as a result of the reduced number of concurrent gates. We present experimental evidence demonstrating the effectiveness of our compiler-based approach. The experimental results collected indicate that our proposed approach can i) execute large circuits that initially cannot fit into small circuits, on small quantum hardware, and ii) significantly improve the PST of the results by 2.1x iii) and 53% reduction in circuit execution time when both the original and our serialized programs can fit into the target quantum hardware.

2 Background and Related Work

2.1 Quantum Computing Basics

Quantum computation is based on the concept of *qubit*, as opposed to classical computing, which is based on *bits*. Compared to a classical bit which represents a value of 0 or 1, a qubit is a *vector* that holds a 'state' *between* 0 and 1, which is defined as follows: $|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle$. This leads to an 'exponential growth' in state space in terms of the number of qubits [32]. For example, having two qubits gives us a state space of $|\varphi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$. A *quantum gate* is the basic building block of a *quantum program/circuit*. Quantum gates fall into two main categories: unitary gates (e.g., rotational gates and X, Z, Y, H gates, etc.) and controlled unitary gates (e.g., CX, CZ, CY, CCX gates, etc.). On current quantum hardware, a quantum gate that operates on multiple qubits can execute only in the presence of a *direct link* between the involved qubits. Qubits are prone to a variety of 'errors', such as coherence error, gate error, and crosstalk. On the other hand, gate errors occur because of the operations being performed on qubits, and crosstalk errors occur due to the interaction between different qubits

[1] In this paper, we will use the terms 'serializability', 'sequential/serial execution', 'circuit reduction', and 'circuit resizing', interchangeably.

during 'concurrently-running' operations. Additionally, during measurement, one can experience a readout error; this type of error affects the reliability of the outputs.

Current quantum computers –NISQ machines– rely heavily on SWAP operations – gate operations that *swap* the state of two linked/neighboring qubits, to perform an operation between two qubits that are not adjacent.

Current NISQ systems operate in a QAOA (Quantum Approximate Optimization Algorithm) [8] fashion. QAOA is a paradigm that *combines* classical computers and NISQ systems [8]. More specifically, a quantum program *compiles* into either a quantum circuit or a batch of quantum circuits. At the end of the execution of each circuit, the resulting qubits (output) are measured (read out) and their measured values are stored in the 'classical computer memory'. Subsequently, the qubits are reset to a state of $|0\rangle$ for the next circuit to execute. Due to the 'probabilistic nature' of quantum algorithms and occurrence of physical errors, this process will be repeated multiple times –shots– and result counts will aggregate to generate the final output.

2.2 Different Types of Dependencies in Quantum Computing

Qiskit [1] provides a *directed acyclic graph* (DAG) representation of the circuit, in which roots and leaves represent qubits and other nodes represent gate operations. Additionally, the edges indicate the qubits used by gate operations, as shown in the example of Fig. 4. Note that this DAG representation i) provides an order for gate operations; ii) indicates dependencies between qubits; iii) specifies parallel operations at each stage; and iv) gives the critical depth of the circuit. While this DAG representation provides the dependencies among different operations, it does *not* differentiate between 'false' and 'true' dependencies. False dependencies in DAG are those that originate from the ordering of the gates, but they do *not* influence the subsequent qubit/operation, meaning that changing their ordering does not affect the outcome of the circuit. However, true dependencies can affect the final result of the system if they are violated, i.e., the order of operations is changed. In Fig. 4, for example, the CNOT operation between q_0 and q_5 has no influence on the value of q_1, following its CNOT operation with q_5; but, in the DAG, these two operations look 'dependent', due to the sequence of the program.

False dependencies provide, in a sense, 'opportunities' for changing the order of operations and transforming circuits [22], to target different objective functions. Attaining these false dependencies is easy during the early stages of compilation process; but, they become impractical to detect during the execution of operations.

2.3 New Features: MM and MR

Starting in 2020, IBM Quantum systems (IBMQ) started to gradually include *Middle Measurement* (MM) and *Middle Reset* (MR) gates into their quantum systems [16], aiming to provide *qubit reuse* during the course of program execution. To our knowledge, all IBMQ quantum systems currently support these features. In the early days of these gates' debut, IBMQ provided an instance of the Bernstein-Vazirani [3] circuit (a 5-qubit version is depicted in Fig. 4) to demonstrate the possibility of serial execution of quantum circuits for improved reliability [16]. This paper proposes a compiler-based 'circuit resizing/serialization' approach that *automates* the disciplined use of MM/MR so that a given quantum circuit can execute on fewer physical qubits in a serial fashion.

We want to emphasize that the MM/MR gates are not in the theoretical quantum gate-set but are achievable via 'imperfections' in the physical properties of the technology. In reality, coherence error causes the qubit state to return to $|0\rangle$. By overclocking a qubit, the coherence error increases exponentially, causing the qubit to return to $|0\rangle$ state faster, allowing to achieve a reset gate (MR gate). Also, in theory, two 'entangled' qubits will remain entangled forever unless we explicitly *disentangle* them via extra logic. However, in practice, two entangled qubits would become disentangled over time – dephasing – allowing us for isolated measurements in case of an entanglement. Formally, we define an MR gate as a gate that returns any state to $|0\rangle$, and an MM gate as a gate that can measure a qubit in isolation [16].

2.4 Related Work

While quantum computing is still in its infancy (in terms of both hardware or software), its potential advantages over so-called classical computing for particular algorithms, e.g., in the context of drug discovery, machine learning and prime factorization, are very promising [17, 32]. Quantum systems are currently being heavily researched, and the major efforts focus on the areas of compiler support [22, 23, 27, 31], operating system support [29], and programming languages [5].

Compilation of quantum programs consists of three main *stages*: i) matrix-to-gates conversion; ii) Intermediate Representation (IR) optimizations; and iii) logical-to-physical qubit mapping and circuit execution. In this context, a *matrix* represents a function/system which is applied on sample inputs (a vector of inputs) to generate a final output (an output vector). In the first step, the matrix is translated into 1-qubit and 2-qubits gates using an algorithm such as Fowler [9, 10]. If, on the other hand, the programmer encodes the quantum circuit directly (i.e., if he/she inputs the gates instead of the matrix), then this stage –Fowler gate production– can be omitted.

The second stage of a quantum compiler performs IR-level optimizations. Recent studies have implemented a variety of LLVM-based optimizations aimed at various domains [27]. For example, Paulihedral [22] is one of the most recent works; it proposes retaining a gate matrix IR abstraction until the final stages of compilation in an attempt to achieve some circuit optimizations, such as depth reduction, gate cancellation optimization and swap reduction, by ignoring false dependencies between layers of gate production and re-ordering the gate operations. This approach fits well to enhance our work in the future, since discovering false dependencies at the final stage of compilation (where our approach is embedded) can be challenging. It can also be used to re-arrange the gate layers according to the volume of their true dependencies in order to maximize serialization opportunities for future enhancements to our work to obtain true dependencies. The last stage of the compilation of a quantum program involves quantum circuit-level optimizations including the mapping of logical qubits to physical qubits [18, 24, 28], gate cancellation, swap reduction, gate scheduling [26] and dynamic decoupling [6].

3 Motivation and Problem Definition

This section describes the three main factors that have motivated us to design a compiler-based strategy for minimizing the size of a given quantum circuit.

3.1 Size Limitation in NISQ Systems

The continuous demand for more processing power requires the development of quantum computers with large number of physical qubits. Furthermore, the notable absence of a physical 'quantum memory' places the whole processing weight on qubits. With all these demands for more qubits on quantum computers, the reliability of physical qubits as well as that of their connecting links remain as the main issue. Furthermore, as the system grows in size, the reliability degradation becomes even harder to avoid. This results in a decrease in the number of links between qubits to eliminate crosstalk noise [2], and as a result, leads to more scattered/distributed qubits, lowering the qubit processing speeds. While, owing to the exponential expansion of processing power, this cost of processing speed may not be the primary concern, the *decrease in the number of links in larger NISQ systems* forces the inclusion of multiple extra SWAP operations to execute a given quantum circuit.

3.2 Reliability Concerns with Larger Systems

The majority of qubits in current IBMQ system architectures, for example, have two links, whereas a few have one or three links [14]. Also, a large quantum circuit will typically have various 'controlled gate' operations, necessitating the use of numerous SWAP gates to complete the execution. This not only results in extra gate errors but also in excessive crosstalk and coherence errors.

For example, executing a 12-qubit circuit like Bernstein-Vazirani [3], which can be categorized as one of the relatively simple 'medium-size' quantum circuits, on a sample IBMQ device results in entirely unreliable outputs (a fidelity of 0.007 is reported in [16]). However, by serial execution, the fidelity can increase up to 0.31 (40x compared to the prior case [16]). Consequently, there is a strong motivation for exploring strategies that *serialize* a given quantum circuit in the absence of a quantum memory, from a reliability viewpoint as well. However, it is important to verify that MM gates are operated in isolation and do not affect the other qubits when measuring a qubits.

3.3 Viability of Isolating MM Gates

One of the most important constraints on the implementation of MM gates is that, ideally, they should have no effect on the entangled qubits to maintain output correctness. To achieve such isolation, the underlying hardware can i) turn off the links connected to the target qubit/reduce the frequency of entangled qubits; ii) similar to middle reset, use decoherence error to disentangle the entangled qubits via dephasing [35]; or iii) use SWAP to move the target qubit to a node isolated from its entangled ones, if the system is not fully-connected (akin to removing a link). While the exact implementation of this gate on current systems remains *unknown*, any hardware that does not support MM in isolation can use one of the three techniques mentioned above to ensure output correctness.

In order to verify that the current middle measurement gates are isolating the qubits from one another, we have designed two distinct circuits, illustrated in Figure 1. The results of both circuits are essentially identical; however, their methods for ensuring that the qubits are not measured simultaneously are different. If the middle measurement is not performed in isolation, upon measuring q_1 (should be equal to $|1\rangle$), q_0 would also be

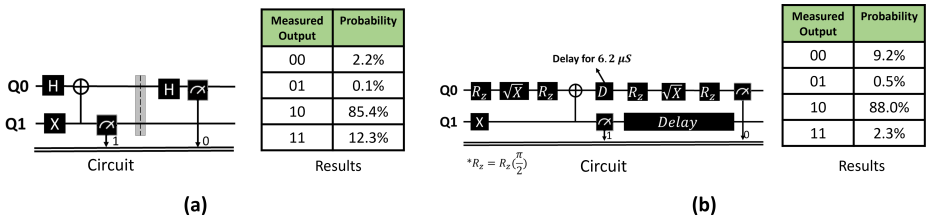

Fig. 1: Checking the functionality of MM (a) by using barrier between measurement operations on ibm_oslo and (b) by applying delay gate in correct placement post-transpile on ibmq_belem.

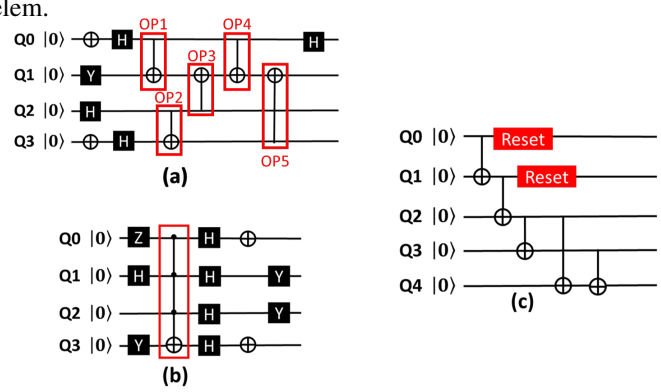

Fig. 2: (a) & (b) 'Unresizable' Circuits, (c) 'Resizable' Circuit

measured. Since the state of q_0 after CNOT is equal to $|+\rangle$, both $|0\rangle$ and $|1\rangle$ would be reported with the same probability upon measurement. Then, by applying the Hadamard gate, we can either reach $|-\rangle$ (if q_0 is measured as $|1\rangle$) or $|+\rangle$ (if q_0 is measured as $|0\rangle$). In both the instances, the measurement results will report $|0\rangle$ and $|1\rangle$ with equal probability. This indicates that the result of our circuit, when the middle measurement is not operated in isolation, should be between $|10\rangle$ and $|11\rangle$ with a probability of 50%. In contrast, if MM is performed in isolation, the measurement would have no effect on q_0's state, and q_0 would retain its $|+\rangle$ state after the measurement. By applying the Hadamard gate, q_0 would report the state $|0\rangle$. The result of our circuit, when MM is executed in isolation, would be $|10\rangle$ with a 100% chance. Therefore, based on the results, we can verify if the MM is operated in isolation or not. In both the cases of entanglements in Fig. 1, $|10\rangle$ is obtained with the highest probability, indicating that the MM gates are operated in *isolation*.

3.4 How to Resize the Circuit via the MM and MR Gates?

The first step in building a circuit for the current NISQ machines is to define the required number of qubits and classical registers for measurement. The programmer/user encounters an error *if* the number of qubits in the circuit to be implemented exceeds the system size. For instance, defining a Bernstein-Vazirani circuit with a size of 10 'logical qubits' and attempting to execute it on a system with 7 'physical qubits' results in a 'compilation error'. Although in principle the Bernstein-Vazirani circuit of any size can be executed *sequentially* using no more than 2 qubits via the employment of the MM and MR gates [16], coherence errors would put a cap on the maximum size of this circuit.

Note that gates involving more than two or three qubits (like those in Fig.2) do not exist in current hardware and are typically implemented via multiple existing physical controlled-gates. This means that achieving full entanglement is doable via sequential approach, like the circuit depicted in Fig. 3 (linear entanglement), which is resizable. This approach holds promise in reducing dependencies and amplifying serialization. **It is crucial to clarify that, though entanglement may result in dependencies, it should not be mistaken as a dependency itself.**

Our goal in this paper is to minimize the size of a given quantum circuit by running it in a serial/sequential manner via the employment of the MM and MR gates, and also to increase the output reliability for circuits executing in architectures with limited number of links per qubit. Thus, our main novelties include i) giving a polynomial time algorithm to minimize the size of a given quantum circuit (i.e., maximize serialization); ii) providing an implementation of this algorithm and proving that it does indeed *minimize* the input circuit; and iii) avoiding the potential reliability issues that could arise from the delays of the MM gates, which can happen through multiple shots.

4 Overall Design

As mentioned before, the constraint for serialization can be expressed as follows:

> **A circuit is 'serializable' if and only if there exists a qubit that can complete its final gate operation without the need of all other qubits on the circuit.**

Fig. 3 shows a serializable circuit and its serial execution with the minimum necessary qubits. whereas Fig. 2 depicts two circuits that do not meet our requirement. In Fig. 2-a, none of the qubits can complete its operation without activating the remaining qubits. On the other hand, in the circuit shown in Fig 3, q_0 can complete its task with the assistance of only q_1. Therefore, the first circuit should be executed in parallel, while the second circuit can be serialized (i.e., its size can be reduced).

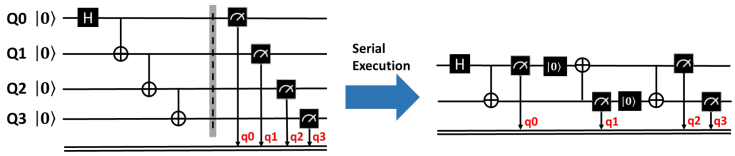

Fig. 3: A sample cat_state_n4 circuit and its serial execution.

4.1 Requirements for Circuit Size Reduction

First, let us discuss the use-cases for which the MM/MR gates are expected to be beneficial. The MM gate enables users to *measure* a qubit *after* it has completed its final action , whereas the MR gate is utilized to *reset* it to a $|0\rangle$ state. Together, these two operations enable any result qubits to be measured and reset so that they can be used by the remaining qubits of the program.

Let us now discuss the reason behind why Fig. 2-a and Fig. 2-b are *not* resizable. For the circuit illustrated in Fig. 2-a, we can see that none of the qubits can perform its task/operation in isolation from the other qubits, owing to the gate that *entangles* them all (i.e., full entanglement). Therefore, resetting any of these qubits is not beneficial for serialization. Fig. 2-b, on the other hand, shows a more complex circuit to demonstrate

the criteria that should be satisfied in order to serially execute a circuit as much as possible. In this example, Q0 is used in OP1 and OP4, needing Q1 for its completion. However, OP2 and OP3 should also be executed before we can reuse Q0 (OP2 and OP2 should finish before OP4), meaning that Q0 needs all the qubits to be available when it finishes its last operation. Q1 has operations with all the other qubits, hence needs them all. Q2 has operations with Q1 and Q3 (OP3 and OP2), still needing OP1 to be executed beforehand, meaning it needs all the qubits to be available before its final operation, and Q3 is similar to Q2. Therefore, we cannot reuse any of these qubits, eliminating the possibility that they can be used as a choice for resizing the circuit. This also means we need to find the *dependencies* between qubits.

The most beneficial qubits for resizing/serializing a given quantum circuit are those qubits that can complete their tasks with the help of the minimum number of other qubits, i.e., *least dependencies*. By prioritizing the qubits based on the number of dependencies in our algorithm, we reduce the circuit size as much as possible. Therefore, in our algorithm, we introduce an additional constraint aiming at maximizing the improvement (note that this is *not* a constraint for resizability of the circuit; rather, it is a constraint to be satisfied to maximize the potential improvements from circuit resizing):

The best nominee for resizing a circuit is the qubit that is least-dependent on other qubits.

One way of finding these qubits is to use a *circuit DAG*, as a DAG precisely captures the sequence of operations, dependencies, and the stage (in the circuit) at which each qubit completes its task. However, the DAG does *not* capture 'false dependencies', which are *not* important for us due to the following reasons: i) finding false dependencies is exponentially complex for a given circuit and has polynomial complexity in the IR-levels of compilation; ii) false dependencies can only cause an adverse effect in our circuit minimization algorithm in one case, which is not expected to be frequent. We further elaborate on this in Section 4.3; and iii) our main goal is to minimize the circuit only through changes in the execution strategy, *not* through the changes in the circuit itself or its gate operation order. Note that, our algorithm can perform *both* with true dependencies and DAG dependencies as inputs. Providing an efficient algorithm to obtain true dependencies as an input to the algorithm is left to a future study.

4.2 Our Proposed Algorithm for Circuit Minimization

This section introduces our approach and goes through an example scenario *step-by-step* to demonstrate how it works in practice. Algorithm 1 gives our proposed algorithm for quantum circuit resizing. To begin, we look for a qubit with the least dependency. We obtain this information by tracing back the DAG for each qubit from *leaf* to *root*. We add the gate operations required before the chosen qubit completes its task, keeping track of the total number of gate operations added in each qubit line. Then, we deduct this value from the total number of gate operations performed in each of the activated qubits by the added gates. This is done mainly to avoid repeating the gate operations to maintain the correct circuit. Note that, when this value reaches zero, we perform the MM/MR procedure. We update the dependency lists by deleting the qubits that have already been allocated from the current lists, and select the new qubit with the least dependency to

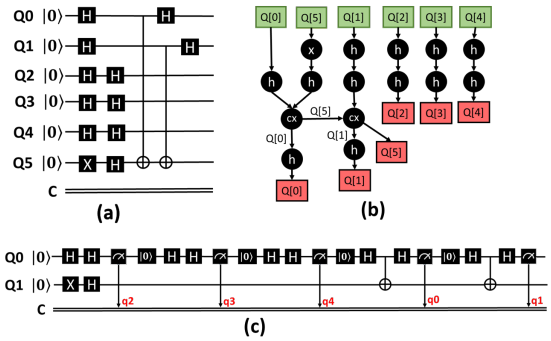

Fig. 4: (a) 5-qubit Bernstein-Vazirani circuit, (b) Corresponding directed acyclic graph (DAG), and (c) Our serialized version.

load on the reset qubit. Thus, we prioritize serialization by decreasing the number of added qubits in each iteration as little as possible.

Fig. 4 shows an example application of our proposed algorithm on an example Bernstein-Vazirani circuit. In this example, $D-list$ for q_0 is $[q_0,q_5]$; and for q_1, it is $[q_1,q_5,q_0]$. Also, $D-list$ for q_2 is $[q_2]$; for q_3, it is $[q_3]$; for q_4, it is $[q_4]$; and finally, for q_5, $D-list$ is $[q_5,q_1,q_0]$. Now, we have the sorted $L-list = [[q_2],[q_3],[q_4],[q_0,q_5],[q_1,q_5,q_0],[q_5,q_1,q_0]]$ as described, the first element in the $L-list$ is $[q_2]$, and we assign all the qubits in the dependency list to the first available physical qubits, which in this case means only assigning q_2 to first available physical qubit (Q_0[2]). We then remove all the assigned qubits from all the elements in $L-list$, which means removing q_2 from all the elements of $L-list$. Additionally, we also update the count list, which shows the number of 'unexecuted' operations for each qubit. In this case, all the 2 Hadamard gate operations for q_2 have already been executed ($count[q_2]=0$), and we perform the MM and MR operations on Q_0 and look for a new qubit for assignment. The new updated list is $D-list = [[q_3],[q_4],[q_0,q_5],[q_1,q_5,q_0],[q_5,q_1,q_0]]$. We perform the same steps for q_3 and q_4, assigning them to the same physical qubit (Q_0). After these two steps, $L-list$ becomes $L-list = [[q_0,q_5],[q_1,q_5,q_0],[q_5,q_1,q_0]]$. In the next step, we choose $[q_0,q_5]$, which is the element for q_0. Since it contains more than one qubit, we assign the first logical qubit (q_0) to the reset qubit (Q_0) and assign the second logical qubit (q_5) to the second available physical qubit (Q_1). By scheduling the gates for these two qubits and updating the list accordingly, $count[q0]$ becomes 0, and $count[q5]$ becomes 1, meaning q_0 is now ready for MM/MR. After the update, the new $L-list$ becomes $L-list = [[q_1],[q_1]]$. We choose the first element, $[q_1]$, and assign q_1 to Q_0. After scheduling all the operations and updating the list, we see that both $count[q_1]$ and $count[q_5]$ are equal to 0. This means that all the operations for both q_1 and q_5 are executed, and we can measure and reset them using the MM and MR gates if needed. Finally, we empty $L-list$, and the algorithm finishes.

4.3 Proof

We now present a *formal proof* showing that our proposed approach does indeed *minimize* a given quantum circuit (size/number of qubits), changing only circuit execution and

[2] Uppercase Q is for physical qubits

ignoring gate reordering/circuit-modifications based on false dependencies (we can also feed the algorithm true dependencies, if they are provided by a previous layer such as [22]).

Algorithm 1: Our proposed compiler algorithm for circuit resizing.

Input:
DAG circuit
Register:
$D-list[q] \longleftarrow$ List of dependencies for q
$L-list[D] \longleftarrow$ list of $D-list$s
$Count[q_i] \longleftarrow List\ of\ number\ of\ q_i's\ gate\ operations$
$PQ \longrightarrow$ A flag list to show available physical qubits
Output:
$New-circuit$
for *qubit* ***in*** *DAG* **do**
    $D-list[qubit] \longleftarrow [qubit]$
    Trace back in DAG from qubit leaf to root and append all interacted qubits
    $L-list.append(D-list)$
$L-list.sort(key = lambda\ i : len(i))$; `// sort this list of D-lists by list length`
$New-circuit \longleftarrow \phi$
$PQ \longleftarrow$ Is a list of Size L-list first element, initialize to 0 meaning available
Algorithm:
if *len(L-list[0]) == circuit's number of qubits)* **then**
    print(circuit is not resizable)
    $New-Circuit = circuit$
    break
else
    while *(len(L-list) != 0)* **do**
        $Chosen \longleftarrow L-list.pop(0)$
        $L-list\ =$ [[q **for** q **in** $D-list$ **if** q **not in** $Chosen$] **for** $D-list$ **in** $L-list$]
        $L-list.sort(key = lambda\ i : len(i))$
        ; `// Update the D-list elements of L-list based on the qubits that became activated and resort the L-list to ensure minimum addition - if any - of qubits will happen at each iteration`
        Assign logical qubit from *Chosen* to physical qubits available in *PQ*, add elements to PQ if *Chosen* can't fit in available
        Update new *PQ* values to 1 meaning occupied
        for q_i ***in*** *Chosen* **do**
            $New-Circuit \longleftarrow New-Circuit +$ Add gate operations if gates are in *Chosen* –only–
            Update count values in $Count[q]$ list
            if $Count[q_i] == 0$ **then**
                $PQ[q] \longleftarrow 0$

We use *proof by contradiction* (this is a proof of size minimization due to a change in execution, not a change in circuit): **Please note that any addition of qubits at any stage or iteration will permanently increase circuit size unless loaded on a reset qubit.** Suppose that q_i is a qubit with more dependencies than the minimum dependency q_m at iteration k; so, it can lead to a smaller circuit than q_m ($S[q_x]$ represents the q_x dependency list size). There are three different scenarios to consider:

- The q_m dependency list is a subset of the q_i dependency list. Obviously, in the dependency list of q_m, the first element is q_m itself (it will be in the q_i dependency list as

well). Note that adding q_i would increase the circuit size by $S[q_i] > S[q_m]$ qubits permanently. Further, adding q_m even right before q_i would lead to $S[q_i] = S[q_i] - S[q_m]$ and release q_m for q_i for use, thereby increasing the size of the circuit by at most $S[q_m] + S[q_i] - S[q_m] - 1$, leading to a smaller circuit, which is clearly a contradiction.

- The q_i and q_m dependency lists have no intersection. In this case, they are like two separate circuits (C_1 and $C2$) we are trying to resize (in the example Bernstein-Vazirani circuit above, based on the dependency lists, for our purposes, we can consider $[q_0, q_1, q_5], [q_2], [q_3], [q_4]$ as separate circuits), with C_1 being the circuit with q_i and C_2 being the one that has q_m. Since they are like two separate circuits, it is clear that any minimization needs them to work in a serial fashion, meaning that either we start with C_1 and then load C_2 or vice-versa. In both cases, irrespective of the selection order of q_m and q_i, the final minimum-sized circuit will be of size $\mathbf{Max}\{C_1, C_2\}$, leading to the two cases with an equal number of physical qubits (not smaller) and will be a contradiction.
- The q_i and q_m dependency lists intersect, but the q_m dependency list is not a subset of the q_i dependency list. In this scenario, we have one of the following two cases:
  - q_m is an element of the q_i dependency list, which is like q_m dependency list is a subset of q_i dependency list case.
  - q_m is not an element of the q_i dependency list. In this case, q_m adds fewer qubits than q_i but guarantees at least one reset qubit, which provides one available qubit for q_i. It is worth to note that, due to the intersection of q_m and q_i dependency lists, some elements of q_i dependency list would have already been added to the target circuit. Now, by adding the remaining elements of q_i, a target circuit with the size of q_i dependency list is created, given that q_m is already in the target circuit. Hence, q_m leads to a final circuit with same or less number of qubits compared to selecting q_i first, contradicting our assumption.

Therefore, by using contradiction, we prove that our algorithm can minimize the size requirement if changing the order of operations is not considered (by only serial execution).

4.4 Complexity Analysis of the Proposed Algorithm

We now study the timing complexity of our proposed algorithm. First, the algorithm must extract the dependencies from the DAG of the circuit. As indicated in Section 2, the roots and leaves of the DAG represent qubits; the remaining nodes represent gates; and the edges capture the qubits of the corresponding gate operations. Assuming a circuit with n qubits and m total gate operations, we need n qubits to check from leaf to root and at most m operations to check for each qubit. Hence, this phase of our algorithm takes $\mathcal{O}(nm)$ to complete.

The subsequent step of the algorithm consists of circuit resizing based on the dependency lists of the qubits obtained in the previous phase and saved as a list of lists named l-list. The outside loop is a while-loop on all n dependency lists of qubits, and inside we sort this l-list based on the dependency list size (each dependency list is an element of l-list), which accounts for $\mathcal{O}(n \log n)$ and may be optimized to $\mathcal{O}n$ if implemented by locating the minimum size element of l-list at each iteration. Note that, each iteration includes the addition of a maximum of m gates, and consequently, this phase

of the algorithm takes at most $\mathcal{O}(mn^2 \log n)$. Overall, the complexity of the algorithm is: $\mathcal{O}(nm) + \mathcal{O}(mn^2 \log n) = \mathcal{O}(mn^2 \log n)$. For example, trying to resize a circuit of size 1000, with one million gates to 100 qubits, takes less than 10 seconds on an Intel core i7 6950X system.

4.5 Iterations vs Shots

As demonstrated in Section 4.4, there is an underlying issue in current quantum hardware that needs to be carefully addressed. In current systems, the quantum circuit is executed over multiple *shots* to attain the probability of success in achieving the correct distribution of results [8]. However, we have observed that, when a circuit containing an MM gate is executed using the same strategy, instead of running the circuit until the final operation, the current systems [14] execute the MM gate for multiple shots upon encountering it, collect the results and continue with the execution of the remainder of the circuit after that. This causes the other qubits to be stalled for the number of shots multiplied by the duration of the executed operations on the measured qubit, which can be substantial in practice. Based on the numbers collected from a sample system of the new generation of ibmq, ibmq_kolkata [14], the readout latency is $675\mu S$ and the best available T1/T2 for the qubits is $214\mu S$ (which is the best value available for this system). If the circuit is stalled for 1000 shots, it means that the other qubits should wait for a minimum of $1000 \times 675\mu S = 675mS$, which is significantly higher than T1/T2 (more than 3000x), causing the other qubits to get $|0\rangle$ state in the process. We want to emphasize that none of the current Dynamic Decoupling [6] techniques can handle an idle period of this magnitude.

To solve this issue, instead of running a circuit over 1000 shots, we execute the circuit using 1 shot and run this circuit in a for-loop for 1000 iterations. By doing so, we solve the issue mentioned above while obtaining the final results. Note that, by using this technique, we are not introducing any new significant change in the current systems; rather, we provide a simple method for solving an issue related to current systems. We would like to encourage the vendors to add the concept of "iterations" into their systems, which can eliminate the need for unnecessary waiting in the queue (1000 shots would translate to 1000 jobs). This can be done by tracking the existing job ID and executing the whole circuit, instead of a portion of it, if the user specifies "iteration count" instead of "shot count".

4.6 Comparison against Concurrent Works

There are two works recently published in arXiv, that also aim to reduce qubit requirements. One of these works [7] was applied on top of ion trap machines with all-to-all connections in an effort to reduce resource usage. For smaller circuits, a SAT-based technique is used to determine optimal utilization, whereas a heuristic method is employed for larger circuits. However, this study does *not* consider one of the main advantages of our strategy, which is to minimize the number of SWAP operations. In addition, our compiler-based approach can outperform the dynamic algorithm in [7] through our greedy strategy, which has been proven to be optimal in this paper. The other work [12, 13] explores a similar approach to fit a program in the selected hardware. Our paper differs from that work in that i) we prove that our algorithm really minimizes the circuit in a 'polynomial' amount of time, whereas the mentioned work does *not*, and ii) we check, via a separate experiment, whether the MM gates are operated in isolation (to

avoid quantum measurement teleportation) to make sure the approach is correct and will not ruin the algorithm logic, and we propose three techniques to achieve that.

5 Experimental Evaluation

We evaluate our proposal under two scenarios. Firstly, by using a small quantum hardware, we serialize the circuits that *do not* normally fit into this hardware using our technique. The goal here is to demonstrate that our approach can be used to execute quantum circuits on quantum hardware with lower qubit capacities. Note that, by default (without our approach) such circuits would *not* execute on the target (small) quantum hardware. Secondly, using a larger quantum hardware, we evaluate and compare our proposal (serialized circuits) to original circuits when *both* of them can be executed on the quantum hardware. This scenario aims at giving a PST comparison of our proposal against the normal (parallel) execution and at revealing the improvements we provide due to the factors such as gate reduction.

5.1 Methodology

We evaluate the effectiveness of our approach using two IBMQ systems: *ibmq_lima* and *ibmq_kolkata* [14]. ibmq_lima is a 5-qubit system that has a T-like network architecture. For simulating ibmq_lima, we execute the experiment using FakelimaV2(), the most recent simulator for the latter system supplied by Qiskit [1, 15].

For simulated ibmq_kolkata, we evaluated our results using FakekolkataV2(), which is the fake-backend for the ibmq_Kolkata hardware. Note that ibmq_kolkata is a new generation IBMQ system with 27 qubits, each having 1 to 3 links connected to it. We used circuits from QASM [5] for our evaluation, which can be accessed at [20].

While our presented results are based on simulation, we want to emphasize that our comparison is *accurate* since we eliminate most of the crosstalk by using serialization. It is because crosstalk occurs when multiple qubits are operated concurrently by different operations (mostly CNOTs) and since most of our quantum circuits can be executed on 2-3 qubits, we are not facing any crosstalk in any of the results presented. For our baseline (parallel execution), on the other hand, there may be some crosstalk cases that are ignored; consequently, the baseline results we report can be 'overestimation', in terms of PST. Therefore, our benefits can be expected to be even *higher* in real quantum hardware.

PST report on ibmq-lima								
Circuit Name	Qubit# in PE	Qubit# in SE	PST	Total Gate#	CNOT Gate#	d_1	d_2	Execution time(mS)
bv_n14 [5]	14	2	51.2%	121	13	117	112	74.54
bv_n19 [5]	19	2	42.2%	166	18	162	157	103.18
wstate_n27 [5]	27	3	35.4%	593	124	182	17	189.21
ghz_state_n23 [5]	23	2	52.2%	70	22	69	8	130.22
swap_test_n25 [5]	25	3	67.6%	482	174	311	31	126.14
cat_state_n22 [5]	22	2	53.2%	66	21	65	8	124.54
rd53_139 [25]	8	5	60.9%	251	140	146	146	68.56
AVG PST(%)	51.8%							

Table 1: PST results for large benchmark circuits on a 5-qubit ibmq-lima machine (our worst-case scenario).

Our serialized execution results are reported using 1000 *iterations*, each containing 1 shot, as discussed in Section 4.5. For our baseline results, on the other hand, all the results are reported using 1000 shots per workload. The mapping policy is set to the default mapping that qiskit/qiskit.transpile employs. We report and compare the results

by using *gate count* and *PST*. Note that gate count is an important metric since it usually tracks the effects of the total gate error. PST is calculated using the formula presented in [18] which as an average PST of expected outcomes.

Table 1 shows our results on a 5-qubit system for benchmarks as large as 27 qubits. While these (original) circuits clearly *cannot* run on 5 qubits in a parallel (normal) fashion, we are able to execute them and obtain reliable outputs by using the proposed strategy. Our results indicate that, on average, we are shrinking the size of the circuits tested by 8.87x while achieving an average PST of 51.7%.

Our results are reported on a 5-qubit system, which is the minimum qubit size for the commercially available quantum systems. We use this system to show that i) while prior works cannot execute these algorithms on small hardware, our approach can execute them and achieve results with high reliability, and ii) there exist some large quantum circuits that benefit from our proposal when targeting even the smallest quantum hardware available. We predict that, by increasing the qubit size and/or using a newer generation of quantum hardware, our proposal can achieve a better PST for a larger set of workloads.

For the Bernstein-Vazirani circuit [3], we report two results with 14 and 19 qubits (the same circuit with two different number of qubits). As shown in Table 1, the PST decreases from 51.2% to 42.2% for 14 qubits to 19 qubits on ibmq_lima, due to increase in coherence error and gate errors predicting a limit on size/depth for reliability. We report 'depth' using two different methods. The first method (d_1), the conventional method used in prior research [4, 11], defines depth as the maximum number of steps between the first operation and the final measurement across all the qubits (DAG layer count). While this method captures the coherence error in 'parallel execution', it does *not* capture well the coherence error in 'serial execution' since in the latter the qubits are *reset* in the middle of the circuit. Therefore, in this work, we define a secondary method, d_2, which is the maximum number of steps between a reset and a subsequent measurement across all the qubits. Note that both these methods report the same value for parallel execution, while they differ for serial execution. Our results indicate that, in the majority of cases, we are even reducing the d_1 due to SWAP reduction when using our technique. By the second definition, a cat_state_n_23, a similar expansion of the circuit shown in Figure 3 on a fully-connected architecture, will have a depth of 23, whereas its serial execution will have a depth (d_2) of 5.

PST report on ibmq-Kolkata											
Circuit Name	PE PST	SE PST	PE Gate #	SE Gate #	PE CX#	SE CX#	PE $d_1 = d_2$	SE d_1	SE d_2	PE exec. time (ms)	SE exec. time (ms)
bv_n14 [5]	26.9%	77.8%	285	121	187	13	68	117	112	45.52	13.66
bv_n19 [5]	21.1%	68.8%	559	166	426	18	126	162	157	73.95	18.88
wstate_n27 [5]	13.7%	56.1%	1016	593	571	124	311	182	17	127.78	68.71
ghz_state_n23 [5]	20.5%	69.8%	307	70	280	22	187	69	8	77.34	22.54
swap_test_n25 [5]	43.9%	53.8%	768	482	484	174	338	311	31	140.40	76.54
cat_state_n22 [5]	31.3%	73%	185	66	159	21	174	65	8	73.33	21.53
rd53_139 [25]	64.5%	67.6%	245	222	137	111	163	156	156	54.53	56.01
Average	31.7%	66.7%	480.7	245.7	301	69	195.29	151.71	69.86	84.66	39.70
PST Gain(%)	210.4% (≃2.1 X)										
Gate reduction(%)	48.9% (≃0.5 X)										

Table 2: Comparison of sequential execution and baseline execution on an ibmq_kolkata (27-qubit system) simulator. PE, SE and exec. time stand for parallel execution, serial execution and execution time, respectively.

5.2 Sensitivity Analysis on Larger Systems

In this part, we compare the results of our serialized execution to baseline (parallel) results on a newer generation quantum system. Table 2 shows the PST and gate count results with the ibmq_kolkata system. We believe that, sequential execution, when applicable, is superior to parallel execution for three major reasons: i) low average number of links on current systems cause excessive SWAP operations in parallel execution (increase in gate counts). Our technique can significantly reduce the severity of this problem; ii) the new generation quantum systems are trending towards improving the measurement gate (readout error), which leads to the minimization of our proposal's overhead; and iii) the new generation systems also have better T1/T2. This causes the coherence error to decrease exponentially, which is the main concern for sequential execution. Therefore, we argue that the current quantum systems are more suited for sequential execution; in fact, their low average link count is not a good fit for parallel execution, which is the state-of-the-art. The fact that serial execution is twice as fast as parallel execution reveals the substantial delay that extra swaps introduce in a system with few links, subsequently lowering system reliability due to increase in gate error. Furthermore, in parallel execution, the prolonged execution time can result in a heightened potential wait time, which in turn will increase the coherence error. Our experimental results indicate that the proposed scheme achieves around 2.1x PST improvement while reducing the number of gates by 48.9%, compared to running the original circuit on the same system. This is because, by serializing the execution, we are reducing the number of SWAP operations/gates needed to migrate the qubits over the links. Compared to the results reported in Table 1, we observe a significant PST improvement with the ibmq_kolkata system. The reason is that the newer generation IBMQ systems have better system characteristics such as readout latency and T1/T2. Additionally, since the newer generation of quantum hardware has lower readout latency, we expect our approach to be more effective in future systems.

6 Concluding Remarks

We present a compiler-based strategy for sequentially executing quantum circuits and downsizing them to the lowest qubit count for execution on smaller systems utilizing MM/MR gates. We demonstrate the correctness of our proposed method, provide a complexity analysis showing that it operates in $\mathcal{O}(mn^2 \log n)$ time, and show its scalability. We report the appropriate level of reliability for a large fraction of the benchmark circuits offered by QASM [19] and propose the notion of 'iteration count' over the frequently-used concept of 'number of shots', to avoid coherence errors and deliver reliable results on a small worst-case system for large benchmark circuits. Finally, we show that, on a modern NISQ system with 27 qubits, our proposed sequential execution can boost reliability by a factor of two (2.1x PST improvement on average) and reduce gate counts and circuit exection time by almost half by minimizing the SWAP counts.

Acknowledgments

We acknowledge the use of IBM Quantum services. The views expressed are those of the authors, and do not reflect the official policy or position of IBM or the IBM Quantum team. The content described in this manuscript is derived from research that has received financial support from the National Science Foundation via Grant Numbers 2119236, 2122155, 2028929, 1931531, and 1763681.

References

1. ANIS, M.S., et al.: Qiskit: An open-source framework for quantum computing (2021). https://doi.org/10.5281/zenodo.2573505
2. Ash-Saki, A., Alam, M., Ghosh, S.: Experimental characterization, modeling, and analysis of crosstalk in a quantum computer. IEEE Transactions on Quantum Engineering **1**, 1–6 (2020)
3. Bernstein, E., Vazirani, U.: Quantum complexity theory. SIAM Journal on computing **26**(5), 1411–1473 (1997)
4. Bhattacharjee, D., Saki, A.A., Alam, M., Chattopadhyay, A., Ghosh, S.: Muqut: Multi-constraint quantum circuit mapping on nisq computers. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). pp. 1–7. IEEE (2019)
5. Cross, A., Javadi-Abhari, A., Alexander, T., de Beaudrap, N., Bishop, L.S., Heidel, S., Ryan, C.A., Sivarajah, P., Smolin, J., Gambetta, J.M., et al.: Openqasm 3: A broader and deeper quantum assembly language. ACM Transactions on Quantum Computing (2021)
6. Das, P., Tannu, S., Dangwal, S., Qureshi, M.: Adapt: Mitigating idling errors in qubits via adaptive dynamical decoupling. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3466752.3480059, https://doi.org/10.1145/3466752.3480059
7. DeCross, M., Chertkov, E., Kohagen, M., Foss-Feig, M.: Qubit-reuse compilation with mid-circuit measurement and reset (2022). https://doi.org/10.48550/ARXIV.2210.08039, https://arxiv.org/abs/2210.08039
8. Farhi, E., Goldstone, J., Gutmann, S.: A quantum approximate optimization algorithm. arXiv preprint arXiv:1411.4028 (2014)
9. Fowler, A.G.: Constructing arbitrary steane code single logical qubit fault-tolerant gates. Quantum Information & Computation **11**(9-10), 867–873 (2011)
10. Fowler, A.G., Mariantoni, M., Martinis, J.M., Cleland, A.N.: Surface codes: Towards practical large-scale quantum computation. Physical Review A **86**(3), 032324 (2012)
11. Gyongyosi, L., Imre, S.: Circuit depth reduction for gate-model quantum computers. Scientific Reports **10**(1), 1–17 (2020)
12. Hua, F., Jin, Y., Chen, Y., Lapeyre, J., Javadi-Abhari, A., Zhang, E.Z.: Exploiting qubit reuse through mid-circuit measurement and reset. arXiv preprint arXiv:2211.01925 (2022)
13. Hua, F., Jin, Y., Chen, Y., Vittal, S., Krsulich, K., Bishop, L.S., Lapeyre, J., Javadi-Abhari, A., Zhang, E.Z.: Caqr: A compiler-assisted approach for qubit reuse through dynamic circuit. In: Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3. pp. 59–71 (2023)
14. IBM: IBMQ System Report guadalupe description. https://quantum-computing.ibm.com/ (2020), accessed on January 2022
15. IBM: Fake provider. https://qiskit.org/documentation/apidoc/providers_fake_provider.html (2021)
16. IBM: How to measure and reset a qubit in the middle of a circuit execution. https://docs.quantum-computing.ibm.com/build/midcircuit_measurement (2021)
17. Kandala, A., Mezzacapo, A., Temme, K., Takita, M., Brink, M., Chow, J.M., Gambetta, J.M.: Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. Nature **549**(7671), 242–246 (2017)
18. Khadirsharbiyani, S., Sadeghi, M., Eghbali Zarch, M., Kotra, J., Kandemir, M.: Trim: crosstalk-aware qubit mapping for multiprogrammed quantum systems. In: 2023 IEEE International Conference on Quantum Software (QSW). IEEE (2023)
19. Li, A., Stein, S., Krishnamoorthy, S., Ang, J.: Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation. arXiv preprint arXiv:2005.13018 (2020)

20. Li, A., Stein, S., Krishnamoorthy, S., Ang, J.: Qasmgit. https://github.com/pnnl/QASMBench (2023)
21. Li, G., Ding, Y., Xie, Y.: 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. pp. 1001–1014 (2019)
22. Li, G., Wu, A., Shi, Y., Javadi-Abhari, A., Ding, Y., Xie, Y.: Paulihedral: a generalized block-wise compiler optimization framework for quantum simulation kernels. In: Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. pp. 554–569 (2022)
23. Litteken, A., Fan, Y.C., Singh, D., Martonosi, M., Chong, F.T.: An updated llvm-based quantum research compiler with further openqasm support. Quantum Science and Technology **5**(3), 034013 (2020)
24. Liu, L., Dou, X.: Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment. In: 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 167–178. IEEE (2021)
25. Maslov, D., Dueck, G.W., Scott, N.: Reversible Logic Synthesis Benchmarks Page (2005), http://webhome.cs.uvic.ca/ dmaslov
26. Murali, P., McKay, D.C., Martonosi, M., Javadi-Abhari, A.: 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. pp. 1001–1016 (2020)
27. Nguyen, T., Mccaskey, A.: Retargetable optimizing compilers for quantum accelerators via a multi-level intermediate representation. IEEE Micro (2022)
28. Ohkura, Y.: Crosstalk-aware nisq multi-programming (2021)
29. Ravi, G.S., Smith, K.N., Murali, P., Chong, F.T.: Adaptive job and resource management for the growing quantum cloud. In: 2021 IEEE International Conference on Quantum Computing and Engineering (QCE). pp. 301–312. IEEE (2021)
30. Ryoo, J., Kandemir, M.T., Karakoy, M.: Memory space recycling. Proceedings of the ACM on Measurement and Analysis of Computing Systems **6**(1), 1–24 (2022)
31. Shi, Y., Gokhale, P., Murali, P., Baker, J.M., Duckering, C., Ding, Y., Brown, N.C., Chamberland, C., Javadi-Abhari, A., Cross, A.W., et al.: Resource-efficient quantum computing by breaking abstractions. Proceedings of the IEEE **108**(8), 1353–1370 (2020)
32. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM review **41**(2), 303–332 (1999)
33. Tannu, S.S., Qureshi, M.K.: 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. pp. 987–999 (2019)
34. Yang, Y., Xiang, P., Kong, J., Zhou, H.: A gpgpu compiler for memory optimization and parallelism management. ACM Sigplan Notices **45**(6), 86–97 (2010)
35. Yu, T., Eberly, J.: Qubit disentanglement and decoherence via dephasing. Physical Review B **68**(16), 165322 (2003)
36. Zhang, C., Chen, Y., Jin, Y., Ahn, W., Zhang, Y., Zhang, E.Z.: A depth-aware swap insertion scheme for the qubit mapping problem. arXiv preprint arXiv:2002.07289 (2020)