SOFTWARE TECHNIQUES TO MITIGATE ERRORS ON NOISY QUANTUM COMPUTERS

A Dissertation
Presented to
The Academic Faculty

By

Swamit Tannu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2020

SOFTWARE TECHNIQUES TO MITIGATE ERRORS ON NOISY QUANTUM COMPUTERS

Approved by:

Date Approved: October 22, 2020

Prediction is very difficult, especially about the future.

Niels Bohr

Dedicated to Aai and Baba

ACKNOWLEDGEMENTS

I am grateful for the opportunities, advice, and encouragement that I got throughout my Ph.D. This is my attempt to acknowledge everyone who shaped my Ph.D. and graduate school journey.

This thesis is dedicated to my parents, who encouraged me to learn and taught me to be curious. I thank my parents for inspiring me and believing in me. Their sacrifices and hard work enabled all the possibilities and opportunities in my life. My mother's love of all the sciences and my father's penchant for creative engineering is the sole reason behind my scientific curiosity. I am lucky to have my wife Amruta by my side, and I thank her for being my pillar. All of this would not have been possible without her. I also want to express my gratitude to Amruta's parents, who supported all our decisions and encouraged us.

I am indebted to my advisor, Moinuddin Qureshi, who gave me the freedom, encouragement, and support to explore new research problems. His patience, empathy, and emphasis on asking the right questions enabled an excellent research environment. I am grateful for his advice on research and life in general. Working with him was the most rewarding outcome of graduate school.

I thank my dissertation committee- Kenneth Brown, Vivek Sarkar, Asif Khan, and Tushar Krishna for their time and advice. I am grateful for Ken's encouragement, kindness, and advice. His optimism and vision of quantum computing systems were crucial in shaping my research attitude. I thank Asif for involving me in ECE-8853 and always encouraging me.

I am grateful to Microsoft Quantum Team and Doug Carmean for giving me the opportunity to work on exciting research problems with amazing people. Working for Doug Carmean was the defining moment in my Ph.D. His vision, technical outlook, and leadership inspired me to work on quantum computing. I am thankful for his patience, mentoring, and encouragement. I was lucky to interact with Burton Smith during two of my internships. The breadth and depth of Burton's technical expertise, his relentless technical curiosity, and his work ethic was truly inspiring.

I want to thank Alan Geller, Karin Strauss, Bob Krick, Mike Lewis, Bobbie Manne, for all the technical discussions, advice, and feedback. I am grateful to Alan for answering all my naive questions and all the encouragement. I wish to thank Leo Kovenhoven, who allowed me to visit his lab and spend time with one of the best experimentalists in the world. I thank Gijs De Lange, Wolfgang Pfaff, Guanzhong Wang for all the discussions and help. MSR Internships and my Ph.D. journey would not have been the same without Zach Myers, Stephen Pruett, Elnaz Ansari, and Ben Lin. I thank all of them for their friendship and all the conversations. I am thankful to Chris Chamberland for all the technical discussions.I also want to thank the IBM for making their platform publicly available and their commitment to build a quantum computing community.

I was lucky to be part of the Moin's Research Group. Everyone in the lab influenced my research. I am grateful to Prashant Nair for mentoring me for over a decade. Many times I feel, I am just walking on the path paved by Prashant. His support and feedback helped me before and throughout grad school. I am also thankful to the fantastic seniors Jian Huang, Mohammad Arjomand, and Chia-Chen Chou for their encouragement and feedback. I thank Vinson Young and Gururaj Saileshwar for their friendship and for tolerating my quirks inside and outside the lab. I will miss all the Bufford highway trips, Sunday brunches, and all the fun at conferences. I also want to thank Sanjay Kariyappa and Poulami Das for all the technical and non-technical conversations. I will miss the post-lunch coffee walks to Blue Donkey. I am grateful to Sahil Shah, Karthik Rao, and Ravi Mangal for their advice and friendship. I am hoping I can continue to work with all my friends and keep learning from them. Six years at Georgia Tech were fantastic, and I thank Georgia Tech faculty and ECE staff for making this possible. I am proud to say that I'm a Ramblin' Wreck from Georgia Tech.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

Quantum computers are domain specific accelerators that can provide large speed up for important problems. Quantum computers with few tens of qubits have already been demonstrated and machines with 100+ qubits are expected soon, these machines face significant reliability and scalability challenges. Due to limited and unreliable qubits, these machines are operated in the Noisy Intermediate Scale Quantum (NISQ) mode of computing. The computation on a NISQ machine can produce incorrect output. Therefore, in the NISQ mode, a program is run thousands of times, and the output log is analyzed to infer the correct output. However, the error rates on current quantum hardware are such that the likelihood of obtaining the right answer is still quite small for NISQ machines, and this problem only becomes worse for programs with a large number of instructions. This dissertation shows how the reliability of near-term quantum computers can be improved by developing software techniques.

Our first work exploits the variability in qubit error rates to steer more operations towards qubits with lower error rates and avoid error-prone qubits. We observe variation in the error rates of different physical qubits and links. This can impact the decisions for qubit movement and qubit allocation. Our experiments on the IBM quantum computer show that the device level variability in error rates has a significant impact on the application fidelity, and by carefully mapping the quantum program on physical devices, we can improve the application fidelity. To that end, we present *Variation-Aware Qubit Movement (VQM)* and *Variation-Aware Qubit Allocation (VQA)*, compiler policies that optimize the movement and allocation by accounting device characterization.

In the second work, we show that on the NISQ hardware, correlated errors produce incorrect answers more frequently than the correct answer. This severely impairs the utility of near-term quantum computers as the correct answer may not be distinguished from the incorrect outcomes. Current NISQ compiler policies (including the work on

variation-aware mapping) exacerbate the problem of correlated errors as they are designed to find one best mapping and use that mapping for all the trials. Using IBM Quantum Computer, we show that "one mapping for all trials" policy increases the correlated errors. If the mapping becomes susceptible to a particular form of error, then all the trials will get subjected to the similar error, which can cause the same wrong answer to appear as the output for a significant fraction of the trials. To mitigate the vulnerability to such correlated errors, we propose an *Ensemble of Diverse Mappings (EDM)*. EDM uses diversity in qubit allocation to run copies of an input program with a diverse set of qubit mappings, thus steering the trials towards making different mistakes. The proposed EDM amplifies the correct answer and suppresses the incorrect answers by combining the output probability distributions of the diverse ensembles.

The third work in this thesis develops program transformation to mitigate state-dependent bias in qubit measurement errors. Qubit measurement is typically the most error-prone operation on the commercially available quantum computers. On the IBM quantum computer, measurement errors show state-dependent bias. For example, qubit in a state-1 is more prone to measurement error than state-0. To mitigate this measurement bias and improve measurement fidelity, we propose *Invert-and-Measure*, which transforms the program data from a vulnerable state to a robust state at runtime and then performs the measurement in the stronger state. We propose two designs for Invert-and-Measure. First, *Static Invert-and-Measure (SIM)*, which executes two instances of the program, one with standard measurements and the other with inverted measurements and combines the results. Second, *Adaptive Invert and Measure (AIM)*, which learns the bias of different states using runtime profiling and produces targeted inversions to increase the likelihood of obtaining the correct answer.

CHAPTER 1
INTRODUCTION

1.1 Motivation

Current microprocessors can perform a computational task million times faster than the early microprocessors designed in the 1970s. This enormous improvement in performance was possible due to advances in semiconductor technology and intelligent system design. The innovations in semiconductor technology doubled the number of transistors on a microprocessor chip every two years. Intel-4004, an early microprocessor, used 2,300 transistors. In comparison, the recently launched IBM Power-10 has 18 billion transistors. Computer designers and architects used this growing number of transistors to maximize the performance of the processors.

To minimize the time to solve a computational problem, we can (1) Decrease the time required for a single computational operation. (2) Perform operations in parallel. (3) Reduce the number of total operations. Computer architects leveraged shrinking transistor feature size with aggressive pipelining to reduce the latency of operations. Furthermore, using efficient microarchitectural designs, architects leveraged instruction-level and memory-level parallelism to reduce time to solution. The growing number of transistors also enabled packing more cores to run multiple tasks concurrently. Using hardware and software techniques, designers have pushed the performance of existing high-performance computing systems beyond petaFLOPS.

Summit, one of the largest supercomputers, can perform 10^{17} operations every second using more than 200,000 cores and two petabytes of DRAM [1]. Although impressive, many computational problems are still beyond the capabilities of supercomputers like Summit. For example, to find prime-factors of a 1000-bit number, we will need about 10^{150} steps in the worst-case. Even when we use all the Summit cores, the time required to factor will be more than the age of the Universe. The high computational complexity is not unique to the prime factorization. There are numerous real-world problems, which require exponentially scaling time. Unfortunately, such problems do not become tractable by reducing the instruction latency or exposing more parallelism at the hardware-level. This limitation is fundamental. As the order of the total number of steps required to solve any computational problem is independent of the computer's design and only depends on the algorithm. Quantum computers promise to solve problems that are beyond the capabilities of conventional computers.

Quantum computer uses properties of quantum bits (qubits) to solve problems that are hard to solve on conventional computers. Qubit device properties enable efficient quantum algorithms. For example, Shor's algorithm on a quantum computer can factor a 1000-bit number within one day. Furthermore, quantum computers can efficiently simulate molecules and materials. Simulation of molecules is at the core of many important applications – from discovering new medicines to building efficient solar panels and manufacturing environment-friendly fertilizers. Unfortunately, simulating a molecule on a conventional digital computer requires exponentially scaling memory and time. Quantum computer using qubits' innate ability to manipulate quantum states provides an efficient way to simulate quantum systems and speed up problems beyond conventional computers' reach.

Quantum computers are domain-specific accelerators. Like conventional accelerators and co-processors, we need a host processor and software layers to execute an application on the quantum computer. In this thesis, we focus on building system software for existing and near-term superconducting quantum computers. In the last two decades, quantum computing has moved from theoretical ideas to realizable systems (albeit at a small scale). The last three years represent significant milestones in quantum computing as Google and IBM demonstrate quantum computers with 50 plus qubits [2, 3], and IBM gave access to their quantum hardware via a cloud service. These prototype quantum computers provide an opportunity to understand the challenges in building practical quantum computing systems and use these insights to improve the design of future quantum computers.

Quantum hardware is unreliable, as qubit devices are fickle and encounter errors. Qubits are used to represent and manipulate a large state space using quantum superposition. Unfortunately, quantum superposition can be easily perturbed, resulting in an erroneous quantum state. Qubits can be protected against errors using *Quantum Error Correction Codes (QEC)*. Unfortunately, QEC requires significant overheads, typically incurring 10-100 physical qubits to encode one fault-tolerant qubit. Existing and near-term quantum computers with tens to hundreds of qubits may not have the capacity to utilize QEC due to the limited number of qubits. Such quantum computers with 10 to 1000 noisy qubits are termed as *Noisy Intermediate Scale Quantum computers (NISQ)* [4]. Even though NISQ machines lack fault-tolerance, they can still provide benefits for a class of quantum applications [5].

The NISQ machines can produce an incorrect output as the computation is subjected to errors. Therefore, to infer the correct answer, the program is run thousands of times on the NISQ machine to produce a probability distribution of the possible output states. However, NISQ machines are significantly limited due to hardware errors. On the IBM quantum computers, the average operational error rate is more than 0.1%. The high operational error rate limits the number of error-free operations we can perform before error corrupts the program state and produces incorrect output with high probability. To leverage the near-term quantum computers with hundreds of qubits, it is essential to mitigate hardware errors.

1.2 Thesis Statement

The limited reliability of quantum hardware poses a significant challenge in leveraging near-term quantum computers. By understanding the nature of hardware errors on a quantum computer, we can design software techniques to mitigate hardware errors.

1.3 Thesis Contributions

The Quantum compiler decomposes the quantum functions into a sequence of quantum operations. Compilers aim to minimize the number of operations to improve program reliability. In addition to improving decomposition, this thesis demonstrates software techniques to mitigate errors by leveraging the noise characteristics of quantum hardware. This thesis makes the following contributions.

1.3.1 Variability-Aware Compiler Policies to mitigate hardware errors

The hardware error rates have high variability on existing quantum computers as some qubit devices are more likely to encounter errors. In the NISQ model, to ensure correctness, all the computations are necessary to be error-free. Thus high error rate on a few qubits can significantly degrade the application fidelity. Mapping more operations on reliable qubits substantially improve the application fidelity. However, mapping operations on the physical qubit devices is challenging due to hardware constraints.

Quantum computers use entanglement to generate correlated states. The creation of an entangled state is a fundamental building block of all quantum algorithms. A two-qubit operation can generate an entangled pair of qubits. However, to execute a two-qubit operation on the hardware, input quantum states must be located on the qubit devices that are connected via a physical link. Unfortunately, on most quantum hardware platforms, qubit devices have only nearest neighbor connectivity, and entangling two non-neighboring qubits is not natively supported by the hardware. However, by using *data movement (known as SWAP) operations* in software, qubit variables can be moved from a device to another such that any two-qubit variables can be entangled.

SWAPs are unreliable, and inserting SWAPs increases the likelihood of errors. The number of SWAPs can be minimized by finding the best possible qubit allocation (mapping of program variables to hardware qubits) and sequence of SWAPs (routing qubits from one location to another for entanglement). Thus, researchers are designing compilers for SWAP minimization. Unfortunately, the SWAP minimization policy does not always demonstrate increased program reliability with decreasing SWAPs on real quantum hardware.

SWAP minimization policies assume that all physical links have an identical failure rate. On the contrary, a significant variation exists in the error rates among qubits and physical links. Our experiments on IBM quantum computers show worst-case error rate can be up to 5x higher than the average error rate. We exploit this non-uniformity in the device error rate to improve application reliability. We propose variation-aware qubit movement and allocation policies that optimize the movement and allocation of qubits to avoid unreliable qubits and links and guide more operations toward reliable qubits and links.

Our evaluations on the IBM machine demonstrate up to 1.9 times improvement in application reliability. Moreover, variability in the error rate is not limited to IBM machines. The characterization of Google's 53-qubit machine also demonstrates significant variability in errors. This variability results from fabrication defects and drifts in operating conditions, and it cannot be eliminated by better device engineering alone. To improve application reliability, we need consolidated system-level efforts.

1.3.2 Diversifying Quantum Programs for Robust Inference

NISQ computers can produce incorrect output, as the computation is subjected to errors. The applications on a NISQ machine attempt to estimate the correct output by running the same program thousands of times and logging the output. If the error rates are low and the errors are not correlated, the correct answer can be inferred as the one appearing with the highest frequency. Unfortunately, quantum computers are subjected to correlated errors, which can cause an incorrect answer to appear more frequently than the correct answer.

We observe that recent work on qubit mapping (including work on variation-aware mapping [6]) attempts to obtain the best possible qubit allocation and uses it for all the trials. This approach significantly increases the vulnerability to correlated errors—if the mapping becomes susceptible to a particular form of error, all the trials will be subjected to the same error, which can cause the same wrong answer to appear as the output for a significant fraction of the trials.

To mitigate vulnerability to such correlated errors, we proposed a compiler transformation that leverages the concept of diversity using *Ensemble of Diverse Mappings (EDM)*. EDM introduces diversity in qubit allocation to run copies of an input program with a diverse set of mappings, thus steering the trials toward making different mistakes. By combining the output probability distributions of the diverse ensemble, EDM amplifies the correct answer and suppresses the incorrect ones. Experiments with the IBM-Melbourne (14-qubit) machine show that EDM improves the relative strength of the correct answer by up to 2.3 times.

1.3.3 Program Transformations for Mitigating Quantum Measurement Errors

Near-term quantum computers may not have enough quantum bits (qubits) to enable fault-tolerant hardware using quantum error correction, and on such NISQ computers, mitigating hardware errors is a primary challenge in enabling quantum speedup. On current IBM quantum hardware, qubit measurement is the most error-prone operation; it has an average measurement error rate of 4 to 8%. Moreover, near-term quantum algorithms that can solve hard optimization and chemistry problems are significantly vulnerable to measurement errors.

On IBM quantum hardware, some quantum states are more vulnerable to measurement errors than others because of repeatable data-dependent measurement bias; the error rate for measurement changes depending on the state being measured. For example, measuring an all-zero state ("00000") on IBM's five-qubit machine has a fidelity of 84%, but this fidelity drops to 62% while measuring an all-one state ("11111").

We developed a compilation technique that transforms the data from a vulnerable state to a stronger state and then performs the measurement. For example, when state "11111" is measured, inverting and measuring ("11111" → "00000") improves the fidelity from 62% to 78%. Moreover, to handle arbitrary bias, we designed an adaptive transformation that learns the error rate for different states using runtime profiling and produces targeted inversions to increase the likelihood of obtaining the correct answer.

The proposed compiler transformation improved reliability (the odds of finding error-free output) three times on IBM quantum hardware. The proposed ideas are not limited to IBM hardware, as a 53-qubit Google machine reported bias in measurement errors (up to 10 times difference in error rate for measuring state "0" *vs.* "1") [7], which can be exploited to mitigate measurement errors.

CHAPTER 2

BACKGROUND

This chapter will discuss the key primitives for gate based quantum computers, programming model and system architecture of near-term quantum computers.

2.1 Quantum Computing Primitives

In this section, we will discuss the fundamental properties of qubits, quantum gates, and qubit measurements.

2.1.1 Quantum Bits

Conventional computers use binary data representation to store and process information. In contrast, a quantum computer represents data using quantum bits (qubits). To understand the difference between a digital bit and a quantum bit, consider a sphere shown in the Figure 2.1. To represent a state of digital bit on this sphere, we can pick any two points on the sphere. For example, the north pole and the south pole on the sphere can represent "1" and "0" respectively. Whereas, the state of a qubit can be viewed as an arbitrary point on the sphere. In the Figure 2.1(a), state of qubit ψ is represented as vector $|\psi\rangle$. This vector $|\psi\rangle$ represents a state of single qubit, is a linear superposition of two basis states $|0\rangle$ and $|1\rangle$, which are orthogonal vectors such that $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α and β are complex numbers. Similarly, the state of two qubits would require four complex numbers as this state ϕ would be superposition of four orthogonal basis vectors $|\phi\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle$ and the state of N qubits would be a superposition of 2^N basis states - $|000..0_n\rangle$ to $|111..1_n\rangle$ represented as a complex vector with 2^N elements. This ability to represent exponentially scaling state with linearly increasing qubits enables a quantum computer's computational power. In quantum programs,

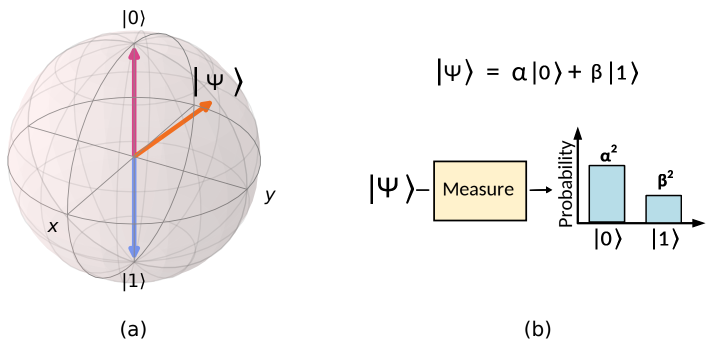


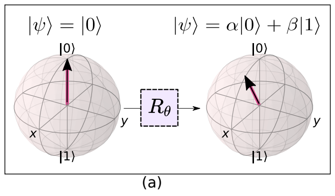
Figure 2.1: (a) Bloch Sphere representation of qubit state (b) Quantum measurement operation

a collection of N qubits can be used as a qubit register, which can be of arbitrary size and, we can perform bitwise qubit operations on the qubit registers.

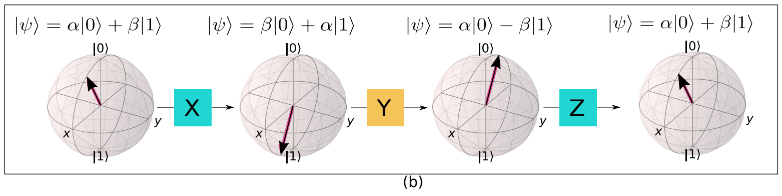
2.1.2 Quantum Measurement

Measurement operation reads the state of the qubit and yields a binary result with the probability depending on the state of the qubit. As shown in Figure 2.1(b), when a qubit with $|\psi\rangle$ is measured, it produces "$|0\rangle$" with probability of $|\alpha|^2$ and "$|1\rangle$" with probability of $|\beta|^2$. Similarly, reading n-qubit state outputs 2^n basis states($000..0_n$ to $111..1_n$) with probabilities corresponding to the collective state.

The measurement of qubit provides partial information. The actual state is a probability distribution, and a qubit measurement is analogous to one sample drawn from this distribution. Moreover, during the process of qubit measurement, the state evolves randomly and losses the original information. Thus to estimate a quantum state, we have to prepare the identical state and measure it repeatedly.



$|\psi\rangle = |0\rangle$
$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$
R_θ
(a)



$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$
$|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$
$|\psi\rangle = \alpha|0\rangle - \beta|1\rangle$
$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$
X
Y
Z
(b)

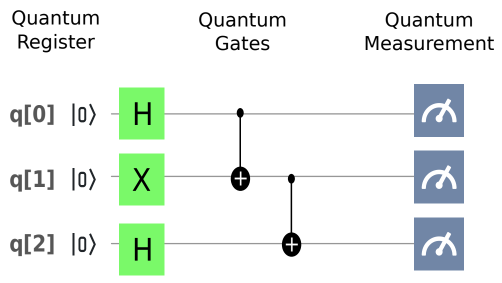
Figure 2.2: (a) Single qubit gate rotates the state vector (b) Sequence of single qubit gate can perform any desired transformation

2.1.3 Quantum Gates

A quantum gate is used to manipulate the qubit state. Ideally, any arbitrary unitary operator can be used as a quantum gate. This unitary is not constrained in size and can be applied on an arbitrarily large number of qubits. However, most quantum hardware platforms support single-qubit and two-qubit gates. Fortunately, we can emulate larger unitary operations using a sequence of the single-qubit and two-qubit gates.

A single-qubit gate rotates the qubit state on the Bloch sphere as shown in the Figure 2.2(a). Theoretically, any arbitrary rotation is possible. However, six single-qubit gates: X, Y, Z, H, T, S are most commonly used in quantum algorithms. X-gate rotates the quantum state, $|\psi\rangle$ by 180^0 about X-axis such that the probability amplitudes α and β are fliped. On the other hand, Z-gate rotates the state vector around the Z-axis by 180^o; this results in inversion of phase, as shown in the Figure 2.2(b). Hadamard (H) gate rotates the state by 90^o along both the X and Z axis. When applied to a qubit in $|0\rangle$ state, the H gate produces a state with an equal superposition of $|0\rangle$ and $|1\rangle$. The T-gate and S-gate introduce phase by rotating state about Z-axis by $\frac{\pi}{4}$ and $\frac{\pi}{2}$, respectively.

It is possible to perform multi-qubit operations. For example, quantum gate which operates on two physical qubits (two-qubit gate) can be used to create a correlated or entangled quantum state. Most quantum computers support *Controlled-NOT (CNOT)* operation. *CNOT* operation is performed on two qubits. The inputs of *CNOT* gate are termed control qubit and target qubit. *CNOT* inverts the state of the target qubit based on the value of the control qubit and does not change the value of the control qubit. For mathematical notation and more details on the type of quantum gates, refer Appendix A.



(a)

```
qreg q[3];
creg c[3];

H q[0];
X q[1];
H q[2];
CNOT (q[0],q[1]);
CNOT (q[1],q[2]);

measure (q[0],c[0]);
measure (q[1],c[1]);
measure (q[2],c[2]);
```

(b)

Figure 2.3: a) Quantum program represented as a circuit (b) Quantum program described using quantum assembly (qasm) language.

2.2 Quantum Circuits

Quantum circuits can describe and visualize quantum programs. Quantum circuits are a sequence of quantum gates applied on the qubit registers, as shown in the Figure 2.3(a). A practical quantum circuit has four components - Qubit Register, Gate Operations, Measurement Operations, and Classical Registers. The sequence of gate operations evolves the collective qubit state of the qubit register in time. The measurement operation generates a binary string upon measurement, and a classical register is used to store the output. To estimate the output, a quantum circuit is executed multiple times, and the output binary string is logged for every trial. Quantum circuits can also be visualized using the quantum assembly (qasm) program as show in the Figure 2.3.

The Quantum circuit representation does not support loops, and operations from left to right represent qubit evolution in time. Moreover, existing quantum computers do not support conditional instructions at runtime. However, existing quantum development libraries mix high-level host languages (such as python) and domain-specific syntax to synthesize quantum circuits at compile time. Note that quantum programs are statically compiled to generate quantum executable circuits that are fully unrolled.

On a quantum computer, copying a qubit is not allowed, and measurement destroys

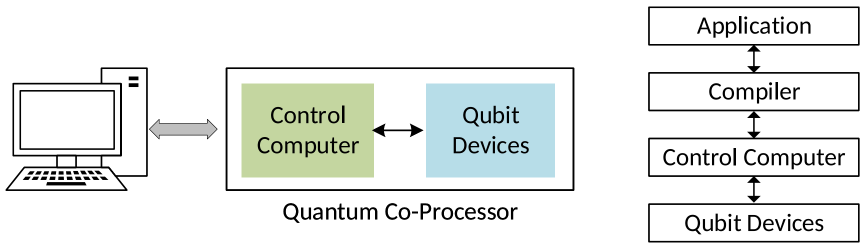


Figure 2.4: a) Quantum co-processor consists of qubits and control computer. Qubits store the information.(b) Simplified quantum computing stack.

the state of the qubit state. These constraints can be effectively represented in the form of a quantum circuit where the total number of qubits is fixed, and once measured, the qubit variable collapses to the classical bit. In this dissertation, we will use the term quantum program and circuit interchangeably.

2.3 Organization of Quantum Computer

Quantum computers are domain-specific accelerators, which can speed up a class of problems by enabling efficient algorithms. Like any other accelerator, quantum computers are connected to a traditional host computer that offloads quantum executable onto the quantum computer. As shown in the Figure 2.4(a), a quantum computer has two components: qubits and control computers. Qubits hold the quantum state, and the control computer manipulates the qubit state by performing quantum gates. In a quantum program, the state of a physical qubit is treated as a quantum variable, and a sequence of quantum operations are used to evolve the quantum state. At the end of the quantum program, the qubits are read by the control computer, and the output is sent back to the host machine.

Figure 2.4(b) shows a simplified computing stack for quantum computers. A quantum program is compiled for a specific target device. The compiler converts the sequence of operations described in the program into low-level gate pulses. The control computer applies theses gate pulses to perform quantum operations. In this thesis, we will focus on compiler techniques, which transforms the original program to mitigate hardware errors. Appendix B summarizes the typical organization and architecture of existing quantum computers.

2.4 Hardware Errors on Quantum Computers

In this section, we will discuss challenges in running quantum algorithms to solve practical problems. Quantum states are fundamentally fickle as the quantum effects are observable only at extremely low energy levels, and even a small noise in the system can disrupt the expected behavior of qubits. During the execution of the quantum program, qubits can encounter Coherence-errors, Gate-errors, and State Preparation and Measurement (SPAM) errors.

2.4.1 Coherence Errors

Coherence errors result from a natural tendency of qubit devices to attain the lowest possible energy state. Coherence errors are analogous to retention errors in conventional systems. However, conventional computers are only subjected to bit-flip errors, whereas quantum computers can experience bit-flip, phase flip errors, amplitude damping, and small rotational errors. The coherence times for current quantum computers are short. For example, on IBM quantum computers, T1 coherence time is about 75μS.

2.4.2 Gate Errors

Quantum operations or gates manipulate the state of a qubit. Unfortunately, quantum gates are not perfect as performing operations on qubits can result in undesired state changes. For example, on an IBM quantum computer, a single qubit gate that is used to manipulate the state of an individual qubit can encounter an error with a probability of 0.1% such that there is about one in thousand chance that a single-qubit gate operation would produce an undesired state change. Whereas, a two-qubit gate that entangles the state of two quantum bits show an average error rate of 4% on IBM quantum computers. The two-qubit operational errors are one of the most dominant forms of errors on quantum computers as they limit the number of operations we can perform before a program encounters an error.

2.4.3 SPAM Errors

Current quantum computers are susceptible to State Preparation and Measurement (SPAM) Errors. For instance, on IBM machine, all qubits are initialized to "$|0\rangle$" state at the beginning of the program. Unfortunately, there is a small chance that a qubit may not be correctly initialized. This is known as the state preparation error. Similarly, reading the state of a qubit can be erroneous. Qubit is a superposition of two basis states:$|0\rangle$ and $|1\rangle$. When measured, qubit produces a binary output: either 1 or 0, depending on the degree of superposition. Unfortunately, the process of measurement is erroneous as sensing the state of the qubit is challenging due to the extremely low energy associated with the qubit. On IBM machines, the average qubit measurement error rate is 5% to 8%, whereas the worst-case measurement error rate can be up to 30%.

2.5 Quantum Error Correction

One of the most significant challenges towards building a scalable quantum computer is the fundamentally fickle states of the qubits. The state of a qubit is analog in nature, and even a small perturbation can modify this state. Fortunately, enabling fault-tolerance in quantum computers is possible using quantum error correction (QEC). The fundamental idea behind QEC is if error modes are limited, then using redundancy in information encoding, we can detect and correct the qubit errors. By generating a correlated entangled state and continuously evolving it, we can create a logical qubit that effectively restricts qubit errors to behave as bit-flip errors or phase flip-errors rather than arbitrary errors. Such phase-flip and bit-flip errors can be detected and corrected using ideas similar to classical error correction.

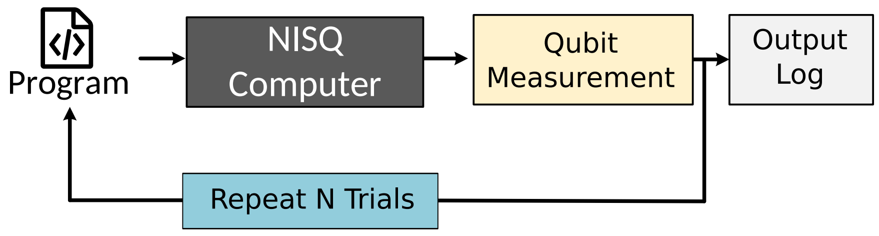


Figure 2.5: Iterative Computing Models for NISQ.

QEC utilizes redundancy in encoding quantum information. For instance, a QEC uses 10 to 100 noisy physical qubits to construct one fault-tolerant logical qubit. The logical qubit can tolerate high operational and decoherence errors by continuously detecting and correcting errors. QEC dominates the tasks that the control processor has to carry-out as QEC protocols execute operations on all qubits at all times. Furthermore, QEC adds extra-steps to ensure fault-tolerance for logical operations. The quantum computing model that assumes fault-tolerance is known as Fault-Tolerant Quantum Computers (FTQC). Building programmable FTQC quantum systems is an open problem.

2.6 Near-term Quantum Computers

Near-term quantum computers with 100 to 1000 qubits do not have enough qubits to leverage QEC completely to achieve fault-tolerance. Such quantum computers with a noisy and limited number of qubits are termed as Noisy Intermediate Scale Quantum (NISQ) computers [4]. Many applications highlighted by Morello [5] can still be viable with such NISQ computers by relying on application properties to perform useful work. Figure 2.5 describes a general computing model for the NISQ. In this model, the given program is run multiple times, and the output of each trial is stored in the output log. As long as the correct results appear with non-negligible probability, we can infer the correct results by analyzing the output log. Quantum application domains such as quantum chemistry simulations and optimization can leverage algorithmic error-resilience and utilize NISQ machines. By mitigating errors on the hardware, these applications can outperform the existing conventional computers.

Designing NISQ machines to suppress noise is of paramount importance as the reliability of the NISQ machines directly correlates to their usefulness. This thesis focuses on compiler policies that can improve the reliability of existing and near-term quantum computers operated in NISQ mode.

CHAPTER 3

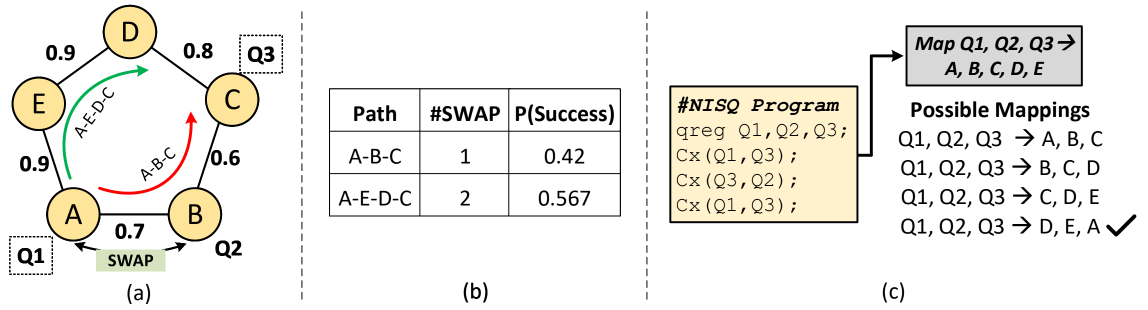
VARIABILITY-AWARE COMPILER POLICIES TO MITIGATE HARDWARE ERRORS

3.1 Introduction

Quantum entanglement is essential for enabling quantum advantage. An entangled qubit state is generated by coupling a pair of qubits using a two-qubit operation. On a superconducting quantum hardware, we can entangle only the qubits that are connected via a physical link. Existing commercial quantum computers are designed using networks that offer limited link connectivity, only to a few of the neighboring qubits, and this connectivity dictates the qubits that can be entangled.

For example, Figure 3.1(a) shows a hypothetical quantum computer with five qubits where circular nodes represent the qubits and edges represent the coupling links between qubits. A pair of qubits can only be entangled if there exists a coupling link between them. Fortunately, quantum computers provide a *SWAP* instruction that can exchange the state of two neighboring qubits. For example, we want to entangle data qubit Q_1 and data qubit Q_3 which are initially residing at physical qubit-A, and physical qubit-C respectively. We can perform this operation in two steps: first swap the data between qubit-A and qubit-B such that Q_1 and Q_2 interchanges positions. Next, entangle qubit data Q_1 and Q_3. In quantum programs, a large number of SWAP instructions are inserted to move data so that entanglement between arbitrary qubits can be performed.

The *Qubit-Movement* policy deals with the problem of selecting a route to move the state of one qubit to another. For example, in Figure 3.1(a), we may choose the route A-B-C for going from A to C, as doing so would minimize the number of SWAP operations. The *Qubit-Allocation* policy deals with the problem of mapping of program qubits to the physical qubits. For example, in Figure 3.1(a), if we want to map three program qubits



Path	#SWAP	P(Success)
A-B-C	1	0.42
A-E-D-C	2	0.567

```
#NISQ Program
qreg Q1,Q2,Q3;
Cx(Q1,Q3);
Cx(Q3,Q2);
Cx(Q1,Q3);
```

Q1, Q2, Q3 →	A, B, C
Q1, Q2, Q3 →	B, C, D
Q1, Q2, Q3 →	C, D, E
Q1, Q2, Q3 →	D, E, A ✓

Figure 3.1: (a) A hypothetical quantum computer with five-qubits – the number on the edge denotes the success probability when that edge is used (b) Variation-Aware Qubit Mapping (VQM) can use more SWAP instructions and yet have higher probability of success (c) Variation-Aware Qubit Allocation (VQA) selects the mapping that improves overall system reliability.

to five physical qubits, we would choose any of three connected qubits (for example, Q_1 maps to A, Q_2 maps to B, and Q_3 maps to C), as placing qubits nearby results in efficient movement. Prior studies [8, 9, 10] have proposed qubit allocation policies based on minimizing the number of SWAPs. These studies assume uniformity in the cost of performing SWAPs. However, in reality, we expect variation in the behavior of different qubits and links, and optimizing for a uniform behavior may not result in the best policy when device variation is taken into account.

To understand the degree of variation in the error-rates of different qubits and links, we analyze the *publicly-available* characterization data for the IBM-Q20 (20 qubits) machine. Such a characterization is performed for the IBM-Q20 several times a day, and we analyze the data for 52 days. We present the statistics of coherence time for all the 20 qubits, the error rate in performing single-qubit operations, and the error-rate in performing two-qubit operations across different qubits. For all these metrics we observe significant variation in the behavior of different qubits and links – in essence, qubits and links are not created equal. For example, our detailed analysis for the links connecting different qubits show that the error rates can vary by as much as 7x across different links in the system. Such variation can have a significant impact on the overall system reliability (Section 3.3).

To analyze the impact of variation on the overall system reliability, we use the *Probability of Successful Trial (PST)* metric. The PST metric indicates the probability that the program finished successfully without any error. As IBM-Q20 is not open to the public, we build a reliability evaluation infrastructure to compute the PST for the IBM-Q20 machine using the machine configuration and error rates based on the characterization data. Our evaluations show that the device variation has a significant impact on the system reliability. To improve system reliability, we should steer more instructions and movement to strong qubits and links, and fewer instructions and movement on weaker qubits and links. We propose such *Variation-Aware* policies to exploit the variation in the behavior of qubits and links, assuming error-rates are known at compile time (Section 3.4).

We propose *Variation-Aware Qubit Movement (VQM)* policy that routes the qubit from source to destination based on minimizing the probability of failure. For example, in Figure 3.1, the success probability of each link is denoted as a weight of the edge. Let us assume, we want to entangle data qubit Q_1 and data qubit Q_3. A conventional variation-unaware policy will use a path that minimizes the number of SWAP instructions, taking the path A-B-C, resulting in an overall probability of success of 42% for these operations. With VQM, we would take the route A-E-D-C, even though this route has more SWAP instructions, since it has an overall probability of success of 56.7%, as shown in Figure 3.1(b). VQM shows a significant improvement in PST (Section 3.5).

We also propose *Variation-Aware Qubit Allocation (VQA)* policy that performs the mapping of program-qubit to physical-qubit with the aim of improving the overall system reliability. For example, in Figure 3.1(c), we want to allocate three program qubits to 5 physical qubits. A conventional mapping policy can choose any of the listed mapping possibilities as they all would have similar cost in terms of SWAP operations. However, with VQA, we would use the mapping D, E, A, as this mapping uses the strongest links, and would improve the overall system reliability. We extend prior proposals for Qubit-

Allocation with VQA and show that VQA+VQM can improve the PST of the IBM-Q20 by up to 1.7x (Section 3.6).

In addition to the simulation-based studies for IBM-Q20, we evaluate our proposed policies on a real quantum machine (IBM-Q5) and demonstrate that our policies continue to provide a significant improvement on the system reliability even in a realistic setting. VQM+VQA improves the PST of the IBM-Q5 by up to 1.9x (and average 1.36x) (Section 3.7).

We also perform a case study, where we analyze programs that require less than half the available qubits, and we have an option of either executing two copies of the program concurrently (to increase the rate of trials) or executing only one copy but map the work on strongest qubits and links (to improve the PST of the given trial). We demonstrate that, in certain cases, having one strong copy has better overall performance (successful trials per unit time) than having two concurrently running copies. Thus, variation-awareness can enable intelligent partitioning for NISQ machines (Section 3.8).

3.2 Data Movement on Quantum Computers

On IBM quantum machines, two-qubit operations are performed using a coupling-link that connects two qubits. For practical reasons, superconducting quantum computers do not allow all-to-all connectivity between the qubits and use a restricted network (such as Mesh) that allows connectivity between only the neighboring qubits. The network structures impose constraints on which qubits can be entangled. Fortunately, there are SWAP operations that can move the qubit from one location to another and enables entanglement of any two arbitrary qubits. Even if the quantum machine does not provide a native SWAP instruction, it can be accomplished using three CNOT gates.

Ideally, we want to engineer quantum computer where each qubit is connected to every other qubit and such unrestricted connectivity would allow any two arbitrary qubits

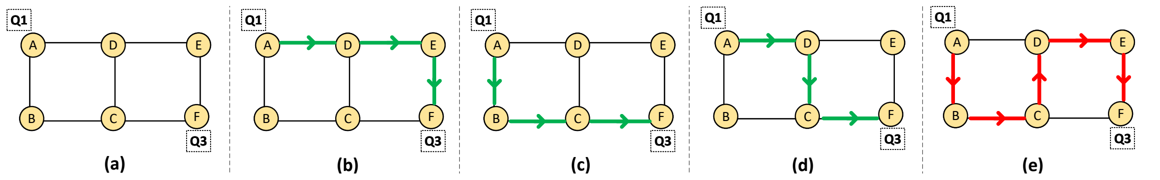


Figure 3.2: (a) Layout of a 6-qubit quantum computer, (b)-(e) are possible routes from A to F. Note that options (b)(c)(d) have an identical number of swaps and (e) incur higher swaps. An intelligent policy would choose one from (b)(c)(d).

to get entangled. Unfortunately, such an organization would require $O(N^2)$ links, which is impractical even for the 49-72 qubit machines that are available today. The links in a quantum machine are not just wires, but resonators that operate at a dedicated frequency, and having a large number of such circuits operate reliably on the chip is a difficult task. Therefore, almost all qubit machines use a Mesh network (or a variant that allows diagonal connections). Such networks restrict that the movement of qubits can occur only between neighboring qubits. For example, for the hypothetical 6-qubit machine shown in Figure 3.2(a) there is no direct connection between qubits A and F. The communication between these qubits must happen via intermediate qubits. Such restrictions give rise to the two sub-problems: (a) Qubit-Movement policy, and (b) Qubit-Allocation policy.

Qubit-Movement Policy: This policy decides the route that should be used while moving the data from one location on the chip to another. Given that such movement is done using SWAP instructions between neighboring qubits, it is reasonable to select the route that minimizes the number of SWAP instructions. Figure 3.2(b)-(e) shows the four possible routes from A to F. The first three (b)-(d) requires only 3 SWAP operations, while (e) requires 4 SWAP operations. The policy may arbitrarily pick one of the routes from (b)-(d).

Qubit-Allocation Policy: This policy decides the initial mapping of program qubits to the data qubits. For example, it is preferred that qubits that communicate frequently be placed near each other. For example, if we wanted to place 4 qubits on the machine shown in Figure 3.2(a), we would not keep these qubits on the four corners, and instead we will try to use the middle two qubits (D and E), as doing so would minimize the SWAPs, required for communication. In fact, recent studies [8, 9, 10] have proposed such allocation policies based on minimizing the number of SWAPs.

In this work, we use the compiler developed by Zulehner et al. [9] as the baseline for both Qubit-Allocation and Qubit-Movement. Our baseline compiler compiles the quantum program for given connectivity to generate the instruction schedule (with additional extra swaps) and initial program-qubit to physical-qubit mapping. It is designed to minimize the number SWAPs by using a greedy search algorithm. Baseline policy for Qubit-Movement and Qubit-Allocation assume uniform cost (specifically reliability impact) in performing SWAP operations. However, in reality, there can be significant variation in reliability of qubits and the links. Policies that take this variation into account can provide better overall system behavior (performance, reliability etc.) To enable such variation-aware policies, we first analyze the publicly available characterization data for the IBM-Q20 machine as IBM-Q20 has the most number of qubits for which characterization data is publicly available.

3.3 Error Characteristics of IBM Quantum Computers

Qubits are fickle as even a small perturbation in the environment can change the state of a qubit. The error rate for a qubit can be defined as the probability of undesired change in the qubit state. Errors in quantum computers can be classified into two categories: retention-errors or operational-errors. Performing operations on qubits can also affect their state incorrectly due to errors, as quantum operations are not perfect. For example, an instruction that rotates the state by some desired angle can introduce extra erroneous rotation. Operational error-rate is defined as the probability of introducing an error while performing the operation [11]. For publicly available quantum-computers from IBM, the single-qubit instruction error-rates are of the order of 10^{-3}, whereas for two-qubit instructions, such as CNOT, it is 10^{-2}. A typical quantum program contains a significant number of two-qubit operations, and given the error-rate of two-qubit operations are an order of magnitude higher than for the single-qubit operations, the two-qubit operations usually dominate the overall error rate. On all superconducting quantum computers, the error-rates significantly vary across physical qubit devices. To understand and quantify the variation in the error-rates of different qubits and links, we analyze characterization data for the IBM-Q20 (20-qubit) machine. IBMQ API provides access to qubit device error rates [12]. We monitored the twenty qubit IBM system for 52 days (between 01/20/2018 to 03/30/2018) and gathered more than 100 different characterization reports. The characterization reports consist of error-rate for all single-qubit operations, two-qubit operations (link errors), and measurement operations. IBM machines are calibrated (one or more times) every day and error-reports are updated after each calibration cycle.

3.3.1 Distribution of Coherence Times

Both T1 and T2 coherence time of a qubit depends on several design, manufacturing and experimental parameters. Due to process variation, biasing and temperature drifts the coherence time can vary significantly. Figure 3.3 shows the T1 and T2 distribution of IBM-Q20. The data is collected for all 20 qubits over 100 observations (so a total of 2000 data points are plotted in the graph). The mean and standard deviation for T1-Coherence time are 80.32μS and 35.23μS respectively. The mean and standard deviation for T2-Coherence time are 42.13μS and 13.34μS respectively.

3.3.2 Error-Rate for Single-Qubit Operations

Single qubit operations rotate the quantum state from one point to other on a state-sphere. On IBM machine, it is performed by applying a microwave signal with a set duration and frequency on the qubit device. Unfortunately, qubit devices are highly

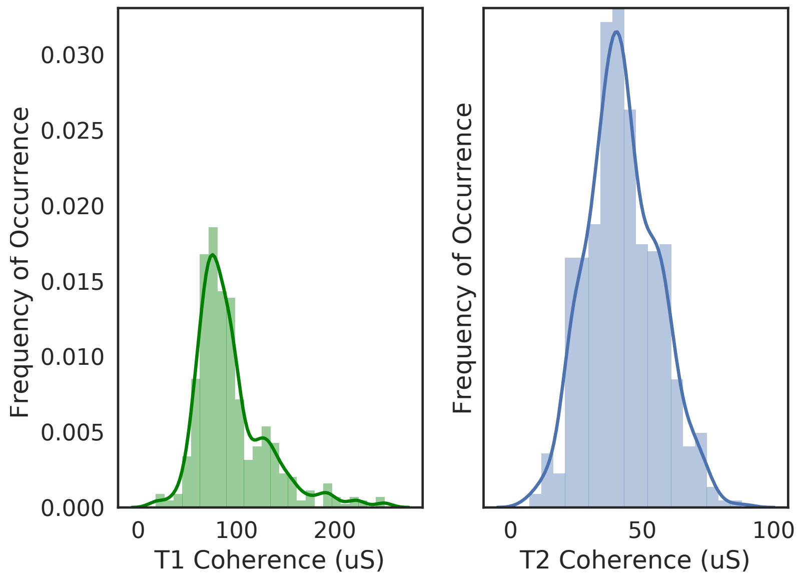


Figure 3.3: Distribution of (a) T1 Coherence time (b) T2 Coherence time for all 20 qubits with 100 samples per qubit.

non-linear and a small perturbation or experimental conditions can cause drift in device characteristics. This can cause variation in the robustness of the quantum operations. Figure 3.4 shows the distribution of error-rate for single-qubit operations. The data shows a large fraction of the error-rate below 1%. Single-qubit operations are more robust than two-qubit operations.

3.3.3 Error-Rate of Two-Qubit Operations

Two-qubit operations are essential to entangle quantum states and move the state of the qubits. In IBM quantum computers, two-qubit operations are performed by applying microwave pulses on target devices, control qubit devices as well as on the coupling link that connects the two. Similar to single-qubit operations, two-qubit operations suffer from variation in error-rate i.e. there is a fraction of coupling links significantly unreliable than most of the links. We analyze the reliability of two-qubit operations for the IBM quantum computer. Figure 3.5 shows the distribution of the error-rate of two-qubit operations for the 20 qubit machine. It consists of data from 76 coupling links col-

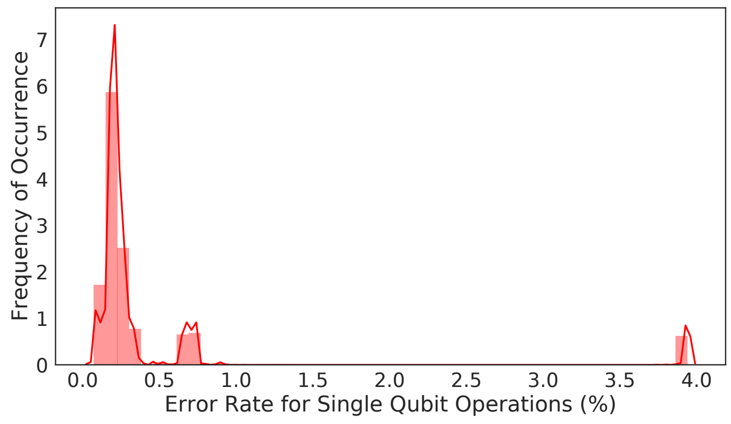


Figure 3.4: Distribution of the error-rates of single-qubit operation for all 20 qubits with 100 samples per qubit.

lected over 100 calibration cycles. The mean two-qubit error-rate is 4.3% and standard-deviation is 3.02%.

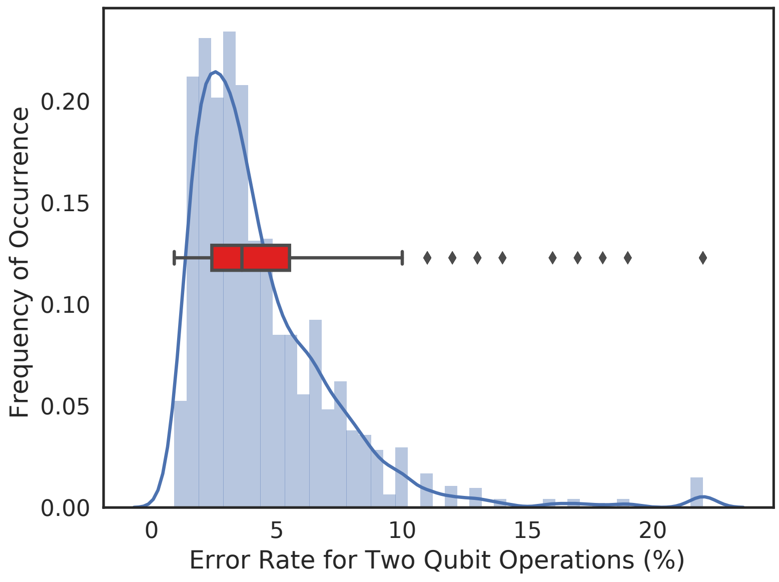


Figure 3.5: Distribution of the error-rates of two-qubit operations for all 76 links of IBM-Q20. The data consists of 100 observations for per link (so a total of 7600 datapoints).

3.3.4 Temporal Variation in Two-Qubit Gate Errors

Error-rate of a link can change with time. IBMQ-20 are frequently re-calibrated to ensure that the characterization is reliable. However, a qubit and the associated coupling links can change their behavior across two different calibration cycles. For example, a qubit pair with a low error rate on one day can have opposite behavior on the other. This might result from tuning parameters, drifts, and other experimental factors. Figure 3.6 shows a time-series of error-rate for three coupling-links. From this data, we observe that error-rate of the links tend to retain their mean error characteristics and stronger links tend to remain strong.

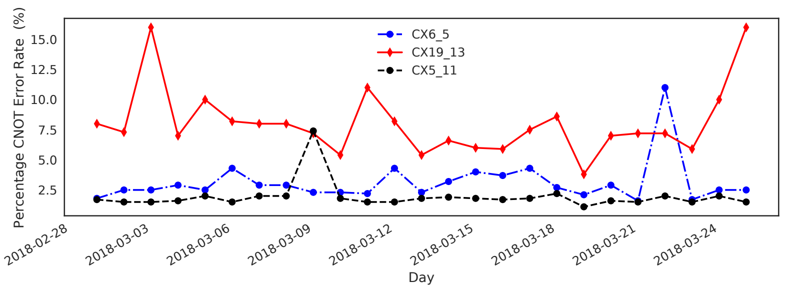


Figure 3.6: Temporal variation in error rate of two-qubit operations for three links. For most periods, the strong link tends to remain strong and the weak tends to remain weak.

3.3.5 Spatial Variation in Two-Qubit Gate Errors

Figure 3.7 shows the layout of the IBM-Q20 qubit computer. Circular nodes represent the qubits and the edges represent a coupling link that is used for performing a two-qubit operation between a pair of qubits. The weight on the edge shows the failure rate of the link and indicates the average probability of failure of the link. For example, the link between $Q14$ and $Q18$ has the highest probability of failure (0.15) and there are several links with a probability of failure as low as 0.02. Thus, there is a variation of 7.5x between the failure rate of the strongest links versus the weakest link.

We observe a significant variation in coherence times, and error-rate of single-qubit and two-qubit operations. Given that the data for this variation can be obtained using characterization (which is performed periodically), we can use the variation data and develop variation-aware compiler policies.

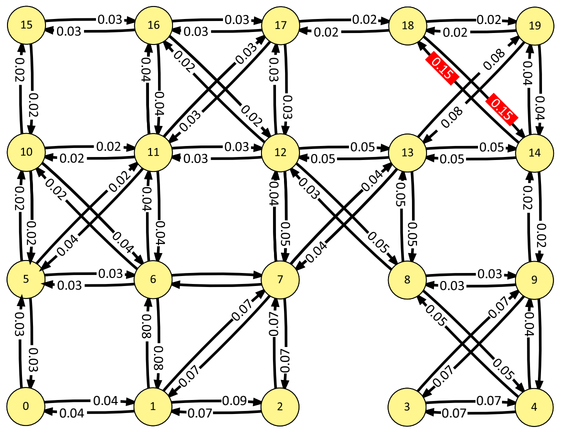


Figure 3.7: Layout of IBM-Q20, each edge represents a possible 2-qubit operation. The label on the edge represent the average probability of failure on that link when an operation is performed. The best link(s) have an error-rate of 0.02 and the worst link has 0.15, so a difference in strength of 7.5x.

3.4 Evaluation Methodology

This section defines figure-of-merit for system-level reliability, benchmarks, and evaluation infrastructure to estimate the effectiveness of proposed variation-aware policies.

3.4.1 Figure-of-Merit for System-Level Reliability

In an iterative model of computing for NISQ programs, the trial contributes to useful information if the trail can be executed without errors. In fact, if the workload can be executed with only a small probability of error, then we may not need a large number of trials to converge on the correct solution. To quantify the overall system reliability, we use the *Probability of Successful Trial (PST)* metric as the primary figure-of-merit. PST can be computed as the ratio of successful trials to the total number of trials performed.

3.4.2 Benchmarks

For our evaluations, we use micro-benchmarks used by the prior studies on quantum compilers and qubit allocation [9, 8] and small kernels demonstrated with IBM quantum computers [12]. These micro-benchmarks are scaled down version of larger quantum applications and subroutines. Table 5.3 show the seven benchmarks used in our study, their description, number of quantum instruction performed, and the number of qubits and the SWAP operations. We choose workloads with diverse qubit entanglement patterns. For example, Quantum Fourier Transform (qft) requires almost all to all entanglement whereas Bernstein-Vazirani (bv) requires one qubit entangled with rest of the other qubits. Whereas, rnd-SD and rnd-LD have random sequence of CNOTs.

Table 3.1: Benchmark Characteristics

NISQ Workload	Benchmark Description	Num Qubits	Total Inst	SWAP Inst
`alu`	Quantum adder [9]	10	299	19
`bv-16`	Bernstein Vazirani [13]	16	66	7
`bv-20`	Bernstein Vazirani [13]	20	90	10
`qft-12`	Quantum Fourier Trans.	12	344	35
`qft-14`	Quantum Fourier Trans.	14	550	53
`rnd-SD`	Rand benchmark with short distance communication	20	100	24
`rnd-LD`	Rand benchmark long distance communication	20	100	35

3.4.3 Evaluation Infrastructure

We perform our studies using the variation data from the IBM-Q20 machine. Unfortunately, the access to IBM-Q20 was not publicly available. Therefore, for our system reliability evaluations, we built a Monte-Carlo based fault-injection simulator using the architecture-level model for the IBM-Q20 machine. We use the iterative model for NISQ where the same workload is executed a large number of times, and the output is analyzed. Figure 3.8 shows an overview of our fault-injection simulator.

The simulator accepts the (a) NISQ program (b) layout, configuration, and error rate, and (c) management policies. The simulator injects errors based on the error rate of the given qubit and link and then tracks if the program completed without an error. We use the IBM-Q20 characterization data to estimate the probability of failure for two-qubit, single-qubit, and measurement operations. We perform 1 million trials for each workload to get PST estimates for the NISQ application by modeling errors as uncorrelated events with independent probability across trials.

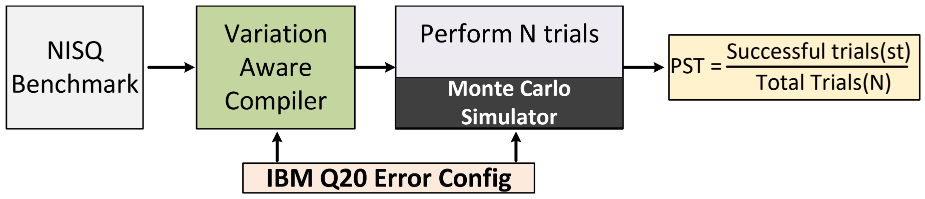


Figure 3.8: Monte-Carlo fault-injection simulator for estimating system level reliability of quantum computers.

Note that the compilation and mapping policies can be evaluated by using first-order simulators to gain insights. Nonetheless, in addition to simulation-based evaluations, we also analyze the effectiveness of our proposal on a real quantum machine, albeit at a smaller scale, using the IBM-Q5 machine and IBMQ-14. In Section 3.7, we demonstrate that our proposal is effective even in a realistic setting and provides a significant improvement in PST for real systems.

3.4.4 Layout and Error-Rate Parameters

The layout configuration specifies the number of qubits and their connectivity. For our studies, we use the IBM-Q20 layout and error-rate collected from IBM-Q20 over 52 days as is. The error-rate parameters describe the error rates for single-qubit, two-qubit and measurement operations. We model the errors in quantum operations as independent trials. We also model the coherence errors for all qubits. For the error-rate of IBM-Q20, the gate errors have a domination impact on the overall system reliability and the impact of coherence errors is negligible (e.g., for bv-20, the gate errors are 16x more likely to cause system failures than the coherence errors).

3.4.5 Baseline for Qubit Movement and Allocation

We use a baseline mapping policy proposed by Zulehner et al. [9] that seeks to minimize the number of SWAP operations. The steps for the baseline scheme are as follows:

1. Initialize an unweighted graph (G) that represents qubits as set of nodes (V) and links as edges (E).

2. Compute distance matrix for minimum number of SWAPs required to entangle any two qubits in G.

3. Partition the input program in layers such that each layer consists of independent operations that can be executed in parallel while respecting data dependencies. For example, an input program is partitioned into L, which is a set of n layers, $L = \{l_0, .., l_i, l_{i+1}, .., l_{n-1}\}$ where n equals the depth of the program.

4. Iterate through all the layers to find the map m_i between program qubit and physical qubit for each layer l_i such that all the CNOTs in the layer can be performed with available physical connectivity.

5. Find optimal set of swap operations ($S_{i \longrightarrow i+1}$) for each pair of layers l_i and l_{i+1} that transforms the map m_i to m_{i+1}. To search for the optimal set of SWAPs in an exponentially scaling search space, authors propose to use A* search that using cost function and heuristics based on the number of hops or Manhattan distance.

Note that the baseline tries to reduce the cost of SWAPs by implicitly assuming a uniform cost for all SWAP operations.

3.5 Variation-Aware Qubit Movement

3.5.1 The Problem of Qubit-Movement

The Qubit-Movement policy is responsible for deciding the route to take while moving qubit data from one device to another.[1] Such a policy can consider all possible routes and pick the one that requires the fewest number of SWAP instructions. Fortunately, most of the designs for quantum computers use a mesh-like network, so all the choices that go either in the X direction or Y direction towards the destination will have identical Manhattan distance, and hence the identical number of SWAP instructions. For example, for the 6-qubit quantum computer shown in Figure 3.9, if we want to go from physical qubit A to physical qubit F, all three routes (A-B-C-F, A-D-E-F, A-D-C-F) have identical hop counts (3), and the Qubit-Movement policy can choose any of these routes. It may consider making the Qubit-Movement decision simple by first going in the "X" dimension and then going in the "Y" dimension (or vice versa) – while such a policy would ensure the shortest route (minimum number of SWAP instructions), such a policy would exclude the selection of path A-D-C-F.

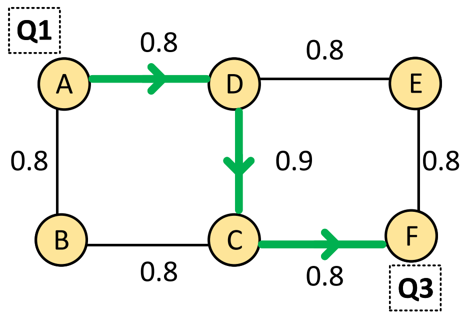


Figure 3.9: A 6-qubit quantum computer, weight indicates probability of success of each link. To move Q1 (at A) to Q3 (at F), a variation-aware policy would use route A-D-C-F as it maximizes the probability of success of the movement.

[1]Qubit-Movement policy is analogous to network-routing algorithms, which decide the path followed by a packet from the source to destination within a network. Similar to how network-routing algorithms try to minimize the "hop count", Qubit-Movement policies try to minimize the number of SWAPs. Network-routing algorithms make localized decisions at each node, so they must be designed carefully to avoid deadlocks. However, Qubit-Movement is orchestrated globally by the compiler, with the knowledge of the usage of all links, so it is easy to avoid schedules that cause deadlocks.

3.5.2 "Variation-Awareness" in Qubit-Movement

Given that there is variation in the error-rates of different links, policies (such as X-first or Y-first) that choose one choice among the list of shortest routes will not always provide the best overall system reliability. For example, the number on each link in Figure 3.9 shows the probability of success of the link. Route A-D-F would maximize the probability of success of the overall movement from A to F, and a variation-aware policy would choose such a route.

3.5.3 Design for Variation-Aware Qubit-Movement

We propose *Variation-Aware Qubit Movement (VQM)* that seeks to perform Qubit-Movement while taking into account the variation in the per-link error rates. VQM selects the paths with the highest reliability for the data movement and actively tries to avoid paths that have poor reliability. Existing mapping policies such as the baseline policy [9] find the optimal path to entangle qubits by formulating a state-space search problem that uses the cost function that is based on the number of inserted SWAPs. Whereas, in VQM, we change the cost function from the number of SWAPs to the overall failure rate incurred by moving the qubit from source to destination. Our variation-aware mapper determines the set of SWAP instructions that minimizes the probability of failure. In case of no variation in error-rates, our policy selects the path with the minimum number of swaps to minimize the probability of failure (identical as a baseline). However, for non-uniform link-errors, VQM picks a path that has the highest reliability. Thus, VQM leverages the locality preserving traits of baseline while using a variation-aware heuristic. Algorithm 1 describes the steps for VQM.

For implementing VQM, we assume that the characterization data of the error rates for different links are available and that this characterization data remains valid during the execution. VQM compiles the application and tries to select the route that tries to

Algorithm 1 Variation-aware qubit movement algorithm

1. Initialize weighted graph (H) with N qubits as vertices (V), links as edges (E) with weights (W) that represent a failure rate of the links, compute distance matrix (D) using Dijkstra's algorithm that holds pairwise shortest distance. Each element in D is the minimum cost for the most reliable path to entangle two qubits on H.

2. Using W, compute the node strength or weighted degree (d_i) of each qubit (v_i) such that $d_i = \sum_j^N w_{ij}$

3. Break the input circuit into layers $(l_i = (l_0, l_1, ...l_n))$ similar to baseline.

4. Find program-qubit to physical-qubit map m_i for each layer l_i such that physical qubits with higher node strengths are prioritized during the mapping process.

5. Find optimal set of swap operations that transforms the map m_i to m_{i+1}. The optimal set of SWAPs minimize the probability of error by choosing most reliable paths using D. To choose the optimal set of SWAP, we use A* search proposed by the baseline with a reliability-aware cost function and with an additional heuristic Maximum Additional Hop (MAH).

maximize system reliability.[2] For selecting the route, VQM simply forms a cost graph where each link has a probability of success, and the overall probability of success of a route is computed as the product of the probability of success of the individual links. VQM selects the route that maximizes the probability of success for the overall route. VQM can select a longer path over the shortest path if the longer path has higher reliability. This will result in an extra number of SWAPs. Furthermore, more qubits get displaced due to a longer chain of SWAPs. The displaced qubits may cause additional set SWAPs for future CNOT operations. We use a parameter that limits the extra SWAP instructions using *Maximum Additional Hop (MAH)*. VQM with such limitations will select the path with the lowest cost, such that the extra hops do not exceed MAH.

[2]In conventional computer systems, applications may be compiled once, and run unchanged for several years. However, it is reasonable in NISQ domain to assume that each time the workload is scheduled, it gets recompiled by the runtime system (using the latest characterization data) and then repeated trials are performed with the updated executable.

3.5.4 Impact of VQM on System Reliability

Figure 3.10 shows the Relative-PST for our seven benchmarks when compiled with VQM and the constrained version of VQM (MAH=4). All benchmarks see a significant improvement in the PST with VQM. Benchmarks such as `qft`, `rnd-LD` require long-distance entanglement and a considerable number of SWAPs (limited locality), therefore they see higher improvement in PST compared to other benchmarks.

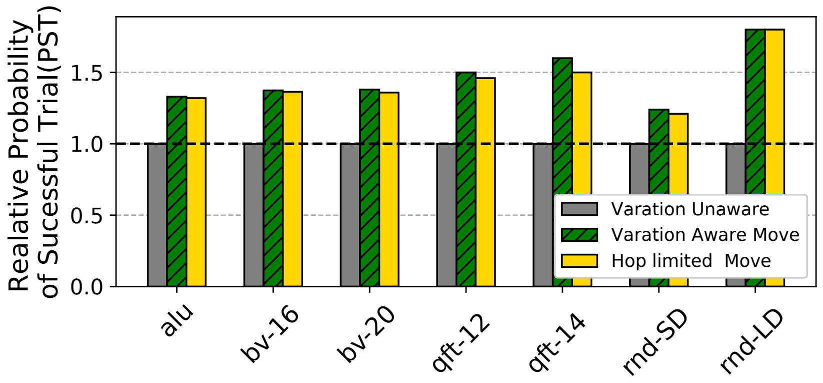


Figure 3.10: Impact of VQM on the Probability of Successful Trials (PST). Note that reported PST numbers are normalized to PST of the baseline policy that selects the shortest route.

We also observe that the hop-limited policy has similar improvement to an unconstrained policy that does not put any limit on the increased hop count. This is especially true for workloads that have locality as the limited hops preserve the locality by restricting the qubit path among active qubits.

3.6 Variation Aware Allocation

The Qubit-Allocation policy is responsible for assigning the program qubits to the physical qubits.

3.6.1 "Variation-Awareness" in Qubit-Allocation

Baseline qubit allocation is oblivious to the variation in the link reliability. It uses an allocation that minimizes the number of SWAPs. For example, if we want to allocate 2 qubits on the machine in Figure 3.13(a), the baseline policy may pick any two neighboring qubits, including D and A, which are connected by the weakest link. If the allocation policy was aware of the variation, it would pick D and C, which are connected by the strongest link. We propose such a *Variation-Aware Qubit Allocation (VQA)* policy.

3.6.2 Design of Variation-Aware Qubit-Allocation

The baseline policy starts with carefully selected initial mapping and then tries to converge to a configuration that has a minimum number of SWAPs. However, doing so does not take into account the variation in the link-errors of the qubits. VQA, on the other hand, maps the frequently used qubits to the qubits with most reliable link to improve reliability and preserve the locality. VQA achieves this by starting with the most reliable initial mapping and restricting frequently used pair of qubits to most reliable links. VQA estimates the most frequently entangled qubits by analyzing the first-N instruction in the program and tracking the number of CNOT operations between each of possible pair of qubits. The steps for VQA are shown in Algorithm 2.

Furthermore, when mapping less number of qubits than the available qubits, baseline exposes all the qubits to the mapping process which can sometimes result in the mapping of frequently used qubits to weak qubits. VQA prevents such undesirable assignments by selecting the strongest (sub-graph) and restricting the qubit mapping that

Algorithm 2 Variation-aware qubit allocation algorithm

```
1. Find the sub-graph ($SG_k$) with k-nodes that has highest aggregate node
   strength (ANS). $ANS = \sum_i^k d_i$ where $d_i = \sum_j^N w_{ij}$.
2. Find qubit activity by calculating the number of CNOTs per qubit for first
   $t$ layers.
3. Map program qubit to physical qubit mapping $m_i$ prioritize the map-
   ping of qubits with high activity to $SG_k$ such that top K active qubits are
   mapped to $SG_k$.
4. Use baseline algorithm to find SWAPs between layer $l_i$ and $l_{i+1}$.
```

maximizes the overall system reliability. VQA computes the strongest set of sub-graphs by using K-core algorithm that recursively prunes nodes with degrees less than k [14].

3.6.3 Impact of VQA on System Reliability

By using VQA, we ensure the mapping of program qubits with high activity (total number of CNOT operations) to the set of physical qubits with the higher node strength. This improves the reliability for workloads that have repeated entanglement operations between few select pairs of qubits. We implement VQA in conjunction with the variation-aware movement. Figure 3.11 shows the relative-PST for the micro-benchmarks normalized to the baseline, VQM, and VQM+VQA. Our evaluations show that VQM+VQA can provide up to 1.7x improvement in PST. Note that, for all the benchmarks, the combination of VQM+VQA provides higher PST than the VQM scheme standalone.

3.6.4 Improvement Relative to Native IBM Compiler

We use a state-of-the-art baseline policy that tries to minimize the number of SWAP instructions. Our baseline is stronger than an alternative policy that uses randomized assignment, such as the native compiler from IBM. Figure 3.11 compares the PST for IBM's native compiler with our baseline and the proposed policies. As the IBM native compiler performs randomized initial mapping, we evaluate 32 configurations (each over 10000

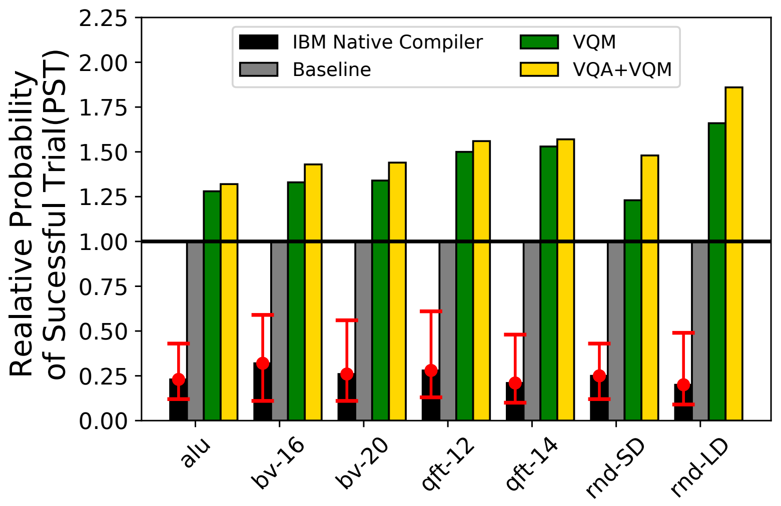


Figure 3.11: PST for VQA and VQM+VQA normalized the baseline policy (variation unaware). We also compare the normalized PST of the native compiler of IBM.

trials) and report the average and the min-max (using the error-bars) PST. Note that our baseline policy has 4x higher PST than the IBM native compiler. Whereas, VQA+VQM improve PST up to 7x over the IBM compiler.

3.6.5 Effectiveness to Per-Day Variation

We perform our evaluations using average behavior of the link/qubit based on characterization data across 52 days. The behavior of the qubit and links can vary over time, and with it the benefit of our scheme. To analyze this, we evaluated `bv-16` with per-period characterization data across the 52 days. Figure 3.12 shows the improvement in PST for `bv-16` benchmark for each day (the dotted line denotes the average). VQA+VQM provides larger PST improvement on days with higher variability and smaller on days with lower variability.

3.6.6 Sensitivity to Scaling of Error Rates

As technology improves, we can expect the error rates to reduce, however the variation may still persist even at lower error rates, meaning our proposal can still be effective.

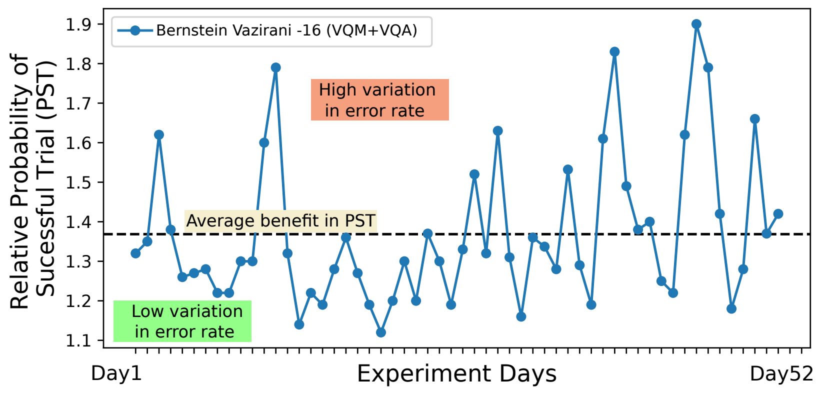


Figure 3.12: Relative improvement in PST for bv-16 benchmark evaluated with error-configs collected over 52 days.

We evaluate `bv-16` benchmark with 10x lower average error rate (standard deviation reducing proportionally or by half as much). As shown in Table 3.2, VQM+VQA provides significant benefits that increases with increased relative variation.

Table 3.2: Sensitivity of VQA+VQM with Error Scaling.

Benchmark Name	Average Error Error-Rate	Covariation of Error Rate	Relative PST Benefit(VQA+VQM)
`bv-16`	1x	Cov-Base	1.43x
`bv-16`	10x lower	Cov-Base	2.02x
`bv-16`	10x lower	2*Cov-Base	2.59x

3.7 Evaluation on Real System: IBM-Q5

Access to IBM-Q20 is not publicly available, so we evaluated our policies for IBM-Q20 using a simulator. We demonstrate the usefulness of our ideas for real quantum systems by performing experiments on the IBM-Q5 machine. For IBM-Q5, the average two-qubit error rate is 4.2%, and the worst link-error is 12%. We use the error configuration of IBM-Q5 to compile the benchmarks that are suitable for IBM-Q5. A compiled program with the baseline policy and with VQA+VQM are then executed on a IBM-Q5 machine and the output is logged. We run each experiment with 4096 trials and analyze the output log to compute the PST for each program and policy. Table 3.3 shows the PST of the baseline and (VQA+VQM). Our proposal improves the PST for IBM-Q5 machine by up to

1.9x improvement (average 1.36x).

Table 3.3: PST for Baseline and Proposed Policies on IBM-Q5.

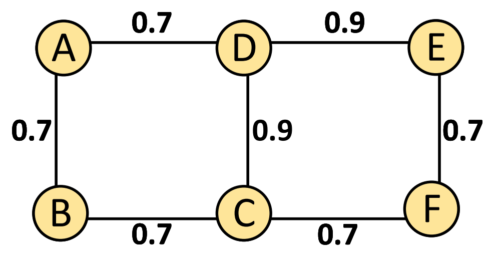
Benchmark Name	PST (Baseline)	PST (VQA+VQM)	Relative Benefit in PST
`bv-3`	0.31	0.38	1.22x
`bv-4`	0.21	0.23	1.09x
`TriSwap`	0.13	0.25	1.90x
`GHZ-3`	0.57	0.77	1.35x
`GeoMean`	0.26	0.36	1.36x

3.8 Partitioning Quantum Computer

We have explored the variation-aware policies for Qubit-Movement and Qubit-Allocation. This concept can be used to provide insights into other design trade-offs that may come in NISQ systems. We do a case study for a scenario, where the workload requires half or fewer qubits than what is physically available, and the computer can be partitioned to run multiple copies of the same workload (to provide more trials per unit time). We analyze whether it makes sense to partition the NISQ computer in such scenarios.

3.8.1 Two Weak-Copies versus One Strong-Copy

When the number of program qubits are less than or equal to half of the physical qubits, we can run two copies of the same program. In an ideal world, the simultaneously running two copies can provide twice as many number of error-free trials per unit time. However, for a quantum computer with variation, running two copies restricts the program qubit to physical qubit mappings. For example, running a single copy provides an opportunity to choose the strongest set of qubits and links in a given quantum computer, whereas, running two copies would constrain us to also use weaker set of qubits and links. Thus, the single copy would try to maximize the PST for a given trial, even if it means sacrificing the increased trials per unit time that would be possible with two copies. Whereas, having two-copies provides more trials per unit time at the expense of PST for each trial. On a given NISQ with variable reliability, should we run two weak copies or run one strong copy of the program?



```
#NISQ Program
qreg Q1, Q2, Q3;
Cx(Q1, Q2);
Cx(Q3, Q2);
Cx(Q1,Q3);
```

(a)

Map Q_{1X}, Q_{2X}, Q_{3X} → A, B, C Map Q_{1Y}, Q_{2Y}, Q_{3Y} → D, E, F	
Copy-X	Copy-Y
Cx (A,B)-- 0.7	Cx (D,E)--0.9
Cx(B,C)-- 0.7	Cx(E,F)--0.7
SWAP(B,C)--$(0.7)^3$	SWAP(D,E)--$(0.9)^3$
Cx(A,B)-- 0.7	Cx(E,F)-- 0.7
PST(X) = 0.12	PST(Y) = 0.32

(b)

Map Q_{1X}, Q_{2X}, Q_{3X} → C, ,D, E Map Q_{1Y}, Q_{2Y}, Q_{3Y} → DON'T MAP	
Single Copy	NA
Cx (A,B)--(0.9)	--
Cx(B,C)--(0.9)	--
SWAP(B,C)--$(0.9)^3$	--
Cx(A,B)--(0.9)	--
PST = 0.53	NA

(c)

Figure 3.13: (a) NISQ with six qubits using mesh connectivity. A CNOT reliability is reported on the top of each link. (b) Mapping policy that runs two copies of a NISQ program (c) Mapping policy that runs one copy using the strongest links.

For a hypothetical machine with six physical qubits with a mesh-layout as shown in the Figure 3.13(a). The edge-weights in the graph show the strength of the coupling links. For a quantum program with three program qubits as shown in the Figure 3.13(a), we can either run two copies by partitioning the quantum computer or run just one copy. Figure 3.13(b), shows two copies of a program: Copy-X and Copy-Y running on a quantum computer. The success probability of individual copy can be calculated by multiplying all the success probabilities of operations in the program. For example, Figure 3.13(b) shows the PST for Copy-X and Copy-Y to be 0.32 and 0.12 respectively. Thus, running two copies does not increase the rate at which successful trials can be done by 2x, instead in our case it is only 37.5% (0.44/0.32).

For the example program, if we choose to run a single copy, we can intelligently select the strongest subset of qubits and links to improve the overall reliability. Figure 3.13(c) shows one such example whereby choosing to run just one strong copy can improve the cumulative PST. When running two copies, the constraints on connectivity restricts the use of link CD which is one of the strongest links. When running two copies, programmer has to resort to the weaker links. Whereas, when running a single copy, we can pick most reliable links and achieve better PST as shown in the Figure 3.13(b).

3.8.2 Benchmark-Based Evaluation

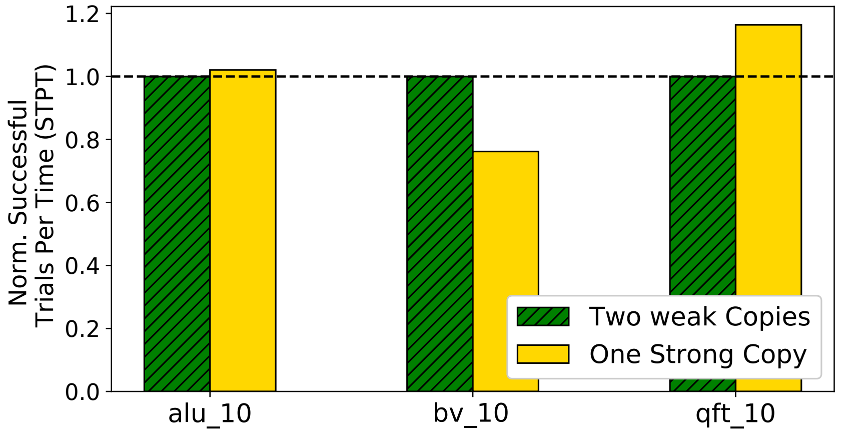


Figure 3.14: Successful Trials per Unit Time (STPT) when running two-copies versus one strong copy.

We extend our evaluation infrastructure to support two copies of the same workload. For the two-copy mode, we explore all possible partitions and select the best. Note that besides the number of copies, movement and the mapping algorithm used for both of the policies are identical. The only difference is the available number of qubits. For the evaluation in this section, we use the figure of merit as *Number of Success Trials Per Unit Time (STPT)*, as it captures both the PST and the increased rate of trials with two copies. We modify these benchmarks to use only 10 qubits. Figure 3.14 shows the STPT of the single strong-copy and two-copies, both normalized to the STPT of the two copies. For this study, we selected the three workloads that can operate with ten qubits. We observe that sometimes two-copy is better (`bv-10`) and sometimes one strong-copy is better (`qft-10`). For NISQ applications, we can estimate which solution is likely to perform better for the workload and use that solution. Thus, our variation-aware policies may be useful in enabling *Adaptive Partitioning* for NISQ machines, where the decision between one strong copy versus two-copies can be based on STPT.

3.9 Summary

In this work, we study the policies for *Qubit-Allocation* and *Qubit-Movement* for current quantum computers. Our experiments on IBM quantum computer show a large variability in the error rates of different qubits and links.The variability in errors significantly impact the probability of success. We show that prior studies that try to minimize data movement (number of SWAPs) may not maximize application reliability. We propose *Variation-Aware Qubit Movement* policy that exploits the variation in error rates by trying to pick a route that has the lowest probability of failure. Furthermore, we develop *Variation-Aware Qubit Allocation* policy that exploits the variation in error rates by allocating program qubits to physical qubits such that the use of links and qubits with high error rates gets minimized. We show that our policies provide significant improvement both in simulated setting and on real IBM quantum computers. Moreover, our insights can also help in understanding the resource sharing and partitioning problems in the near-term quantum computers, such as deciding between running one strong-copy versus two concurrent copies of NISQ program.

CHAPTER 4
DIVERSIFYING QUANTUM PROGRAMS FOR ROBUST INFERENCE

4.1 Introduction

The NISQ machines can produce an incorrect output as the computation is subjected to errors. Therefore, to infer the correct answer, the program is run thousands of the times on the NISQ machine to produce a probability distribution of the possible output states. This distribution is analyzed to infer the correct answer, for example, by selecting the most frequently occurring output. Consider the Bernstein-Vazirani (BV) algorithm that allows the program to infer the hidden key in a single shot. On an idealized machine, this program will provide the correct answer with a probability of 1, as shown in Figure 4.1(a). However, if we execute BV on a NISQ machine, then we will get the correct answer for some trials and wrong answer for others. Figure 4.1(b) shows the output distribution for BV, where the correct answer occurs with 30% probability and the most dominant incorrect answer occurs with 25% probability. The correct answer can be inferred by selecting the most frequent output. Unfortunately, the NISQ machine can have correlated errors that cause the same incorrect answer to appear with a high frequency. Inferring the correct answer can become challenging in such scenarios. For example, consider Figure 4.1(c), where the correct answer still occurs with 30% probability, but one of the incorrect answers occurs with 35% probability. We observe that the task of inferring the correct answer can be achieved via two means: increasing the probability of the correct answer or by reducing the probability of the dominant wrong answer. Recent work on qubit allocation policies (swap minimizing or variation-aware) have focused on the former, whereas, in this work, we focus on the latter.

Qubit allocation policies deal with the problem of assigning the program qubits to

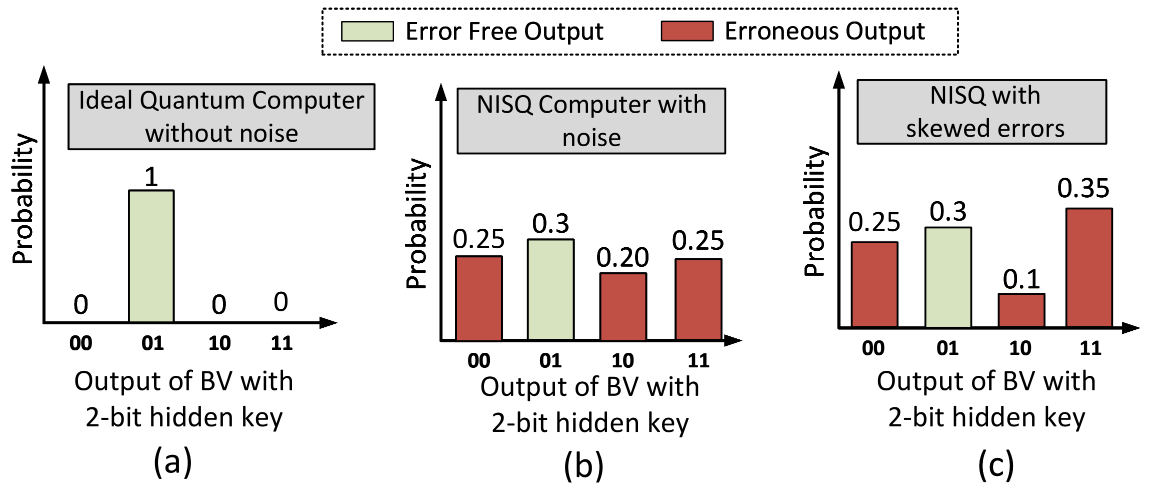


Figure 4.1: Output of Bernstein-Vazirani with 2-bit key on (a) Ideal machine (b) NISQ machine providing a correct answer (c) NISQ machine providing a wrong answer.

the physical qubits (qubit assignment) and moving the qubit from source to destination for performing two-qubit operations (qubit routing). Qubit allocation policies have a significant impact on the reliability of the NISQ machine as these policies can determine the number of operations required to execute a given program. Routing of the qubit from source to destination is typically accomplished by inserting additional SWAP instruction that can swap two neighboring qubits. Recently proposed qubit mapping policies try to minimize the number of SWAP instructions. Recent studies have also investigated variation-aware qubit mapping policies that try to use the strongest qubits and links (the ones with lowest error rates) to perform the computation. All of the prior proposals on intelligent qubit mapping (both SWAP minimizing and variation-aware) try to determine the best mapping and use that mapping for running all of the trials on the NISQ machine. Unfortunately, such an approach also makes the application vulnerable to correlated errors – if the computation is subjected to a particular error, the computation for all of the trials will continue to be performed on the same set of qubits and links, causing the same erroneous output to occur for a large number of trials.

To mitigate the vulnerability to such correlated errors, this work leverages the con-

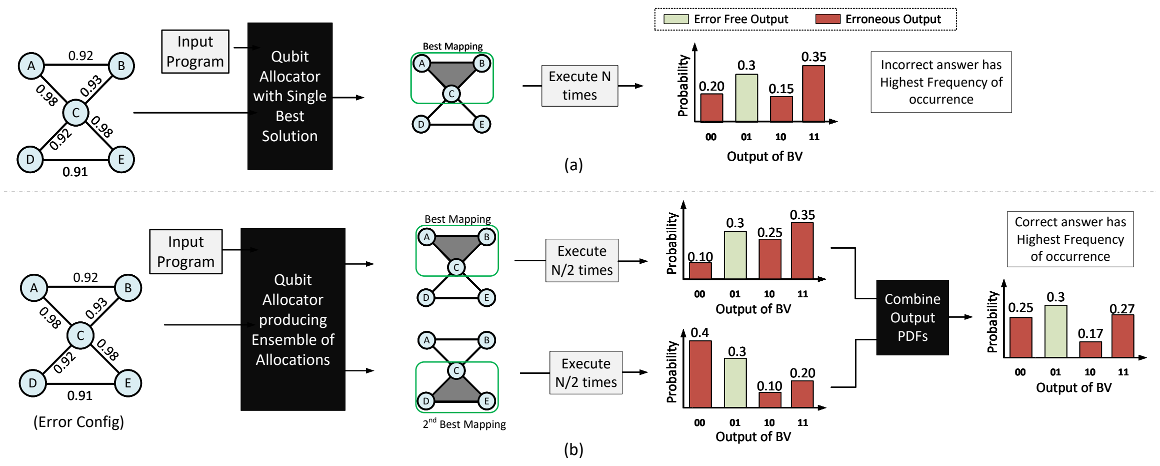


Figure 4.2: Bernstein-Vazirani using (a) Single best mapping (b) Ensemble of Diverse Mappings (EDM), running two allocations and merging the outputs (EDM infers correct answer even if both mappings have a dominant incorrect answer).

cept of diversity,[1] and proposes *Ensemble of Diverse Mappings (EDM)*. EDM is based on the insight that rather than having all the trials be subjected to the same sources of errors, split the trials into multiple groups, and have a different mapping for each group so that the trials in each group get subjected to different sources or errors and hence different incorrect outputs. For example, consider the scenario as shown in Figure 4.2(a) where the baseline performs N trials using the best mapping and still obtains an incorrect output. EDM splits the N trials into two groups and uses a different mapping (best and the second-best) for these groups. Even though both of these groups individually produce an incorrect answer with the highest probability, these incorrect answers are different – so when we merge the output distributions, the incorrect outputs end up getting attenuated, and the correct answer ends up getting accentuated. Even though the two groups individually failed to produce the correct answer, the diversity in EDM allows the ensemble to infer the correct answer. While we explain EDM with two mappings, EDM can be implemented with more than two mappings. For our studies, we use

[1]We note that when a team is formed with members of very similar skills and backgrounds, then all the members may share the same *blind-spot* and the team overall becomes vulnerable to that blind-spot. Whereas, when teams are formed with members of a diverse set of skills and backgrounds, then each member may have a different blind-spot, which may not be present in the other team members, making the overall group more resilient to such blind-spots.

EDM with four mappings, with each mapping used for one-quarter of the trials.

4.2 Motivation

4.2.1 Variation-Aware Qubit Mapping

The different qubits and links of a NISQ machines can have widely varying error rates as not all qubits have the same level of vulnerability to errors, and this variation in error rates has a large impact on the reliability of NISQ applications [15, 16, 17, 18]. For example, if we can map a program on the most reliable qubit, then the probability of errors can be reduced significantly (up to 10x). Especially for a class of NISQ programs that use less than available physical qubits, a programmer can choose the most reliable qubits to improve the reliability. Moreover, we can extend the idea of variation-aware allocation to qubit movement. For example, SWAP operations are unreliable and show significant variation in reliability (up to 20x on IBM-Q14), by using quantum links with high reliability and avoiding links with low reliability the system can reduce the impact of noise on NISQ machines. This makes the overall system reliability be dictated less by the worst-case qubits and links, and more by the average-case qubits and links.

To enable variation-aware techniques, we need error characterization data that describes the error rates for all the qubits and the links on a quantum computer. Fortunately, the error rates can be evaluated using randomized bench-marking and gate tomography. For IBM machines the error rates are evaluated after every calibration cycle, and the error characterization data is available to the programmer using IBM's qiskit API. However, the estimated error rates are not constant as qubits are non-linear devices that can have time-varying deviations due to drift and changing operating conditions. Our experimental evaluations show the relative reliability of collection of qubits and quantum links to largely have repeatable behavior. To estimate the reliability of the circuit in a variation-aware manner, prior works have used the *Estimated Probability of Success (ESP)* metric [18]. ESP for an executable can be computed by taking a product of all the gate success rates (g^s) and measurement success rates (m^s). The gate success rate is the probability of performing a gate without any error, which is calculated using the gate error rate (g^e). The measurement success rate (m^s) captures the probability of performing all measurements without any error. ESP is given by the equation below. Variation-aware mapping scheme tries to find the mapping that has the highest ESP. We use a variation-aware mapping policy as our baseline.

$$ESP = \prod_{i=1}^{N_{gates}} g_i^s * \prod_{j=0}^{N_{meas}} m_i^s$$

$$g_i^s = (1 - g_i^e), \quad m_i^s = (1 - m_i^e)$$

4.2.2 The Inference Problem for NISQ

A NISQ machine is subjected to errors. Therefore, to infer the right answer, the given program is run for thousands of trials, and the output of each trial is logged. In the end, we get an output probability distribution that is influenced by both correct and incorrect answers. The task of inferring the correct answer becomes challenging at high error rates. For example, if the error rate is small, then the correct answer would appear with the highest frequency. As qubit error rate increases, the likelihood of correct answer decreases significantly such that the incorrect answers may be produced as frequently as correct answers.

We can improve the inference quality of the NISQ machine by either increasing the frequency of the correct answer or by reducing the occurrence of the most common wrong answer. Existing mapping policies focus only on the first option and try to perform the computation using the strongest qubits and links. Therefore, they run all the trials using the mapping that maximizes the probability of getting the correct answer.

4.2.3 The Challenge: Correlated Errors

We observe that even with the mapping that maximizes the ESP, NISQ machines can fail to provide the correct answer as the most frequently occurring outcome. In such cases, a particular incorrect answer dominates the correct answer. The wrong answer occurring with a high frequency happens because the computation gets subjected to similar types of error repeatedly leading the same wrong outcome. Thus, quantum computers can have correlated errors. Current approach to performing all the trials with a single mapping policy makes the application vulnerable to correlated errors – if the computation is subjected to a particular error, the computation for all of the trials will continue to be performed on the same set of qubits and links, causing the same erroneous output to occur for large number of the trials. Correlated errors is a real problem on IBM quantum machines, for example, recent study reports the correlated nature of SPAM errors [19]. In this work we develop solutions for addressing the correlation in the incorrect answer. We provide the characterization for correlated errors next.

4.3 Experimental Methodology

In this section, we briefly describe the benchmarks, system configuration, and the metrics used in our work.

4.3.1 Benchmarks

Existing quantum computers such as publicly available IBM fourteen qubit machine are severely limited due to noise. Due to low coherence and high gate error rates it can execute circuits with small number of qubits for short duration (low depth). Table 4.1 describe benchmarks and total number of single qubit gate operations (SG), CNOT operations (CX), and measurement operations (M) for the respective benchmarks.

Greycode Decoder: Grey code decoder decodes a binary string to a grey code string using a reversible circuit. For this benchmark, number of two qubit and measurement operations scale linearly with number of qubits. We use six bit circuit described in RevLib [20]. The greycode benchmark is used to understand the effects of correlated errors on shallow circuit that measure qubits in standard basis. Moreover, greycode has identical number of measurement and two-qubit gates, which is useful to understand if the correlation in errors stem from measurement or two qubit operations.

Bernstein-Vazirani (BV): BV finds a n-bit binary secret encoded in the quantum oracle by querying the oracle once. On execution, `BV` outputs a binary string corresponding to the secret key. For `BV`, number of two qubit and single qubit gates scale linearly with number of qubits. BV is sensitive to phase and T2 errors as it measures qubits in Hadamard basis. We use two instances of BV to understand if SWAPs can cause correlated errors as BV-7 has one additional SWAP operation compared to BV-6.

Quantum Approximate Optimization Algorithm: QAOA is a generalized algorithm that can be used to solve combinatorial optimization problems. We use QAOA to solve the max-cut problem, which tries to partition an input graph into two subsets (S_1,S_2) of nodes to maximize the number of edges between the first (S_1) and the second(S_2) subset. Note that QAOA-5, QAOA-6, QAOA-7 do not require any SWAP operations. For QAOA, number of two qubit gates scale super linearly with number of qubits. Whereas number of single qubit operations scale quadratically. QAOA is believed to be robust against certain class of two and single qubit errors.

Reversible circuits: We use three reversible circuits (Fredkin gate, two bit adder, and 2:4 decoder) to understand how correlated errors would affect the reliability of short width circuits. For instance, all reversible circuits use three to four qubits, but it contains more than 10 two-qubit gates. For these circuits, T1 decoherence might be the dominant error mechanism and such workloads can provide insights into how decoherence can cause correlated errors.

Table 4.1 shows the characteristics of the benchmarks used in our study. The terms "SG", "CX" and "M" respectively denote the number of single-qubit, two-qubit, and measurement operations in the workload. Workload evaluation on existing quantum computers is severely limited due to high error rates, which limits the length of the programs that can be run reliably on the current machines. Therefore, similar to prior studies [16, 15, 18, 21] we perform our experiments on small benchmarks.

Table 4.1: Benchmark Characteristics

Benchmark Name	Benchmark Description	Output	Number of Gates
`Greycode`	Greycode decoder	output: 001000	SG: 13, CX: 5, M: 6
`bv-6`	Bernstein-Vazirani	key: 110011	SG: 13, CX: 7, M: 5
`bv-7`	Bernstein-Vazirani	key: 1101011	SG: 13, CX: 11, M: 6
`qaoa-5`	max-cut 5 node graph	cut: 10101	SG: 24, CX: 8, M: 5
`qaoa-6`	max-cut 6 node graph	cut: 101010	SG: 30, CX: 10, M: 6
`qaoa-7`	max-cut 8 node graph	cut: 10101010	SG: 36, CX: 12, M:7
`Fredkin`	Fredkin gate	output:110	SG: 26, CX: 13, M:3
`adder`	1bit adder	output:011	SG: 12, CX: 15, M:3
`Decode-24`	2:4 Decoder	output: 100000	SG:119, CX:71, M:6

4.3.2 System Configuration

For all our evaluations, we use publicly available IBM quantum computer with fourteen qubits *ibmq-16-melbourne* [12]. For clarity we refer *ibmq-16-melbourne* as IBMQ-14. Moreover, for all the evaluations, we use a variation-aware mapping policy [15] as the baseline. As the error-characteristics of the NISQ machine can change dramatically between two calibrations, to guarantee statistical significance, we always execute baseline and the proposed policy for 16 thousand trials within a short succession of each other in each round. We repeat 10 such rounds and report the improvement for the median round.

4.3.3 Figure-of-Merit for Reliability

The goal of running the workload on a NISQ machine is to be able to infer the correct answer. This can be achieved by either increasing the probability of the correct answer or by suppressing the strongest sets of wrong answer or both. We need reliability metrics that account for both effects and has an intuitive implication on what it would mean to the ability to infer the correct answer on the NISQ machine.

The metric commonly used to indicate the reliability of a NISQ machine is the *Probability of Successful Trial (PST)*. PST is calculated by computing the ratio of a number of error-free trials to the total number of trials. PST is a good metric to compare two design points, for example comparing an ion-trap machine with a superconducting machine [22]. Moreover, recent papers on noise adaptive and variation-aware qubit mapping polices also use similar metrics to capture the reliability of applications [15, 16, 18].

$$\boxed{PST = \frac{\text{Number of Trials with Correct Solution}}{\text{Total Number of Trials}}}$$

Unfortunately, PST does not always indicate the ability to infer the output of a NISQ machine correctly. For example, with PST=0.2 we can have reliable inference if all incorrect answers occur with less than 0.2 probability. However, another system with PST=0.2 will be unable to infer the correct output if one of the wrong answers is more dominant, say, for example, it occurs with 30% probability. To account for the magnitude of both the correct and the incorrect answers, we define a metric, *Inference Strength (IST)*. IST is a ratio of the frequency of correct output to the frequency of the most commonly occurring erroneous output.

$$\boxed{IST = \frac{\text{Probability of Correct Solution}}{\text{Probability of Strongest Incorrect Solution}}}$$

If IST exceeds 1, the system will be able to correctly infer the output, whereas if IST is significantly lower than 1, then the wrong answer(s) would mask out the correct answer. As our objective is to improve the ability to infer the correct answer on a NISQ machine, we use IST as the primary figure of merit in our evaluations.

4.3.4 Limitations of Evaluation on IBMQ-14

Running a large instance of the benchmark has two problems: first, large instances of benchmarks require significantly more operations. For example, in theory, BV has linearly increasing CNOT cost, but in reality, it requires a number of CNOT operations that scale super-linearly due to extra SWAPs. This problem is severe on IBMQ-14 machine due to unidirectional and limited connectivity. The second problem arises due to variability on IBMQ-14 machine. For small benchmarks we can pick the strongest set of qubits, but for larger ones we have to include even the weak qubits. During our evaluations, we could avoid two consistently weak qubits on IBMQ-14(Qubit-12 and Qubit-11 with readout error of 15% and 30%). If we include weak qubits, the PST drops by 3x.

To avoid experimental inconsistencies, we focus on small instances of NISQ algorithms such as BV, and QAOA as these low depth circuits promise linear scaling of gates with the number of qubits. Whereas, for sensitivity, we use reversible circuits that scale polynomially with the width of the circuit. Note that the circuits such as adder, decoder, Fredkin gates may not be representative of NISQ applications as they require 100s of two-qubit operations even for 3 to 4 qubit circuits as shown in Table 4.1. But these long benchmarks can provide insights on how output distribution changes due to T1 errors vs Measurement errors.

4.4 Correlated Errors on NISQ

In this section, we analyze the correlation in errors on IBM's fourteen qubit machine (IBMQ-14) and study how the correlated errors produce incorrect answers such that the

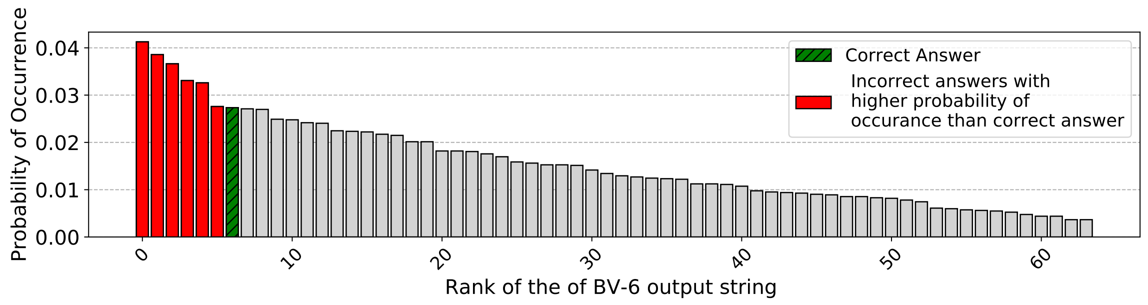


Figure 4.3: Output probability distribution for *Bernstein-Vazirani* (BV-6) with 6-bit hidden key executed on the IBM-Q14 machine (note that the states are sorted by the frequency of occurrence, from the highest to the lowest).

frequency of some incorrect answers is more than the correct answers.

4.4.1 Impact of Noise on Application Reliability

IBMQ-14 suffers from high measurement and gate error-rates. To understand the nature of errors and the impact on the system reliability, we execute the Bernstein-Vazirani (BV) benchmark with a 6-bit secret key. We perform each experiment for 16 thousand trials. Figure 4.3 shows the probability distribution for the different outcomes, with the outcomes arranged from the highest frequency of occurrence to the lowest. Notice that due to high error rates, the probability of getting the correct answer is fairly low (2.8%) and the output log consists all 64 possible outcomes (63 incorrect answers plus one correct answer). Furthermore, some of the incorrect outputs occur with almost 1.5x the frequency of the correct answer. We observe that the relative strength of the correct answer (probability normalized to the most frequent incorrect answer) is only 68%, and therefore, inferring the correct answer is not straightforward. In the Appendix-A, we describe how correlated errors can degrade quality of inference in NISQ model.

4.4.2 Correlation in Errors

To test if using the same set of qubits cause correlated errors, we execute two sets of experiments. The first set containing eight runs using the best mapping and the second set contains eight runs with different mappings (top-8 mappings).

BV-6 with Single Best Mapping: We run eight copies of BV-6 with single best mapping that maximizes the reliability. To understand if the trials with single mapping produce similar incorrect answers, we measure the divergence or dissimilarity between the output probability distribution using the *KL-divergence*. KL-divergence estimates the distance between the pair of probability distributions. If KL-divergence is close to zero, then the output distributions are similar. Figure 4.4 shows the heat map (darker shades are close to zero, indicating similarity) that illustrates the pairwise divergence between output probability distribution of BV-6 runs. All non-diagonal elements (d_{ij}) represent the divergence between the output of i^{th} and j^{th} run when BV-6 is executed with the single best mapping. Note that the pairwise KL-divergence between all the runs are close to zero. Thus with identical mapping, NISQ programs tend to produce similar incorrect outputs.

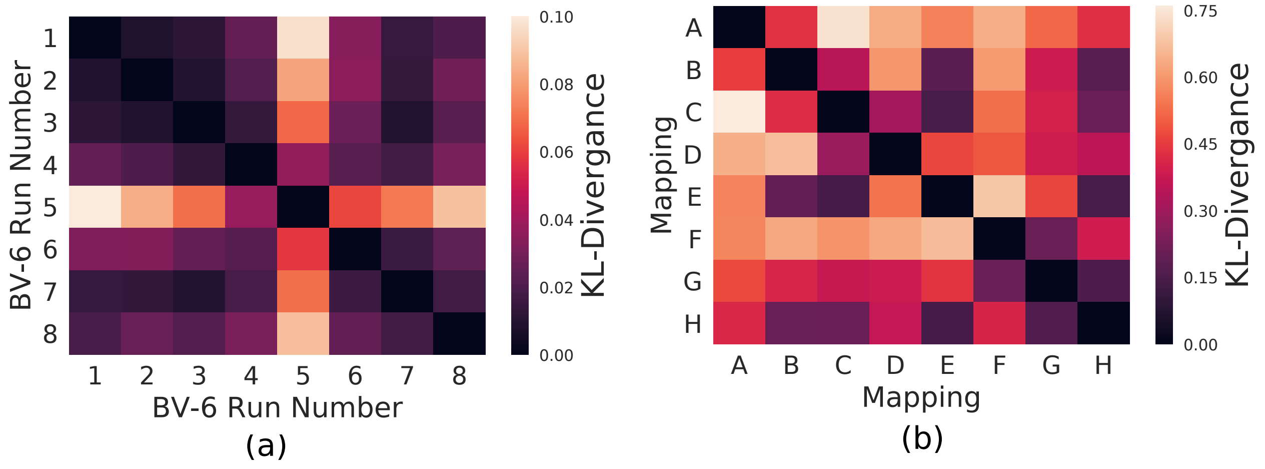


Figure 4.4: (a) Divergence between output of eight BV-6 runs with the strongest mapping. Dark squares indicate a value close to zero (indicating that the distributions are close to identical). (b) Pairwise divergence for the output of eight copies of BV-6 that are run with eight different mappings (light colors indicate divergent distributions).

BV-6 with Diverse Mappings: In the second experiment, we run BV-6 benchmark with eight completely different mappings and estimate the divergence between output probability distributions. Figure 4.4 shows a heat-map corresponding to the pairwise KL divergence for the eight copies of BV-6. We observe significant dissimilarity between all the eight copies of BV-6 such that average KL-divergence between two copies is 0.5, which is significantly higher as compared to eight runs of single best copy with average KL-divergence of 0.03. Furthermore, the most frequently occurring incorrect answers show a large variation across eight copies of BV-6. Thus by introducing diversity in the qubit mapping, we can enable diversity in the output probability distribution. Note that all the mappings used were within 10% of the ESP of best mapping and the executed identical number of gates.

4.5 Analyzing Correlation in Errors via Buckets-and-Balls Analysis

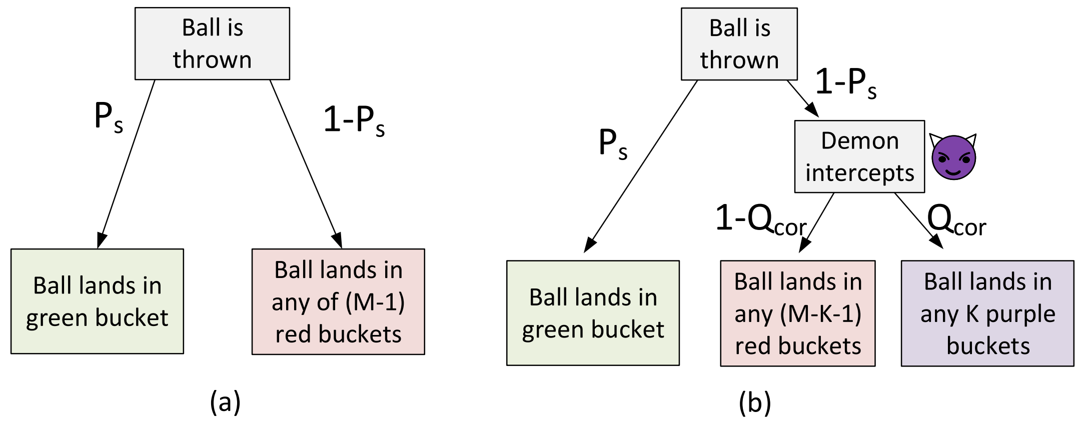
To understand the impact of correlated errors on the inference quality of a NISQ machine, we use buckets and balls analysis.

4.5.1 Execution on NISQ as Buckets-and-Balls

The output of NISQ programs can be analyzed as buckets and ball problem. On NISQ machines, running a program that outputs m-bit string for N trials is equivalent to throwing N balls at the M buckets where $M = 2^m$. In this experiment, we have two types of buckets: green bucket that represents the correct answer and red buckets that represent all possible incorrect answers. We don't know the green bucket, but we can guess it by throwing a large number of balls and tracking the bucket with the most number of balls.

4.5.2 Analytical Model for Uncorrelated Errors

For N balls and M buckets, if P_s is the probability of the ball landing in a green bucket then $(1 - P_s)$ is the probability of the ball landing in the any of the $M - 1$ red buckets as shown in Figure 4.5(a). With no correlation, the likelihood of ball landing in any of the $M - 1$ red buckets would be identical. For large N, number of balls in the green bucket (correct answer) would approach expected value of a Bernoulli trial: NP_s



Ball is thrown
P_s
1-P_s
Ball lands in green bucket
Ball lands in any of (M-1) red buckets
(a)
Ball is thrown
1-P_s
P_s
Demon intercepts
1-Q_{cor}
Q_{cor}
Ball lands in green bucket
Ball lands in any (M-K-1) red buckets
Ball lands in any K purple buckets
(b)

Figure 4.5: Buckets and Ball Model for NISQ (a) uncorrelated errors (b) correlated errors.

whereas number of balls in red bucket that has highest occupancy would be at the most $NP_e + 2 * \sqrt{(N * P_e * (1 - P_e))}$ (with 95% confidence), where $P_e = \frac{1-P_s}{M-1}$.

We use an analytical model (confirmed with Monte Carlo simulator) to understand how IST changes with P_s and M. For instance, Figure 4.6 describes the relationship between IST and P_s for M=64 buckets. The uncorrelated error model suggests that even with P_s=2%, we can distinguish the green bucket from rest as IST>1. Unfortunately, on real quantum computers, this model does not hold. Figure 4.6 show experimental P_s and IST data (blue dots) for three 6-bit applications (QAOA-6, BV-7, Grey-code) for 120 experiments executed on IBM-Q14 quantum computer. The experimental data show significantly smaller IST compared to the uncorrelated model for an identical P_s. To understand the mismatch, let's change our model and account for correlated errors.

4.5.3 Analytical Model for Correlated Errors

Correlated errors break the assumption that all incorrect answers are equally likely. To account for correlated errors, let's introduce a Demon in our model. This Demon biases errors such that k outcomes out of $2^m - 1$ incorrect outputs are more likely than the rest of the $2^m - k - 1$ outputs. These k more likely incorrect answers can be represented as purple buckets.

As shown in Figure 4.5(b) correlation-factor (Q_{cor}) determines what fraction of balls land in the k purple buckets after demon intercepts. The probability of balls hitting in the purple buckets is $(1 - P_s) * (Q_{cor})$ and probability of ball hitting in any of the k purple bucket is $\frac{(1-P_s)*(Q_{cor})}{k}$. Figure 4.6 shows the result of the Monte Carlo simulation displaying the relationship between IST and P_s for $M = 64$, and $k = log(M) = 6$ and range of Q_{cor}. For simplicity, we assume that k scales with $O(log(M))$ as the correlation among errors tend to be local.

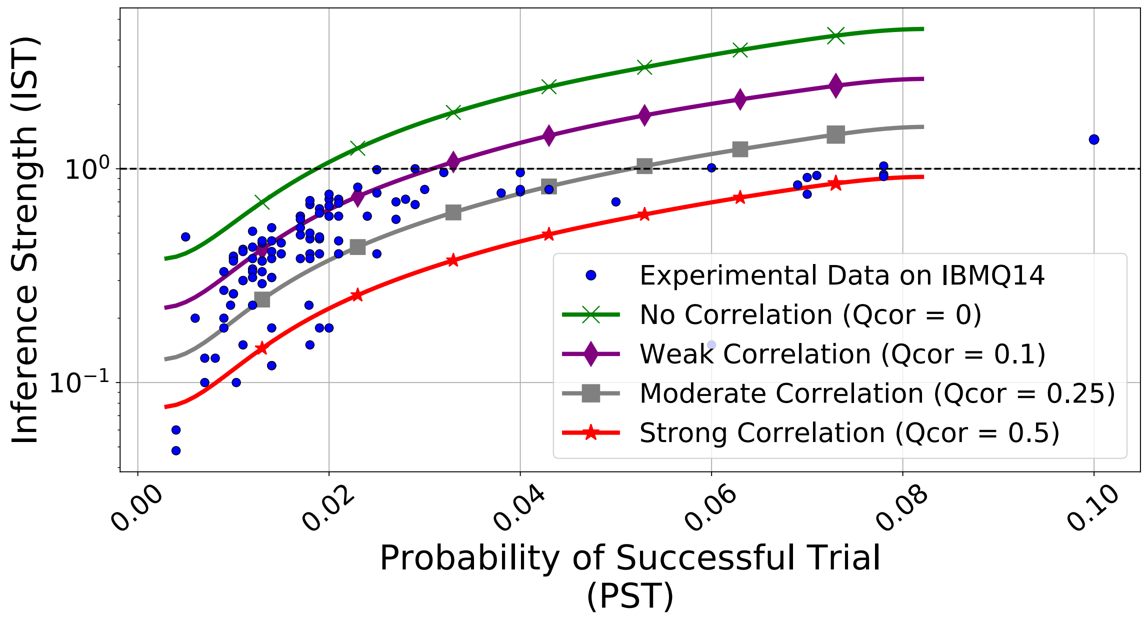


Figure 4.6: Inference Strength (IST) vs Probability of Successful Trial (PST) for Buckets and ball model and experimental data for 120 runs (each run was evaluated with 8192 trials) of QAOA-6, BV-6, and greycode-decoder on IBMQ-14 machine.

To understand the impact of correlated errors on reliability, we compute PST frontier Using Monte Carlo simulations. PST Frontier is the minimum PST required to infer the correct answer from given output distribution (PST at which IST=1). For the model with no correlation, PST frontier is at 1.8%, that means for 6-bit application with PST>1.8%, we can always deduce the correct answer. The PST Frontier shifts right to 3.6% with correlated errors that have weak correlation (Qcor=10%). Moreover, it shifts even further at 8% for strong correlation model (Qcor=50%).

Unfortunately, there is no simple way of deducing the correlation-factor on the real machine as it depends on the device characteristics and the type of application that we are running. High PST frontier degrades the effectiveness of NISQ applications like QAOA. For example, our experiments show that QAOA-6 with baseline policy consistently fails to meet PST Frontier criteria as it has a median PST of 2.5% and IST of 0.78 on IBMQ-14 for 30 experimental runs.

4.6 Ensemble of Diverse Mappings

To mitigate the correlated errors on NISQ machines, we propose *Ensemble of Diverse Mapping (EDM)*. In this section, we will discuss design and reliability improvement provided by the EDM.

4.6.1 Motivation

Variation-aware qubit allocation improves the reliability of NISQ machines [16, 15, 6, 18]. However, running a NISQ application with just one mapping can increase its vulnerability to correlated errors. Running the program with single mapping multiple times produces incorrect outcomes with correlated errors. To mitigate the correlation, we need to introduce diversity in the program. One way to introduce variety in the program is by running the input program using a diverse set of qubit devices rather than being restricted to always using the same program assignment for all of the trials.

To test if the diverse mappings provide better reliability, we use BV-6 benchmark. Similar to the previous experiment, we use eight different logical to physical mappings (`A,B,C,D,E,F,G,H`) to run BV-6 on IBMQ-14 each for 16,384 trials. Figure 4.7 shows the *IST* for BV-6 with different mappings. IST captures the relative strength of the correct answer compared to the incorrect answer. When we use different mappings, we can expect variation in the reliability of individual qubit assignments. For example, Mapping C produces the output probability distribution with highest IST as compared to the other mappings. However, no single mapping has the IST exceeding 1. IST greater than one means the correct answer occurs with the highest frequency. To test if an ensemble of mapping can improve the IST, we execute the BV-6 for 4096 trials with mappings A, B, C, and D and merge the output probability distributions to generate EDM. We use 4096 trials each to match the number of trials in the baseline that runs with the single best solution.

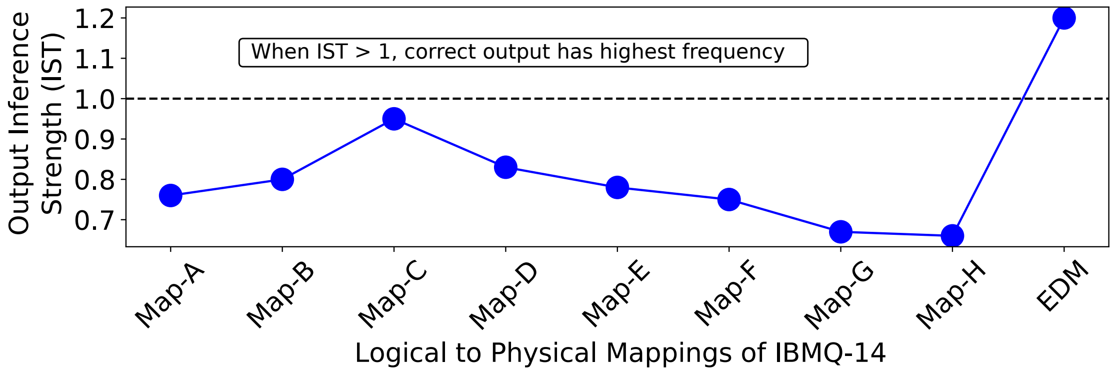


Figure 4.7: IST for BV-6 executed with the eight different mappings (A-H) on IBMQ-14 and the Ensemble of Mappings (EDM: A+B+C+D). Note that, none of the individual mappings have an IST ≥ 1, but the EDM has IST of about 1.2.

Figure 4.7 shows the IST of 1.2 for BV-6 when executed with an ensemble of qubit assignments. The Ensemble of mapping improves the IST as incorrect answers get averaged out when we merge output probability distributions that are not similar. Use of Ensembles is one of the proven machine learning techniques that can improve the accuracy and robustness of classification tasks [23].

EDM is inspired by the principle of maximum entropy that suggests the probability distribution, which best represents the current state of knowledge is the one with largest entropy [24]. By using diverse mappings, EDM tries to avoid the repeated incorrect answers such that the incorrect results are spread across multiple outcomes.

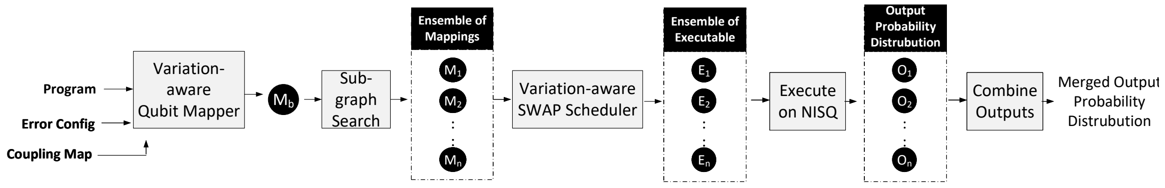


Figure 4.8: Overview of EDM: Design contains four steps (1) get top "K" mappings (2) generate K executable (3) Run the trials for each executable (4) merge the output probability distributions to create the combined output for the ensemble.

4.6.2 EDM: Overview and Design

Our proposed *Ensemble of Diverse Mapping (EDM)* enables diversity in the output distributions by using an ensemble of qubit mappings. Figure 4.8 provides an overview of EDM. EDM contains four steps. In the first step, a compiler generates the best initial mapping and SWAP schedule for a given input program using coupling map (network topology) of a quantum computer and the error rate characterization data. In the second step, we use the initial mapping, and find all the isomorphic sub-graphs for the given quantum computer, and rank the sub-graphs as per the Estimated Success Probability (ESP). EDM picks the top "k" sub-graphs based on the ESP. In the third step, we re-compile the program by using the ensemble of initial mappings $(M_1, M_2, ..., M_n)$ to produce an ensemble of executable $(E_1, E_2, ..., E_n)$, and run all executable on a NISQ machine as shown in the Figure 4.8, to produce set of output probability distributions $(O_1, O_2, ...O_n)$. Finally, we merge the probability distributions of all the members in the Ensemble to generate the final result.

For the first step, EDM can use any variation-aware quantum compiler. In this work, we use variation-aware qubit mapper that uses A* search with reliability-aware heuristics proposed by [9, 15]. Furthermore, we use ESP as a cost function to select the strongest mapping on IBMQ-14 [18]. ESP incorporates measurement and single qubit gate errors. We also use benchmark specific heuristics to ensure optimal mapping. For example, a path graph satisfies the CNOT constraints for QAOA such that no SWAPs are required to perform QAOA. We verify the cost of all the mappings by using a brute force search

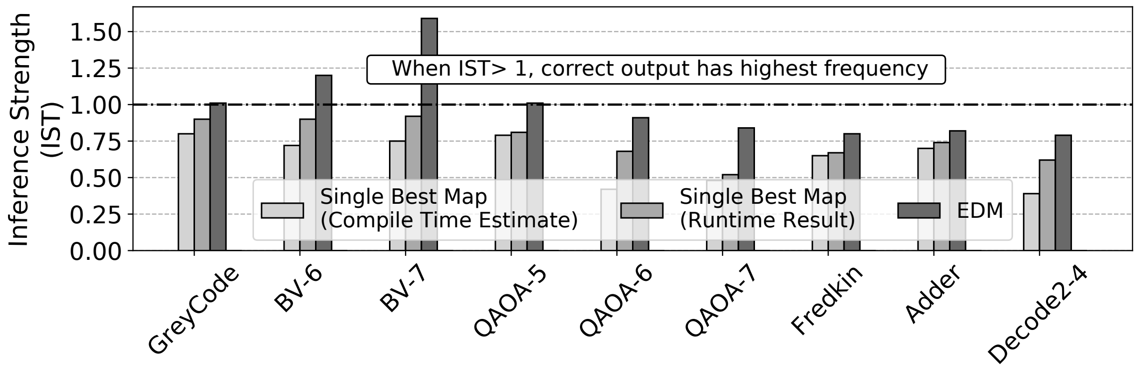


Figure 4.9: Improvement in IST with EDM, compared to single-best mapping. EDM has significantly higher IST compared to the best single mapping (both: that is estimated at compile time and the one that is observed at runtime).

to check the optimally of the mapping. For BV and QAOA, our compiler produces an optimal mapping.

To generate the Ensemble of initial qubit assignments, we need to ensure that the selected mapping has high reliability. When assigning program qubits to physical qubits, two major factors impact the output reliability: measurement errors and two-qubit gate errors. To leverage the mapping produced by the variation-aware mapper, we use graph isomerism to transfer the mapping from one set of qubits to another set of qubits. We search for all isomorphic sub-graphs, on the IBMQ-14 coupling graph using VF2 algorithm [25]. Once we have the list of all isomorphic graphs, we compute ESP and select the sub-graphs with highest ESP.

For the final step of producing the combined probability distribution, we use a simple average to merge the probability distributions of all of the members in the Ensemble.

4.6.3 Why Select the Top-K Mappings?

Variation aware allocation policies use compile-time information to estimate reliability by using metrics such as ESP. However, maximizing the ESP at compile time may not always result in maximizing the PST at runtime, as the behavior of the devices can change unpredictably at runtime. Figure 4.10 shows ESP and the corresponding PST after evaluation for eight maps used for BV-6. There is a good correlation between ESP and PST. However, this correlation is not perfect. For example, Map-A is estimated to be the best mapping at compile-time; yet, at runtime, Map-C has the highest PST. Moreover, picking mapping with highest ESP cannot guarantee the highest IST. As error calibration data used to estimate ESP is not perfect due to temporal variations in qubit reliability and error-rates can change substantially due to cross-talk. Nonetheless, there is a good correlation between mappings that are good at compile time with the mappings that produce the highest PST at the runtime. Hence, we use the top K mappings to generate our Ensemble for EDM.

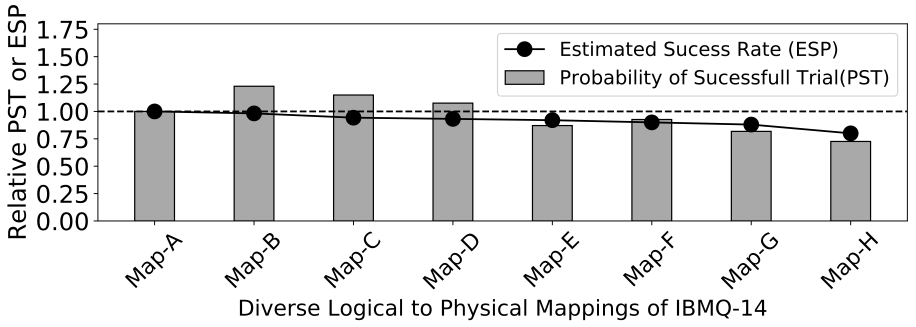


Figure 4.10: Comparing estimated reliability (ESP) at compile-time and observed reliability (PST) at run-time for BV-6 with eight different qubit mappings.

Our evaluations also show a weak correlation between PST and IST. For example, a slight improvement in PST for a given mapping does not result in increase in IST as the probability of the wrong answer can increase as well. Our analysis encountered several cases where a mapping with the highest ESP had lower IST compared to other mappings. We could form an ensemble of mappings that is estimated to produce the highest IST, however, to keep the design simple, we select the top K mappings that are deemed to have the highest PST for forming EDM. [2]

[2]In extreme cases, the noise profile of the machine can change quickly, and cause the output distribution to be close to uniform. We can identify such cases by computing the relative standard deviation (σ/μ) of the probability distribution, comparing it with that of the uniform distribution, and discarding the results if the distance is quite small. We found such a strategy to be quite useful under such cases of

4.6.4 Impact of EDM on Inference Strength

EDM is designed to mitigate the correlation in errors and improve the IST such that the frequency of individual incorrect answer reduces by spreading the mistakes. Figure 4.9 shows the improvement in IST for QAOA and BV. We compare EDM against two different mappings: *single best mapping at compile time* that is estimated using ESP and single best mapping post-execution which is evaluated after running an ensemble of mappings. For example, as shown in the Figure 4.10, we estimated Map-A as the most reliable mapping based on its ESP. However, after running BV-6 with other mappings, we may realize that Map-C has the highest PST. To understand if the benefits of using Ensemble are due to diversity in the mappings or because of uncertainty in ESP, we also compare EDM with another baseline, *single best mapping post execution*, which represents the best mapping encountered at runtime. For example, this would be Map-C, as shown in the Figure 4.10.

The ensemble of mappings not only outperforms the best-estimated mapping at the compile time but also beats the best-single mapping encountered at runtime. This suggests that uncertainty in ESP is not a key reason behind the success of EDM. As for QAOA-5, the estimated best mapping at compile time is identical to the mapping at runtime, and even then EDM outperforms the baselines. EDM increases the entropy of output distribution such that, for the resulting output probability distributions, errors are spread across multiple possible incorrect answers.

4.6.5 Impact of Ensemble Size

There is an inherent trade-off in ensembles selection. By increasing the size of Ensemble, we can introduce more diversity, but at the same time, we expose the program to relatively unreliable qubits. Finding the right size of an ensemble is especially crucial for the IBM machine, as it shows high variability in error rates. Our default implementa-

extreme noise.

tion of EDM uses four mappings in the Ensemble. The number of ensembles is dictated by our ability to find the initial mapping that has similar SWAP cost and the ESP. EDM finds the graphs that are isomorphic to the initial mapping produced by the baseline. For IBMQ-14 due to limited connectivity, and high variability[3] in error rate, we observe that number of strong ensembles are limited two to four.

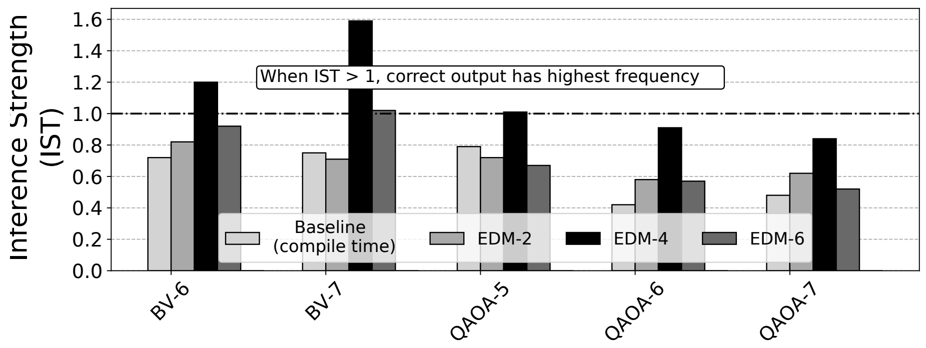


Figure 4.11: Sensitivity of EDM to the number of members in the Ensemble. With increasing ensemble, computation gets mapped to weaker qubits. Hence the benefit of EDM with larger ensemble size starts to reduce.

We evaluate the sensitivity of EDM to the number of members in the Ensemble. We form Ensemble with two mappings (EDM-2), four mappings (EDM-4, default), and six mappings (EDM-6) and run the workloads with the differently sized ensembles. Figure 4.11 shows the IST of the EDM with varying ensemble sizes. We observe that with only two members in the Ensemble, we do not add enough diversity, and in fact, the other copy can reduce the overall PST slightly for some cases and reduce the IST compared to even the baseline (BV-7 and QAOA-5). When the Ensemble contains four members, there is a good balance between the increase in diversity and the loss of PST. Overall we see significant improvement in IST. When the Ensemble contains six members, the mapping is forced to choose qubits that may have significantly lower reliability than the best qubits, and the overall degradation of PST is significantly greater than the gain from combining the diverse outputs. Therefore, in our experiments, we use a default size of

[3]IBM-Q14 machine has two significantly noisy qubits Q12 and Q11 with readout error-rates up to 30%, we avoid using these qubits, which puts more constraints on finding a right isomorphic subgraph

4 members in the Ensemble to balance both the increase is diversity and the pitfall of being forced to use more unreliable qubits for computation.

Note that the best number of ensembles will depend on the machine and the correlation in errors on that machine. So, there is no single best number of members in EDM that will always work well across variety of machines. We would recommend that users of EDM perform sensitivity while deciding the ensemble size.

4.7 Weighted EDM

One of the limitations of our proposed implementation of EDM is that the members of the ensemble are based on prioritizing the maximization of ESP rather than maximizing the diversity in forming the ensemble. Therefore, outputs of some of the mappings can have a similar output probability distribution if mappings in an ensemble have a common set of qubits. Unfortunately, on existing IBM machine, due to a large variation in error rates, a small number of qubits finding two sub-graphs that use a completely different set of qubits but have comparable ESPs is challenging. The effectiveness of EDM stems from the diverse set of outputs, and even a few unique qubit mappings can produce diverse incorrect answers. For example, in the case of BV-6, all the eight mappings had two to three common qubits. However, the diversity of the output was significant as illustrated by the Figure 4.4(b). Moreover, for all eight mappings, the common qubits are the qubits that are less likely to get errors. It might be possible to have enough diversity with few common qubits between two mappings in an ensemble.

4.7.1 Design of Weighted EDM

To maximize the diversity without deteriorating the reliability, we propose *Weighted Ensemble of Diverse Mappings (WEDM)*. Weighted EDM uses runtime information to maximize the diversity in the output probability distributions. In essence, it is risky to improve diversity at compile time by picking mapping with lower ESP. Whereas, we can solve this problem more efficiently at runtime. For instance, we can evaluate the diversity in the probability distributions and then perform scaling operation to increase the diversity. In contrast to EDM where we merge output probability distributions with identical weights, WEDM uses weighted average such that the weight is proportional to the cumulative mutual entropy of the output as shown in the Figure 4.12.

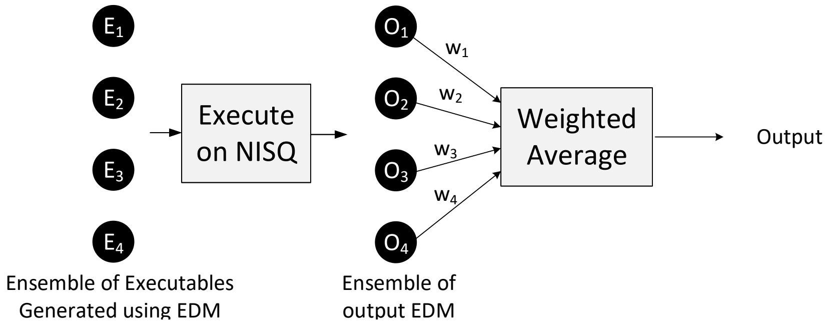


Figure 4.12: Design of Weighted EDM (WEDM).

The cumulative entropy of the output represents the uniqueness of the output probability distribution. For example, if we have four member ensemble with A, B, C, and D, we would calculate the KL-divergence of A with B, C, and D respectively and average these three values. The output of A will receive this weight before getting merged with the aggregated output distribution. A similar process will be repeated for B, C, and D.

For weighted EDM (WEDM), we use symmetric KL divergence (SD_{KL}) to compute resultant output probability distribution (O_{WEDM}) that is a weighted sum of ensemble output probability distributions (O_i). For N ensembles, the output probability distribution (O_{WEDM}) and normalized ensemble weights ($\overline{W}$) are evaluated as follows:

$$O_{WEDM} = \sum_{i=0}^{i=N} \overline{W_i} * O_i \tag{4.1}$$

$$W_i = \sum_{j=0}^{j=N} SD_{KL}(O_i, O_j) \quad \& \quad \overline{W_i} = \frac{W_i}{\sum_{i=0}^{i=N} W_i} \tag{4.2}$$

4.7.2 Impact of WEDM on Inference Strength

Figure 4.13 shows the improvements in IST with EDM and WEDM. WEDM improves the IST by up to 2.3x over the estimated single best mapping such that the correct answer has 1.73x higher likelihood compared to the incorrect answer. Both WEDM and EDM not only outperformed the estimated best mapping at compile time, but also showed improvements even over the single best mapping that we would have picked if we knew the behavior at runtime. With WEDM, all the workloads enter a regime where the correct answer has the highest frequency of occurrence. Thus achieving our goal of having higher confidence in the inference for NISQ.

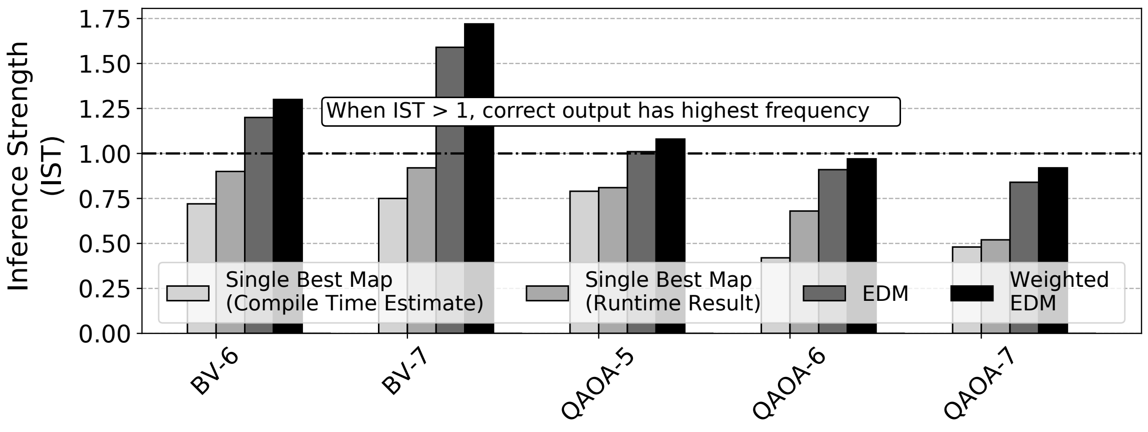


Figure 4.13: IST improvement with EDM and Weighted EDM (WEDM) over the baseline which uses the single best allocation for all of the trials. EDM and WEDM provide significant improvement in system reliability.

Using ensembles, we improve the IST but we can degrade the PST slightly as we use mappings that are not the most optimal when running EDM and WEDM. In both EDM and WEDM, at the end of the execution, we combine output probability distributions such that each entry in the distribution is averaged. The PST of an ensemble is bounded by the best and worst mapping in an ensemble. As we scale the workload, small improvements in PST or degradation does not change the effectiveness of NISQ applications. Whereas, improving the IST can correlate with the ability of the NISQ machine to infer the correct answer.

4.8 Summary

The arrival of quantum computers with dozens of qubits will enable a better understanding of the impact of qubit errors on applications. This can help us in developing efficient solutions to mitigate errors. In NISQ computing model, the program is run thousands of times, and the output log is used to infer the outcome. The ability to infer the correct outcome depends on both the probability of the correct outcome and the probability of the most-frequently occurring incorrect outcome. We focus on the latter to improve the ability to infer the correct answer on NISQ machines.

Existing qubit allocation schemes search for one best mapping, and this mapping is used for all the trials. Unfortunately, such a method is vulnerable to correlated errors. The correlation in errors causes a few wrong answers to repeat for a large number of trials. To mitigate correlated errors, we leverage the principle of diversity, and propose an *Ensemble of Diverse Mappings (EDM)*. With EDM, the total number of trials are divided into multiple groups and a different mapping is applied to each group. To keep the implementation of EDM simple, we use the top-4 mappings produced by the underlying mapping policy. We show that with EDM, the magnitude of the dominant wrong answer decreases and the reliability of the NISQ system increases by up to 1.6x.

EDM merges the probability distributions generated by each of the mappings using an equal weight. We make an observation that the runs that have similar output have less information than the runs that have different outputs. Based on this insight, we propose *Weighted Ensemble of Diverse Mappings (WEDM)* that scales the output distributions generated by each of the mappings with ensemble weights. WEDM improves reliability by up to 2.3x.

The key idea in our paper is to have multiple versions of the same quantum program, each tailored for a diverse set of mistakes. In this work, we specifically use mapping policies to create such diverse programs. However, there are other sources of program transformations that can provide diversity as well. Exploring such diversification of quantum programs using alternative transformations is a part of our future work.

CHAPTER 5
PROGRAM TRANSFORMATIONS TO MITIGATE MEASUREMENT ERRORS

5.1 Introduction

A program running on a NISQ machine can encounter an error due to coherence, gate operation, or due to measurement at the end of the computation. Measurement is typically the most error-prone operation on the current quantum computers. For example, on the IBM machine, the average error rate for readout (measurement operation) is 6%-8% with the worst-case error rates for the readout ranging from 25%-30%. Reading a qubit is fundamentally challenging as qubits are extremely low energy devices and during a process of readout, qubit devices interact with the noise of the measurement circuitry. Thus, even if a quantum machine performs all the computation without encountering an error, in the end, the measurement operation can still result in an erroneous output. The goal of this paper is to improve the reliability of near-term quantum computers by mitigating the impact of measurement errors.

Measurement operations are designed to collapse the qubit in a state of superposition into a classical binary state, 0 or 1. Thus, a measurement error manifests itself as either a "1" being read as a "0" or vice versa. In this work, we observe that measurement errors do not affect all states equally. For example, on IBM machines, measurement errors have state-dependent bias such that the state "1" is erroneously read "0" ($1 \rightarrow 0$) more frequently as compared to state "0" measured as the state "1" ($0 \rightarrow 1$). While measuring a collective state of N qubits, we would expect to encounter more errors for states that have a large number of ones. We can exploit this bias to mitigate the impact of measurement errors.

To show the state dependent bias in measurement errors we conduct a simple ex-

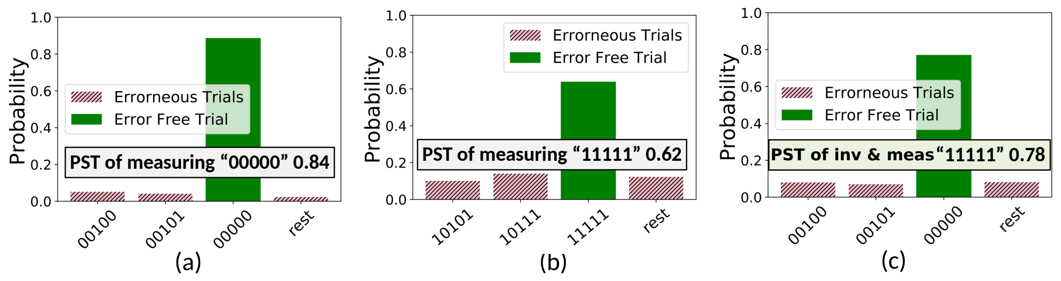


Figure 5.1: Probability of successfully measuring a state (a) All-zero state "00000" (b) All-ones state "11111" (c) Measuring the All-ones state by first inverting the state and then performing the measurement (expected output "00000").

periment. We initialize the IBM-Q5 (five qubit) machine into an all-zero state (00000) and measure the state. This experiment was repeated for one thousand trials. The experiment is deemed to give the right output if we obtain the 00000 state and incorrect if it gave any of the other 31 possible values. Figure 5.1(a) shows the probability of obtaining the correct answer and the probability of obtaining a few of the dominant incorrect states. We note that the probability of successful measurement for the all-zero state is 84%. Conversely, if we initialized the machine in an all-ones (11111) state, then the probability of successful measurement drops to 62%, as shown in Figure 5.1(b). Thus, reading a state of "1" is usually more error-prone than reading a "0" state.

We also conducted an exhaustive experiment will all 32 states ("00000" to "11111") and observed that the probability of successful measurement shows a strong inverse correlation with the Hamming Weight (number of ones) of the state being measured. So, states with higher number of ones are more susceptible to measurement errors than the states with fewer ones. Furthermore, this state-dependent bias is observed even for qubits in the state of superposition. For example, a GHZ state is an equal superposition of all-one and all zero state, when measured it is expected to produce the all-zero state and the all-one state with 50% probability each. We observe that on IBM-Q5, the all-zero state was four times as likely as the all-ones state. We note that our experiments with all three publicly available IBM machines (two 5-qubit machines and one 14-qubit machine) show such state-dependent bias in measurement errors.

The key insight is to exploit the state dependent bias to reduce the impact of measurement errors. For example, if we are likely to read a vulnerable state (high Hamming Weight), then we can transform it into a stronger state (low Hamming Weight) by inverting qubits before the measurement, performing the measurement, and inverting the measured result. We refer to such a method of conditionally converting the state to obtain lower measurement errors as *Invert-And-Measure*. For example, reconsider the example of Figure 5.1(b) where we read the all-one state. Instead, if we inverted the state (by using an additional "X" gate at each of the qubits)[1] before the measurement and then we perform the measurement, then the probability of successful measurement increases from 62% to 78%, as shown in Figure 5.1(c). Note that the measurement produces a complementary state (all-zeros on correct output) and we must invert this output to get the desired results. Unfortunately, prior to measurement, we do not know the Hamming Weight of the system (and hence we do not know if the system is in a weak state or strong state). Always using inversion before measurement can degrade reliability if the system was already in the strong state. We design two practical policies for Invert-and-Measure.

5.2 Qubit Measurement Errors

Measurement is the most error-prone operation in current quantum computers. Table 5.1 shows the minimum, average, and maximum error-rates for the measurement operation (readout operation in IBM terminology) for the three IBM machines that we use in our evaluations. We note that the average error rate for measurement operation in the range of 4%-8% and as high as 31%. The high rate of measurement errors can reduce the application level reliability, especially for low depth programs.

[1]Note that Invert-and-Measure increases the overall gate count due to the addition of the X gate. Fortunately, single qubit gates (such as the X gate) have a low error rate of approximately 0.1%, which is almost 100x smaller than the error rate of the measurement operations. Therefore, the vulnerability due to the extra X gate itself does not have a significant impact on the overall system reliability.

Table 5.1: Error Rates of Measurement Operation

Machine	Error Rate		
Name	Min	Mean	Max
`ibmqx2 (IBM-Q5A)`	1.20%	3.8%	12.8%
`ibmqx4 (IBM-Q5B)`	3.4%	8.2%	20.7%
`ibmq-melbourne (IBM-Q14)`	2.2%	8.12%	31%

5.2.1 Impact of Measurement Errors on NISQ Applications

Errors can cause the NISQ machine to produce an erroneous output. Figure **??** (b) shows the distribution of output for a Bernstein-Vazirani (BV) kernel storing 2-bit key "01". On an idealized, error-free machine, we would get the secret key with 100% probability. Errors can cause the trial to produce incorrect output, and we should see a distribution of all possible outputs. For example, in Figure **??**(c) shows a case where the correct output occurs with 50% probability, and each of the incorrect output occurs with no more than 25% probability. Thus, we can correctly infer the key, even in the presence of errors. Now, lets say we stored a different key ("11"), as shown in Figure **??**(d). The correct output occurs with 30% probability, and one of the incorrect outputs occurs with 35% probability. In such cases, we will not be able to infer the correct key by analyzing the log. This can especially happen if certain states are more vulnerable to measurement errors, causing a bias in output to go from the correct output to incorrect output.

5.3 Bias in Measurement Errors

This section provides characterization data for measurement errors, in particular highlighting the data pattern dependence of measurement errors and the correlation with the Hamming Weight (the number of 1s in the pattern).

5.3.1 Data Dependent Bias

Qubit devices have a natural tendency to relax to the low energy state (0) from the high energy state (1). This creates a data dependent bias in measurements as qubit which already in State 0 is less likely to change its state due to natural relaxation process as compared to the qubit in State 1. For example, our evaluations on IBM quantum computers show that a qubit in an excited state is more likely to encounter an error as compared to the qubit in a low energy state. To understand the data-dependent bias in measurement errors, we generate all the 32 possible states (00000 to 11111) and measure each state 16 thousand times. We compute the *Probability of Successful Measurement (PSM)* as the ratio of correct output to the number of trials. Figure 5.2 show the relative PSM for all 32 basis states on IBM's five qubit`ibmqx2` machine. Note that the x-axis is in the ascending order of Hamming Weight. We calculate relative PSM by dividing the PSM of each basis state with the highest PSM. For `ibmqx2`, state "00000" is the strongest basis state, whereas states "11111" is the weakest state with relative PSM of 0.38, with the relative PST reducing with increasing Hamming Weight.

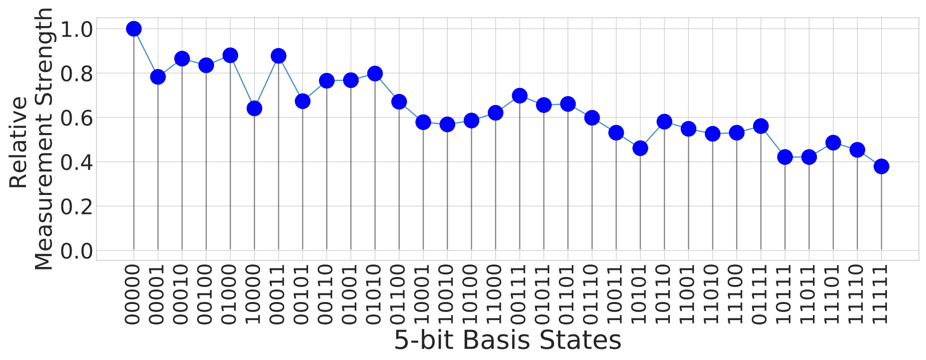


Figure 5.2: Probability of Successful Measurement for `ibmqx2` basis states (X-axis shows five bit basis states in ascending order of hamming weights).

With unbiased measurements, PSM for all the basis states should be similar. However, our evaluations suggest that the probability of successful measurement is inversely proportional to the Hamming weight of basis states (Correlation coefficient = 0.93). Thus, larger the Hamming weight higher the probability of measurement error.

To understand the impact of the size of the quantum computer, we run similar experiments with 10-bit basis states on `ibmq-melbourne` for 150 thousand trials. Figure 5.3 show the relative PSM for all 1024 basis categorized as per the Hamming Weight of the basis state. The data again shows a strong inverse correlation between the measurement strength and the Hamming Weight.

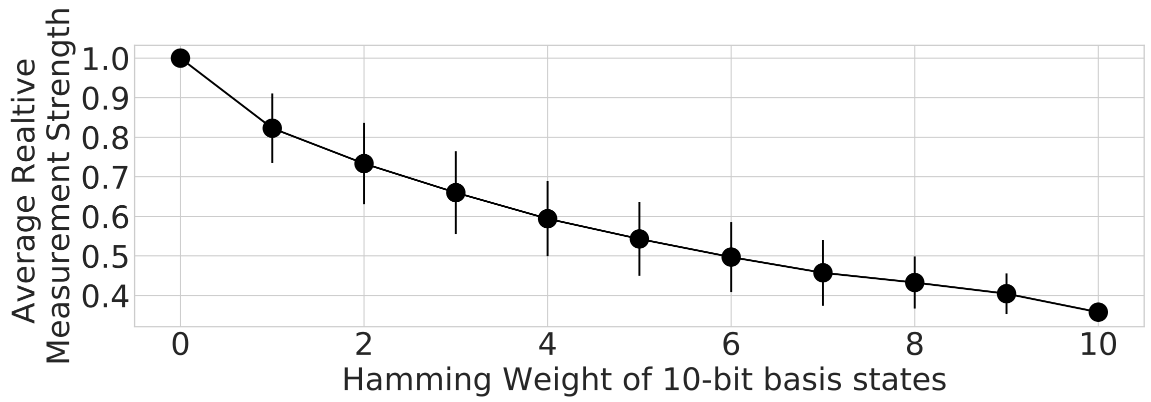


Figure 5.3: Probability of successful measurement for the `ibmq-melbourne` machine. The data is averaged over all the basis states with the identical Hamming Weight.

5.3.2 Impact of Bias on Superposition of States

Thus far, we have measured the vulnerability of measurement errors to states that do not have any superposition. However, measurement errors can have data dependent bias even for qubits that have superposition. To understand how state dependent measurement bias affects the superposition of states, we create the *Greenberger-Horne-Zeilinger (GHZ)* state, which is an equal superposition of basis-states "00000" and "11111" (GHZ-5 $=\frac{1}{\sqrt{2}}(|00000\rangle + |11111\rangle)$).

If the GHZ-5 state is prepared and measured on a quantum computer with no errors, the output will be either "00000" or "11111" with 0.5 probability respectively. However, when prepared and measured on the IBM machine, the probability of measuring state "00000" and state "11111" are unequal. As shown in the Figure 5.4, PSM degrades from 0.5 to 0.4 for state "00000" and from 0.5 to 0.1 for sate "11111". Our experiments suggest that measurement bias extends to the superposition of basis states. Note that GHZ states are considered to be the maximally entangled state; thus measurement bias affects qubits that are in superposition and entanglement.

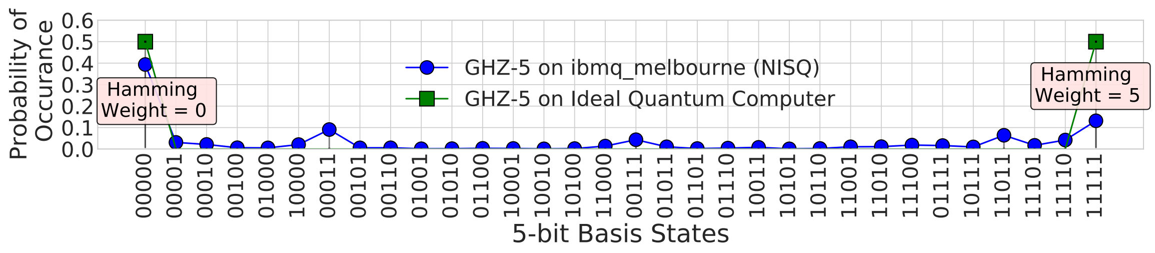


Figure 5.4: Output probability distribution for GHZ5 (X-axis is in ascending order of the Hamming Weight). On ideal machine both "00000" and "11111" occur with 0.5 probability – errors affect state 11111 four times as state 00000.

5.3.3 Impact of Bias on NISQ Applications

State-dependent bias in the measurement errors produces non-uniformity in the measurement strength of basis states. The skew in measurement strength degrades the reliability of NISQ machines. For example, if the desired or optimal answer is a weak basis state, then the probability of measuring the answer drops significantly. Furthermore, weak states are often incorrectly measured as strong states. To understand this masking effect, we execute an instance of QAOA on the `ibmq-melbourne` machine. We use QAOA to solve the max-cut problem for five input graphs (Graph-A to Graph-E) such that each graph consists of six nodes and the desired output (the partition that maximizes the cost function) is in the increasing order of Hamming Weight, as shown in the Table 5.2. We execute each graph for 32 thousand trials. All five graphs use an identical number of gates and the optimal mapping (aware of variation in error rates of different qubits).

Table 5.2 also shows the Probability of Successful Trials (PST), Inference Strength (IST) and the rank of the correct answer for the five graphs. We observe that PST is inversely correlated to the Hamming Weight. As for input graphs A and B, the PST is 2x

Table 5.2: Impact of measurement bias on QAOA

Input Graph	Optimal Output	Hamming Weight	Metric		
			PST	RS	Rank
`Graph-A`	010000	1	6.5%	1.3	1
`Graph-B`	010100	2	5.5%	1.01	1
`Graph-C`	101001	3	5.0%	0.70	7
`Graph-D`	101011	4	1.9%	0.59	14
`Graph-E`	110110	4	1.5%	0.23	24

higher as compared input graph D and E. Thus, state-dependent measurement bias can significantly deteriorate the reliability of the application.

The bias in measurement has a significant impact on the Inference Strength (IST) and Rank of correct answer as well. IST drops significantly for the input graphs E and F. When the expected output is a weak state (Hamming Weight= 3 or 4) the incorrect answers have a higher frequency of occurrence compared to the correct answer.

To understand how measurement bias impacts the ability to infer the optimal answer, we use the rank of the correct answer. For A and B the correct answer appears with the highest frequency, whereas, for D and E (high Hamming Weight) the incorrect answers mask the correct answers as the correct output corresponds to a weak basis state.

5.4 Evaluation Methodology

We perform our evaluations using publicly accessible IBM machines. We use key kernels that are typically used for evaluating the performance of NISQ machine. Finally, we discuss several figure-of-merits that are appropriate for assessing the reliability of a NISQ machine.

5.4.1 NISQ Benchmarks

Developing applications for near-term quantum computers is an open problem [4, 5]. *Quantum Approximate Optimization Algorithm (QAOA)* [26] has emerged as an appealing use for NISQ machines as it can be used to solve discrete optimization problems. We use QAOA as one of the kernels for our evaluation. The other kernel we use is Bernstein-Vazirani (BV)[13] where a secret key is stored and the kernel allows the inference of the key. Note that both `bv` and `qaoa` produce a binary string as the solution. For example, the output of QAOA when solving a max-cut problem represents a partition that maximizes the cost function. Whereas, BV is an oracle that finds a secret key and outputs a binary string corresponding to the secret key. On an ideal quantum computer (with no qubit errors), these applications will produce correct output with certainty. In the case of BV, the correct output is produced with the probability of one, whereas in the case of QAOA, the correct output string has the highest frequency of occurrence. Note that QAOA solves an optimization problem – solutions produced with QAOA can be used to calculate the cost function, and the answer corresponding to optimal value among all the tested solution can be used as the good enough solution. Table 5.3 shows the different configuration of BV and QAOA used in our study.

Table 5.3: Benchmark Characteristics

Benchmark	Problem/Algorithm	Output
`bv-4A`	4 bit Bernstein-Vazirani	Secret: 0111
`bv-4B`	4 bit Bernstein-Vazirani	Secret: 1111
`bv-6`	6-bit Bernstein-Vazirani	Secret: 011111
`bv-7`	7-bit Bernstein-Vazirani	Secret: 0111111
`qaoa-4A`	max-cut for 4 node graph	Output cut: 0101
`qaoa-4B`	max-cut for 4 node graph	Output cut: 0111
`qaoa-6`	max-cut for 6 node graph	Output cut: 101011
`qaoa-8`	max-cut for 8 node graph	Output cut: 10101101

5.4.2 Reliability Metrics

Our goal is to improve the reliability of NISQ machines. While PST has been commonly used at the metric to denote the system level reliability, we discuss two additional metrics that can provide further insight into assessing the reliability of NISQ machines, depending on the application.

Probability of Successful trial (PST):

PST has been used to evaluate the reliability of NISQ applications [22]. To calculate PST, a NISQ program is run multiple times, and the output of each trial is logged. PST is evaluated by computing the ratio of a number of error-free trials to the total number of trials. PST can be evaluated for a class of problems that have known correct solution. Note that by only knowing the PST, we do not know if the execution will lead to a successful outcome or not. For example, a PST of 20% may be sufficient if all other incorrect answers appear with less than 20% but insufficient otherwise.

$$\boxed{PST = \frac{\text{Number of Trials with Correct Solution}}{\text{Total Number of Trials}}}$$

Inference Strength (IST):

When running a NISQ application, we need to consider the frequency of error-free outcomes along with the erroneous outcome, as incorrect outputs can mask the error-free outputs. Thus, suppressing erroneous trials is essential to determine the error-free answer from the erroneous ones. To quantify this, we propose *Inference Strength (IST)*, which is the ratio of the frequency of error-free output to the number of most frequently occurring erroneous output. RS can easily capture the cases where the incorrect answer can dominate the correct answer. For example, the correct answer appears with the highest frequency in the output log only if RS exceeds 1.

$$\boxed{IST = \frac{\text{Probability of Correct Solution}}{\text{Probability of Strongest Incorrect Solution}}}$$

Rank of Correct Answer:

For discrete optimization problems, such as QAOA, we get a solution from the NISQ machine, and we would compute the cost associated with the solution. However, instead of only testing the most frequently occurring solution on the NISQ machine, we could also test "top K" answers for possible solution (note that, if the quantum speedup is 2^N the such a solution will have a speedup of $2^N/K$, which is still substantial for small K). To compute the rank of the correct solution, we sort the output log in descending order using frequency of occurrence and use the rank of Correct Answer as a metric to capture the effectiveness of the NISQ machine.

$$\boxed{Rank = \text{Rank of Correct Answer based on Frequency}}$$

5.4.3 Machine Configuration and Parameters

Table 5.4: Quantum Machines

Platform	`ibmqx2`	`ibmqx4`	`ibmq-melbourne`
Number of Qubits	5	5	14

For our evaluations, we use the publicly available quantum cloud service from IBM [12]. We conduct our experiments on three quantum machines, as shown in the Table 5.4. We use multiple machines to understand the machine specific and general measurement bias. We evaluate all the benchmarks using the most optimal qubit allocation for both the baseline experiments and for proposed bias mitigation techniques. We use handcrafted allocations that are cognizant of underlying noise and variation in the error rate. When running the benchmarks, we ensure that the identical program (number of gates, and position of qubits) are performed for the baseline and proposed policy. Moreover, we run each benchmark for more than 32,000 trials and ensure that the baseline and the

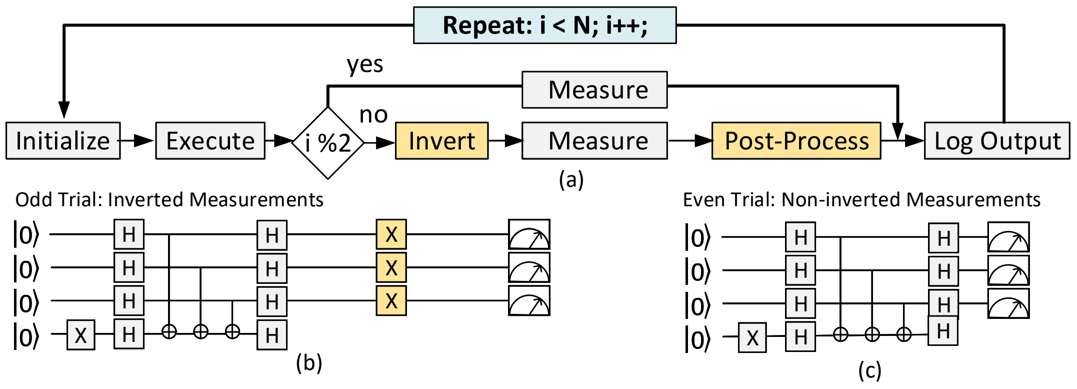


Figure 5.5: Static Invert-and-Measure (SIM): Split trials into standard-mode and inverted-mode, and merge results

proposed policy are evaluated in the same calibration window such that the evaluations for the baseline and proposed policy are executed in intertwined batches.

5.5 Exploiting Bias Using Inversion

Our characterization data shows that there is a significant bias in the measurement errors, and these errors tend to show a strong correlation with the Hamming Weight of the output being measured. We could exploit this bias to reduce the impact of measurement errors on the NISQ machines.

5.5.1 Invert-and-Measure: The Basic Concept

If we could guess the state being measured, and if it was a state that is highly vulnerable to measurement errors, then we could instead perform the measurement in an inverted mode. In the inverted mode, the qubits would first be inverted (using the X gate), and then the measurement is performed. The measurement would give an output complementary to what is expected; however, we can perform inversion on the measured output to get the desired state. We exploit this insight in our proposal, *Invert-and-Measure*. For example, if we were reading an all-ones state (highly error-prone) then inverting the state will allow us to perform our measurement in the all-zeros state (less error-prone).

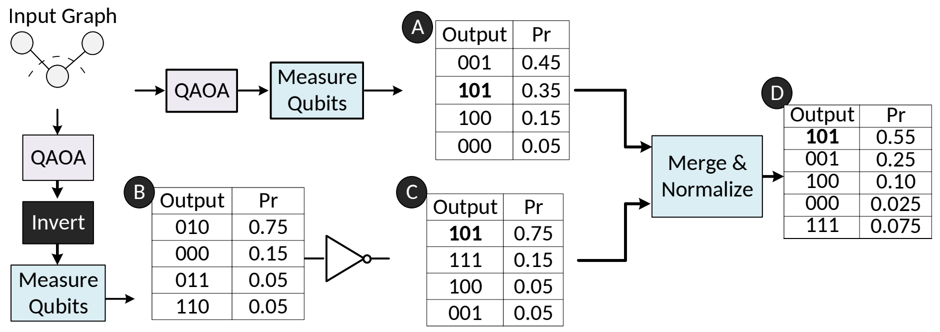


Figure 5.6: Output of QAOA on the IBM-Q14 machine using the (a) Baseline policy (b) Static Invert and Measure (SIM)

Performing all the measurement in an inverted form is not always beneficial. In fact, it can lead to more errors. For example, if we were reading a strong state and we inverted and measured, then we would increase the error rate of the system. Therefore, inverted measurement makes sense if we know the state we are reading. Applying an inverted measurement thus faces a practical challenge as we do not know the state we are measuring before performing the measurement. We propose *Static Invert-and-Measure* that avoids the reliance on knowing which state is being measured.

5.5.2 Static Invert-and-Measure

Rather than performing all of the trials in the same measurement mode, our Static Invert-and-Measure (SIM) policy divides the trials into multiple groups and performs a different measurement mode on each group. The simplest form of SIM is to have two measurement modes: standard and inverted, and use each measurement mode for half the trials, as shown in Figure 5.5(a). Note that for measurements performed in the inverted mode, we flip the measured results to get the expected output. The distribution obtained from both modes of measurement is then combined to obtain an aggregate distribution over all the trials. As SIM divides the measurements into groups, the system may perform only half of the measurements in the vulnerable state and the other half in the stronger state so that the errors can get averaged out.

We explain the operation and effectiveness of SIM with an example, as shown in Figure 5.5(b). Consider a QAOA application that produces a 3-bit output. The expected output of QAOA on an ideal quantum computer is "101". If we run the application on a NISQ machine, we can get different output in different trials. The PST is 0.35, and the incorrect answer "001" (with lower Hamming Weight) is measured more frequently as compared to the correct answer Ⓐ. Whereas, for the inverted measurements, our expected measurement would be "010" (with high probability) Ⓑ. We can flip the measurements obtained in an inverted mode to obtain the desired probability distribution for the inverted mode Ⓒ. Combining the distributions from the standard mode and inverted mode produces the correct answer with the highest probability (PST=0.55), as shown in Ⓓ. Thus, SIM limits the vulnerability of measurement errors to bias towards only one group and averaging the results from different measurement mode can improve the overall reliability of the system.

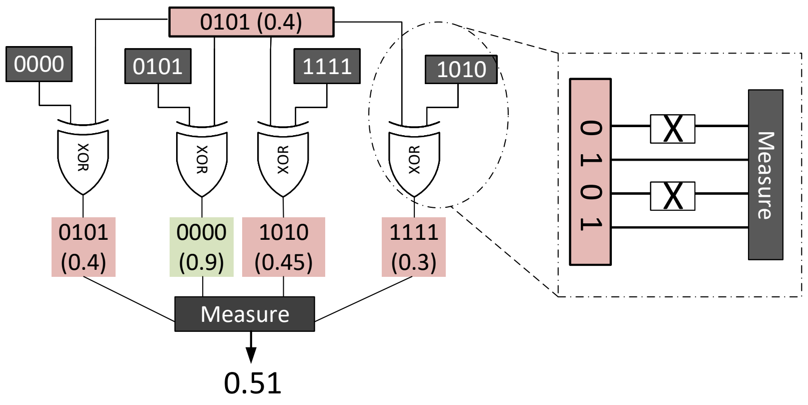


Figure 5.7: SIM with four inversion-Strings: all-zeros, all-ones, even-bits-1, and odd-bits-1. Averaging over four modes increases the likelihood of getting a stronger state.

5.5.3 Generalizing SIM to Multiple Modes

The basic insight in SIM is to transform a state being measured into another state which might be less vulnerable to measurement errors. This is achieved by inverting the measurement for some of the trials. We can generalize this concept of measuring in a different basis as having an *Inversion-String* that is applied to the set of qubits being measured before doing the measurement. In standard mode, the Inversion-String is all zeros, so no inversion is applied, and the state is read as is. In the inverted mode, the Inversion-String is all ones, so all the qubits are inverted before performing the measurement. For an N-qubit machine, there are 2^N possible Inversion-Strings. In fact, if we divide all the trials into 2^N groups, and applied a unique Inversion-String to each of the group, then we would get an average value of measurement error, regardless of the state being measured.

Applying all possible Inversion-Strings may not be a viable option, especially for a machine with dozens of qubits, given that the number of Inversion-Strings grows exponentially with the number of qubits. However, even if we choose a handful of Inversion-String, we can still average out the measurement errors on those measurement modes. For example, the SIM policy described earlier had two Inversion-Strings (all-zeros and all-ones) and is optimized for cases where the state being measured is either very low Hamming Weight or very high Hamming Weight. We could add additional measurement modes which are designed for states that may have moderate Hamming Weight. For example, we could add an Inversion-String that has an alternating string of 1s and 0s. So, our policy may have four Inversion-Strings altogether: all-ones, all-zeros, even-bits-one, and odd-bits-one. Such a design with four Inversion-Strings is shown in Figure 5.7.

We find that such an implementation of SIM with four Inversion-Strings is effective at providing measurement errors close to average, without requiring us to know the state that is being measured and without knowing the machine characteristics (which states are vulnerable and which are not vulnerable). For our experiments with SIM, we split the trials into four equal groups. We use the four Inversion-Strings: no inversion ($00000..0_n$), full inversion ($11111..1_n$), even qubit inversion ($10101..1_n$), and odd qubit inversion ($01010..0_n$). Using these inversion strings, we generate four copies of a program and run each copy for an equal number of trials. For partial and fully inverted copies we perform post-measurement flips and combine all the outputs.

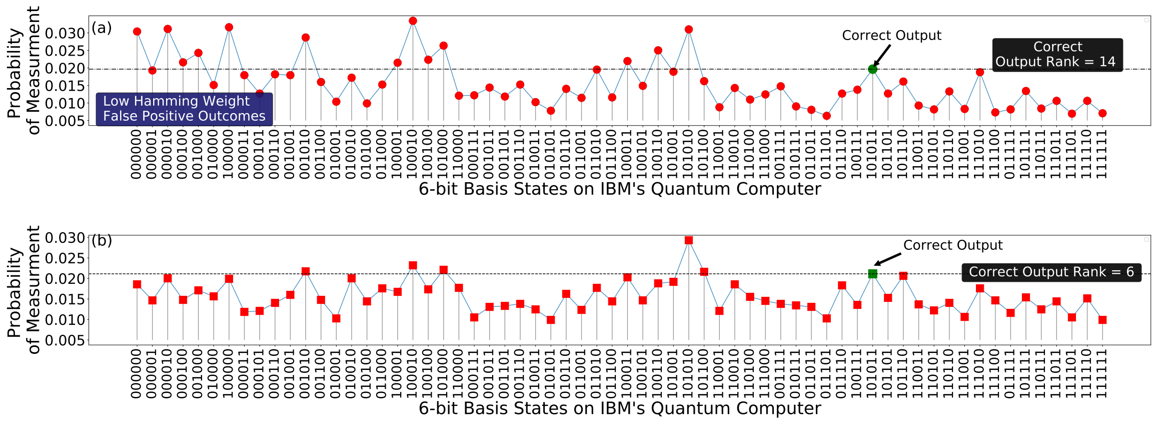


Figure 5.8: Output of QAOA on the IBM-Q14 machine using the (a) Baseline policy (b) Static Invert and Measure Policy (SIM)

5.5.4 Impact of SIM on Reliability of QAOA

To understand the effectiveness of SIM, we use QAOA to find the max-cut for the input graph-D (output string: 101011). We run the application on IBM-Q14 for 16384 trials in the baseline configuration. The output of QAOA with the baseline policy is shown in the Figure 5.8(a). The baseline PST is only 1.9% and the Relative Strength is 0.59. With the baseline policy, a significant number of incorrect answers are generated, especially incorrect answers tend to have low Hamming Weight. The baseline produces thirteen incorrect outcomes with a higher frequency of occurrence than the correct answer, such that the Rank of the correct answer is fourteen.

To improve reliability, we run QAOA with SIM. We prepare, four copies of the executable such that first copy uses non-inverted measurements, second and third copy uses alternating partial inversions, and the fourth copy uses fully inverted measurements. We run each copy for 4096 trials and combine all four distributions after post-correcting the outputs. Figure 5.8(b) shows the distribution of outputs produced by SIM. SIM improves PST by 10% and the IST by 23%. The Rank of the correct answer improves from 14 to 6. Thus, SIM can mask out a significant number of incorrect outputs by averaging out the measurements over a larger number of measurement modes.

5.5.5 Impact of SIM on PST

We conduct our experiments on three IBM machines: `ibmqx2`, `ibmqx4`, `ibmq-melbourne`. Figure 5.9 shows the PST of the system with SIM, normalized to the PST of the baseline machine. We observe that across all three machines, SIM improves PST. SIM can improve PST by as much as 2X for `ibmqx4`. SIM provides an improvement in other reliability metrics (such as Relative Strength and Rank) as well; however, we report results on those metrics in Evaluation Section.

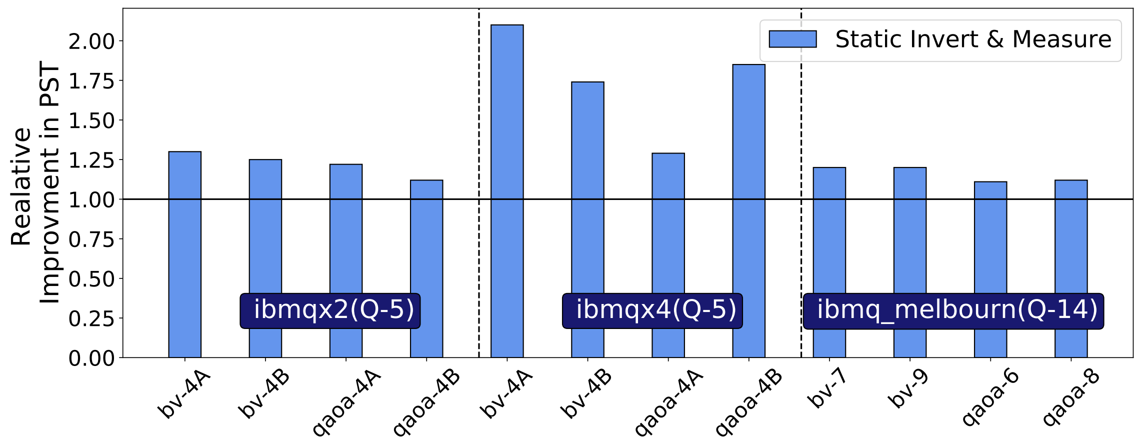


Figure 5.9: Impact of SIM on PST of the three machines.

5.6 Adaptive Invert-and-Measure

With SIM, we perform measurement in four different modes with the expectation that these inversion strings will average out the errors, and overall reliability of system will be dictated by the average error-rate for measurement rather than the worst-case error rate for the measurements. We do this because we do not know the state being measured. However, if we had a way of predicting (using profile information or a few "canary" trials) the state being measured, then we would use an Inversion-String that maps the given state to the strongest state. For example, if the strongest state is all-zeros, then the Inversion-String will be the same as the state being measured. This would ensure that the system always perceives the error rate corresponding to the state with the minimum measurement error rate.

While running a few "canary" trials to estimate the likely output looks like a promising option, however, the success of getting the right answer with a non-negligible probability again depends on the measurement error rate for the state being measured. If the state that we are interested in measuring is a highly error-prone state, then even in the canary trials we would encounter an error and get the wrong output. Therefore, we need both the profile information of the application as well as the profile information of the machine to make use of the canary trials. Unfortunately, the measurement error does not have a perfect correlation with the Hamming Weight, and for some machines (e.g.`ibmqx4`) this correlation is weak.

5.6.1 Arbitrary Measurement Bias and Impact

On `ibmqx2` and `ibmq-melbourne` measurement strength of the basis state is inversely proportional to its Hamming Weight. However, not all quantum computers have measurement strengths that scale predictably with Hamming Weight. For example, on `ibmqx4`, IBM's five-qubit quantum machine, we observe an arbitrary measurement bias such that measurement strength is not strongly correlated with the Hamming weight of the basis state. Figure 5.10 show the relative measurement strength for all 32 basis states on IBM's five-qubit machine. The data shows that the strength of the measurement is not monotonically decreasing with the Hamming Weight of the basis state. To test if the bias is repeatable, we evaluated the measurement strength of different five-qubit basis states for 35 days over 100 calibration cycles. We observe a repeatable bias in the measurement strength of basis states.

The natural state-dependent bias, variability, and machine-specific errors can collectively produce an arbitrary bias. Also, measurement errors can have a significant impact on the reliability of the NISQ machine. For example, we execute 32 instances of

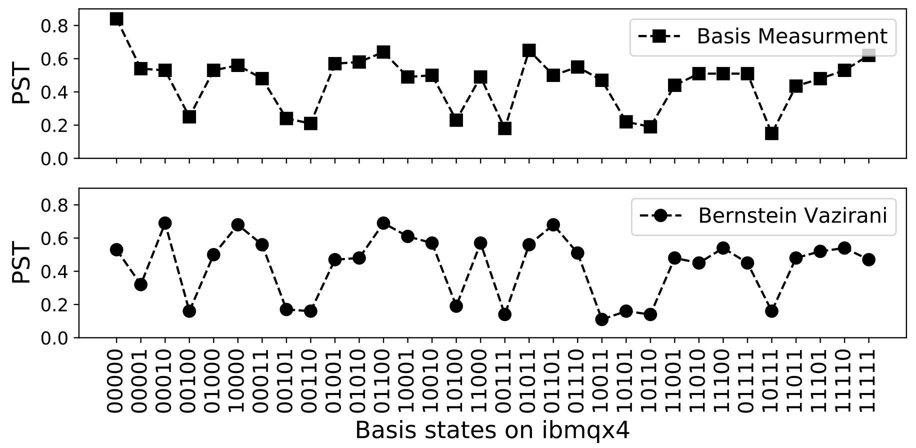


Figure 5.10: (a) Probability of Successful Measurement (PSM) for `ibmqx4` states (b) Probability of Successful Trial (PST) for BV for different desired output states.

BV-4 (each for 24 thousand trials) on `ibmqx4` such that each instance outputs 5-bit basis state (4-bit secret key and 1-bit ancillary qubit). The PST for the experiments with 32 different keys is shown in Figure 5.10(b). The x-axis is the secret input key, and the states are arranged in the increasing order of the Hamming Weight. We can observe a positive correlation between the PST and the measurement strength as weak basis states have significantly lower PST as compared to the stronger states.

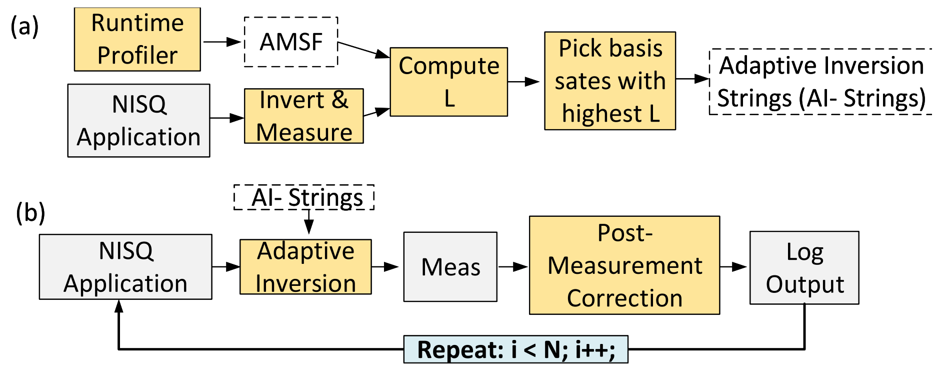


Figure 5.11: Design of Adaptive Invert-and-Measure.

5.6.2 Design of Adaptive Invert-and-Measure

To adapt to any arbitrary bias in the measurement, we propose *Adaptive Invert and Measure (AIM)*. AIM uses run-time profiling to build the measurement strength curve for a given quantum computer and uses targeted inversions. Figure 5.11 shows an overview of AIM. AIM contains three parts: (1) generating machine profile for measurement strength, (2) generating likely outputs for the application using canary trials, and (3) running the application with tailored Inversion-Strings that map the likely output states to the strongest state of the machine. We describe these steps below.

Generating Measurement Strength Function:

For a small machine (such as IBM-Q5) we can build an *Approximate Measurement Strength Function (AMSF)* by measuring the probability of successful measurement for each of the possible states. However, for a larger machine (such as IBM-Q14), evaluating all possible measurement states is not a viable option due to the exponential growth in the number of states. For IBM-Q5, we use a brute-force approach (similar to Figure 5.10(a)). Whereas, for IBM-Q14, when run QAOA-8, we use a divide-and-conquer approach, where we learn the AMSF using one window of 4-qubits at a time (sliding window).

Generating Candidates for Likely Output:

AIM performs canary trials using the four Inversion-Strings used in SIM to produce an output distribution that removes global bias. However, note that a state that is low strength may appear with low probability in this distribution simply due to measurement errors. Therefore, we scale the output distribution with an inverse value of measurement strength (for example, if the measurement strength of state X is 0.1 and state Y is 0.2, but both occur with same frequency, then we scale the likelihood of X by a factor of two compared to Y).

We denote L_i as the likelihood that the measured basis state i is correct. L_i is defined by Equation 1.

$$L_i = \frac{\text{Probability of occurrence of state i in output}}{\text{Measurement strength of the state i}} \tag{5.1}$$

We sort and select top "k" strings with the highest L value. These "k" strings or the strings within one or two hamming distance are the most likely to be the correct output.

Generating Inversion-Strings for Execution:

When the likely outputs are available, we use the Inversion-String that can map them to the strongest state. For simplicity, if the strongest state is an all-zero state, then the Inversion-String is the same as the predicted output. We run the execution for a given number of trials with this tailored Inversion-String. We do this for all of the "k" predicted outputs (we use K=4 in our study). For our evaluations, if we have N number of trials in the baseline, we use 25% of the trials as canary trials to generate the possible outputs for the application. For the remaining 75% of the trials, we use the tailored Inversion-String to perform the experiments. The total number of trials of the application remains the same for the baseline and AIM.

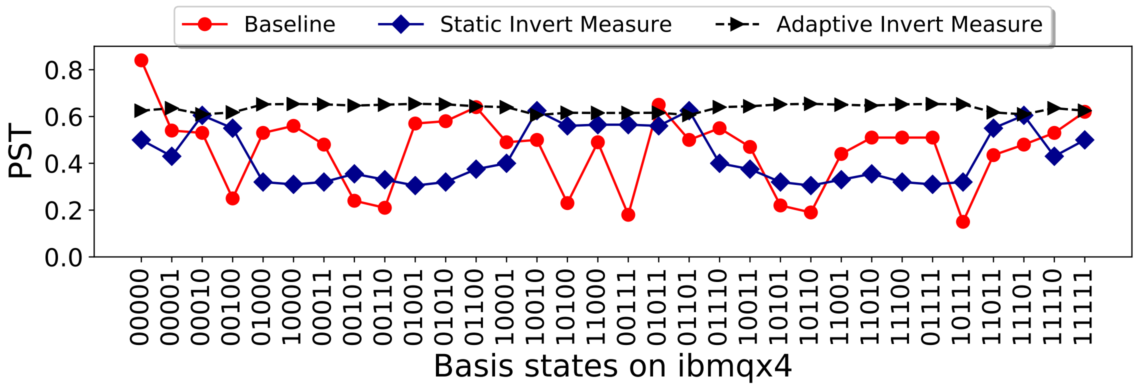


Figure 5.12: Bernstein-Vazirani for all possible keys using: Baseline, SIM, and AIM. Note that, with AIM the PST remains high for all possible states (and is close to the maximum, except for the all-zero state).

5.6.3 Estimating Measurement Strength Function

Measurement Strength Function (MSF) encapsulates the measurement strength of each basis state on a quantum computer. It is crucial to learn MSF, for detecting the measurement bias and and correct it using *Adaptive Invert and Measure (AIM)*. To evaluate the MSF, we can prepare each basis state and measure the state multiple times to estimate it's measurement strength function. However, the evaluation of absolute measurement strength is expensive. Because the number of basis states scales exponentially with the number of qubits. To mitigate this challenge, we propose two techniques:

Measuring equal Superposition of N qubits: To avoid the cost of preparing every basis state, we can prepare a quantum state with equal superposition of N-qubit state and measure this state repeatedly. By measuring the equal superposition of basis states, we can estimate the relative measurement strength of the basis states. By using technique we achieve highly accurate valley curve with about 1% mean squared error rate. However, even with superposition, we don't really solve the exponential scaling problem. For example, for 30 qubit machine, there are billion possible states just to mitigate the sampling error, we will require number of trials that also scale exponentially with Number of qubits.

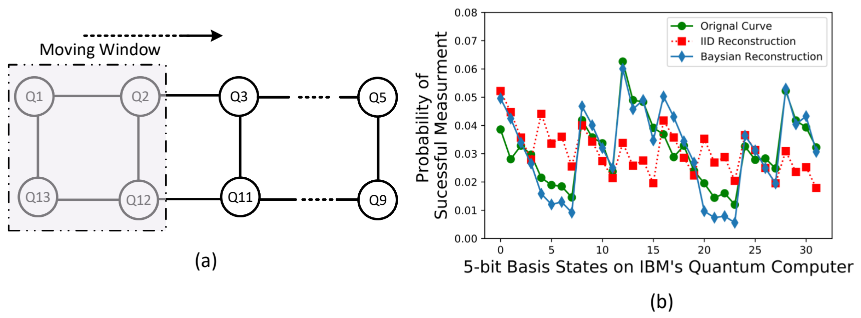


Figure 5.13: (a) Moving window characterization to estimate MSF (b)Comparison between MSF estimates with IID and Bayesian reconstruction

Moving Window Characterization: To enable evaluation of MSF with linearly scaling number of trials, we propose *approximate windowed characterization technique (AWCT)*. AWCT uses divide an conquer approach by partitioning the N qubit in smaller groups with m qubits and perform characterization of "m" qubits at a time using uniform superposition. When characterizing N qubit machine, AWCT characterizes fraction of "m" qubits at a time and moves to next set of m qubits until we characterized all the N qubits. With AWCT, the number of trials required to estimate the relative strengths of the basis states scale with $O(2^m)$ rather than $O(2^N)$. AWCT can be used in two modes: overlapping window. Figure 5.13 measurement errors on 14 qubit machine, using three techniques – baseline, overlapping window and,We can combine all the local characterizations to generate one global characterization using an analytically model described in .

5.6.4 Impact of AIM on Reliability of BV-5 Benchmark

Both SIM and AIM try to transform the state being measured into another state using Inversion-Strings. SIM does so without any knowledge of the application and the system characteristics using statically selected strings. AIM performs system and application profile to generate specialized Inversion-Strings to get the strongest state for the measurement. Figure fig:bvall shows the PST of BV for the baseline, SIM, and AIM on the `ibmqx4` machine. We experiment with all possible basis states. We note that the PST with the baseline and SIM are quite variable and the fidelity depends on the states, with some states having quite low fidelity. With AIM, the PST remains rather stable across all the possible states. Compared to the baseline and SIM, AIM continues to have a consistently high PST, with the exception of state all-zeros (the all-zero state is the strongest, so the baseline has the highest PST). Thus, AIM not only improves the PST but also makes the system have less dependence on the values used by the applications.

5.7 Evaluations

We defined three reliability metrics as the figure-of-merit for our evaluations: Probability of Successful Trial (PST), Relative Strength, and the Rank of the correct answer. We provide the effectiveness of SIM and AIM on these three metrics, on the three machines that we used in our study.

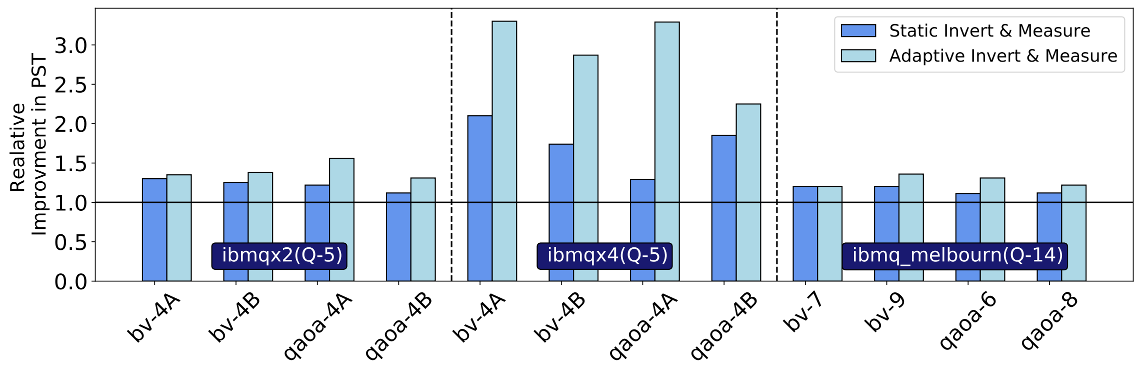


Figure 5.14: Probability of Successful Trial (PST) for SIM and AIM, normalized to the baseline. SIM improves PST by up-to 2X, whereas AIM improves PST by up-to 3X.

5.7.1 Impact on Relative Strength

Table 5.5 shows the Relative Strength (ratio of the frequency of the correct output to the frequency of the strongest incorrect output) of the baseline, SIM, and AIM. For `ibmqx2`, SIM improves the relative strength by 1.2x and AIM improves it by 1.56x. For `ibmqx4`, SIM improves the relative strength by 3.4x and AIM improves it by 7.2x. For `ibmq-melbourne`, SIM improves the relative strength by 1.9x and AIM improves it by 2.8x. We note that for `ibmqx4`, SIM improves the relative strength from 0.46 to 2.85 (6.2x) and AIM improves the relative strength to 10.38 (22.5x improvement). The relative strength of more than 1 means that the correct answer appears with the highest frequency. If the relative strength is less than 1, then the state will not have the top-most Rank. We analyze the Rank of correct answer next.

5.7.2 Impact on Rank of Correct Answer

Table 5.6 shows the Rank of the correct answer for the baseline, SIM, and AIM for the three machines that we evaluate. SIM and AIM substantial improvement in the rank of the correct answer, especially for the larger workloads such as QAOA-8 and BV-9. For small workloads such as BV-4, an incorrect answer is second or third with a baseline policy and with AIM and SIM it jumps to the most frequent answer. For `ibmqx4`, both

Table 5.5: Relative Strength for Baseline, SIM, and AIM

Benchmark	Platform	Baseline	SIM	AIM
BV-4A	ibmqx2	1.22	1.12	1.3
BV-4B	ibmqx2	0.9	1.25	1.8
QAOA-4A	ibmqx2	0.73	0.86	1.2
QAOA-4B	ibmqx2	0.72	0.96	1.12
BV-4A	ibmqx4	0.46	2.85	10.38
BV-4B	ibmqx4	4.8	6.4	5.7
QAOA-4A	ibmqx4	0.82	1.94	2.03
QAOA-4B	ibmqx4	0.72	2.67	1.98
BV-6	ibmq-melbourne	0.70	0.93	1.02
BV-7	ibmq-melbourne	0.62	0.84	1.09
QAOA-6	ibmq-melbourne	0.23	0.72	0.86
QAOA-8	ibmq-melbourne	0.18	0.36	0.78

Table 5.6: Rank of Correct Answer

Benchmark	Platform	Baseline	SIM	AIM
BV-4A	ibmqx2	1	1	1
BV-4B	ibmqx2	2	1	1
QAOA-4A	ibmqx2	3	2	1
QAOA-4B	ibmqx2	4	2	2
BV-4A	ibmqx4	3	1	1
BV-4B	ibmqx4	1	1	1
QAOA-4A	ibmqx4	3	1	1
QAOA-4B	ibmqx4	4	1	1
BV-7	ibmq-melbourne	8	4	1
BV-9	ibmq-melbourne	7	3	1
QAOA-6	ibmq-melbourne	14	6	2
QAOA-8	ibmq-melbourne	38	9	4

SIM and AIM obtain the highest rank. For `ibmq-melbourne`, the rank of the 8-bit QAOA improves from 38 to 9 with SIM, and 4 with AIM.

5.7.3 Impact of SIM and AIM on PST

Figure 5.14 shows the Probability of Successful Trial (PST) of SIM and AIM, normalized to the baseline. For `ibmqx2`, SIM improves the PST by 22% (up-to 30%) and AIM improves it by 40% (up-to 56%). For `ibmqx4`, SIM improves the PST by 74% (up-to 85%) and AIM improves it by 290% (up-to 329%). For `ibmq-melbourne`, SIM improves the PST by 16% (up-to 20%) and AIM improves it by 27% (up-to 36%). Both SIM and AIM are simple techniques that improve the reliability of the NISQ machines by mitigating measurement errors.

5.8 Summary

In this work, we focus on mitigating measurement errors, which tend to have the highest error-rate on current machines. We observe that there is state-dependent bias in measurement errors, with some states experiencing significantly higher error rates compared to the other states. Furthermore, the disparity between the measurement strength of basis states can significantly affect the reliability of NISQ applications, especially while measuring states with a high Hamming Weight.

We propose to exploit the bias in measurement errors to improve the overall system reliability. For example, while measuring a state which is highly susceptible to measurement errors, we invert the state of the qubits and perform measurement in the inverted mode. To avoid the reliance on a-priori knowing the state being measured, we propose *Static Invert-and-Measure (SIM)*, which splits the trials into multiple groups and applied a different inversion string to each group. SIM obtains measurement errors close to the average and improves the application reliability by up-to 2X.

If we could predict the state that is being measured, and the error rate profile of the machine, then we can proactively map the predicted state to the strongest state using a specifically designed inversion string. We use this insight to propose *Adaptive Invert-and-Measure (AIM)*. AIM estimates the measurement strength function of the machine for each state. AIM also conducts a few "canary" trials to learn the likely outcomes for the given application and uses the inversion string that maps the predicted output to the strongest state before performing the measurement. Our evaluations, using three IBM machines (ibmqx2, ibmqx2, ibmqx14), shows that AIM improves the reliability of the system by up to 3X.

CHAPTER 6
RELATED WORK SURVEY

In this chapter, we will survey the related work in the broad area of quantum computer architecture and compilers. Although the primary focus of this dissertation is on designing compiler policies for near-term quantum computers, we would like to provide context to help understand how the field of quantum computing and quantum computer architecture has moved from early abstract ideas to concrete system design. To that end, we will divide this survey in two parts. First part will focus on organization of quantum Computing systems and in the later part we will focus on Quantum Software Systems for Near-term Quantum Computers.

6.1 Hardware Architecture of Quantum Computers

The quantum hardware architecture focuses on: (1) Qubit Device Architecture (2) Control Computer Microarchitecture (3) Quantum System Organization. In this chapter we will focus on Microarchitecture and Systems Organization, topic relevant to this thesis. To understand how qubit device architecture evolved please refer to recent survey on quantum hardware and qubit technologies [22, 27, 28, 29] .

In the three decades, the experimental quantum computers have greatly improved the quality and number of qubit devices, which significantly affected the Microarchitecture and System Organization research in traditional systems communities. Initially the field of Quantum computing was discussed as a purely theoretical and gradually it has started to focus on engineering and systems challenges. Early work focused on the understanding capabilities of the quantum computing model [30, 31, 32, 33, 34]. However, with the proposal of Peter Shor's polynomial integer factorization algorithm [35], generated significant interest in understanding the system level challenges in building quantum computers to access the threat of quantum computers to encryption systems. At the similar time, experimentalists demonstrated first physical implementations of quantum bits using a variety of devices – trapped ions, for example, were the devices originally developed for building precise atomic clocks, Monroe with his fellow researchers at NIST demonstrated universal two-qubit CNOT gate using trapped ions qubits [36] based on schema presented by Cirac et al. [37], Whereas bouchiat et al. demonstrated early superconducting qubits [38]. Subsequently, numerous experimental and theoretical ideas proposed to use solid-state devices, neutral atoms, ions, and molecules as qubits [39, 36].

In 2000, David DiVincenzo, proposed a concrete criteria to evaluate the scalability of qubit technology. At the same time, many classical computer architects focused on developing system organization, microarchitecture, the hardware-software interface for quantum computers. Early works in quantum system architecture provided a blueprint for quantum systems by defining ISA, and microarchitectural primitives [40, 41, 42, 43]. The primary focus for these papers was to create system abstractions for large scale systems and understand bottlenecks in building and scaling the quantum computers. In the subsequent sections we will review related work in the area of qu

6.1.1 System Organization for Quantum Computers

Although quantum computers have significantly different physical constraints compared conventional co-processor. The general idea and research methodology developed by traditional computer architects can be usesd expose and exploit trade-offs in designing quantum systems and optimize required resources for computing.

Quantum System Organization focuses on design of control computer architecture and HW/SW interface. Initial papers in quantum systems used abstract models of control processor, and the majority of the articles focused on tradeoffs in organizing quantum memory, compute, and interconnects [44, 45, 46, 47, 48]. Work in [40] and [49] quantified the overhead of error correction for the ion-trap computers and proposed a framework to understand hardware constraints on the system reliability. These were one of the first papers that took a system design perspective towards understanding the overheads of enabling fault-tolerance in quantum computers. Followup research highlighted the impact of quantum error correction on the design of microarchitecture and proposed solutions to optimize for area and performance overheads. For instance, Thakkar et al. [44] focused on the organization of compute qubits and memory qubits. The authors observed the temporal locality and reuse patterns in the typical large scale quantum algorithms and proposed a trade-off between reliability and latency to optimize the area and execution time. Researchers have also proposed distributed SIMD architectures and associated scheduling policies to support quantum operations on a distributed ion-trap quantum substrate [50]. Besides ion-trap technology, architects have also analyzed the architectural trade-offs in other quantum technologies such as eSHe qubits and silicon qubits [51, 52]. For large scale systems, Jones et al. provided an abstraction of quantum computing stack that describes the functional components in scalable fault-tolerant quantum computers and provides an insight into resources required to build a large-scale fault-tolerant quantum computer [42]. Whereas, authors in [53, 54], estimated the computational resources required to perform error decoding. Moreover, they highlighted the architectural challenge of performing low-latency error decoding and emphasized the limited scalability of minimum weight perfect matching decoder on Intel CPU due to large working set size. Several recent works highlight the engineering challenges in building solid-state quantum computers [55, 56]. Tannu et al. focused on building organizational and microarchitectural challenges in building large scale fault-tolerant quantum computers, and proposed an effective use of thermal hierarchy and a microarchitecture that uses specialized hardware units to minimize software bloat [57].

6.1.2 Micro-architecture for Near-term Quantum Computers

Recent experimental breakthroughs have encouraged physicists and system architects to focus on small-scale quantum system designs. To that end, several ideas on architecting near-term control computers for superconducting qubit systems have been proposed. For example, Ofek et al. [58] used an FPGA based control that can provide real-time feedback to tune superconducting qubits. Recently. Fu et al. [59] proposed a specific microarchitecture to control more than ten superconducting qubit devices. In this work authors, make use of microcode to trigger instruction specific wave-forms. Followup work by Fu et al. [60] developed a microarchitecture to support control flow and conditional execution to scale the current experiments. In case of ion-trap quantum computers, experimentalists have been using semi-customized FPGA based control frameworks such ARTIQ.

Cryogenic control hardware is essential to scale the superconducting quantum computer beyond 100 qubits. Several researchers are developing insights into using conventional off-the-shelf hardware at cryogenic temperatures (4K to 77K). For example, Conway et al. [61] and Homulle et al. [62] characterized the functionality of CMOS based FPGAs at 4K. Followup work by Homulle et al. [63] also characterized the performance, failure modes, and power dissipation of Altera and Xilinx FPGAs built with 28nm process technology at 4K temperature. On the other hand, several groups are focusing on Josephson junction technology that can be used to construct a multiplexing scheme to control qubits [64].

6.2 Software Architecture for Quantum Computer

Early quantum quantum software work focused quantum programming languages, circuit synthesis, resource estimation and quantum simulation. The primary objective was to understand the resources required to support large scale quantum algorithms such as

Shor's Algorithm, Ground State Estimation. The early work in focused on unique challenges (no cloning, destructive measurement) of quantum computing paradigm. Work by Salinger

6.2.1 Quantum Circuit Synthesis and Compilation

The development of compilation tool-chain and resource estimation tools made quantum architecture research accessible to systems community. For example, ScaffCC – high-level programming language and llvm based compiler tool-chain enabled compilation of large-scale quantum applications [65]. Whereas, Suchara et al. developed a resource estimation tools to evaluate the number of qubits and execution-time required to run a full-scale quantum application such as Shor's algorithm [66]. Research in quantum compilers can be categorized into two broad categories front-end and back-end compilers. Front-end of quantum compilers design interfaces to build quantum programs that can be either simulated on quantum simulators or run on existing and near-term quantum substrates. For example, IBM's QISKit frame-work uses python based front-end to develop quantum programs and execute on IBM machine [67]. Moreover, there are programming languages, and APIs can be used for developing and simulating quantum programs [67, 68, 69, 70].

The back-end of quantum compiler performs circuit synthesis and qubit allocation, and optimize quantum programs to improve performance. Quantum circuit synthesis focus on a decomposition of general unitary operation, where complex fault-tolerant instruction is translated into the sequence of simpler instructions that can be performed within a fault-tolerance protocol. Several works focus on generating efficient instruction sequences [71, 72, 73, 74, 75]. All quantum circuits are required to be reversible. Back-end of a compiler can synthesize common circuits such as bit-wise logical operations, adders, multipliers that satisfy constrain of reversibility [76, 77].

6.2.2 Qubit Mapping

Quantum compiler maps the qubit variables used in the program on the physical qubits of the quantum hardware. This task is known as qubit allocation, transpilation, or quantum circuit layout synthesis in the literature. The primary objective of program mapping is to enable all the logical two qubit operations in the program by satisfy qubit connectivity constraints. Typically compiler achieves this goal by inserting extra SWAP operations, reversing CNOT gate direction using additional Hadamard gates, or performing non-neighbouring CNOTs. In this process, without changing the semantics of input program. This problem was described by Maslov et al. [77] for early NMR and Ion Trap quantum computers. Furthermore, Moslov et al. provide an excellent theoretical analysis and heuristic solution for general qubit mapping problem and show that the qubit mapping is an NP-Complete problem. Recently, Zulehner et al. focused on the qubit mapping problem for IBM NISQ computers. Zulehner et al. posed the mapping problem as a search problem and developed A* search based solution [9].Subsequent work provide heuristic solutions for mapping problems [10, 78, 79, 80, 81]. Whereas, Sirachi et al. use dynamic programming to minimize the gate depth and number of gates [8]. Furthermore, researchers have also focused on formal approach to solve this problem using general temporal planning frameworks using constraint programming [82]. Furthermore Wille et al. use SMT solver to reduce additional SWAP cost [**WilleSMT**].

6.2.3 Variability-Aware Qubit Mapping

Our work on variability-aware and noise-aware ideas are on of the first ideas that exposes the high variability in errors and study its impact on the application fidelity. The philosophy behind the variation-aware and noise adaptive qubit allocation is that the distribution of errors across qubits is unequal, with some qubits being more susceptible to errors than others, so application reliability can be improved by performing computation on the strongest set of qubits and links [15, 6]. Subsequent work by Murli et al. show similar approach, which uses device characterization data to mitigate the hardware errors using noise-aware qubit mapping. They use Z3 SMT solver to search optimal solution for small instances of the problems running on IBM quantum computers [16]. In the subsequent work, Murli et. al demonstrate the ideas can be used for Ion-Trap platforms using different technology specific tradeoffs [21]. Whereas, Nishio et al. [18] use beam search to find set of SWAP gates to maximize the application fidelity.

6.2.4 Low-Level Pulse Compilation

Superconducting quantum computers utilize microwave control pulses to perform quantum gates the state of quantum bit devices. Typically a quantum program is compiled to produce a sequence hardware supported gates in the form of symbolic executable, which is converted into a sequence of physical microwave pulses, using pulse templates to substituting a gate operation with a microwave pulse. Although this design is practical and efficient, it masks certain optimization opportunities that can be leveraged by breaking the abstractions. Several recent compiler strategies advocate synthesizing code block using custom pulses to leverage opportunities in fusing gates to reduce total execution time [83, 84, 85, 86]. Furthermore, IBM's open pulse environment makes such optimization possible on existing superconducting hardware platforms [87].

6.2.5 Software for Noise Mitigation

With the growing number of qubits, quantum software can play an important role in improving the reliability of near-term quantum applications [88, 89, 90]. To that end, theorists have proposed application specific techniques [91, 92, 93, 94, 95, 96, 97] for error mitigation. Another promising area to mitigate errors is by the use of low-cost detection codes [98]. Recent work on IBM and Rigeti hardware demonstrates the effectiveness of Dynamical Decoupling in suppressing coherence errors for two-qubit experiments [99]. Furthermore, Strikis et al. [100] proposed an error mitigation strategy that inserts an extra gate before and after each operation to reduce both active and idle errors. They propose a learning scheme to identify the type of extra gates for effective error mitigation. Similarly, Zlokapa et al. [101] propose to train a deep neural network to learn the noise characteristics of a 5-qubit machine and use this network to identify the best Dynamical Decoupling sequence for the quantum circuit.

6.3 Commercial Software Development Tools

Most quantum computing vendors have released software frameworks that enable end-to-end development and deployment of applications. Many of the too-chains use host language that works as a wrapper around low-level machine instruction languages. IBM Qiskit, uses python as a host language, whereas a low level openqasm is used as the embedded domain specific language to support quantum operations [67]. On the other hand, Xanadu's PennyLane focuses on facilitating aspects of development of (quantum) machine learning algorithms as a python package [102], whereas Microsoft's Quantum Development Kit focuses on resource estimation for fault-tolerant quantum computers [68], using a domain specific language (called Q#).

CHAPTER 7
CONCLUSION

We are witnessing an exciting time in quantum computing. Quantum hardware platforms are showing steady increase in the number of qubits and their computational capabilities. Furthermore, there is a growing interest in using quantum hardware to solve practical problems. Unfortunately, with the existing scale of quantum hardware, we can not enable fault-tolerance. As near-term quantum computers are expected to be noisy, mitigating qubit errors will be a primary challenge in enabling quantum speedup for practical problems. To that end, we focus on developing software techniques to suppress hardware errors on near-term quantum computers. We propose compiler policies that use device-level noise characterization to improve application fidelity.

In Chapter 3, we show that worst-case operational error rates can be up to ten times higher than the best device error rates on IBM quantum computers. The worst-case qubit device significantly reduces the application fidelity. To improve the application fidelity, our compiler exploits the variability in error rates when mapping program programs on quantum hardware such that it assigns more operations on reliable qubits and avoid unreliable qubits and coupling links. We build a variability-aware compiler policy that maps the program variables and inserts data movement operations to maximize the application fidelity. Furthermore, we develop a simple yet powerful experimental methodology for evaluating and optimizing the application reliability. Our evaluations on IBM quantum computers show up to three times improvement in the reliability of quantum benchmarks over baseline policy that does not account variability.

We also show how the variability in error rate can impact resource partitioning problems and create counter intuitive scenarios when partitioning the quantum hardware. The variability in errors result from fabrication defects and drift in operating conditions and cannot be eliminated by better device engineering alone. By using hardware noise characterization data, we can build software to mitigate the impact of hardware errors.

Programs executed on near-term NISQ machines can produce erroneous answers due to device errors. To infer the correct answer, a quantum program is executed for a large number of trials, and the output of the program is logged for each trial. At the end of all trials, we infer the correct answer by selecting output with the highest frequency. Our ASPLOS-2019 paper (Chapter 3) and many subsequent works have developed compilation strategies to maximize the probability of a correct answer. These strategies exploit the variability in qubit errors to map programs on the most reliable set of qubits. All the prior compilers search a single best program mapping and advocate using it for all the trials.

In chapter 4, we challenge the policy of “one mapping for all trials" as we observe selecting one mapping for all trials results in correlated errors. The correlated errors produce select few incorrect answers more frequently, resulting in incorrect answers appearing with the highest frequency and masking the correct answer. We propose a new figure of merit for NISQ systems - Inference Strength that quantitatively captures the ability to infer correct answers on NISQ machines. In our paper, we propose compiler strategies that bolster Inference Strength in addition to Probability of Success. We propose *Ensemble of Diverse Mappings (EDM)* to tolerate correlated errors. Rather than using a single mapping for all the trials, EDM splits the execution trials into groups and uses a diverse mapping to each group. Moreover, we extend EDM to *Weighted Ensemble of Diverse Mappings (WEDM)* that places different weights to the output produced by each mapping, to maximize diversity in errors. A key enabler for an ensemble of diversified programs is the model of computing, where a program is repeatedly executed for a large number of trials. We advocate the execution of functionally identical but structurally different programs for different trials. Furthermore, we can generalize the idea of program diversification by introducing diversification at different levels – from algorithms to low-level control pulses, introducing diversity can make programs robust against errors.

In Chapter 5, we focus on mitigating measurement errors on quantum computers. Measurement errors are the most dominant errors on both most commercially available quantum computers. On publicly available IBMQ computers, the worst-case measurement error rate can be up to 30%. Moreover, potential near-term quantum algorithms are significantly vulnerable to measurement error. On IBM quantum computers, measurement errors show a significant bias depending on the state being measured, with some states significantly more error-prone than other states. For example, on the IBMQ-14 machine, measuring all one state is about 2.5x times more error-prone than measuring all-zero state. The bias is caused by the natural tendency of qubits to relax to low energy or zero states.

We propose techniques to exploit state-dependent bias to mitigate measurement errors. We propose *Static Invert-and-Measure (SIM)* that splits the trials into standard-mode and inverted-mode measurement and merge the results to mitigate bias in measurement. We propose *Adaptive Invert-and-Measure (AIM)* that tailors the inversion profile to suit the machine characteristics and maps the likely solutions to be read in the strong state. Reducing measurement errors is challenging as measurement exposes otherwise well-isolated qubit to the noisy measurement circuitry. Data from recent quantum hardware suggest that measurement errors can be even worse for large scale superconducting quantum computers. We demonstrate a directional bias in measurement errors and use intelligent flip-and-measure techniques to exploit this bias and mitigate measurement errors.

In the near-term, software techniques can significantly improve quantum computers' reliability and utility by using the noise charismatics of quantum hardware.

Appendices

APPENDIX A

QUBIT STATES AND QUANTUM GATES

In this thesis, we focus on gate based quantum computers. Gate based model performs computations using quantum register (collection of qubits), and evolves the collective state of the quantum register using quantum gates.

Single Qubit State: Quantum computation leverages superposition and entanglement to enable quantum speedup. A state of a qubit ($|\psi\rangle$) is a vector in a Hilbert space that is represented as a linear superposition of two basis states $|0\rangle$ and $|1\rangle$ as shown in the equation A.1. Qubit Measurement is used to measure/read the state of the qubit. Note that $|0\rangle$ and $|1\rangle$ are termed as basis vectors, and alternatively $|\psi\rangle$ can be represented as the column vector (A.2). When qubit is measured, it produces binary output (1 or 0) with probability depending on the state of the qubit. For example,when measured, qubit with state $|\psi\rangle$ produces "$|0\rangle$" with probability of α^2 and "$|1\rangle$" with probability of β^2.

Qubit state

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle\,, \quad where\ |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} and\ |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{A.1}$$

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}, \quad where \quad \alpha, \beta \in \mathbb{C}, \quad |\alpha|^2 + |\beta|^2 = 1 \tag{A.2}$$

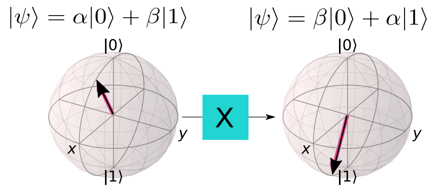
Single Qubit Gates: Quantum gates are unitary operators. When quantum gate is operated upon a qubit, it’s state vector rotates on the Bloch sphere. Equation A.3 and A.4 show typical single qubit gates used in quantum programs.Whereas, equations A.5 - A.10, illustrate how different gates can be used to manipulate the qubit state.

Single Qubit Gates

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix},\ Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} \tag{A.3}$$

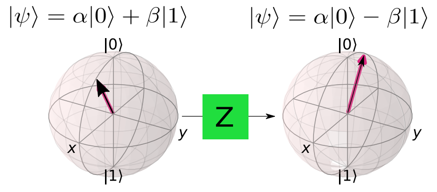
$$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix},\ S = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 0 \\ 1 & i \end{pmatrix},\ R_\theta = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix} \tag{A.4}$$

Single Qubit Gate Examples


$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$
$|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$
$|0\rangle$
$|0\rangle$
X
x
y
x
y
$|1\rangle$
$|1\rangle$

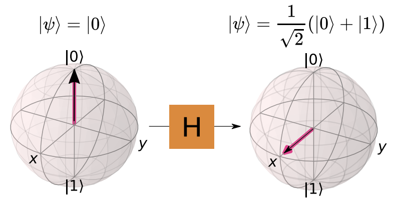
(A.5)

$$X|\psi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} * \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \beta \\ \alpha \end{pmatrix} = \beta|0\rangle + \alpha|1\rangle \tag{A.6}$$


$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$
$|\psi\rangle = \alpha|0\rangle - \beta|1\rangle$
$|0\rangle$
$|0\rangle$
Z
x
y
x
y
$|1\rangle$
$|1\rangle$

(A.7)

$$Z|\psi\rangle = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix} * \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} \alpha \\ -\beta \end{pmatrix} = \alpha|0\rangle - \beta|1\rangle \tag{A.8}$$


$|\psi\rangle = |0\rangle$
$|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$
$|0\rangle$
$|0\rangle$
H
x
y
x
y
$|1\rangle$
$|1\rangle$

(A.9)

$$H|\psi\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \tag{A.10}$$

Two Qubit State: A state of the pair of qubits can be represented using four basis vectors as shown in equation A.11, which uses a tensor product to represent the two qubit state as a vector in Hilbert space .When the two qubits are measured, the output $|00\rangle$, represents the scenario when both qubits read "0", is produced with probability $|a_0|^2$. Similarly the output $|01\rangle$, $|10\rangle$, $|11\rangle$ are produced with probability $|a_1|^2$,$|a_2|^2$,$|a_3|^2$.

Two Qubit State

$$if\ |\phi\rangle = \alpha|0\rangle + \beta|1\rangle, and |\psi\rangle = \gamma|0\rangle + \delta|1\rangle \tag{A.11}$$

$$|\phi\rangle \otimes |\psi\rangle = (\alpha|0\rangle + \beta|1\rangle) \otimes (\gamma|0\rangle + \delta|1\rangle) \tag{A.12}$$

$$|\phi\rangle \otimes |\psi\rangle = (\alpha\gamma|00\rangle + \beta\gamma|01\rangle + \alpha\delta|10\rangle + \delta\beta|11\rangle \tag{A.13}$$

$$\alpha\gamma \rightarrow a_0, \alpha\delta \rightarrow a_1, \beta\gamma \rightarrow a_2, \beta\delta \rightarrow a_3$$

$$|\phi\rangle \otimes |\psi\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle$$

$$|\phi\rangle \otimes |\psi\rangle = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} \tag{A.14}$$

To represent, a collective state of N qubits will require 2^N complex numbers.

Two Qubit Gates

$$cnot(|\phi\rangle \otimes |\psi\rangle) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 \\ a_1 \\ a_3 \\ a_2 \end{pmatrix} \tag{A.15}$$

(A.16)

$$swap(|\phi\rangle \otimes |\psi\rangle) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} a_0 \\ a_2 \\ a_1 \\ a_3 \end{pmatrix}$$

This two qubit state shown in equation A.13, is separable as the collective state can be written as a product of states $|\psi\rangle$ and $|\phi\rangle$. If we perform a cnot operation between two qubits with states $|\psi\rangle$ and $|\phi\rangle$, then the collective output state of two qubits can be represented as shown in the equation (A.15) where a target input ($|\phi\rangle$) is flipped when control input ($|\psi\rangle$) is one.

APPENDIX B

QUANTUM COMPUTER ORGANIZATION

B.1 Organization of Quantum Computer

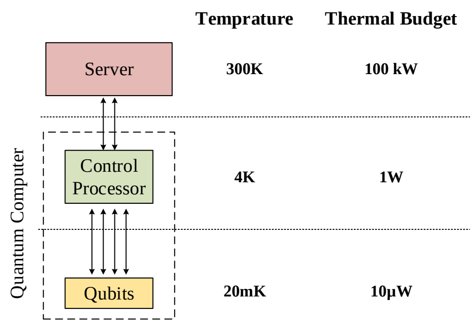


Figure B.1: (a) Quantum co-processor consists of qubits and control computer. Qubits store the information.(b) Quantum computing stack

Quantum computers are the co-processors that accelerate a class of fundamentally hard problems. On a conceptual level, a quantum co-processor is connected to a conventional computer or host machine, just as a graphical processing unit or any other co-processor. Quantum computer is constructed by connecting quantum bits (qubits) to a control computer as shown in the Figure 2.4. In a quantum computer, the state of qubits is used to represent information, whereas a control computer executes operations to manipulate the qubit states. State of a qubit is described as a mathematical abstraction just as conventional/classical bit. For example, if the digital bit is represented as two points on a sphere, say north pole or south pole, a state of the qubit is any point on the sphere. The analog nature (superposition of states) of the qubits is leveraged to build efficient algorithms. On a physical-level, qubits can be realized using photons, trapped-ions, or superconducting solid-state circuits. Typically, the control computer manipulates the state of qubits via sending technology-specific signals to qubit device. For example, ion-trap qubits use laser pulses, whereas superconducting qubits are manipulated with the microwave pulses.

The quantum computer is kept inside a cryogenic refrigerator with multiple thermal domains. We envision that a 77K domain can holds cryogenic DRAM that keeps the instruction and data of the quantum application. Whereas 4K domain holds a control processor. A 20 mK domain contains the quantum substrate. In future systems, DRAM can be used to contain the working instruction set of the quantum application as the instruction footprint for quantum algorithms is typically large (10s GB) [71, 103]. Conventional memories such as DRAM are empirically found to continue operating at 77K [104].

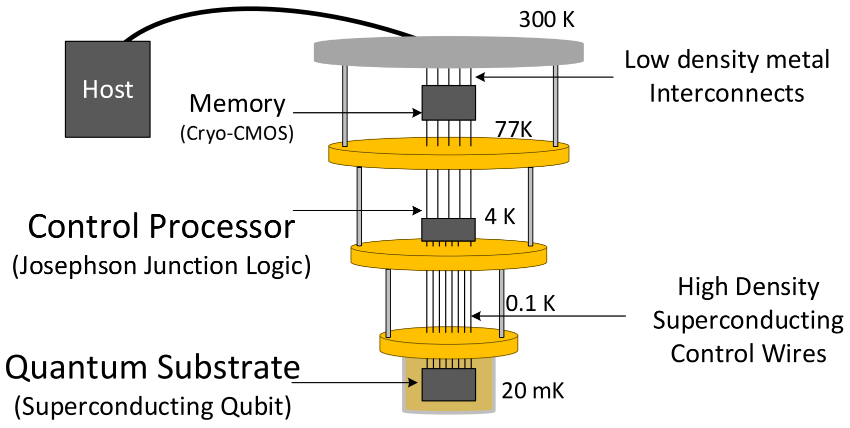

300 K
Host
Low density metal Interconnects
Memory
(Cryo-CMOS)
77K
Control Processor
(Josephson Junction Logic)
4 K
0.1 K
High Density Superconducting Control Wires
Quantum Substrate
(Superconducting Qubit)
20 mK

Figure B.2: Organization of a scalable quantum computer.

B.2 How to execute a Quantum Program?

Quantum computers execute a program by transforming a quantum program into qubit-technology specific control signals. For example, on an IBM's quantum computer, a programmer writes a quantum program using quantum assembly language (QASM), that describes the sequence of instructions performed on each qubit. The program is compiled on a host machine to generate: mapping between program qubits and physical qubit devices, and the instruction schedule to satisfy the data dependence and machine specific instruction ordering constraints. In the next step, the compiled program is loaded on the control computer. On an IBM machine, control computer executes the instruction by sending an instruction specific microwave pulse to every qubit via a coaxial cable. At the end of every quantum program, qubit states are measured, and the output of measurement is sent to host.

APPENDIX C
SIMILARITY METRICS FOR EVALUATING APPLICATION FIDELITY

C.1 A Primer on KL-Divergence

NISQ machines can produce a probability distribution over all the possible outputs. For our study, we are interested in measuring the similarity (or dissimilarity) of two probability distributions. The Kullback-Leibler divergence (or KL-divergence) is a measure of how one probability distribution is different from another probability distribution. We use the KL-divergence to analyze the diversity in output distributions generated by different mappings. We also used symmetric KL-divergence to estimate the weights for merging the outputs of different mappings in the WEDM design. In this Appendix, we will discuss a few illustrative examples. For example, if we have two discrete probability distributions P and Q defined over a state of N values, the KL divergence between P and Q, denoted as $D_{KL}(P||Q)$, is shown by Equation C.1.

$$D_{KL}(P||Q) = \sum_{i=1}^{N} P_i \log \frac{P_i}{Q_i} \tag{C.1}$$

For example, consider the two distributions P and Q over four values (0-3), as shown in Table C.1.

Table C.1: Example Probability Distributions

Distribution	0	1	2	3
P(x)	0.2	0.3	0.4	0.1
Q(x)	0.25	0.25	0.25	0.25

Then, $D_{KL}(P||Q)$ and $D_{KL}(Q||P)$ can be calculated as follows:

$$\begin{aligned} D_{KL}(P||Q) &= 0.2 \cdot ln(\frac{0.2}{0.25}) + 0.3 \cdot ln(\frac{0.3}{0.25}) \\ &+0.4 \cdot ln(\frac{0.4}{0.25}) + 0.1 \cdot ln(\frac{0.1}{0.25}) = 0.046 \end{aligned} \tag{C.2}$$

$$\begin{aligned} D_{KL}(Q||P) &= 0.25 \cdot ln(\frac{0.25}{0.2}) + 0.25 \cdot ln(\frac{0.25}{0.3}) \\ &+0.25 \cdot ln(\frac{0.25}{0.4}) + 0.25 \cdot ln(\frac{0.25}{0.1}) = 0.052 \end{aligned} \tag{C.3}$$

Thus, KL divergence may not be symmetric and can not qualify as a distance metric. However, it can be symmetrised to enable symmetric KL divergence (SD_{KL}) such that $SD_{KL}(P,Q) = SD_{KL}(Q,P)$.

$$SD_{KL}(P,Q) = D_{KL}(Q||P) + D_{KL}(P||Q) \tag{C.4}$$

REFERENCES

[1] J. Hines, “Stepping up to summit,” *Computing in science & engineering*, vol. 20, no. 2, pp. 78–82, 2018.

[2] S. K. Moore, *IBM Edges Closer to Quantum Supremacy with 50-Qubit Processor*, `https://spectrum.ieee.org/tech-talk/computing/hardware/ibm-edges-closer-to-quantum-supremacy-with-50qubit-processor`, [Online; accessed 3-April-2018], 2017.

[3] O. of the Director of National Intelligence, *IARPA Quantum Computer Science Program (2010) Broad Agency Announcement IARPA-BAA-10-02.Available from:* `https://www.fbo.gov/index?s=opportunity&mode=form&tab=core&id=637e87ac1274d030ce2ab69339ccf93c`, [Online; accessed 3-April-2017], 2010.

[4] J. Preskill, “Quantum computing in the nisq era and beyond,” *arXiv preprint arXiv:1801.00862*, 2018.

[5] A. Morello and D. Reilly, “What would you do with 1000 qubits?” *Quantum Science and Technology*, vol. 3, no. 3, p. 030 201, 2018.

[6] S. S. Tannu and M. K. Qureshi, “Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, ACM, 2019, pp. 987–999.

[7] B. Villalonga, D. Lyakh, S. Boixo, H. Neven, T. S. Humble, R. Biswas, E. G. Rieffel, A. Ho, and S. Mandrà, “Establishing the quantum supremacy frontier with a 281 pflop/s simulation,” *arXiv preprint arXiv:1905.00444*, 2019.

[8] M. Siraichi, V. F. Dos Santos, S. Collange, and F. M. Q. Pereira, “Qubit allocation,” in *CGO 2018-IEEE/ACM International Symposium on Code Generation and Optimization*, 2018, pp. 1–12.

[9] A. Zulehner, A. Paler, and R. Wille, “Efficient mapping of quantum circuits to the ibm qx architectures,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2018*, IEEE, 2018, pp. 1135–1138.

[10] A. Shafaei, M. Saeedi, and M. Pedram, “Optimization of quantum circuits for interaction distance in linear nearest neighbor architectures,” in *Proceedings of the 50th Annual Design Automation Conference*, ACM, 2013, p. 41.

[11] E. Knill, D Leibfried, R Reichle, J Britton, R. Blakestad, J. Jost, C Langer, R Ozeri, S. Seidelin, and D. J. Wineland, “Randomized benchmarking of quantum gates,” *Physical Review A*, vol. 77, no. 1, p. 012 307, 2008.

[12] I. B. M. Corporation, *Universal Quantum Computer Development at IBM:* `http://research.ibm.com/ibm-q/research/`, [Online; accessed 3-April-2017], 2017.

[13] C. H. Bennett, E. Bernstein, G. Brassard, and U. Vazirani, “Strengths and weaknesses of quantum computing,” *SIAM journal on Computing*, vol. 26, no. 5, pp. 1510–1523, 1997.

[14] V. Batagelj and M. Zaversnik, “An o (m) algorithm for cores decomposition of networks,” *arXiv preprint cs/0310049*, 2003.

[15] S. S. Tannu and M. K. Qureshi, “A case for variability-aware policies for nisq-era quantum computers,” *arXiv preprint arXiv:1805.10224*, 2018.

[16] P. Murali, J. M. Baker, A. J. Abhari, F. T. Chong, and M. Martonosi, “Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers,” *arXiv preprint arXiv:1901.11054*, 2019.

[17] W. Finigan, M. Cubeddu, T. Lively, J. Flick, and P. Narang, “Qubit allocation for noisy intermediate-scale quantum computers,” *arXiv preprint arXiv:1810.08291*, 2018.

[18] S. Nishio, Y. Pan, T. Satoh, H. Amano, and R. Van Meter, “Extracting success from ibm’s 20-qubit machines using error-aware compilation,” *arXiv preprint arXiv:1903.10963*, 2019.

[19] M. Sun and M. R. Geller, *Efficient characterization of correlated spam errors*, 2019. eprint: `arXiv:1810.10523`.

[20] A. Parent, M. Roetteler, K. M. Svore, and K. M. Svore, “Revs: A tool for space-optimized reversible synthesis,” in *Proceedings of the 9th International Conference on Reversible Computation (RC 2017)*, 2017, pp. 90–101.

[21] P. Murali, N. M. Linke, M. Martonosi, A. J. Abhari, N. H. Nguyen, and C. H. Alderete, “Full-stack, real-system quantum computer studies: Architectural comparisons and design insights,” in *Proceedings of the 46th International Symposium on Computer Architecture*, ACM, 2019, pp. 527–540.

[22] N. M. Linke, D. Maslov, M. Roetteler, S. Debnath, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe, “Experimental comparison of two quantum computing

architectures," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3305–3310, 2017.

[23] T. G. Dietterich, "Ensemble methods in machine learning," in *International workshop on multiple classifier systems*, Springer, 2000, pp. 1–15.

[24] E. T. Jaynes, "Foundations of probability theory and statistical mechanics," in *Delaware seminar in the foundations of physics*, Springer, 1967, pp. 77–101.

[25] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

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

[27] P. Krantz, M. Kjaergaard, F. Yan, T. P. Orlando, S. Gustavsson, and W. D. Oliver, "A quantum engineer's guide to superconducting qubits," *Applied Physics Reviews*, vol. 6, no. 2, p. 021 318, 2019.

[28] M. Saffman, "Quantum computing with neutral atoms," *National Science Review*, vol. 6, no. 1, pp. 24–25, 2019.

[29] K. R. Brown, J. Kim, and C. Monroe, "Co-designing a scalable quantum computer with trapped atomic ions," *npj Quantum Information*, vol. 2, no. 1, pp. 1–10, 2016.

[30] R. P. Feynman, "Simulating physics with computers," *Int. J. Theor. Phys*, vol. 21, no. 6/7, 1982.

[31] S. Lloyd, "Universal quantum simulators," *Science*, pp. 1073–1078, 1996.

[32] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, no. 1907, pp. 553–558, 1992.

[33] E. Bernstein and U. Vazirani, "Quantum complexity theory," *SIAM Journal on computing*, vol. 26, no. 5, pp. 1411–1473, 1997.

[34] A. Berthiaume and G. Brassard, "The quantum challenge to structural complexity theory.," in *Computational Complexity Conference*, Citeseer, 1992, pp. 132–137.

[35] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM review*, vol. 41, no. 2, pp. 303–332, 1999.

[36] C. Monroe, D. Meekhof, B. King, W. M. Itano, and D. J. Wineland, “Demonstration of a fundamental quantum logic gate,” *Physical review letters*, vol. 75, no. 25, p. 4714, 1995.

[37] J. I. Cirac and P. Zoller, “Quantum computations with cold trapped ions,” *Physical review letters*, vol. 74, no. 20, p. 4091, 1995.

[38] V. Bouchiat, D Vion, P. Joyez, D Esteve, and M. Devoret, “Quantum coherence with a single cooper pair,” *Physica Scripta*, vol. 1998, no. T76, p. 165, 1998.

[39] L. M. Vandersypen and I. L. Chuang, “Nmr techniques for quantum control and computation,” *Reviews of modern physics*, vol. 76, no. 4, p. 1037, 2005.

[40] S. Balensiefer, L. Kregor-Stickles, and M. Oskin, “An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures,” in *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, ser. ISCA ’05, Washington, DC, USA: IEEE Computer Society, 2005, pp. 186–196, ISBN: 0-7695-2270-X.

[41] R. Van Meter and C. Horsman, “A blueprint for building a quantum computer,” *Commun. ACM*, vol. 56, no. 10, pp. 84–93, Oct. 2013.

[42] N. C. Jones, R. Van Meter, A. G. Fowler, P. L. McMahon, J. Kim, T. D. Ladd, and Y. Yamamoto, “Layered architecture for quantum computing,” *Physical Review X*, vol. 2, no. 3, p. 031 007, 2012.

[43] K. M. Svore, A. V. Aho, A. W. Cross, I. Chuang, and I. L. Markov, “A layered software architecture for quantum computing design tools,” *Computer*, vol. 39, no. 1, pp. 74–83, 2006.

[44] D. D. Thaker, T. S. Metodi, A. W. Cross, I. L. Chuang, and F. T. Chong, “Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing,” in *ACM SIGARCH Computer Architecture News*, IEEE Computer Society, vol. 34, 2006, pp. 378–390.

[45] M. Oskin, F. T. Chong, I. L. Chuang, and J. Kubiatowicz, “Building quantum wires: The long and the short of it,” in *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, IEEE, 2003, pp. 374–385.

[46] V. Giovannetti, S. Lloyd, and L. Maccone, “Quantum random access memory,” *Physical review letters*, vol. 100, no. 16, p. 160 501, 2008.

[47] R. V. Meter and M. Oskin, “Architectural implications of quantum computing technologies,” *J. Emerg. Technol. Comput. Syst.*, vol. 2, no. 1, pp. 31–63, Jan. 2006.

[48] N. Isailovic, M. Whitney, Y. Patel, and J. Kubiatowicz, "Running a quantum circuit at the speed of data," in *ACM SIGARCH Computer Architecture News*, IEEE Computer Society, vol. 36, 2008, pp. 177–188.

[49] M. Oskin, F. T. Chong, and I. L. Chuang, "A practical architecture for reliable quantum computers," *Computer*, vol. 35, no. 1, pp. 79–87, 2002.

[50] J. Heckey, S. Patil, A. JavadiAbhari, A. Holmes, D. Kudrow, K. R. Brown, D. Franklin, F. T. Chong, and M. Martonosi, "Compiler management of communication and parallelism for quantum computation," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 43, 2015, pp. 445–456.

[51] E. Chi, S. A. Lyon, and M. Martonosi, "Tailoring quantum architectures to implementation style: A quantum computer for mobile and persistent qubits," *SIGARCH Comput. Archit. News*, vol. 35, no. 2, pp. 198–209, Jun. 2007.

[52] D. Copsey, M. Oskin, F. Impens, T. Metodiev, A. Cross, F. T. Chong, I. L. Chuang, and J. Kubiatowicz, "Toward a scalable, silicon-based quantum computing architecture," *IEEE Journal of selected topics in quantum electronics*, vol. 9, no. 6, pp. 1552–1569, 2003.

[53] S. J. Devitt, A. G. Fowler, T. Tilma, W. J. Munro, and K. Nemoto, "Classical processing requirements for a topological quantum computing system," *International Journal of Quantum Information*, vol. 8, no. 01n02, pp. 121–147, 2010.

[54] A. G. Fowler, A. C. Whiteside, and L. C. Hollenberg, "Towards practical classical processing for the surface code," *Physical review letters*, vol. 108, no. 18, p. 180 501, 2012.

[55] C. G. Almudever, L Lao, X. Fu, N Khammassi, I. Ashraf, D. Iorga, S Varsamopoulos, C Eichler, A Wallraff, L Geck, *et al.*, "The engineering challenges in quantum computing," in *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2017, pp. 836–845.

[56] X. Fu, L Riesebos, L. Lao, C. G. Almudever, F. Sebastiano, R Versluis, E. Charbon, and K. Bertels, "A heterogeneous quantum computer architecture," in *Proceedings of the ACM International Conference on Computing Frontiers*, ACM, 2016, pp. 323–330.

[57] S. S. Tannu, Z. A. Myers, P. J. Nair, D. M. Carmean, and M. K. Qureshi, "Taming the instruction bandwidth of quantum computers via hardware-managed error correction," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17, Cambridge, Massachusetts: ACM, 2017, pp. 679–691, ISBN: 978-1-4503-4952-9.

[58] N Ofek, A Petrenko, Y Liu, B Vlastakis, L Sun, Z Leghtas, R Heeres, K. Sliwa, M Mirrahimi, L Jiang, *et al.*, "Demonstrating real-time feedback that enhances the performance of measurement sequence with cat states in a cavity," in *APS Meeting Abstracts*, 2015.

[59] X. Fu, M. A. Rol, C. C. Bultink, J. van Someren, N. Khammassi, I. Ashraf, R. F. L. Vermeulen, J. C. de Sterke, W. J. Vlothuizen, R. N. Schouten, C. G. Almudever, L. DiCarlo, and K. Bertels, "An experimental microarchitecture for a superconducting quantum processor," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-50 '17, Cambridge, Massachusetts: ACM, 2017, pp. 813–825, ISBN: 978-1-4503-4952-9.

[60] X Fu, L Riesebos, M. Rol, J van Straten, J van Someren, N Khammassi, I Ashraf, R. Vermeulen, V Newsum, K. Loh, *et al.*, "Eqasm: An executable quantum instruction set architecture," *arXiv preprint arXiv:1808.02449*, 2018.

[61] I. Conway Lamb, J. Colless, J. Hornibrook, S. Pauka, S. Waddy, M. Frechtling, and D. Reilly, "An fpga-based instrumentation platform for use at deep cryogenic temperatures," *Review of Scientific Instruments*, vol. 87, no. 1, p. 014 701, 2016.

[62] H. Homulle, S. Visser, and E. Charbon, "A cryogenic 1 gsa/s, soft-core fpga adc for quantum computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 11, pp. 1854–1865, 2016.

[63] H. Homulle and E. Charbon, "Performance characterization of altera and xilinx 28 nm fpgas at cryogenic temperatures," in *Field Programmable Technology (ICFPT), 2017 International Conference on*, IEEE, 2017, pp. 25–31.

[64] O Naaman, J. Strong, D. Ferguson, J Egan, N Bailey, and R. Hinkey, "Josephson junction microwave modulators for qubit control," *Journal of Applied Physics*, vol. 121, no. 7, p. 073 904, 2017.

[65] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "Scaffcc: Scalable compilation and analysis of quantum programs," *Parallel Computing*, vol. 45, pp. 2–17, 2015.

[66] M. Suchara, A. Faruque, C.-Y. Lai, G. Paz, F. T. Chong, and J. Kubiatowicz, "Comparing the overhead of topological and concatenated quantum error correction," *arXiv preprint arXiv:1312.2316*, 2013.

[67] I. B. M. Corporation, *Quantum Software Development Kit for writing quantum computing experiments, programs, and applications*, `https://github.com/QISKit/qiskit-sdk-py#license`, [Online; accessed 3-April-2018], 2017.

[68] K. Svore, A. Geller, M. Troyer, J. Azariah, C. Granade, B. Heim, V. Kliuchnikov, M. Mykhailova, A. Paz, and M. Roetteler, “Q#: Enabling scalable quantum computing and development with a high-level dsl,” in *Proceedings of the Real World Domain Specific Languages Workshop 2018*, ACM, 2018, p. 7.

[69] D. S. Steiger, T. Häner, and M. Troyer, “Projectq: An open source software framework for quantum computing,” *Quantum*, vol. 2, p. 49, 2018.

[70] F. T. Chong, D. Franklin, and M. Martonosi, “Programming languages and compiler design for realistic quantum hardware,” *Nature*, vol. 549, no. 7671, p. 180, 2017.

[71] D. Kudrow, K. Bier, Z. Deng, D. Franklin, and F. T. Chong, “Dynamic machine-code generation for quantum rotations,” *GSWC 2013*, p. 23, 2013.

[72] V. Kliuchnikov, D. Maslov, and M. Mosca, “Practical approximation of single-qubit unitaries by single-qubit quantum clifford and t circuits,” *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 161–172, 2016.

[73] ——, “Fast and efficient exact synthesis of single qubit unitaries generated by clifford and t gates,” *arXiv preprint arXiv:1206.5236*, 2012.

[74] A. Bocharov, Y. Gurevich, and K. M. Svore, “Efficient decomposition of single-qubit gates into v basis circuits,” *Physical Review A*, vol. 88, no. 1, p. 012 313, 2013.

[75] P. Selinger, “Efficient clifford+ t approximation of single-qubit operators,” *arXiv preprint arXiv:1212.6253*, 2012.

[76] A. Parent, M. Roetteler, K. M. Svore, and K. M. Svore, “Revs: A tool for space-optimized reversible synthesis,” in *Proceedings of the 9th International Conference on Reversible Computation (RC 2017)*, 2017, pp. 90–101.

[77] D. Maslov, S. M. Falconer, and M. Mosca, “Quantum circuit placement: Optimizing qubit-to-qubit interactions through mapping quantum circuits into a physical experiment,” in *Proceedings of the 44th annual Design Automation Conference*, ACM, 2007, pp. 962–965.

[78] A. Cowtan, S. Dilkes, R. Duncan, A. Krajenbrink, W. Simmons, and S. Sivarajah, “On the qubit routing problem,” *arXiv preprint arXiv:1902.08091*, 2019.

[79] G. G. Guerreschi and J. Park, “Two-step approach to scheduling quantum circuits,” *Quantum Science and Technology*, 2018.

[80] K. E. Booth, M. Do, J. C. Beck, E. Rieffel, D. Venturelli, and J. Frank, "Comparing and integrating constraint programming and temporal planning for quantum circuit compilation," *arXiv preprint arXiv:1803.06775*, 2018.

[81] A. Zulehner, H. Bauer, and R. Wille, "Evaluating the flexibility of a* for mapping quantum circuits," in *International Conference on Reversible Computation*, Springer, 2019, pp. 171–190.

[82] D. Venturelli, M. Do, E. Rieffel, and J. Frank, "Compiling quantum circuits to realistic hardware architectures using temporal planners," *Quantum Science and Technology*, vol. 3, no. 2, p. 025 004, 2018.

[83] P. Gokhale, Y. Ding, T. Propson, C. Winkler, N. Leung, Y. Shi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Partial compilation of variational algorithms for noisy intermediate-scale quantum machines," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ACM, 2019, pp. 266–278.

[84] P. Gokhale, A. Javadi-Abhari, N. Earnest, Y. Shi, and F. T. Chong, "Optimized quantum compilation for near-term algorithms with openpulse," *arXiv preprint arXiv:2004.11205*, 2020.

[85] Y. Shi, P. Gokhale, P. Murali, J. M. Baker, C. Duckering, Y. Ding, N. C. Brown, C. Chamberland, A. Javadi-Abhari, A. W. Cross, *et al.*, "Resource-efficient quantum computing by breaking abstractions," *Proceedings of the IEEE*, 2020.

[86] Y. Shi, N. Leung, P. Gokhale, Z. Rossi, D. I. Schuster, H. Hoffmann, and F. T. Chong, "Optimized compilation of aggregated instructions for realistic quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1031–1044.

[87] D. C. McKay, T. Alexander, L. Bello, M. J. Biercuk, L. Bishop, J. Chen, J. M. Chow, A. D. Córcoles, D. Egger, S. Filipp, *et al.*, "Qiskit backend specifications for openqasm and openpulse experiments," *arXiv preprint arXiv:1809.03452*, 2018.

[88] M. Martonosi and M. Roetteler, "Next steps in quantum computing: Computer science's role," *arXiv preprint arXiv:1903.10541*, 2019.

[89] F. T. Chong, D. Franklin, and M. Martonosi, "Programming languages and compiler design for realistic quantum hardware," *Nature*, vol. 549, no. 7671, p. 180, 2017.

[90] E. National Academies of Sciences and Medicine, *Quantum Computing: Progress and Prospects*, E. Grumbling and M. Horowitz, Eds. Washington, DC: The National Academies Press, 2019, ISBN: 978-0-309-47969-1.

[91] K. Temme, S. Bravyi, and J. M. Gambetta, "Error mitigation for short-depth quantum circuits," *Physical review letters*, vol. 119, no. 18, p. 180 509, 2017.

[92] T. Tsunoda, A. Patterson, X. Yuan, S. Endo, J. Rahamim, P. Spring, M. Esposito, S. Jebari, K. Ratter, S. Sosnina, *et al.*, "Implementing the variational quantum eigensolver with native 2-qubit interaction and error mitigation," *Bulletin of the American Physical Society*, 2019.

[93] S. Endo, S. C. Benjamin, and Y. Li, "Practical quantum error mitigation for near-future applications," *Physical Review X*, vol. 8, no. 3, p. 031 027, 2018.

[94] A. Kandala, K. Temme, A. D. Córcoles, A. Mezzacapo, J. M. Chow, and J. M. Gambetta, "Error mitigation extends the computational reach of a noisy quantum processor," *Nature*, vol. 567, no. 7749, p. 491, 2019.

[95] V. Havlíček, A. D. Córcoles, K. Temme, A. W. Harrow, A. Kandala, J. M. Chow, and J. M. Gambetta, "Supervised learning with quantum-enhanced feature spaces," *Nature*, vol. 567, no. 7747, p. 209, 2019.

[96] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, p. 242, 2017.

[97] L. Cincio, K. Rudinger, M. Sarovar, and P. J. Coles, "Machine learning of noise-resilient quantum circuits," *arXiv preprint arXiv:2007.01210*, 2020.

[98] R. Harper and S. Flammia, "Fault tolerance in the ibm q experience," *arXiv preprint arXiv:1806.02359*, 2018.

[99] B. Pokharel, N. Anand, B. Fortman, and D. Lidar, "Demonstration of fidelity improvement using dynamical decoupling with superconducting qubits," *arXiv preprint arXiv:1807.08768*, 2018.

[100] A. Strikis, D. Qin, Y. Chen, S. C. Benjamin, and Y. Li, "Learning-based quantum error mitigation," *arXiv preprint arXiv:2005.07601*, 2020. arXiv: `2005.07601 [quant-ph]`.

[101] A. Zlokapa and A. Gheorghiu, "A deep learning model for noise prediction on near-term quantum devices," *arXiv preprint arXiv:2005.10811*, 2020.

[102] V. Bergholm, J. Izaac, M. Schuld, C. Gogolin, C. Blank, K. McKiernan, and N. Killoran, "Pennylane: Automatic differentiation of hybrid quantum-classical computations," *arXiv preprint arXiv:1811.04968*, 2018.

[103] D. Kudrow, K. Bier, Z. Deng, D. Franklin, Y. Tomita, K. R. Brown, and F. T. Chong, "Quantum rotations: A case study in static and dynamic machine-code generation for quantum computers," in *ACM SIGARCH Computer Architecture News*, ACM, vol. 41, 2013, pp. 166–176.

[104] W. H. Henkels, N. C. C. Lu, W. Hwang, T. V. Rajeevakumar, R. L. Franch, K. A. Jenkins, T. J. Bucelot, D. F. Heidel, and M. J. Immediato, "A low temperature 12 ns dram," in *VLSI Technology, Systems and Applications, 1989. Proceedings of Technical Papers. 1989 International Symposium on*, 1989, pp. 32–35.

VITA

Swamit Tannu is a Ph.D. candidate in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. Where he is advised by Dr. Moinuddin Qureshi, his work identifies challenges in building quantum computing systems and software. At Georgia Tech, he is developing compilers and runtime techniques to improve the reliability of noisy quantum computers. During 2015 to 2019, Swamit worked with Microsoft as Research Intern and contributed towards building architectural abstractions for quantum and superconducting accelerators. His work on Superconducting Accelerators won the Best Paper Award at Computing Frontiers 2019. Swamit was awarded MS (2019) in Electrical and Computer Engineering from Georgia Tech, and Bachelor of Engineering in Electronics Engineering from University of Mumbai.