A Case for Multi-Programming Quantum Computers

Poulami Das
poulami@gatech.edu
Georgia Tech

Swamit S. Tannu
swamit@gatech.edu
Georgia Tech

Prashant J. Nair
prashantnair@ece.ubc.ca
Univ. of British Columbia

Moinuddin Qureshi
moin@gatech.edu
Georgia Tech

ABSTRACT

Existing and near-term quantum computers face significant reliability challenges because of high error rates caused by noise. Such machines are operated in the *Noisy Intermediate Scale Quantum (NISQ)* model of computing. As NISQ machines exhibit high error-rates, only programs that require a few qubits can be executed reliably. Therefore, NISQ machines tend to underutilize its resources. In this paper, we propose to improve the throughput and utilization of NISQ machines by using multi-programming and enabling the NISQ machine to concurrently execute multiple workloads.

Multi-programming a NISQ machine is non-trivial. This is because, a multi-programmed NISQ machine can have an adverse impact on the reliability of the individual workloads. To enable multi-programming in a robust manner, we propose three solutions. First, we develop methods to partition the qubits into multiple reliable regions using error information from machine calibration so that each program can have a fair allocation of reliable qubits. Second, we observe that when two programs are of unequal lengths, measurement operations can impact the reliability of the co-running program. To reduce this interference, we propose a *Delayed Instruction Scheduling (DIS)* policy that delays the start of the shorter program so that all the measurement operations can be performed at the end. Third, we develop an *Adaptive Multi-Programming (AMP)* design that monitors the reliability at runtime and reverts to single program mode if the reliability impact of multi-programming is greater than a predefined threshold. Our evaluations with IBM-Q16 show that our proposals can improve resource utilization and throughput by up to 2x, while limiting the impact on reliability.

CCS CONCEPTS

• **Hardware: Quantum technologies**;

KEYWORDS

Quantum Computer, Multi-programming, Reliability, NISQ

ACM Reference Format:
Poulami Das, Swamit S. Tannu, Prashant J. Nair, and Moinuddin Qureshi. 2019. A Case for Multi-Programming Quantum Computers. In *The 52nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-52), October 12–16, 2019, Columbus, OH, USA*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3352460.3358287

MICRO-52, October 12–16, 2019, Columbus, OH, USA

ACM ISBN 978-1-4503-6938-1/19/10…$15.00
https://doi.org/10.1145/3352460.3358287

1 INTRODUCTION

Quantum computing can solve classically intractable applications such as breaking cryptographic codes [37], physics and chemistry simulations [7, 20, 32] by using quantum bits (qubits) and quantum-mechanical properties. Recently, IBM, Intel, and Google have showcased prototypes comprising of 50, 49, and 72 qubits respectively [13, 14, 16]. The size of quantum computers is expected to get to hundred(s) of qubits in the near future. Quantum computers require extremely complex control circuitry, sophisticated microwave devices, cryogenic refrigerators that can maintain temperatures of few milli-kelvins, and shielding from environmental noise. Therefore, quantum computers are typically accessible to users via cloud services [2, 9, 34] instead of using a per-user system.

Quantum cloud services provide accessibility, ease of use, and scalability [2, 9, 23, 34]. Cloud providers try to relieve the users from resource management tasks by hiding the complexity involved in operating a quantum computer. The recent advances in quantum algorithms and hardware have fueled research (and commercial) interests, and there is an exponential growth in number of users of quantum computers, far outpacing the growth in number of quantum computers. As the demand for access to quantum computers grow, this would lead to increasing wait-times, and therefore there is a need to increase the throughput of quantum computers, so that the same machine can scale to a larger number of users (or a larger number of experiments from the same user within a shorter period of time). In this paper, we study multi-programming as a means to improve the throughput of near-term quantum computers.

Quantum computers are susceptible to errors. While quantum computers can be made fault-tolerant using quantum error correction (QEC) codes, such codes are expensive (20-100 physical qubits to form a fault-tolerant logical qubit). Therefore, we are unlikely to see QEC being adopted until quantum computers have several thousands of qubits. Consequently, near-term quantum computers are likely to be operated without error correction, in a mode called *Noisy Intermediate-Scale Quantum (NISQ)* [31] computing. In the NISQ model of computing, computations are susceptible to errors and therefore, an application is executed several times (called *trials*) and the final answer is determined statistically. The likelihood that a NISQ machine can execute large programs without encountering an error is quite small, especially due to high error rates of 2-qubit gate operations (CNOT) [40].

As CNOT operations dominate most quantum algorithms, we explain the limitations of CNOT operations on the success rate of a program using an example. The typical CNOT error-rate of current machines is in the range of 2% to 4%. Figure 1(a) shows the probability that a program containing a given number of CNOT operations can be finished without the computation encountering any error. We analyze three different CNOT error-rates, ranging from 1% (optimistic) to 3% (current). For near-term quantum computers, a program with several tens of CNOT operations will have

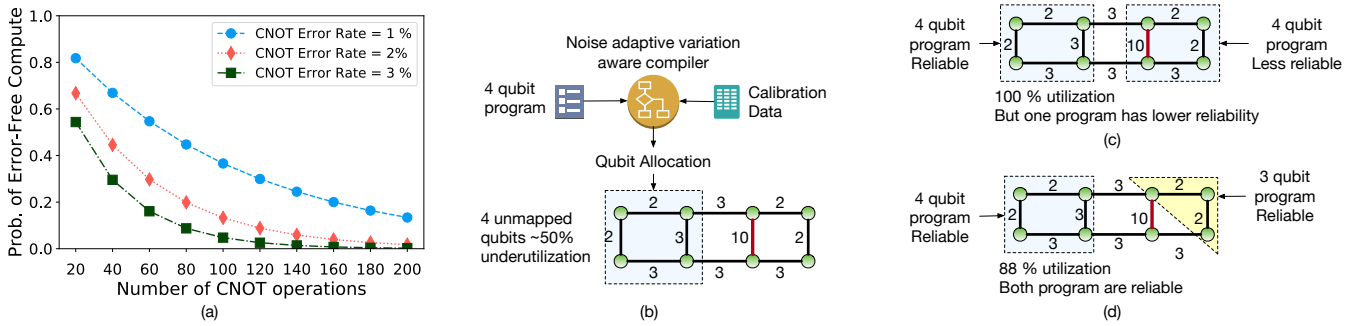

Figure 1: (a) High CNOT error rates on NISQ computers limit the number of operations that can be completed reliably (b) Intelligent compilers use calibration data to locate reliable qubits but can leave resources underutilized (c) Two programs can offer 100% utilization but one will have lower reliability. (d) Two different programs offer 88% utilization but both are reliable.

an extremely low likelihood of being completed without any error. Therefore, it is extremely difficult to run large circuits that use more qubits and greater circuit-depth. To execute applications reliably, compilers use gate fidelity data obtained from machine calibration and map programs onto the most reliable qubits [25, 40]. Even then, only smaller programs with low circuit depths and fewer qubits execute reliably. The problem continues to persist as the size of NISQ computers scale because improvements in gate error rates is extremely slow. Consequently, as shown in Figure 1(b), an application tends to utilize only a fraction of the available resources. The throughput of a NISQ computer can be improved by using the remaining unused qubits to perform computations. To that end, this paper advocates multi-programming for quantum computers, such that multiple independent programs can be executed concurrently on a NISQ machine to improve the system throughput.

Unlike conventional computers, sharing resources on a NISQ computer is fundamentally challenging. When multiple programs execute concurrently on a NISQ machine, the activity of one program can negatively affect the reliability of a co-executing program. So, the interference from resource sharing is not limited to performance but also dictates the overall fidelity of the computation performed by the NISQ machine. The goal of this paper is to propose solutions that improve the throughput and utilization of NISQ machines while limiting the impact on reliability associated with multi-programming NISQ computers. We study three specific sources of reliability impact and develop policies to mitigate them.

To enable multi-programming on NISQ computers, we first study the problem of developing *Fair and Reliable Partitioning (FRP)* algorithms. Existing compiler policies reduce the cost of communication to overcome the restricted connectivity on NISQ computers [19, 38, 41] and perform computations on qubits and links with the lowest error rates [25, 40]. For a NISQ computer that simultaneously executes two or more applications, the compiler may not be able to optimize the reliability of each individual application and an application may end up with executing on error-prone qubits. We observe that not all reliable qubits are co-located in space. Thus, even an intelligent compiler cannot completely avoid all the weak links and qubits. Hence, it is likely that some of the reliable qubits remain unused. However, as shown in Figure 1(c), the reliability of at least one of the programs could be very low on a multi-programmed NISQ computer with 100% utilization. On the contrary, if there exists two separate regions, each with reliable qubits and links, it is possible to map two programs separately onto these regions. As shown in Figure 1(d), such a scheme would help improve qubit utilization without compromising the reliability of either of the two programs.

The second problem we try to address is to reduce the impact of scheduling the measurement operations. When two programs of unequal lengths are executed concurrently, and the shorter program finishes, then the measurement operations of the shorter program can interfere with the longer program. We propose the *Delayed Instruction Scheduling (DIS)* policy that delays the scheduling of the shorter program such that both programs finish their gate operations at similar times, and then performs the qubit measurements.

The third problem we try to address is proactively ensure robustness under multi-programming. When multiple programs are run concurrently, the activity of the NISQ machine is increased, which can lead to increased cross talk noise and cause the error rate of the operations to increase. To limit the reliability impact due to multi-programming, we develop an *Adaptive Multi-Programming (AMP)* design, which monitors the reliability impact at runtime, and reverts the system to isolated execution if the reliability impact due to multi-programming exceeds a certain threshold. AMP allows the system to use multi-programming when it is beneficial, while ensuring that the reliability of programs that are sensitive to concurrency can still be maintained at conventional levels.

Overall, this paper makes the following contributions:

(1) We advocate the use of multi-programming to improve the utilization and throughput of NISQ computers, whereby the qubits are used to concurrently run multiple workloads.
(2) We develop *Fair and Reliable Partitioning (FRP)* algorithms that try to split the qubit resources into multiple groups in a fair manner, while avoiding the qubits/links that have extremely high error rates.
(3) We develop the *Delayed Instruction Scheduling (DIS)* policy to mitigate the interference of measurement operations of one program on the gate operations of the co-running programs.
(4) We propose an *Adaptive Multi-Programming (AMP)* design that monitors the reliability impact at runtime and reverts the system to isolated execution mode if the reliability impact is high.

Our experiments on the IBM quantum computer IBM-Q16 show that multi-programming can improve the throughput and qubit utilization up to 2x with minimal impact on reliability.

2 BACKGROUND AND MOTIVATION

2.1 Basics of Quantum Computing

Quantum computers can solve problems deemed hard on conventional computers [7, 11, 20, 32, 37]. A qubit is the basic unit of information on a quantum computer. Qubits can be in a *superposition* of multiple states which provides parallelism. For instance, a system with n qubits can exist in a superposition of 2^n possible states at the same time. Quantum computers use *entanglement* to create a correlated state over multiple qubits, where manipulating one of them directly impacts the other qubits. In most quantum algorithms, qubits are initialized in a superposition state, manipulated through quantum gate operations, and measured at the end of the program. Measurements collapse qubits to a classical state.

2.2 Errors in Quantum Computers

Qubits are extremely sensitive to noise and prone to errors. Qubit errors can be broadly classified into the following categories:

(1) *Coherence errors*: Qubits retain their quantum state only for an extremely small duration of time (coherence time), limiting the circuit depth that can be run on a NISQ machine.

(2) *Operational errors*: Quantum gate operations are imperfect and cannot be applied with precise accuracy. Operational errors or gate errors refer to errors that occur during computations.

(3) *Measurement errors*: Qubits are measured to retrieve the final result of a quantum program. Measurement errors lead to incorrect results even if a program does not encounter other errors.

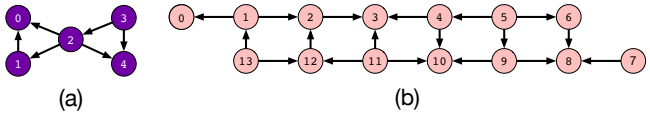

Figure 2: Publicly available quantum computers (a) IBM QX4 Tenerife architecture (b) IBM Q16 Melbourne architecture

2.3 NISQ Model of Computing

Quantum computers can be protected against errors by leveraging Quantum Error Correction (QEC) codes [8, 10, 17], however, such codes require an overhead of 20x-100x (physical qubits per fault-tolerant qubit), so we are unlikely to see systems with such codes until the system reaches a capacity of several thousands of qubits. Figure 2 shows the architectures of current generation of quantum computers from IBM, QX4 *Tenerife* and Q16 *Melbourne*.[1] Quantum computers are expected to scale up to hundreds of qubits in near future [31]. Scaling quantum computers to a large number of qubits is extremely challenging primarily due to two reasons. Firstly, additional noise channels are introduced as systems grow in size. Secondly, the complexity of the circuitry used to control individual qubits increases. Current quantum computers are operated in a mode called *Noisy Intermediate Scale Quantum (NISQ)* computing [13, 14, 16]. We are likely to see NISQ machines over the next decade and such machines are expected to solve important algorithms such as *Quantum Approximate Optimization Algorithm (QAOA)* [6] and *Variational Quantum Eigensolver (VQE)* [22, 30].

[1]IBM Q16 *Melbourne* is a publicly accessible 14-qubit quantum computer from IBM based on superconducting qubits. For the rest of the paper, we refer it as IBM Q16.

2.4 Application Fidelity Under NISQ

Reliably executing applications on NISQ machines is hard because of very high error rates and lack of fault-tolerance. Hence, programs are executed multiple times. Each execution is called a trial. The application reliability of a NISQ application, measured as the *Probability of a Successful Trial* (PST) [40], depends on several factors, such as the number of quantum gates and circuit depth. It also depends on the link error rates, coherence times, and measurement errors of the physical qubits on which the program is mapped [25, 40]. We measure the PST of programs comprising of two-qubit gate and measurement on IBM QX4 and IBM Q16.

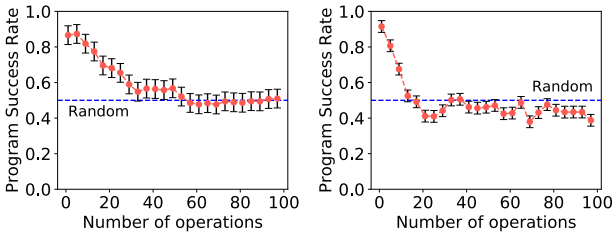

Figure 3: Experimental demonstration of impact of gate errors on program fidelity. Both IBM QX4 (left) and Q16 (right) show an decrease in the program success rate (Probability of Successful Trial) as the number of operations increase.

We form a synthetic program that performs a specified number (N) of CNOT operations on a single link and measure the target qubit. The probability that a random guess will provide a correct answer is 50%. Figure 3 shows the probability of getting the correct answer as the number of CNOT operations in the program is varied for IBM-QX4 and IBM-Q16. We note that once the program has only few tens of CNOT operations, the output of the machine is no better than a random guess. A program can use the CNOT budget by performing few CNOT operations on several qubits (shorter depth) or several CNOT operation on a few qubits of qubits (longer depth). However, a program that has a large number of CNOT operations on a large number of qubits will have intolerable error rates.

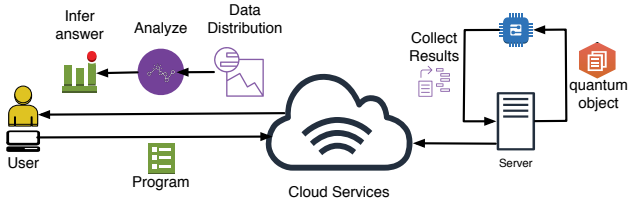

Figure 4: Cloud based user-quantum computer interface

2.5 Quantum Cloud Services and Challenges

Users can access quantum computers through cloud services as shown in Figure 4. Quantum computers have huge deployment, maintenance, and operating costs. This originates from the fundamental resources and infrastructure required to run a quantum computer such as cryogenic coolers, superconducting wires, microwave devices, and sophisticated control circuitry. Cloud services ensure ease of use where users can take their hands off from actual resource management tasks. However, quantum cloud services faces a few key challenges.

2.5.1 Resource Underutilization: Qubit error rates improve extremely slowly and currently 2-qubit gate error rates range in the order of 2-4%. Consequently, only a few operations (ranging in tens) can be executed without encountering errors. The reliability of workloads using larger number of qubits, with greater circuit depths is much lower than smaller workloads. Alternately, smaller workloads can significantly leave resources unused on the quantum chip. The resource underutilization becomes more significant as NISQ computers grow in size and more qubits are available on a quantum chip. For example, allocating a 4-qubit program on IBM Q16 can lead to more than 70% resource underutilization.

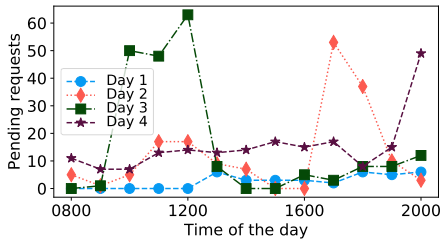

Figure 5: Pending traffic for IBM Q16 on different days

2.5.2 Contention for Quantum Computers: Quantum computing research has accelerated in recent years with demonstrations of prototypes and investments from the industry to leverage these machines for commercial advantage. This has led to an increase in the number of active researchers from a wide spectrum of scientific domains. However, the rate at which quantum computers are being developed lags the rate of increase in the number of users. The limited number of systems leads to increasing contention for access and wait times in the queues can be large as shown in Figure 5. Increasing the rate at which a quantum computer services the requests can allow the system to handle a larger number of requests.

2.6 Proposal: Multi-programming NISQ

In this paper, we advocate multi-programming for improving the utilization of NISQ computers and show that it is practical to concurrently run multiple applications instead of letting the unused qubits remain idle. The increased throughput has added benefits for cloud providers as it helps in servicing jobs quicker as the number of users grow and quantum computers remain limited in number. However, multi-programming a NISQ computer is non-trivial as it can directly impact individual application reliability. When a NISQ computer is multi-programmed, its physical qubits must be distributed between the applications. But, physical qubits exhibit variance in error rates and current compilers [25, 40] account for such variation while performing qubit allocation and movement to improve the reliability. Partitioning a quantum computer to share resources may eventually allocate weaker qubits to an application and restrict a compiler's optimization capabilities for choosing reliable qubit movement paths. A degradation in reliability may also result from additional unintended crosstalk channels [35] and on-going measurement operations. Therefore, multi-programming must be adaptive with a mechanism to detect degradation in reliability. We identify three challenges in multi-programming NISQ machines and provide effective solutions. We describe our evaluation methodology before discussing our solutions.

3 EVALUATION METHODOLOGY

We perform our evaluations on the IBM Q16 quantum computer. To ensure fairness, for any two workloads which are used to compare reliability, all experiments are performed in the same calibration cycle and by launching experiments one after another.

3.1 Figure-of-merit: Trial Reduction Factor

Multi-programming enables sharing of quantum resources and improves machine throughput. When the throughput increases, the effective number of trials performed on both programs combined is reduced. We define *Trial Reduction Factor* (TRF) as the ratio of the number of trials performed when both programs execute individually (baseline) to the total number of trials performed when the programs share resources. If $T_i^{independent}$ is the number of trials performed when the i^{th} program is executed independently and T_{ij}^{shared} is the number of trials performed when i^{th} and j^{th} programs share resources, their TRF is defined by Equation (1).

$$TRF_{ij} = \frac{T_i^{independent} + T_j^{independent}}{T_{ij}^{shared}} \quad (1)$$

3.2 Quantifying Application Reliability

To evaluate application reliability we use the metric: *Probability of a Successful Trial* (PST), defined as the ratio of number of successful trials to the total number of trials performed [40]. For greater reliability, a higher PST is desirable.

$$PST = \frac{\text{Number of successful trials}}{\text{Total number of trials}} \quad (2)$$

3.3 NISQ Benchmarks

For our baseline we use NISQ benchmarks derived from prior works on noise aware compilation policies [25, 40] described in Table 1. Since out of 14 qubits on IBM Q16, only a few of them offer low gate and measurement error rates, we explore small benchmarks. For the evaluation of our proposed policies for multi-programming, we use a set of workload mixes derived from the NISQ benchmarks specified in Table 1.

Table 1: NISQ Benchmarks

Benchmark	Description	Qubits	Number of Instructions	Number of CNOTs
bv_n3	Bernstein-Vazirani [1]	3	8	2
bv_n4	Bernstein-Vazirani [1]	4	11	3
Toffoli_n3	Toffoli gate	3	15	6
Fredkin_n3	Fredkin gate	3	16	8
Peres_n3	Peres gate	3	16	7

3.4 Baseline Setup

For the baseline, we estimate the PST of each NISQ benchmark by executing it independently on IBM Q16 for 8192 trials using the best qubit allocation derived from calibration data.

3.5 Overview of the Proposed Framework

Our evaluation framework is shown in Figure 6. It accepts two workloads W1 and W2 (equivalent to two independent jobs) and the most recent calibration data. Depending upon the number of qubits required for each workload, the partitioning algorithm decides if they can both be executed reliably on the given quantum computer. If there exists two regions where W1 and W2 can be mapped and executed reliably, they are compiled together using the qubit allocations received from the partitioning algorithm and executed. The PST of each individual workload is calculated. The impact on reliability of each program is computed by comparing with the PST obtained by individually executing the same program using the best qubit allocation (baseline).

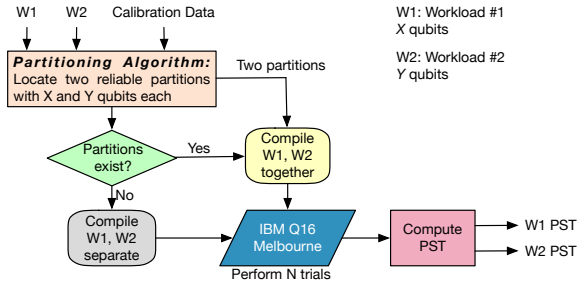

Figure 6: Overview of the proposed multi-programming framework. The partitioning algorithm locates two reliable regions on the NISQ computer, with X and Y qubits each. If it can find two such regions, both workloads execute together. If it is unable to locate the requested regions, it defaults to the baseline and each benchmark is run individually

4 FAIR AND RELIABLE PARTITIONING

We advocate multi-programming NISQ computers to improve the throughput by executing multiple programs concurrently. Since the reliability of a NISQ application depends on the physical qubits allocated to the program, it is important to ensure fairness while allocating qubits to multiple programs in a shared environment.

4.1 Challenges in Fair Resource Allocation

The challenges in fair resource allocation arises from the uniqueness of each physical qubit that is exhibited in the non-uniformity in coherence times, gate and measurement error rates. Furthermore, these error rates vary in time. Thus, the physical qubits allocated to a program directly impacts its reliability [25, 40]. Compilers account for this variation to perform qubit allocation and select qubit movement paths to enable SWAP operations.

4.1.1 Restrictions on Qubit Allocation: Multi-programming constrains the compiler to use a restricted set of physical qubits, limiting its capability to optimize for greater reliability. In order to understand the restrictions imposed on qubit allocation, we look at the allocations of a 4-qubit program P1 and a 5-qubit program P2 on a hypothetical NISQ architecture. As shown in Figure 7(a), when mapped independently, P1 is allocated physical qubits A, B, I, and J

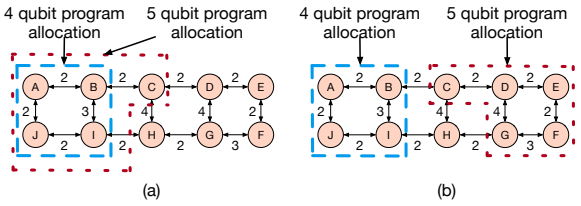

Figure 7: (a) Qubit allocation of a 4-qubit program P1 and a 5-qubit program P2. (b) Qubit allocations of P1 and P2 on a multi-programmed NISQ computer. Each node represents a qubit and label on each edge represents the link error rate.

whereas, P2 is allocated physical qubits A, B, C, I, and J. Figure 7(b) shows a qubit allocation for both programs together. The average link error rate of the regions allocated to P2 for independent execution and in the shared environment are 2.2 and 2.6 respectively. The allocation in the shared environment is 18% weaker.

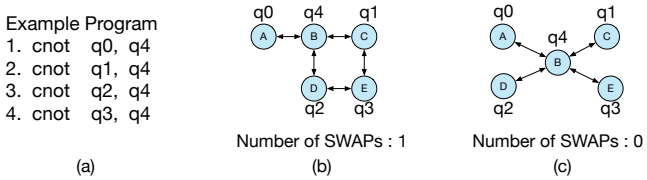

Figure 8: (a) An example NISQ program (b) this topology requires 1 SWAP to perform Instruction 4 (c) this topology does not require any extra SWAP to execute the program

4.1.2 Restrictions on Qubit Movement: Application reliability not only depends on qubit allocation, but also depends on program characteristics and network topology of the allocated region. A well-connected region can minimize the total cost of SWAPs inserted to bring two non-adjacent qubits physically next to each other so that a CNOT gate can be executed. For instance, Figure 8 shows a program that executes 4 CNOT instructions and two possible network topologies. In the partition shown in Figure 8(b), the compiler needs to insert a SWAP operation in order to perform the 4th CNOT instruction. However, a better connected region as shown in Figure 8(c) requires lesser number of SWAPs (in this case 0). When a quantum computer is partitioned for multi-programming, application reliability can vary based upon the number of SWAPs inserted. This depends on the network topology of the assigned partition.

4.2 Qubit Allocation for Multi-programs

We study the average 2-qubit gate error rate on each physical link and measurement error rates for each qubit of IBM Q16 as shown in Figure 9.[2] We make two key observations:

- Not all good links are spatially co-located. A region with good links has weak links as well. For example, qubits *Q2* and *Q12* have two links each with error rates of 4%, but the link that connects them physically has an error rate of 17%.

[2]Error rates in this Figure are based on calibration data collected on 03.14.2018

- Qubits with good connectivity and reliable links can still suffer from high measurement error rates (for example: *Q3, Q11*).
- Qubits connected to link(s) with low gate error rate(s) do not necessarily have a large degree of freedom (number of links) and may also suffer from high measurement errors (for example: *Q7*).

We observe similar trends on IBM Q20 based on previously reported numbers [40] since recent data is unavailable.

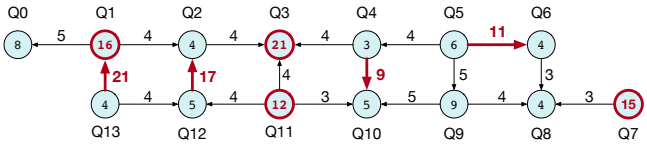

Figure 9: Error rates on IBM Q16 architecture. A node represents a qubit and the label on each edge represents the 2-qubit gate error rate for that link. Links marked in bold have above mean 2-qubit gate error rate whereas qubits circled bold have greater than mean measurement error rate

Reliable qubits are thus usually distributed across the entire architecture, rather than being situated next to each other. Current noise aware compilers try to find a sweet spot by locating reliable qubits as well as allocating program qubits to physically close and well-connected qubits. The latter policy is crucial to minimize the total number of SWAPs inserted. Thus, the compiler is compelled to use some of the qubits and links that may not have the lowest error rates. As a result, some of the reliable links and qubits with lower measurement error rates may remain unused. Therefore, we draw a key insight that as long as there exists more than one reasonably good cluster of qubits on a quantum substrate with similar error rates, it may be possible to run two independent programs on each cluster without significantly affecting their reliability. Using this insight we design a qubit allocation algorithm that partitions the quantum computer for enabling multi-programming.

4.2.1 Fairness in Qubit Allocation: The qubit allocation algorithm described in Algorithm 1 analyzes the underlying architecture and ranks links and qubits depending upon their *utility*, and classifies the physical qubits into 3 groups of high, medium and low utility. *Utility* of a physical qubit is defined as the ratio of the number of links (degree of freedom) to the sum of link error rates. The algorithm chooses a high utility qubit with good neighboring qubits as the root and grows a graph by adding nodes along the boundary. Compiler parameters α and β assists in choosing a high quality root node as described in the Algorithm 1.

Accounting for CNOT error rates: The algorithm locates qubit clusters such that most of the bad links are avoided by the programs together. For example, Figure 10 represents two potential partitions while allocating qubits for two programs with 4 and 5 qubits each. The partition in Figure 10(b) is considered better than Figure 10(a) since it avoids 75% of the weak links. The algorithm achieves this by choosing a different starting rank for generating sub graphs and observing the total number of weak links in both regions.

Accounting for measurement errors: The qubit allocation in Algorithm 1 minimizes the use of qubits with high measurement error

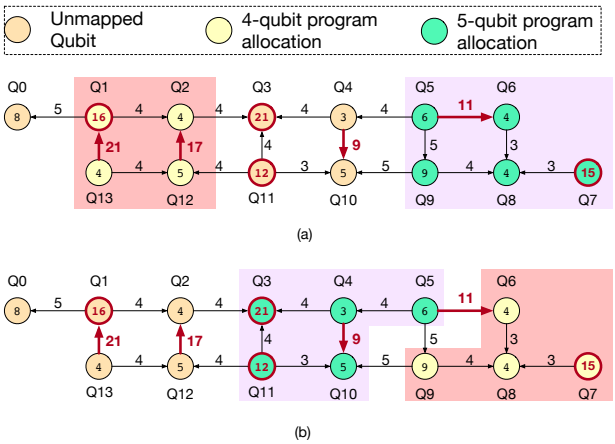

Figure 10: Two possible partitions (a) and (b) for a 4-qubit program and a 5-qubit program that are scheduled to be run concurrently on the NISQ machine. Partition (b) is more reliable compared to partition (a).

rates using the parameter β while allocating qubits. We observe that there is an inherent trade-off involved between gate error rates and measurement error rates in certain regions on the architecture. The algorithm optimizes for gate error rates over measurement errors if a program has large number of 2-qubit gate operations using the compute to measurement ratio (*CMR*).

Accounting for program characteristics: Each quantum program has its own characteristics and uses individual program qubits differently. Our proposed partitioning algorithm accounts for these characteristics while allocating qubits by profiling the *usage* and *interaction graph* of each program qubit. *Usage* of a program qubit is defined as the number of operations performed on it and its *interaction graph* is the set of other program qubits it interacts with. Few quantum algorithms use ancilla qubits in the program that are not measured (for example, the target qubit in Bernstein-Vazirani algorithm). These ancilla qubits are mapped to physical qubits with higher measurement error rates. Program qubits with high usage are mapped to physical qubits with higher utility followed by mapping its neighbors to program qubits from its interaction graph. The process is repeated until all program qubits are allocated.

The qubit allocation obtained from Algorithm 1 is used as the initial mapping by SABRE [19]. SABRE is a recently proposed compiler that maps program qubits to physical qubits and offers low time complexity. We enhance SABRE (called *Variation Aware SABRE*) to use error data instead of Djikstra's distance that is used in SABRE originally to perform qubit movement operations. We also use the reverse circuit of the program to aid mapping ancillae qubits on qubits with high measurement errors. It also assists in scheduling two or more programs under the delayed instruction scheduling described later in Section 5.

4.2.2 To partition or not to partition? Post qubit selection, the compiler analyzes the reliability of each partition and flags a warning if at least one of the programs is allocated qubits with lower reliability as compared to its qubit allocation in an isolated environment. The

compiler uses the *tolerance* factor to decide the difference in reliability it can tolerate. The ability to partition a quantum computer not only depends on the size of the quantum computer, size of the programs in context, and distribution of error rates but also on the daily variation in error rates. Certain regions on a quantum computer exhibiting completely different characteristics on different days thereby limiting the opportunities for multi-programming.

Algorithm 1 Fair and Reliable Partitioning

```
// Locate a reliable cluster on the chip
function CREATE_SUB_GRAPH(Program, utility, CMR)
    rank ← starting rank ; root ← node[rank];
    while root node not found do
        if α% of root's neighbors have high utility and
            β% nodes including root have measurement
            errors < mean measurement error then
            root node found
        else
            rank ← rank + 1
        end if
    end while
    Grow graph from root by adding neighbors along
    the boundary until program can fit and If CMR is
    low, exclude nodes with large measurement errors.
    return sub graph
end function
// Perform qubit allocation for Fair and Reliable Partitioning
function FAIR_AND_RELIABLE_PARTITION(graph,usage, inter-
action)
    if ancillae in program then
        while all ancillae not mapped do
            Allocate ancilla (AnQ) to qubit (PhyQ) in SG
            with high measurement errors. Allocate
            non-allocated neighbors of PhyQ to program
            qubits that interact with AnQ.
        end while
    end if
    while all program qubits not mapped do
        Allocate program qubit PrgQ in order of usage
        to high utility qubit PhyQ in SG. Allocate non-
        allocated neighbors of PhyQ to unmapped
        program qubits that interact with PrgQ.
    end while
    return qubit allocation
end function
```

4.2.3 Scalability: We implement our proposed algorithm and run a mix of workloads on IBM Q16 with error rates shown in Figure 11. For developing the mapping for a system with larger number of qubits, information on reliable regions can be saved once they are located and reused by other programs. As detection of reliable regions only depend on calibration data; it can be done offline. Furthermore, instruction scheduling can be performed in parallel and merged together with appropriate barriers. Similarly, as we are analyzing a cloud environment, the compilation latency can be pipelined with execution of other jobs.

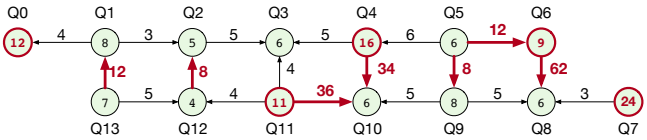

Figure 11: Error rates on IBM Q16 during experiment

4.3 Impact on Throughput and Reliability

With multi-programming for certain workload sizes on IBM Q16, the resource utilization and throughput can be improved by 2x with slight loss of program reliability. The Trial Reduction Factor (TRF) is 0.5, indicating that the throughput of the machine is doubled and the overall number of trials is reduced by 50%. Table 2 shows the PST of each individual workload for the baseline and under multi-programming.

Table 2: Probability of Successful Trial (PST) under isolated execution and under multi-programming

Mix	Number of CNOTs			Baseline PST in %age		Multi-Program PST in %age	
$W_1 - W_2$	W_1	W_2	$W_1 - W_2$	W_1	W_2	W_1	W_2
bv3-peres3	2	8	10	78.4	39.7	69.5	32.7
bv3-toff3	2	9	11	77.0	39.8	73.9	30.6
bv3-fredkin3	2	11	13	77.9	38.8	60.1	41.8
bv3-bv3	2	2	4	78.6	78.2	71.9	73.4
bv4-bv3	3	2	5	23.9	78.0	19.8	73.2
Average PST (%)				61.0		54.7	

Both Figure 12 and Table 2 show that PST decreases by 11% on an average and by 22% in the worst case even when the error rates are very high and there exists a large variance in the error rates as shown in Figure 11. The worst case degradation in PST motivates us to design policies that can minimize the impact of multi-programming on reliability. Our proposed techniques are discussed in Sections 5 and 6.

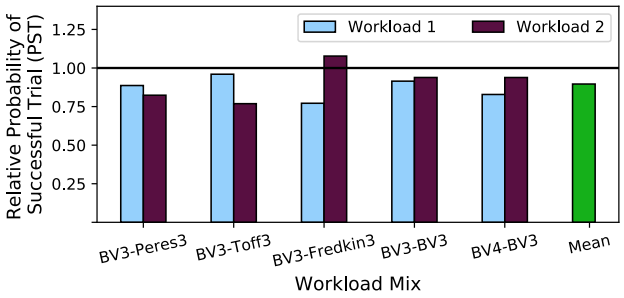

Figure 12: Probability of Successful Trial (PST) relative to baseline for multi-programming model

5 DELAYED INSTRUCTION SCHEDULING

In this section, we discuss interference from measurement as one of the potential challenges in multi-programming a NISQ computer. Qubits are extremely fragile and signals applied to one qubit can leak on to the other qubits causing unwarranted fluctuations in their quantum states. Interference may occur due to the crosstalk introduced by additional operations and qubit measurement operations. We discuss how qubit measurements are performed on current NISQ computers and propose an instruction scheduling policy that minimizes the interference caused by them.

5.1 When Should Qubits be Measured?

Superconducting qubits [4, 18] at 15-20 milli-Kelvins are controlled and measured using microwave signals generated at room temperature. The measurement setup also consists of signal attenuators, amplifiers, and circulators designed to operate at 4 Kelvin. This thermal gradient introduces noise channels into the system. When two programs execute in parallel, conventional wisdom suggests that they be launched as decoupled independent threads as shown in Figure 13(a). In this scheduling, qubits of a program are measured as soon as all its gate operations complete. Unfortunately, on a NISQ computer, two programs cannot be completely decoupled since measurements corresponding to the shorter program can inject noise channels into the operating environment of the co-running program, lowering its reliability. Another possible approach is to schedule measurements after all gate operations are executed as shown in Figure 13(b). This approach is currently used by the IBM *Qiskit* compiler as the default. However, for two programs of different lengths, if the qubit measurements of the shorter program are delayed to protect the program in progress, they may decohere by the time they are measured. To overcome this problem, we propose that two parallelizable programs be context aware.

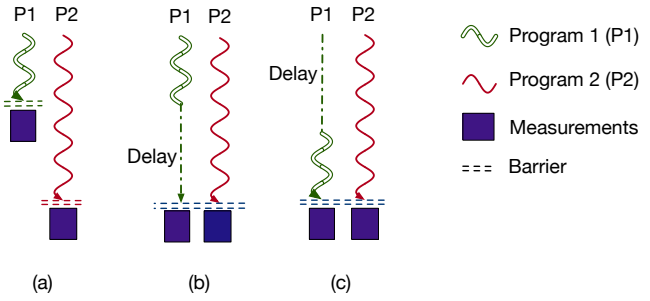

Figure 13: (a) Programmer expects two programs to be decoupled completely (b) Default instruction scheduling from IBM Compiler where all measurements are performed at the end (c) Our proposed delayed instruction scheduling policy where the shorter program thread starts late

5.2 Scheduling Algorithm for Multi-programs

In order to minimize the impact of qubit measurement operations on on-going gate operations and to maximize the reliability of both the applications, we propose a *Delayed Instruction Scheduling (DIS)* policy such that both the applications complete around the same time and all qubit measurements are performed only after gate operations for both programs conclude. The same two programs shown in Figure 13(a) will be scheduled as shown in Figure 13(c) as per the delayed scheduling policy. The scheduling policy moves all the measurements to the end by inserting appropriate *barriers* as shown in Algorithm 2 and statically schedules instructions by analyzing the data flow graph of both the applications. The overall schedule of the multi-programs is generated by Algorithm 3.

Algorithm 2 Instruction Scheduling

```
1:  // Generate schedule for multi-programming
2:  function GENERATE_SCHEDULE(N program schedules)
3:      if Delayed Instruction Scheduling then
4:          Add all measurement instructions for
5:          N programs in order of decoherence times
6:          Add global barrier
7:      end if
8:      for Each program (P_i) in N programs do
9:          Add instruction of P_i to global schedule.
10:         if all instructions of P_i are scheduled then
11:             if Delayed Instruction Scheduling then
12:                 Insert barrier
13:             end if
14:             Decrement N
15:         end if
16:     end for
17:     return global schedule
18: end function
```

5.3 Impact of Measurement Scheduling

We evaluate our partitioning scheme along with the proposed DIS policy. We execute the compiled code on IBM Q16 and determine the PST for the same mix of workloads used in Section 4.2. The results are shown in Figure 14. Our partitioning scheme when combined with the proposed instruction scheduling policy improves the reliability. Overall, the TRF is still 0.5 and the reliability drops by 10% on an average. However, we see an improvement in the reliability of the longer program (3% average improvement in PST) and the worst-case degradation in PST is about 22%. We expect the impact of our policy will be higher when measurements can be done in parallel and takes lower latency than current technologies.

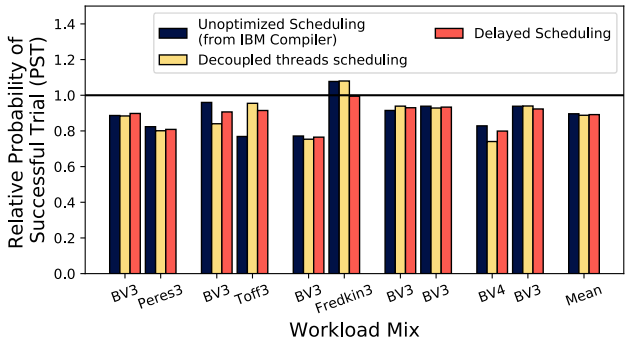

Figure 14: Probability of Successful Trial (PST) of proposed delayed instruction scheduling policy relative to baseline

Algorithm 3 Multi-program Compilation

function (Chip coupling graph, N programs, calibration data)

  for Each physical qubit (phy_i) on chip **do**
    Compute $utility[i] = \frac{\text{Number of links of } phy_i}{\sum \text{Error rates of links of } phy_i}$
    Create 3 utility groups in order of number
    of links and utility
  end for

  if Delayed Instruction Scheduling **then**
    Program $P_i \leftarrow$ Reverse P_i
  end if

  for Each program P_i in N programs **do**
    for Each program qubit ($PrgQ_j$) in P_i **do**
      $Usage[i][j] \leftarrow \frac{\text{Instructions using } PrgQ_j \text{ in } P_i}{\text{Total instructions in } P_i}$
      $Interaction[i][j] \leftarrow$ Set of program qubits
      that interacts with $PrgQ_j$
      Calculate compute to measurement ratio:
      $CMR_i \leftarrow$ Number of Operations
    end for
    Rank Program qubits in order of their usage
  end for

  // Independent qubit allocation and scheduling
  for each program (P_i) in N programs **do**
    $Graph_i \leftarrow create\_sub\_graph(P_i, utility, CMR_i)$
    $qalloc_i \leftarrow fair\_and\_reliable\_partition(Graph_i, usage[i],$
    $interaction[i])$
    $indp\_sched_i \leftarrow variation\_aware\_SABRE(Graph_i, P_i)$
  end for

  // Shared qubit allocation and scheduling
  for each program (P_i) in N programs **do**
    $Shared_i \leftarrow create\_sub\_graph(P_i, utility, CMR_i)$
    Re rank unmapped physical qubits
    $qalloc_i \leftarrow fair\_and\_reliable\_partition(Shared_i, usage[i],$
    $interaction[i])$
    $sched_i \leftarrow Variation\_aware\_SABRE(Graph\_shared_i, P_i)$
  end for

  for each program (P_i) in N programs **do**
    $Glob_Sched \leftarrow generate_schedule($all $sched_i)$
    if mean error rate of all links in
      ($Graph_i$ x tolerance < $Shared_i$) **then**
      Generate *Warning*
    end if
  end for

  return *Glob_Sched, indp_sched for N programs*
  ($sched_i$), Warning

end function

6 ADAPTIVE MULTI-PROGRAMMING

NISQ computers are sensitive to noise and sharing resources to execute multiple programs concurrently can increase crosstalk and interference between programs. Although our proposed FRP algorithm and DIS policy attempts to minimize the impact on application reliability through optimized resource allocation and instruction scheduling, it may still be possible that the reliability of one of the applications is low. Therefore, it is important that there is a mechanism to detect such occurrences and disable multi-programming for those instances. We develop a design that can dynamically monitor a program's reliability at execution time and mitigate the impact on reliability caused by multi-programming by reverting back the program to isolated execution.

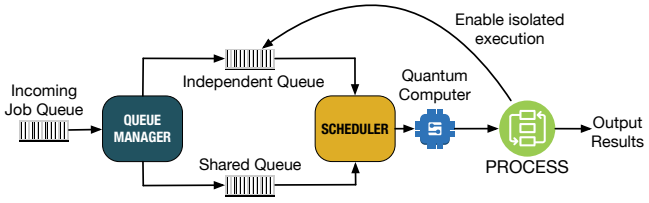

Figure 15: Design of Adaptive Multi-Programming

6.1 Adaptive Multi-Programming Design

We describe a light-weight *Adaptive Multi-Programming (AMP)* design as shown in Figure 15, that resides at the user-quantum computer interface, manages the multi-programming environment, monitors program reliability at execution time, and reverts back a program to isolated execution when the impact on its reliability is beyond an acceptable threshold. The *AMP* design uses the following mechanism to perform the aforesaid tasks:

(1) Multiple programs sharing resources are run together for **S** trials (called *shared trials*)
(2) Each program is individually run for **I** independent trials with the best possible qubit allocation (called *independent trials*)

Our proposed approach can be either implemented by the programmer by writing a parallel program; or by the cloud provider by locating two independent jobs in the requests queue. The *AMP* design accepts jobs from different users in the *incoming job queue*. The Queue Manager (QM) maintains two queues: a) *shared queue and b) independent queue*. Besides accepting incoming jobs from the cloud and managing the additional outgoing queues to the Job Scheduler, it also hosts the compiler. The QM monitors incoming jobs and performs either of the following actions:

(1) Compiles a program and puts its object code in the *independent queue* under any of the four conditions: 1) if it requires access to the entire quantum computer, 2) a user requests exclusive access to entire machine, 3) the program cannot share resources with other programs, or 4) a user has already provided or requested a specific qubit allocation. This gives users an opportunity to control which region their program gets mapped to. Users can still choose to not share resources or request exclusive access to the machine. Rigetti Computing for instance already allows users to reserve specific regions on their quantum computer at a specific time [33].

(2) Monitors the resource requirements of the jobs in the incoming queue and if it finds two independent jobs that can share resources, the QM allocates them qubits as per the algorithm described in Section 4 and compiles them. Post compilation, it puts the object code on the *shared queue*. It also compiles the individual jobs using the best available resources and puts the compiled job object in to the *independent queue*. The size of reliable partitions where programs can be mapped may be predetermined using the calibration data. For larger machines, we expect the number of such available partitions to be higher and therefore more than two jobs can be compiled and executed together.

The scheduler receives job objects from the shared and independent queues and uses arbitration or other advanced scheduling policies to run them on the NISQ computer. Job scheduling is a well-studied problem for current data centers and the adopted policy depends on the agreement between the user and service provider, the pay model, and priority advocated by the service provider. For instance, users requesting exclusive access to the entire machine or a specific region may be required to pay more as opposed to users willing to share resources. Jobs sharing resources can be scheduled earlier than those requiring exclusive access to the machine as long as fairness is guaranteed. Even a simple round robin policy and first-come-first-served approach may be sufficient.

Each job in the *shared queue* is run **S** times, corresponding to the *shared trials*. On the other hand, each job object in the *independent queue* is run **I** times, corresponding to the *independent trials*. Once a program is executed, the Process block analyzes the outputs. The outputs are directly returned to the user for programs that did not share the quantum computer, whereas, the outputs from the *shared trials* are split into two output distributions because it corresponds to programs that shared resources. Then, these distributions are merged with their corresponding distributions from the *independent trials* and the impact on reliability is analyzed before returning the results to the users. The PST of certain workload mixes for different multi-programming modes is shown in Figure 17.

6.2 Dynamic Reliability Monitoring Scheme

The design of our proposed reliability monitor is shown in Figure 16. It resides in the Process block shown in Figure 15. Programs that execute in parallel execute two types of trials: *independent trials* and *shared trials*. The reliability monitor leverages these two types of trials to dynamically detect any significant impact on reliability at execution time. It accepts the distributions from the two types of trials per application and decides to re-execute trials in an isolated environment for an affected program when it detects a deterioration in its reliability. For detection, it uses two statistical metrics: *Entropy* and *Hellinger Distance*.

Entropy [36] describes the amount of disorganization or randomness in a system quantitatively. Entropy in a system is higher in the presence of noise. In quantum systems, reliability or the probability to get the correct answer is highest when the noise is minimum. Depending upon the amount of noise, the system entropy changes. Thus, by computing the difference in the entropy between the two types of trials conducted on a program, the reliability monitor can detect any impact on reliability. If $X = \{x_1, x_2, x_3, ..., x_N\}$ is a set of random phenomena, and $p(x_i)$ is the probability of occurrence of

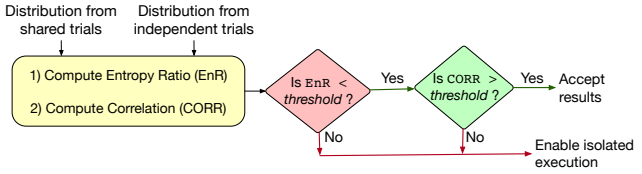

Figure 16: Reliability monitor and controller

x_i, entropy can be computed using Equation (3). In quantum computing context, X is the state space of the N qubits. If $Entropy(I)$ is the entropy of the independent trials and $Entropy(S)$ is the entropy of the shared trials, the entropy ratio, *EnR*, can be computed using Equation (4). A higher *EnR* indicates that the trials conducted while sharing are noisier.

$$Entropy(x) = -\sum_{i=1}^{N} p(x_i) \log_{10} p(x_i) \tag{3}$$

$$\text{Entropy Ratio (EnR)} = \left| \frac{Entropy(S)}{Entropy(I)} \right| \tag{4}$$

Entropy only estimates the amount of noise in the system and cannot capture the similarity between the distributions obtained from both types of trials. For example, two probability distributions $d_1 = \{0.2, 0.2, 0.2, 0.4\}$ and $d_2 = \{0.4, 0.2, 0.2, 0.2\}$ corresponding to the state space of 2 qubits can lead to an *EnR* of 1, even though the merged distribution will have low reliability.

Hence, we also compute the correlation (*CORR*) between these two output distributions, using a metric derived from Hellinger distance. Hellinger distance (*H-Dist*) [12] is used to quantify the similarity between two probability distributions. If $X = \{x_1, x_2, x_3, ..., x_k\}$ and $Y = \{y_1, y_2, y_3, ..., y_k\}$, are two discrete probability distribution functions, the Hellinger distance is computed by Equation (5). It is bounded between 0 and 1. Two identical probability distributions have a *H-Dist* of 0, whereas, the maximum distance of 1 is obtained when there is most dissimilarity. We define *correlation* (*CORR*) using Equation (6). This means, two identical probability distributions have a *CORR* of 1 whereas completely non-identical distributions have a *CORR* of 0. If S and I corresponds to probability distribution functions obtained from shared trials and independent trials, for higher reliability, higher correlation (*CORR(S, I)*) is desirable.

$$H\text{-}Dist(X, Y) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^{k} (\sqrt{p_i} - \sqrt{q_i})^2} \tag{5}$$

$$CORR(X, Y) = 1 - H\text{-}Dist(X, Y) \tag{6}$$

As shown in Figure 16, depending upon the computed *EnR* and *CORR(S, I)*, the monitor performs one of the following:

(1) If *EnR* is lower than a preset threshold and *CORR(S, I)* is greater than a preset threshold, the trials are deemed reliable. The distributions are merged and returned to the user.
(2) If *EnR* is greater than and *CORR(S, I)* is lower than corresponding preset thresholds, the trials are deemed unreliable. In that case, the monitor enables isolated execution for the affected program as shown in Figure 15.

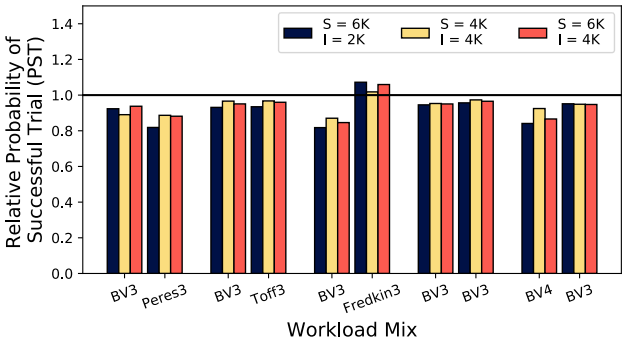

Figure 17: Probability of Successful Trial PST relative to baseline for different multi-programming modes

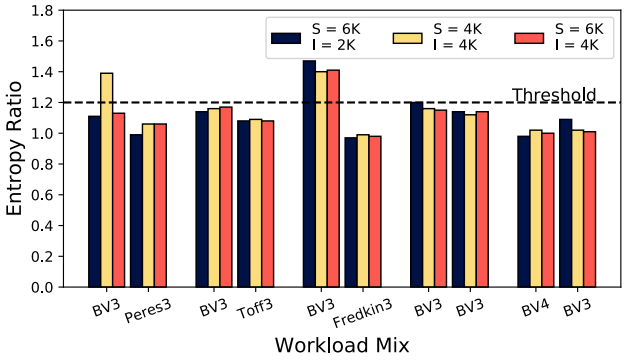

Figure 18: Entropy Ratio (*EnR*) between shared trials and independent trials for different multi-programming modes

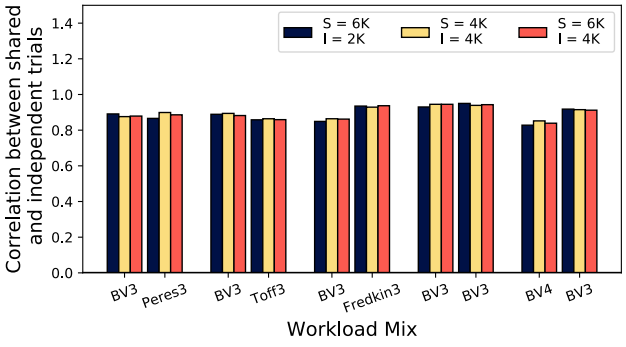

Figure 19: Correlation between shared and independent trails (*CORR(S, I)*) for different multi-programming modes

The threshold for *EnR* depends upon acceptable range of reliability, whereas the threshold for *CORR(S, I)* can be predetermined conservatively for a given program size by profiling typical kernels and benchmarks. For the workload mixes we studied whose PST is shown in Figure 17, the entropy ratio (*EnR*) and correlation (*CORR(S, I)*) are shown in Figures 18 and 19 respectively. We set a threshold for EnR to be lower than 1.2 (accepting 20% more noisy trials). For the thresholds we set, the impact on reliability for *BV3* in mix *BV3-Fredkin3* is detected and can be corrected by retrying. We also observe that in certain cases such as in mix *BV3-BV3*, the *ENR* is extremely close to the threshold, but since *CORR(S, I)* is above the acceptable threshold, the programs are not re-executed. At its heart, the AMP design incurs the overhead of minimal changes to existing job schedulers, additional queues, and logic to merge two distributions making it scalable. The statistical tests in the run-time system involve arithmetic operations on classical machines which are available at data centers. The operations are not on the critical path and can be performed in conjunction with other jobs.

6.3 Impact on Throughput and Reliability

The TRF and reliability varies depending upon the selected multi-programming mode as shown in Figure 20 and Table 3 respectively. The average impact on reliability is within 8% across all the multi-programming modes. We observe that the multi-programming mode S=4K, I=4K offers minimal impact on reliability, and the average and maximum loss in PST are 6% and 12% respectively. We must highlight that extreme PST degradation such as 12% are detected by our dynamic reliability monitor. We must also highlight that the IBM Q16 machine has a huge variance in error rates with 7 out of 18 links and 5 out of 14 qubits having error rates higher than the mean as shown in Figure 11.

Table 3: Probability of Successful Trial (PST) (in percentage) for various modes of multi-programming

Mix	Baseline PST		Mode 1 PST S= 6K, I= 2K		Mode 2 PST S= 4K, I= 4K		Mode 3 PST S= 6K, I= 4K	
$W_1 - W_2$	W_1	W_2	W_1	W_2	W_1	W_2	W_1	W_2
bv3-peres3	78.4	39.7	72.4	32.5	69.8	35.2	73.5	35.0
bv3-toff3	77.0	39.8	71.7	37.2	74.4	38.5	73.2	38.2
bv3-fredkin3	77.9	38.8	63.7	41.6	67.8	39.5	65.9	41.1
bv3-bv3	78.6	78.2	74.3	74.8	74.9	76.1	74.7	75.5
bv4-bv3	23.9	78.0	20.1	74.2	22.1	74.0	20.7	73.9
Average PST	61.0		56.3		57.2		57.2	

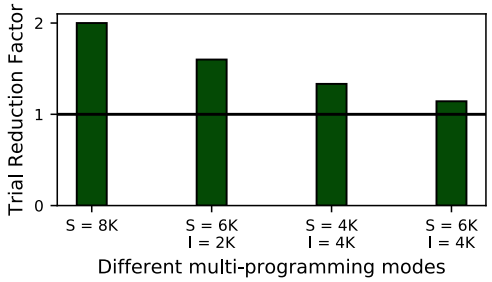

Figure 20: Effective reduction in the number of trials enabled through multi-programming

7 DISCUSSION

7.1 Scalability

Near-term quantum computers are expected to scale up to a few hundreds of noisy qubits in the next few years [31]. The error rates improve very slowly (based on past trends) and non-uniformity in qubit characteristics are expected to be present in the future generations of quantum computers [15, 27] because these variations emerge from material defects introduced into the system during the fabrication of qubits. More importantly, the origin of some of these defects are not yet fully understood and the lithographic processes used to fabricate qubits are still not mature enough to get rid of these imperfections. Consequently, as systems grow in size, resource underutilization scales too. Under such circumstances, our proposed approach can improve the utilization of these larger systems by running more than two programs in parallel.

7.2 Who is Responsible: User or Server?

Our multi-programming solution is scalable and can be implemented by both the programmer and service provider. If cloud providers opt for a utility computing model, programmers can optimize for both reliability and resource utilization by writing parallel programs themselves and requesting a specific qubit allocation. On the other hand, the APM design proposed in Section 6.1 enables cloud providers to improve the throughput of near-term quantum computers by scheduling programs from two independent users.

7.3 Limitations of Our Proposal

The fundamental properties of qubits limit how quantum systems must operate. Our proposed multi-programming policies launch parallel applications in a tightly coupled manner. Thus, the overall time to run them together depends on the length of the longer program. We expect this to be less concerning since cloud providers receive a large number of requests with varying demands.

The ability to locate more than one region for qubit allocation is based on the insight that good qubits are spread across the entire architecture and not always spatial neighbors. Therefore, the capability to improve the utilization and throughput of a NISQ computer depends on the architecture of the quantum substrate and how the error characteristics are distributed in the architecture. However, our proposed solution is adaptive and multi-programming may be turned off when required. The ability to partition a quantum computer also depends on the daily variations in error rates and on certain days the errors may be too large to enable multi-programming.

8 RELATED WORK

Progress in NISQ algorithms and hardware has provided impetus towards building practical applications and rapid commercialization of quantum technologies [2, 24]. It is expected that this growth will bring an interdisciplinary community of researchers together to solve computational problems in the wide range of areas, from high energy physics to financial analysis [5, 26, 29]. To cater to such a diverse set of end users, we will require system abstractions and solutions that can enable seamless access to quantum computers [3, 21]. For example, cloud services are currently being used to hide the complexities involved in operating and maintaining a quantum computer and provide an effective interface that allows users to access these machines. In the absence of fault-tolerance, quantum compilers play a seminal role in the success of near-term quantum computers and several prior works [25, 28, 39–41] have focused on developing compiler algorithms and optimizations that can improve program reliability on these noisy quantum computers. However, there exists a gap between the growth in the number of users of quantum resources and the resources themselves, and designing architectural solutions to close this gap remains an open problem. In this paper, we focused on understanding some of the challenges in sharing quantum resources to improve utilization and reliability of near-term quantum computers. A prior work [39] explores the feasibility of partitioning a quantum computer to run two copies of the same quantum program, whereas, in this paper, we study the ability to run two programs of same or different sizes with minimal impact on reliability.

9 CONCLUSION

Near term quantum computers are operated in the NISQ model of computing. Such machines are characterized by large error rates and can only perform a limited number of gate operations before the fidelity of the computation degrades to unacceptable levels. Consequently, many programs only use a fraction of the available qubits, leaving unused resources on a quantum computer. We propose that these idle qubits be used to run other independent applications, thereby enabling multi-programming and increasing the throughput of NISQ machines. However, NISQ machines are extremely sensitive to noise and multi-programming can impact program reliability. We discuss the challenges in sharing a NISQ machine and propose a Fair and Reliable Partitioning (FRP) algorithm to allocate resources to multiple programs. We also propose a Delayed Instruction Scheduling (DIS) policy to reduce the impact of qubit measurement operations on the on-going gate operations of concurrent programs, when they are of unequal lengths. Lastly, we present an Adaptive Multi-Programming (AMP) design that is embedded with a reliability monitor. This reliability monitor detects instances of severe degradation in program reliability using statistical tests and reverts the program's execution to an isolated mode. Overall, for the workloads we evaluated on the IBM Q16 quantum computer, the throughput can be improved by up to 2x with a maximum loss in reliability of 12%.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their feedback. We also thank Bobbie Manne, Doug Carmean, and Nicolas Delfosse for the technical discussions and feedback. We are grateful to Keval Kamdar, Anshuman Verma, Moumita Dey, Gururaj Saileshwar, and Vinson Young for their editorial feedback. This work was supported by a gift from Microsoft.

REFERENCES

[1] Ethan Bernstein and Umesh Vazirani. 1997. Quantum complexity theory. *SIAM Journal on computing* 26, 5 (1997), 1411–1473.
[2] Davide Castelvecchi. 2017. IBM's quantum cloud computer goes commercial. *Nature News* 543, 7644 (2017), 159.
[3] Frederic T Chong, Diana Franklin, and Margaret Martonosi. 2017. Programming languages and compiler design for realistic quantum hardware. *Nature* 549, 7671 (2017), 180.

[4] John Clarke and Frank K Wilhelm. 2008. Superconducting quantum bits. *Nature* 453, 7198 (2008), 1031.

[5] Eugene F Dumitrescu, Alex J McCaskey, Gaute Hagen, Gustav R Jansen, Titus D Morris, T Papenbrock, Raphael C Pooser, David Jarvis Dean, and Pavel Lougovski. 2018. Cloud quantum computing of an atomic nucleus. *Physical review letters* 120, 21 (2018), 210501.

[6] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. 2014. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028* (2014).

[7] Richard P Feynman. 1982. Simulating physics with computers. *International journal of theoretical physics* 21, 6 (1982), 467–488.

[8] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. 2012. Surface codes: Towards practical large-scale quantum computation. *Physical Review A* 86, 3 (2012), 032324.

[9] Google. 2018. Quantum computing playground. https://opensource.google.com/projects/quantum-computing-playground

[10] Daniel Gottesman. 1997. Stabilizer codes and quantum error correction. *arXiv preprint quant-ph/9705052* (1997).

[11] Lov K Grover. 1996. A fast quantum mechanical algorithm for database search. *arXiv preprint quant-ph/9605043* (1996).

[12] Ernst Hellinger. 1909. Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen. *Journal für die reine und angewandte Mathematik* 136 (1909), 210–271.

[13] Jeremy Hsu. 2018. CES 2018: Intels 49-qubit chip shoots for quantum supremacy. *IEEE Spectrum Tech Talk* (2018).

[14] Julian Kelly. 2018. A preview of Bristlecone, Google's new quantum processor. *Google Research Blog* 5 (2018).

[15] PV Klimov, Julian Kelly, Z Chen, Matthew Neeley, Anthony Megrant, Brian Burkett, Rami Barends, Kunal Arya, Ben Chiaro, Yu Chen, et al. 2018. Fluctuations of energy-relaxation times in superconducting qubits. *Physical review letters* 121, 9 (2018), 090502.

[16] Will Knight. 2017. Ibm raises the bar with a 50-qubit quantum computer. *Sighted at MIT Review Technology: https://www. technologyreview. com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer* (2017).

[17] Emanuel Knill and Raymond Laflamme. 1996. Concatenated quantum codes. *arXiv preprint quant-ph/9608012* (1996).

[18] Jens Koch, M Yu Terri, Jay Gambetta, Andrew A Houck, DI Schuster, J Majer, Alexandre Blais, Michel H Devoret, Steven M Girvin, and Robert J Schoelkopf. 2007. Charge-insensitive qubit design derived from the Cooper pair box. *Physical Review A* 76, 4 (2007), 042319.

[19] Gushu Li, Yufei Ding, and Yuan Xie. 2018. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. *arXiv preprint arXiv:1809.02573* (2018).

[20] Seth Lloyd. 1996. Universal quantum simulators. *Science* (1996), 1073–1078.

[21] Margaret Martonosi and Martin Roetteler. 2019. Next Steps in Quantum Computing: Computer Science's Role. *arXiv preprint arXiv:1903.10541* (2019).

[22] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. 2016. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics* 18, 2 (2016), 023023.

[23] Microsoft. 2018. Microsoft Quantum. https://www.microsoft.com/en-us/quantum/

[24] Masoud Mohseni, Peter Read, Hartmut Neven, Sergio Boixo, Vasil Denchev, Ryan Babbush, Austin Fowler, Vadim Smelyanskiy, and John Martinis. 2017. Commercialize quantum technologies in five years. *Nature News* 543, 7644 (2017), 171.

[25] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. 2019. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 1015–1029.

[26] Engineering National Academies of Sciences and Medicine. 2019. *Quantum Computing: Progress and Prospects*. The National Academies Press, Washington, DC. https://doi.org/10.17226/25196

[27] C Neill, A Megrant, R Barends, Yu Chen, B Chiaro, J Kelly, JY Mutus, PJJ O'Malley, D Sank, J Wenner, et al. 2013. Fluctuations from edge defects in superconducting resonators. *Applied Physics Letters* 103, 7 (2013), 072601.

[28] Shin Nishio, Yulu Pan, Takahiko Satoh, Hideharu Amano, and Rodney Van Meter. 2019. Extracting Success from IBM's 20-Qubit Machines Using Error-Aware Compilation. *arXiv preprint arXiv:1903.10963* (2019).

[29] Roman Orus, Samuel Mugel, and Enrique Lizaso. 2019. Quantum computing for finance: overview and prospects. *Reviews in Physics* (2019), 100028.

[30] Alberto Peruzzo, Jarrod McClean, Peter Shadbolt, Man-Hong Yung, Xiao-Qi Zhou, Peter J Love, Alán Aspuru-Guzik, and Jeremy L OâĂŹbrien. 2014. A variational eigenvalue solver on a photonic quantum processor. *Nature communications* 5 (2014), 4213.

[31] John Preskill. 2018. Quantum Computing in the NISQ era and beyond. *Quantum* 2 (2018), 79.

[32] Markus Reiher, Nathan Wiebe, Krysta M Svore, Dave Wecker, and Matthias Troyer. 2017. Elucidating reaction mechanisms on quantum computers. *Proceedings of the National Academy of Sciences* 114, 29 (2017), 7555–7560.

[33] Rigetti. 2018. Introduction to Quantum Cloud Services. https://www.rigetti.com/qcs/docs/intro-to-qcs

[34] Rigetti. 2018. Rigetti quantum cloud services. https://rigetti.com/

[35] Chad Tyler Rigetti. 2009. *Quantum gates for superconducting qubits*. Yale University.

[36] Claude Elwood Shannon. 1948. A mathematical theory of communication. *Bell system technical journal* 27, 3 (1948), 379–423.

[37] Peter W Shor. 1999. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review* 41, 2 (1999), 303–332.

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

[39] Swamit S Tannu and Moinuddin K Qureshi. 2018. A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. *arXiv preprint arXiv:1805.10224* (2018).

[40] Swamit S Tannu and Moinuddin K Qureshi. 2019. 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*. ACM, 987–999.

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