

**SOFTWARE AND ARCHITECTURE TECHNIQUES FOR IMPROVING
FIDELITY OF EMERGING QUANTUM COMPUTERS**

A Dissertation
Presented to
The Academic Faculty

By

Poulami Das

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

Georgia Institute of Technology

August 2023

© Poulami Das 2023

**SOFTWARE AND ARCHITECTURE TECHNIQUES FOR IMPROVING
FIDELITY OF EMERGING QUANTUM COMPUTERS**

Thesis committee:

Dr. Moinuddin Qureshi
School of Computer Science
Georgia Institute of Technology

Dr. Vivek Sarkar
School of Computer Science
Georgia Institute of Technology

Dr. Frederic T. Chong
Department of Computer Science
The University of Chicago

Dr. Arijit Raychowdhury
School of Electrical & Computer Engineering
Georgia Institute of Technology

Dr. Kenneth R. Brown
Department of Electrical & Computer Engineering
Duke University

Date approved: July 26, 2023

Dedicated to Baba and Maa

ACKNOWLEDGMENTS

As my PhD journey comes to an end, I want to thank the amazing people I had the opportunity to come across, without whom this dissertation would not have been written.

I am indebted to my advisor Moin Qureshi for accepting me in his research group, giving me the opportunity and freedom to pursue research in an emerging computing paradigm, and nurturing my growth. His eye for problems, ability to connect with the intuitions and details of research, quick evaluation of ideas, work ethic, and emphasis on effective communication have significantly impacted my growth as a researcher. But most importantly, I am grateful to him for his holistic influence on my life. His excitement about research, passion for teaching, patience, kindness, caring personality, simplicity, and empathy have deeply impacted my thinking and overall approach toward life. Any amount of words will be insufficient to describe how his humility and tall personality have overshadowed the research aspects of my PhD and transformed me into a better person.

I am also grateful to have some of the greatest researchers I admire in my PhD committee. They have helped me at various stages of my journey. I would like to thank Fred Chong for providing me with valuable advice and feedback regarding my research whenever I met him. I interacted with Fred for the first time at ISCA-2018, when I was still debating whether or not to pursue my PhD research in quantum computing. Talking to him heavily influenced my decision to continue in this direction and over the years, speaking to him has always enabled me to think about new research ideas and improvise existing works. I am grateful to Vivek Sarkar for the discussions on my works on quantum compilation, future research directions, and his immense support during my academic job search. His availability to answer even minor questions and prepare me for the interviews genuinely helped me navigate the process with ease. His commitment to students' success is unfathomable and my admiration for him only increases each time I meet him. I can only hope that someday, I will become like him. I would like to thank Ken Brown for his appre-

ciation and feedback on my works in quantum error correction. His words motivated me to continue working in this area and deeply influenced some of the current research directions that I am pursuing. Last, but not the least, I am grateful to Arijit Raychowdhury for his feedback on my research, mentorship, and crucial discussions during my job search.

I have been also blessed to have the opportunity to interact with an amazing set of faculty members who were always available whenever I needed their help. I want to thank Sudhakar Yalamanchili as I would not have joined the PhD program at Georgia Tech in the first place without his support. He is one of the best human beings I have ever met and my limited interactions with him made me recognize one of the most valuable traits of a good advisor- one must advise keeping in mind the best interests of the student. I am grateful to Hyesoon Kim, Tom Conte, Tushar Krishna, Alex Daglis, Saibal Mukhopadhyay, and Yingyan (Celine) Lin. It is impossible to enlist and describe the numerous small and big things they have helped me with, and all I can say is that my life would have been very different without their support at Georgia Tech.

Over the years, I have realized that much of the learning during one's PhD comes from the interactions within their research group. I have been blessed in this regard to have come across such an amazing group of researchers in my lab. I want to thank Prashant for being the most fantastic mentor and a true friend. He has helped me with honest criticism and crucial advice at different stages of my PhD, from writing my first paper to now as I write this dissertation. Swamit has been an excellent research mentor since the beginning of my PhD. His enthusiasm, criticism, genuine feedback, and emphasis on writing good technical papers have only improved me as a researcher. My conversations with Gururaj and Sanjay influenced my thinking, approach to problem solving, and time management skills. I also want to thank Jian for sharing his words of wisdom during my job search, Muhammed for being so welcoming when I was new to the group, Vinson for pushing me to have a healthier lifestyle, Suhas for the multitude of discussions on different topics in quantum computing, and Anish for a wide range of technical as well as non-technical discussions.

I will definitely miss grabbing coffee with each of them. I also want to thank Narges and Ramin for their company and wish them the best in their future endeavors.

I have also been fortunate to collaborate with many researchers from the industry during my internships who have been instrumental in shaping my research and overall personality. I am grateful to Doug Carmean for having me as his intern and giving me the freedom to pursue my interests in quantum and continues to help me with many important things even today. ¹ His quick evaluation of ideas, thinking about problems that could potentially arise when we build systems at scale, and presentation style have heavily influenced my outlook towards research. I am also fortunate that he introduced me to Bobbie, who has been an extremely valuable mentor over the years. In addition to her technical feedback, I have always cherished and appreciated her words of wisdom (sometimes on sudden phone calls at random odd hours of the day) that have made me remain calm during rejections and helped me make decisions more confidently. Many a time, she understands my thoughts much better than myself. I am grateful to Nicolas Delfosse- he is the reason I ended up working on quantum error correction. His patience while mentoring me and answering my naive questions has only increased my admiration for him over time. I also cherish the interactions with my other collaborators- Bob Krick, Michael Lewis, Krysta Svore, and Chris Pattison as they helped me significantly during my internships. I also got the opportunity to meet Madan Musuvathi and Karin Strauss at Microsoft who have helped me with useful advice since then. I am thankful to Microsoft Research for providing me access to various opportunities and giving me the PhD fellowship.

I am also grateful to Cody Jones for mentoring me during my internship at Google. My conversations with Aditya Locharla, Kunal Arya, Austin Fowler, Mike Neumann, and Craig Gidney helped me learn a lot about the practical challenges in enabling quantum error correction on real systems. At Google, I also had the opportunity to meet Partha Ranganathan. I want to thank him for all the useful advice he has given me ever since. I

¹He also offered me a desk with an unforgettable view of the beautiful Mt. Rainier and the North Cascades, fulfilling my childhood dream of working from a hill station.

am also grateful to Yunong Shi and Eric Kessler for their mentorship during my internship at Amazon that helped me understand the key challenges in quantum cloud services (and also for spoiling me with a desk in the clouds in downtown Seattle and another adjacent to the Empire State Building in New York City).

I am indebted to Yale Patt for his teaching and guidance. His 460N course introduced me to some of the most fundamental topics in computer architecture. He convinced me to pursue a PhD and supported me to enroll at Georgia Tech. His teaching philosophy will influence my life forever. I thank Derek Chiou for teaching me many of the complexities of advanced computing systems, the challenges in building machines at scale, and the economic aspects that drive the computing industry. His comments over the years have heavily influenced my outlook on computing research. I am also grateful to Babak Falsafi and Onur Mutlu for their mentorship and advice. I want to thank my undergraduate advisor Anirudha Chandra for guiding me during my Bachelor's program, helping me with my graduate school applications, and suggesting my parents to send me abroad for higher studies.

I also want to acknowledge my friends at Georgia Tech who became family over time- Moumita, Guru, Anand, Srikanth, Aheli, Prashanth, and Divya. The study group discussions for different courses, post deadline hangout sessions, night sky viewing trips, dosas and biryanis, and so many other memories- I will cherish them forever. My friends outside Georgia Tech also helped me sail through various challenges in the last few years- Biying (for being there without any questions since our UT days), Sravana (for innumerable things and accompanying me to the same concert multiple times), Esha (for her understanding and continuing to be my friend even when I fail to keep up with my social commitments almost every time), Sabyasachi (for patiently fixing my parents' electronics gadgets unlimited number of times so that I could reach out to them hassle-free), Siddharth (for making conferences more fun and eventful), Biraja (for helping me capture some of my favorite photographs in the national parks), Ravi and Fran (for the lazy breaks and making California even more beautiful). I will also be forever grateful to Anshu for his patience, for being

there amidst the toughest times of my life, for challenging me technically and setting the bar too high, and for keeping me grounded while polishing my wings to fly.

Finally, I thank my loving parents, Milan Kanti Das and Tandra Das. They placed a lot of emphasis on providing me access to education, worked very hard, and made several sacrifices so that I could pursue my dreams. They have been a source of constant love, support, and encouragement that gave me the strength to move to the US to pursue my graduate education. I dedicate this thesis to both of them.

TABLE OF CONTENTS

Acknowledgments	iv
List of Tablesxviii
List of Figures	xx
Summary	xxv
Chapter 1: Introduction	1
1.1 The Limits of Classical Machines	1
1.2 Why Quantum Computing?	3
1.3 Evolution of the Quantum Computing Landscape	5
1.4 The Need for Full-Stack Solutions	7
1.5 Hardware Errors: The Biggest Challenge in Quantum Computing	8
1.6 Quantum Error Correction is Essential but Expensive	9
1.7 Enter the Noisy Intermediate Scale Quantum or NISQ Era	10
1.8 Overview of the Quantum Roadmap: From NISQ-Era to Fault-Tolerance	11
1.9 Error Mitigation and Error Correction	13
1.10 Thesis: Leverage Software and Architecture Solutions to Overcome Errors	15
Chapter 2: Background	19

2.1	Basics of Quantum Computing: A Primer on Quantum Information Theory	19
2.2	Quantum Algorithms	21
2.3	Quantum Hardware: Qubit Devices and Control Processor	23
2.4	Organization of a Quantum Computer	26
2.5	From Quantum Application to Qubit Devices	27
2.6	Errors in Quantum Hardware	30
2.7	Software Error Mitigation: Prospects, Challenges, and Related Works	32
2.7.1	Qubit Allocation and Routing Policies for NISQ Applications	33
2.7.2	Noise-Aware Compiler Policies for NISQ Applications	34
2.7.3	Circuit Decomposition Techniques	35
2.7.4	Error-Specific Mitigation Policies	35
2.7.5	Application-specific Error Mitigation Policies	38
2.7.6	Pulse-level Compilation Policies	39
2.8	Quantum Error Correction using Surface Codes	40
2.9	Error Decoding: Prospects, Challenges, and Related Works	42
2.9.1	Software Approaches to Decoding	44
2.9.2	Paradigm Shift Towards Hardware Decoders	45
Chapter 3: Efficient Techniques for Mitigating Measurement Errors on Emerging Quantum Computers		46
3.1	Motivation	46
3.1.1	Measurement Crosstalk	47
3.1.2	Impact of Spatial Variation	48
3.2	Key Insights of Proposed Solution	48

3.3	Challenges in Measurement Subsetting: Correlation versus Fidelity Trade-off	49
3.4	Overview of Proposed Design: JigSaw	49
3.5	Global-Mode: Generation of Global-PMF	50
3.6	Subset-Mode: Generation of Local-PMFs	51
3.6.1	Circuits with Partial Measurements	51
3.6.2	Optimizations to Improve Fidelity of the CPM	52
3.7	Post-processing via Bayesian Method	53
3.8	Multi-Layer JigSaw (JigSaw-M)	56
3.8.1	Global and Subset-Modes for JigSaw-M	56
3.8.2	Adapting Reconstruction for JigSaw-M	56
3.9	Evaluation Methodology	57
3.9.1	Quantum Hardware Platforms	57
3.9.2	Baseline Compiler	57
3.9.3	Benchmarks	57
3.9.4	Experimental Setup: Number of Trials	57
3.9.5	Figure-of-Merit	59
3.10	Final Evaluations	60
3.10.1	Results for Probability of Successful Trial	60
3.10.2	Results for Fidelity	60
3.10.3	Impact of Number of Circuits with Partial Measurements and Selection Method	61
3.10.4	Impact of Recompile	62
3.11	Scalability of Bayesian Reconstruction Algorithm	64

3.11.1	Insight: Bounded by Observations	64
3.11.2	Memory Complexity	65
3.11.3	Time Complexity	67
3.11.4	Results for Scalability Analysis	67
3.12	A First Order Estimation of Number of Trials	68
3.13	Evolution of the Measurement Error Mitigation Landscape and Follow-up Works	70
Chapter 4: Efficient Techniques for Mitigating Idling Errors on Emerging Quantum Computers		71
4.1	Motivation	71
4.2	Challenges in Reducing Idling Errors	73
4.3	Characterizing the Effectiveness of Dynamical Decoupling	74
4.3.1	Mitigating Idling Errors using DD	74
4.3.2	Effectiveness of DD under Crosstalk	75
4.3.3	Factors Influencing Idling Errors and DD	76
4.3.4	Impact of DD on Application Fidelity	78
4.4	Overview of Proposed Design: Adaptive Dynamical Decoupling	79
4.5	Clifford Decoy Circuits (CDC)	79
4.5.1	Design of Clifford Decoy Circuits (CDC)	80
4.5.2	Effectiveness of Clifford Decoy Circuits	81
4.6	Overcoming the Limitations of CDCs: Seeded Decoy Circuits	81
4.7	Managing Search Complexity	83
4.8	ADAPT: Design Implementation	84

4.8.1	Integration in the Compiler Tool-Flow	84
4.8.2	Finding Idle Qubits	84
4.8.3	Pulses for Implementing DD	85
4.9	Evaluation Methodology	86
4.9.1	Compiler	86
4.9.2	Quantum Hardware Platforms	86
4.9.3	Benchmarks	87
4.9.4	Figure-of-Merit	87
4.9.5	Number of Trials	88
4.9.6	Competing Policies	88
4.10	Final Evaluations	89
4.10.1	Results for IBMQ-Paris	89
4.10.2	Results for IBMQ-Toronto	90
4.10.3	Results for IBMQ-Guadalupe	91
4.10.4	Impact of DD Pulse Type	92
4.11	Evolution of the Idling Error Mitigation Landscape and Follow-up Works	95
Chapter 5: Efficient Solutions For Improving The Throughput of Emerging Quantum Computers via Multi-programming		96
5.1	Motivation	96
5.2	Challenges in Increasing Throughput of Quantum Computers	98
5.3	Overview of Proposed Solution: Multi-Programming	100
5.4	Fair and Reliable Resource Allocation Policies	100
5.4.1	Qubit Allocation for Multi-programs: Key Insights	101

5.4.2	Fairness in Qubit Allocation:	102
5.4.3	To partition or not to partition?	104
5.4.4	Algorithm Implementation	105
5.5	Delayed Instruction Scheduling Policy in Shared Resource Environments	106
5.5.1	When Should Qubits be Measured?	106
5.5.2	Scheduling Algorithm for Multi-programs	107
5.6	Dynamically Switching between Multi-programming and Isolated Execution	110
5.6.1	Adaptive Multi-Programming Design	110
5.6.2	Dynamic Reliability Monitoring Scheme	112
5.7	Evaluation Methodology	115
5.7.1	Figure-of-merit: Trial Reduction Factor	115
5.7.2	Quantifying Application Fidelity	115
5.7.3	NISQ Benchmarks	116
5.7.4	Baseline Setup	116
5.7.5	Overview of the Proposed Framework	116
5.8	Final Evaluations	117
5.8.1	Impact of Fair and Reliable Partitioning Algorithm	117
5.8.2	Impact of Delayed Instruction Scheduling Policy	119
5.8.3	Impact of Adaptive Multi-Programming	120
5.9	Discussion	121
5.10	Evolution of the Multi-Programming Landscape and Follow-Up Works	122

Chapter 6: Efficient Real-Time Decoding Solutions For Quantum Error Correction on Emerging Quantum Computers 123

6.1	Motivation	123
6.2	Challenges in Real-Time Decoding	124
6.3	Overview of Proposed Solution: LILLIPUT	125
6.3.1	Detection of Errors from the Stabilizer Measurements	126
6.3.2	Error Identification as a Matching Problem	126
6.3.3	Handling Data and Measurement Errors	127
6.3.4	Handling Boundary Cycles	129
6.4	Error Assignments in LILLIPUT	129
6.4.1	Streaming Mode of Operation	130
6.4.2	Error Assignment	130
6.4.3	Tracking the Internal State	131
6.4.4	Programming the LUT	133
6.4.5	Decoding in Presence of Non-Clifford Gates	134
6.4.6	Determination of Logical Error	134
6.5	Evaluation Methodology	135
6.5.1	Surface Code Parameters	135
6.5.2	Monte Carlo Simulation Infrastructure	135
6.5.3	Noise Model	136
6.5.4	Target Hardware Platforms	136
6.6	Final Evaluations	137
6.6.1	Results for Accuracy	137
6.6.2	Results for Hardware Complexity	139
6.6.3	Results for Latency	141

6.7	A Case for Compressed LUTs	142
6.7.1	Not All Error Events are Equally Likely	142
6.7.2	Compressing the LUTs in Software	143
6.7.3	Accessing CLUTs and Decompression in Hardware	145
6.7.4	Performance and Overheads of CLUTs	145
6.7.5	Scaling to Other Decoder Configurations	146
6.8	Evolution of the Decoding Landscape and Follow-Up Works	148
Chapter 7: Conclusion and Future Directions		149
7.1	Summary of Presented Works	151
7.2	Key Takeaways	152
7.2.1	A Case for Specialized Fine-grained Software Error Mitigation . . .	152
7.2.2	Breaking the Abstractions	153
7.2.3	Quantum Overheads of Software Error Mitigation Must Be Mini- mized	153
7.2.4	Quantum Cloud Resource Management Requires A Different Ap- proach	154
7.2.5	Expand the Scope: A Case For Full-stack Solutions	154
7.2.6	Adapting Solutions To Meet System Constraints	154
7.3	Future Research Directions	155
7.3.1	Co-Designing Software Error Mitigation Using Application and Hard- ware Constraints	155
7.3.2	Striking A Balance Between Overheads and Benefits Of Software Error Mitigation	156
7.3.3	Adapting Device-level Software Error Mitigation To Large-scale Applications	156

7.3.4	Exploring Trade-offs in Qubit Device Technologies	157
7.3.5	Automated Software Tool-chains	157
7.3.6	System-level Solutions for Quantum Cloud Services	158
7.3.7	Enabling QEC on Real-Systems: Rethinking Existing Approaches .	158
7.3.8	Modeling architecture and system-level solutions for cryogenic control and QEC	159
7.4	What Can We Do With Quantum Computers In The Next 5, 10, 20 and 30 years?	160
	References	161
	Vita	182

LIST OF TABLES

2.1	Comparison of different qubit technologies	25
3.1	Measurement Errors on Google Sycamore	47
3.2	Details of NISQ benchmarks	58
3.3	Fidelity obtained from EDM, JigSaw, and JigSaw-M relative to the baseline	61
3.4	Number of Outcomes in the Global-PMF for a Graycode-18 Benchmark on IBMQ Hardware	65
3.5	Scalability Analysis of JigSaw and JigSaw-M: Memory (in GB) and Num- ber of Operations (in million)	68
4.1	Idling Times for Programs on IBMQ-Rome	72
4.2	Correlation between Decoy and Input Circuits (higher is better)	83
4.3	Error Characteristics of IBMQ Hardware	86
4.4	Quantum Benchmark Characteristics	87
4.5	Summary of Results	92
5.1	NISQ Benchmarks	116
5.2	Probability of Successful Trial (PST) under isolated execution and in a multi-programmed environment using the FRP algorithm.	118
5.3	Probability of Successful Trial (PST) for various modes of multi-programming	120
6.1	Parameters of Rotated Surface Codes	135

6.2	LUTs for different decoder configurations	140
6.3	FPGA utilization for different configurations	140
6.4	Maximum Frequency (in MHz) and Latency (in ns)	141
6.5	Logical Overhead for Implementing CLUTs	146
6.6	Memory Requirement for LILLIPUT with CLUTs	147

LIST OF FIGURES

2.1	(a) Bloch sphere representation of a qubit. (b) X gate performs an inverse or bit-flip operation. (c) H gate creates an equal superposition state.	19
2.2	(a) A GHZ circuit and its (b) output distribution. (b) The equivalent quantum program is written in IBM's OPENQASM language.	22
2.3	Physical qubits roadmap of quantum computers (the past and promised future)	24
2.4	Organization of a quantum computer	26
2.5	Overview of NISQ compilation.	27
2.6	(a) A 4-qubit Bernstein Vazirani circuit (b) Mapping of program qubits to the physical qubits of an example NISQ hardware (c) Compiler SWAPs qubits Q_1 and Q_2 to enable the CNOT operation between qubits Q_2 and Q_3 as they are not physically connected.	28
2.7	(a) A two-qubit GHZ circuit and its (b) ideal output distribution on a noise-free quantum computer. (c) The output distribution on a real noisy quantum computer.	30
2.8	Distance 3 (a) regular and (b) rotated surface code. Z stabilizers (in blue) detect X errors and X stabilizers (in red) detect Z errors. (c) Z stabilizer circuit.	41
2.9	Overview of steps required for QEC.	42
3.1	Spatial variation in measurement error rates of qubits on IBMQ-Toronto . .	48
3.2	Overview of JigSaw. JigSaw runs the program in two modes: global-mode (all program qubits are measured) and subset-mode (only a subset of qubits are measured), and then uses the local-PMFs generated in subset-mode to update the global-PMF obtained during the execution in global-mode. . . .	50

3.3	(a) Example of a CPM (b) Compiler avoids vulnerable qubit. Mapping that avoids vulnerable qubit (c) with extra SWAP (d) without incurring extra SWAP	51
3.4	Steps in Bayesian Reconstruction for a 3-qubit program (qubits Q_2, Q_1, Q_0) using a CPM measuring Q_1, Q_0 .	55
3.5	Impact of Number of Trials on Probability of Successful Trial of Applications.	58
3.6	Probability of Successful Trial (PST) from JigSaw and JigSaw-M relative to baseline and comparison with prior work Ensemble of Diverse Mappings (EDM) [88]. Number below the label shows absolute PST for the benchmark.	60
3.7	Impact of (a) Number of CPM and (b) CPM Selection Method on the Performance of JigSaw.	62
3.8	Probability of successfully measuring a qubit of a 6-qubit BV program on IBMQ-Toronto in the (a) baseline (b) in each CPM after recompilation.	63
3.9	Comparison of Mean PST relative to the Baseline.	63
3.10	Memory required to store (a) the global, (b) N intermediate and 1 output PMF, and (c) N local-PMFs.	66
3.11	(a) Number of Global-PMF entries and (b) Epsilon (ϵ) with increasing trials (T) on IBMQ-Paris.	66
4.1	(a) 4-qubit Bernstein-Vazirani circuit (b) Impact of SWAPs on the idle time of Q_0 when BV circuits with increasing sizes are executed on IBMQ-Toronto.	72
4.2	(a) Qubit-rotation operation on a single qubit (b) Dynamical Decoupling (DD) using XYYX (XY4) sequence.	73
4.3	Circuit to evolve qubit q_0 (a) freely (b) with DD. (c) Fidelity of q_0 with free evolution and with DD. Circuit to evolve q_0 (d) freely (e) with DD in the presence of crosstalk from ongoing CNOT operations. (f) Fidelity of q_0 in the presence of crosstalk, with and without DD. Distribution of fidelity of the idle qubit (g) without and (h) with DD when circuits (d-e) are executed on every qubit-link combination on IBMQ-Guadalupe.	74
4.4	Distribution of Relative Fidelity of q_0 in the presence of DD for 700 qubit-link combinations on IBMQ Toronto.	76
4.5	Relative Fidelity of Qubit-12 in the presence of CNOTs on Link:17-18 for different calibration cycles.	77

4.6	Fidelity of QFT and BV benchmarks with all possible DD sequences on IBMQ-Toronto. The sequence 0 (000000) denotes DD on none of the qubits and sequence 63 (111111) denotes DD on all qubits.	78
4.7	Overview of ADAPT with key building blocks.	79
4.8	Correlation between the Fidelity of a 4-qubit Adder circuit on IBMQ-Guadalupe and the corresponding Clifford decoy circuit. We observe a strong correlation.	81
4.9	(a) Example quantum circuit. Decoy circuit construction using (b) Clifford gates (c) only CNOT (d) Mostly Clifford and few non-Clifford gates.	82
4.10	Overall Workflow of ADAPT.	84
4.11	a) XY-4 DD sequence, (b) decomposition of XY-4 sequence on IBMQ systems, (c) IBMQ-DD using $X(\pi)$ – $X(-\pi)$ gates, and (d) decomposition of IBMQ-DD sequence inserted in the idle window between two gates.	85
4.12	Relative fidelity on 27-qubit IBMQ-Paris using XY4 DD sequence.	89
4.13	Relative Fidelity of different policies on 27-qubit IBMQ-Toronto using (a) XY4 and (b) IBMQ-DD sequences.	90
4.14	Relative Fidelity of different policies on 16-qubit IBMQ-Guadalupe using (a) XY4 and (b) IBMQ-DD sequences.	91
4.15	A characterization circuit (a) without any DD (b) with the XY4 DD sequence (c) IBMQ DD sequence. (d) Mean fidelity from individual DD sequences on 16-qubit IBMQ Guadalupe when the idle time is increased.	94
5.1	Pending traffic for IBM Q16 on different days.	97
5.2	(a) Qubit allocation of a 4-qubit program P1 and a 5-qubit program P2. (b) Qubit allocations of P1 and P2 on a multi-programmed NISQ computer. Each node represents a qubit and the label on each edge represents the link error rate.	99
5.3	(a) An example NISQ program (b) this topology requires 1 SWAP to perform Instruction 4 (c) this topology does not require any extra SWAP to execute the program.	100

5.4	Error rates on IBM Q16. A node represents a qubit and the label on each edge is the 2-qubit gate error rate for that link. Links marked in bold have above mean 2-qubit gate error rate whereas qubits circled bold have greater than mean measurement error rate.	101
5.5	Two possible partitions (a) and (b) for a 4-qubit program and a 5-qubit program that are scheduled to be run concurrently on the NISQ machine. Partition (b) is more reliable compared to partition (a).	103
5.6	(a) Programmer expects two programs to be decoupled completely (b) Default instruction scheduling from IBM Compiler where all measurements are performed at the end (c) Our proposed delayed instruction scheduling policy where the shorter program thread starts late.	107
5.7	Design of Adaptive Multi-Programming	110
5.8	Reliability monitor and controller	112
5.9	Overview of the proposed multi-programming framework. The partitioning algorithm locates two reliable regions on the NISQ computer, with X and Y qubits each. If it can find two such regions, both workloads execute together. If it is unable to locate the requested regions, it defaults to the baseline and each benchmark is run individually.	117
5.10	Probability of Successful Trial (PST) relative to baseline for multi-programming model	118
5.11	Error rates on IBM Q16 during multi-programming experiments	119
5.12	Probability of Successful Trial (PST) of proposed delayed instruction scheduling policy relative to baseline	119
5.13	Effective reduction in the number of trials enabled by multi-programming	120
6.1	Summary of demonstrations of QEC codes on real systems.	124
6.2	Overview of LILLIPUT.	125
6.3	Steps in translating the stabilizer measurement outcomes (red denotes an error is identified) to error detection events in each QEC cycle. The bitstream for stabilizer measurements is specified from left to right (by convention) on the surface code lattice.	126

6.4	A distance 4 repetition code decoding graph, where nodes and edges denote parity and data qubits respectively. Note that this decoding graph is for the purpose of illustration only and our evaluations in this paper are based on surface codes. Examples of (a) space-like (b) space-time like and (c) time-like detection events.	128
6.5	(a) The sliding window uses detection events from cycles 1, 2, and 3 and assigns error to the oldest layer i.e., cycle 1. Note that all detection events in the window are considered for matching but only the edges touching the oldest cycle are considered during error assignments, shown in blue. (b-d) The sliding window streams forward one cycle at a time.	131
6.6	(a) LILLIPUT finds the optimal matching in the current window but adds a detection event in cycle 2 which is tracked in the internal state register. (b-c) The additional event is considered from the register. In the boundary cycle, zeros are padded. By convention, detection events of the oldest cycles occupy the least significant bits in our design. The bits for each cycle run bottom to top from the most significant position to the least.	132
6.7	Valid possible error assignments for the detection event at the center of the distance 4 surface code lattice.	133
6.8	Monte Carlo simulation framework.	136
6.9	Accuracy of distance (a) 3 and (b) 4 surface codes for the different number of syndrome measurement rounds (m). Note that the p^2 line is only for visual aid purposes.	137
6.10	Accuracy of distance 5 surface codes for different number of syndrome measurement rounds (m).	138
6.11	Impact of number of cycles on LER.	139
6.12	Probability distribution of the Hamming weight of memory addresses for (a) $p = 0.1\%$ and (b) $p=1\%$ for 5 cycles	143
6.13	Steps (in software) involved in compressing LUTs.	144
6.14	Steps (in hardware) involved in obtaining error assignment data from CLUTs.	145
6.15	(a) Comparison of logical error rate for the baseline and LILLIPUT with CLUTs (b) Logical error rate and decoder failure rate with CLUTs.	146
6.16	Probability distribution of Hamming weight of LUT accesses for (a) $d=4, m=3$ and (b) $d=5, m=2$	146

SUMMARY

Quantum computers promise to solve important computational problems in many application domains such as chemistry, material science, high-energy physics, cryptanalysis, and machine learning. Most of these applications are intractable on conventional systems. Quantum computers get their computational advantages by leveraging quantum-mechanical properties to store and manipulate information. A quantum bit or *qubit* is the fundamental unit of information on a quantum computer. Quantum algorithms manipulate the state of the qubits using quantum operations. After years of research and development, quantum computers with a few hundred qubits are available today. Unfortunately, the qubit devices are noisy, and imperfections in the quantum operations lead to incorrect outcomes during program execution and limit the fidelity of these systems.

Quantum information can be protected by using quantum error correction (QEC) codes at the expense of redundancy (50-1000x). These codes project errors into failed parity checks which are used to identify or decode errors in real-time. However, it is impractical to run applications in a fully fault-tolerant manner on emerging systems with only a few hundred to thousands of qubits. Instead, these systems run applications in the presence of errors and promise to accelerate certain domain-specific applications. Quantum hardware errors serve as a major bottleneck in running most practical quantum applications and their impact must be minimized.

Thesis statement

By using software techniques to mitigate the impact of specific types of qubit errors and architecture solutions to detect errors in real-time, we can improve the fidelity of emerging quantum computers.

CHAPTER 1

INTRODUCTION

1.1 The Limits of Classical Machines

The first transistor, built in 1947, laid the foundation of the computing industry as we see it today. Regarded as one of the pivotal developments in the world of computing, the transistor is a semiconductor device that acts as a switch and enables us to encode and process binary information (“0” or “1”). Combining multiple transistors allows us to create complex digital integrated circuits and incorporate advanced functionality in computing systems. However, the field really started maturing in the 1960s when substantial efforts were made to improve the manufacturing process of transistors at an industrial scale. Over time, the improvements in process technology continued and enabled us to fabricate twice the number of transistors almost every two years for decades, increasing the number of transistors on a chip exponentially from a few thousand in the 1970s (2300 in Intel’s first microprocessor 4004 [1]) to many billions today. With more transistors becoming available, computer architects were able to create hardware structures and organizations that enhanced performance further. Rapidly changing hardware combined with algorithmic and software developments have enabled computing systems to evolve from solving basic arithmetic operations to performing complex artificial intelligence and machine learning tasks.

The time taken to run an application depends on the algorithmic complexity, capabilities of the software stack, and computing hardware. For example, an application implemented using an algorithm with linear complexity written in C++ running on a newer generation Intel processor will provide solutions much faster than an equivalent implementation using an algorithm with quadratic complexity written in Python and executed on an older generation core. Therefore, despite rapid advancements in hardware, some problems still remain

intractable even on today's most powerful supercomputers. A classic example of such a problem is integer factorization- the computational complexity of this problem grows exponentially with the size of the problem due to the lack of efficient classical algorithms. In 2013, Martinis estimated that it would take about 30 CPU years to factor a 640-bit number [2]. Furthermore, computing the prime factors of a 2048-bit number could take up to ten years on a supercomputer farm almost the size of North America by consuming 10^6 TerraWatts of power (all of earth's energy in one day) and costing about 10^6 trillion dollars. Thus, it is practically impossible to factor a 2048-bit number in a reasonable amount of time. There exist many such classically intractable applications including molecular simulations in chemistry and many body simulations in physics.

Moreover, hardware improvements are slowing down in recent years. To understand this, we must discuss the two fundamental trends that shaped transistor scaling over the years. The first trend was observed by Gordon Moore from Intel, in 1965, in which he noted that transistor densities would double almost every two years, leading to the formulation of *Moore's Law* [3]. The second trend was observed by Dennard and others from IBM, in 1974, in which they noted that power densities of transistors would remain constant even if transistors shrank in size, leading to the formulation of *Dennard's Scaling Law* [4]. The combined effect of these two trends led to transistors shrinking in size by half, switching speeds or core frequencies increasing by 2x, and the transistor count per chip increasing by 2x every two years without increasing the power consumption, leading to increased performance per core. However, since the mid-2000s, power densities started increasing with diminishing transistor sizes, limiting the maximum clock frequencies and performance of the cores. More recently, there has been a paradigm shift towards building multi-core systems and domain-specific accelerators that are dedicated to increasing the performance of certain dedicated tasks. However, these approaches will still be insufficient to build custom hardware solutions for solving fundamentally classically intractable problems.

1.2 Why Quantum Computing?

Quantum computers promise massive speedups for these classically intractable problems by leveraging quantum mechanical properties. Quantum computers encode and process information using quantum bits (*qubits*) and leverage the properties of *superposition* and *entanglement* to implement efficient quantum algorithms that offer computational speedup. For example, the same study by Martinis in 2013 [2] estimated that a sufficiently large quantum computer with about 200 million qubits could compute the factors of the same 2048-bit number in only about twenty-four hours using a quantum algorithm [5]. More recent estimates suggest that the factors can be computed in less than eight hours on a quantum computer with only 20 million qubits [6]. Solving some of these classically intractable problems efficiently on a quantum computer enables us to design better fertilizers, analyze chemicals for efficient batteries, understand material sciences, and discover better medicines for healthcare. In other words, quantum computers promise to revolutionize the computing landscape by enabling us to solve important problems that can greatly benefit our society at large. Let us take protein folding as an example. Proteins are chains of amino acids with very high degrees of freedom as they can exist in any possible combinations of their constituent amino acids. For example, a protein molecule with 100 amino acids can exist in one of the 10^{47} potential arrangements. However, in reality, proteins fold to their native combination very quickly [7] but determining this native configuration is extremely hard on conventional systems due to the enormously large number of possibilities and lack of efficient search algorithms in this space. Consequently, the behavior of most protein molecules is not yet fully understood which further limits our understanding of processes involving them, such as those observed in Alzheimer's disease, vaccine development, and crop improvement research. On the contrary, an efficient quantum algorithm reduces the complexity of the protein folding problem from $O(2^N)$ (classical) to $O(N^4)$, where N is the number of amino acids in the protein [7, 8].

Unlike the state of a classical bit, which can be only either a “0” or “1”, the state of a qubit can be a *superposition* or linear combination of both “0” and “1”, also known as the basis states. Each basis state is associated with a complex probability amplitude. The state of a classical system with “n” bits can only represent a single n-bit binary value or combination of “0”s and “1”s at any given time, whereas the state of a quantum computer with “n” qubits can represent all possible 2^n states simultaneously. Hence, a quantum computer spans across all possible basis states of the equivalent classical machine and operates in exponentially larger state space at any given time, giving rise to massive levels of parallelism. However, only having an exponentially large state space is not enough and a quantum computer must be able to manipulate it simultaneously and efficiently. To enable this, quantum computers leverage the property of *entanglement* to create highly correlated states such that qubits become intrinsically inter-connected or *entangled*.

Quantum algorithms leverage these quantum mechanical properties to enable efficient algorithms. In a typical quantum application, the qubits are initialized to encode a given problem including the data inputs, the probability amplitudes of the different quantum states or basis states are manipulated by using quantum gate operations such that the probabilities of the desired outcomes in the state space are amplified, which can be then obtained by measuring the qubits. For example, an algorithm may start with a fully entangled state of “n” qubits where the probabilities associated with each of the 2^n states are equal. Next, by applying quantum operations, the probabilities associated with each of the 2^n states are manipulated such that only the probabilities of the desired state (corresponding to the program solution) are boosted while the probabilities associated with the undesired states (corresponding to incorrect solutions to the problem) are reduced. Measuring the qubits collapses the superposition and produces the desired classical outcomes (with each qubit corresponding to a 0 or 1) because the probability of observing a certain outcome upon measurement depends on the probability amplitudes associated with the corresponding basis state in the computational state space.

1.3 Evolution of the Quantum Computing Landscape

Richard Feynman first proposed the potential of using quantum behavior for computing to best model or simulate the behavior of complex physical systems in 1982 [9]. A decade later, in 1992, Deutsch and Jozsa proposed an algorithm that offered exponential speedup for a black-box problem implementing a constant (0 on all inputs or 1 on all inputs) or balanced function (1 for exactly half of the input domain and 0 for the other half), where the algorithm determines the nature of the function in a single evaluation [10]. In the following year, Bernstein and Vazirani proposed a quantum algorithm that offered an exponential speedup over any classical algorithm for identifying a hidden string encoded in a function [11]. Soon after, Simon proposed an algorithm, that determined the period of a black box function exponentially faster than a classical algorithm [12]. Drawing on the insights of period finding from Simon's algorithm, Shor proposed quantum algorithms for discrete logarithm and integer factoring problems in polynomial time [5]. The introduction of Shor's algorithm served as a major catalyst in the field of quantum computing because it implied that cryptographic applications which relied on the classical hardness of integer factorization could now be broken if one could build a large fully functional quantum computer, compromising the security and privacy of encrypted communications and confidential data. In subsequent years, other quantum-easy classical-hard algorithms were also introduced for various application areas such as database search [13], quantum chemistry applications [14], and quantum machine learning [15].

Similar to transistors in conventional systems, qubit devices lie at the heart of a quantum computer. The quantum devices stack continued to evolve over the years but gained significant momentum in the last five to seven years. A pivotal moment in the field occurred in 2016 when IBM released public access to their 5-qubit quantum computer [16]. By empowering almost anyone anywhere in the world with the ability to run a quantum program on a real machine consisting of five qubits via their cloud services, IBM's Quantum Experi-

ence venture led to a widespread curiosity among a broader set of researchers, enlarged the quantum workforce, and accelerated progress in the field. There are different approaches to building qubit devices— ranging from the usage of natural atoms to artificial atoms and photons. While the scale of devices studied in academic research labs typically ranges up to a handful of qubits and making each of them better in terms of exhibiting quantum properties, industrial giants, such as IBM, Intel, Google, Microsoft, Amazon, and Honeywell, are making significant progress toward building large-scale quantum computers that can power real-world applications.

In 2019, Google demonstrated computational speed-up for certain random sampling tasks using their 54-qubit Sycamore machine, entering the *supremacy* or *beyond-classical* regime. Quantum supremacy refers to the regime in which a quantum computer outperforms the best-known implementation on classical systems for a certain task, which may or may not be useful. The random circuit sampling study by Google performed the computation in 200 seconds while the same task would take 10,000 years on the world’s most powerful and fastest supercomputer. Although the claims were immediately challenged by researchers, the study set forward an important milestone in the quantum computing community. In the same year, the National Academies of Sciences, Engineering, and Medicine released their *Quantum Computing: Progress and Prospects* report which stated that there were no fundamental reasons why a large-scale, fault-tolerant, gate-based quantum computer could not be built [17]. Quantum supremacy-focused experimental studies were also released by other device providers on other machines in the subsequent years [18, 19, 20].

The field has also progressed forward considerably from free access to a 5-qubit machine from IBM to machines with few tens and hundreds of qubits. Furthermore, other cloud providers such as Amazon and Microsoft today offer paid services to access quantum computers. Quantum computers continue to rapidly scale in size, with systems up to 433 qubits already available today [21], 1,000-plus qubits to become available by the end of the year 2023, and a 100,000 qubit computer planned for by 2033 [22, 23].

1.4 The Need for Full-Stack Solutions

The key to computing is building the right abstractions. Just like conventional systems, quantum computers too require full-stack solutions ranging from quantum algorithms to quantum programming languages, compilers, control processor architectures, and qubit devices. A quantum computer consists of three key components- (1) qubits, (2) a control processor that sends instructions to the qubits to perform computation and transforms the measurement signals obtained by reading the qubits into classical outcomes, and (3) interconnects that enable communication between the control processor and the qubits.

Quantum programs are usually written in high-level languages to allow seamless software development [24, 25, 26]. High-level languages also offer strong abstractions between the target operations in an algorithm and the underlying hardware on which the programs will be executed. On the other hand, the *native* gates of a quantum computer refer to the set of low-level instructions closer to the hardware that can be directly executed on the device. Native gates are specific to each device. For example, the native gate set on IBMQ systems differs from that on Rigetti or Google machines. To run a quantum program, a compiler translates each high-level instruction of a program into an equivalent set of operations using the device-specific native gate set. For example, any quantum program must be translated by the compiler to a combination of gates from the set $[X, RZ, CX, ID, SX]$ for execution on IBMQ systems [27]. Each native gate is essentially implemented as an analog pulse. Therefore, the low-level assembly code is further translated to a sequence of control pulses that are used to control the actual qubit devices. To obtain the optimal control parameters for performing high-fidelity quantum operations, the on-chip device components are regularly calibrated. The control processor sends the accurate sequence of pulses scheduled in time with proper synchronization to perform operations on the qubits. Thus, in addition to qubits, a quantum computer needs tremendous amounts of software and classical resources or a full-stack approach in order to be fully functional.

1.5 Hardware Errors: The Biggest Challenge in Quantum Computing

Despite the availability of quantum computers with up to a few hundred qubits, the demonstration of quantum advantage in real-world applications remains an open problem. The performance of real quantum systems is severely limited by the high error rates of the noisy quantum hardware. A quantum computer cannot intrinsically eliminate errors. As any combination of “0” and “1” denotes a valid quantum state, even the slightest perturbation in the state of a qubit, the smallest inaccuracies in performing quantum operations, or stray signals that couple into the quantum substrate or interconnects during computations lead to the generation of incorrect outcomes during program execution.

Limited Qubit Lifetimes: The performance of quantum computers is plagued by the inability of the qubits to retain quantum information indefinitely. Depending on the qubit device technology, qubits retain their states for a few tens of microseconds to seconds.

Poor Operation Fidelities: Quantum operations cannot be performed with precise accuracy. Moreover, unwarranted couplings with stray signals and device crosstalk encountered while performing quantum gate and measurement operations reduce the operational fidelities even further. For example, the average error rate of a quantum operation ranges between 1-4% on some of the existing machines [28].

Qubits accumulate errors from their unwarranted interactions with various noise channels in their environment, many of which are not yet fully understood, and therefore, ways to mitigate them fully at the device level remain unknown. When the noise levels are too high or the number of computations performed is very large, hardware errors produce incorrect results that are often indistinguishable from the correct solution of a program, even if the computation is repeated for multiple trials. The error rates of the quantum hardware severely limit us from running most practical quantum applications today. However, qubit qualities continue to improve in newer generations of quantum computers, albeit slowly, providing us the impetus to invest in this promising field.

1.6 Quantum Error Correction is Essential but Expensive

Despite improving device qualities, we are still far from reaching the target error-rates required to run most practical quantum applications (below 10^{-10}). Quantum error correction (QEC) codes enable us to protect quantum information at the expense of redundancy. The simplest form of QEC code was introduced by Shor in 1996 [29] but later other QEC codes were developed. QEC codes encode a *fault-tolerant* or *logical* qubit by distributing information over many physical qubits [29, 30, 31, 32, 33]. The error rate of the logical qubit reduces exponentially with increasing redundancy of the QEC code, if the error-rate of the physical qubits is below a *threshold* [30]. By periodically executing certain quantum operations and measuring some of the redundant qubits, errors can be projected into failed parity checks. A classical *decoder* uses the parity checks to locate errors and sends the correction to the control processor so that future operations can be updated to revert the errors.

QEC enables us to achieve the error-rate required to run an application by controlling the redundancy of the code. A fault-tolerant quantum computer (FTQC) performs computations by interspersing QEC operations in between logical operations. Given the redundancy of QEC codes, each logical qubit requires (1) many physical qubits which typically range between 50-1000x, (2) redundant gate operations, and (3) real-time error decoding to maintain its state. Note that operations on the logical qubit must be translated into operations on the underlying physical qubits. Consequently, QEC codes incur huge resource overheads in terms of the number of physical qubits, redundant operations, and classical resource overheads for control and error decoding. Thus, there is growing interest in demonstrating and studying the performance of QEC codes as systems with improved device qualities and system sizes become available. For example, Google recently demonstrated the ability to suppress errors on their Sycamore machine [34, 35]. However, building a fully fault-tolerant quantum computer could take several years due to the number of qubits, complex control, and readout circuitry required in these systems.

1.7 Enter the Noisy Intermediate Scale Quantum or NISQ Era

Quantum algorithms such as Shor’s algorithm or Grover’s database search algorithm require us to run too many computations which are nearly impossible to run on emerging systems with high error-rates. Over the next few years, we will be operating in the *Noisy Intermediate Scale Quantum (NISQ)* regime where quantum hardware comprising of a few hundred to thousands of noisy qubits with high operational error-rates and short qubit lifetimes will become available [36]. The prospects of running applications that can tolerate certain amounts of errors at the algorithmic level and requires relatively fewer computations on these NISQ machines look promising. Instead of attempting to obtain the exact solution of a problem, NISQ applications typically target to achieve (1) good-enough approximate solutions that are more accurate than what is achievable using the best-known classical algorithm or (2) obtain better or similar quality solutions faster.

The two most promising NISQ algorithms are *Variational Quantum Eigensolver (VQE)* [37] and *Quantum Approximate Optimization Algorithm (QAOA)* [38]. They promise to accelerate certain tasks in optimizations, quantum chemistry, and machine learning domains. Referred to as *variational quantum algorithms* or *VQAs*, these algorithms are executed in a hybrid or quantum-classical variational loop in which (1) the problems are mapped onto parametric quantum circuits that encode the cost function being optimized, (2) an initial guess is made for the circuit parameters and the corresponding circuit is executed on the quantum computer for thousands of trials, (3) the outputs from the quantum computer are used to guess the next set of circuit parameters using a classical optimizer and an updated circuit is generated, and (4) the process is repeated for hundreds and thousands of iterations until the optimal circuit parameters are identified. The output distribution of the circuit corresponding to the optimal parameters produces the optimal solution to the problem at hand. Recent studies on a 127-qubit quantum computer from IBM shows how variational circuits produce accurate results and are useful in the NISQ-era [39].

1.8 Overview of the Quantum Roadmap: From NISQ-Era to Fault-Tolerance

While emerging quantum hardware cannot unlock the full potential of quantum computing, they still promise computational advantages for various real-world important applications such as climate modeling [40], material science [41], financial optimizations [42], etc. In fact, NISQ machines have the potential to introduce a virtuous cycle in the field of quantum computing, similar to Moore’s law in the semiconductor industry.¹ Accelerating real-world applications on NISQ machines has the potential to demonstrate commercial advantages using quantum computers, which in turn has the potential to attract more investments and the involvement of a larger workforce in the field. This will further accelerate progress. Consequently, developing practical NISQ applications is an active area of research and is heavily emphasized as a crucial need for the field in the NAE report on the progress and prospects of quantum computing [17]. With quantum computers consisting of 1000-plus and 4000 qubits projected to become available in the next few years, we are expecting some substantial progress in the context of NISQ applications.

Emerging quantum systems are also used to study QEC at a smaller scale to pave the path toward FQTCs. While building a large-scale fault-tolerant quantum computer is a grand engineering challenge that could take up to several years, it still remains the ultimate objective as it can unlock the full potential of quantum computing. The QEC roadmap is expected to evolve from the demonstration of error suppression using logical qubits to the ability to perform logical operations, and scaling up to a greater number of logical qubits as the systems mature. Recently, some preliminary steps have been taken in this direction with the demonstration of error suppression on emerging quantum computers [43, 44, 45].

¹Moore’s law is often interpreted as a law of economics rather than a law of device physics. It created a virtuous cycle in the semiconductor industry by enforcing investments in research and development to improve device fabrication technology and attracting a workforce critical to innovate and sustain growth to the next level. Improvements in the device manufacturing process enabled chip-makers to reduce the price of their products which led them to sell more products, increasing their revenue. The increased revenue enabled chip-makers to spend more capital investments in research and development and in hiring talent to improve their manufacturing capabilities for the next cycle. Thus, Moore’s law set the right pace in the industry for one to conceive new product ideas or upgrade their earlier product to keep up with their competitors. Overall, the semiconductor market too grew exponentially, adhering to the trends of Moore’s law.

Below is a timeline of some of the key events in the field of quantum computing ranging from algorithms to devices that have led to developments that we see today.

- 1982 • Feynman proposes the idea of simulating physics using quantum behavior
- 1992 • Deutsch–Jozsa algorithm shows speedup for a black-box problem
- 1993 • Bernstein-Vazirani algorithm or the "Quantum Hello World" developed
- 1994 • Simon's algorithm speeds up period finding in black-box problems
- 1994 • Shor's algorithm threatens the integrity of secure communications
- 1996 • Grover invents quantum database search algorithm
- 2016 • Anyone anywhere on earth can access a real quantum computer on IBM's cloud
- 2018 • Preskill coins the term *NISQ*
- 2019 • Google claims quantum supremacy, enters into a war of words with IBM
 - NAE says practical quantum computing is not fundamentally impossible
 - Amazon launches their quantum cloud service, AWS Braket
- 2021 • Microsoft launches their quantum cloud service, Azure Quantum
- 2022 • Other demonstrations of quantum supremacy on random circuit sampling
 - Google shows it is possible to suppress errors using quantum error correction
- 2023 • *[Expected]* IBM's promised 1000-qubit machine launch
- 2029 • *[Expected]* Google's timeline to deliver a 1,000,000-qubit machine
- 2033 • *[Expected]* IBM's timeline to deliver a 100,000-qubit machine

1.9 Error Mitigation and Error Correction

Hardware errors pose a significant challenge in running most practical quantum applications. While QEC is the only way to drastically reduce hardware error rates, running applications in a fully fault-tolerant manner is impractical on emerging quantum systems. Error mitigation is an orthogonal approach, specially tailored for the NISQ regime, that targets to reduce the impact of hardware errors via improved program scheduling and/or output post-processing schemes.² The role of software error mitigation became more prominent around 2018 with the introduction of the noise-adaptive compilation approach- two concurrent studies [47, 48] noted that qubit devices exhibit temporal and spatial variation in error-rates meaning that the error rates of the devices change over time and that qubits even on the same quantum chip have non-uniform error-rates with some being more vulnerable to errors than others. The studies further showed that exposing the error model to the software can enable the compiler to steer more computations on the less error-prone devices and generate a program schedule that is more resilient to errors. Chong mentioned in his keynote address at the Architectural Support for Programming Languages and Operating Systems (ASPLOS) Conference in the same year how such software techniques coupled with other architecture solutions could close the gap between quantum algorithms and devices, accelerating the quantum computing field by 10-20 years [49]. This led to a rapid rise in error mitigation solutions both in academia and industry in the subsequent years.

The key objective of software error mitigation techniques is to improve application fidelity by averting hardware errors as much as possible. These approaches are typically integrated at (1) the compiler level (before the execution on the quantum platform) to generate highly optimized quantum object code or pulse schedules with reduced vulnerability to errors or (2) later at the output processing stage (after the quantum hardware returns the

²It is possible to use some of the error mitigation techniques in QEC as well. For example, dynamical decoupling (DD) is a device-level method to keep idle qubits busy so that they can retain their quantum states for longer periods of time by introducing a set of low-cost single-qubit operations. DD has been shown to be effective for NISQ applications, including Quantum Volume circuits used for benchmarking quantum computers [46]. However, recent QEC studies from Google also use DD pulses while running QEC [35].

execution results) to clean up or rectify the output distributions obtained. There are various approaches for software error mitigation ranging from (1) generating program schedules with the fewest operations to (2) noise-adaptive scheduling, (3) diversifying the noise profile of the execution by mapping the same program onto different qubit devices, and (4) using additional characterization circuits to fine-tune the compiled code and/or post-process the results. For example, matrix-based measurement error mitigation [50] is a promising scheme that prepares a noise matrix by characterizing the measurement errors on the qubit devices and uses the inverse of this matrix to alter the noisy output distribution in an effort to improve the quality of the solutions. More recently, in addition to generalized software error mitigation schemes, several recent proposals in the literature suggest fine-tuning these optimizations by leveraging application-specific characteristics to further improve fidelity.

Most software error mitigation techniques improve the application fidelity at the overhead of executing additional circuits on the quantum hardware and post-processing the results using classical computing resources. For example, full-scale matrix-based measurement error mitigation described above requires the characterization of an exponentially large number of quantum circuits and a classical post-processing step with exponential complexity. Consequently, the adoption of this method has been limited to quantum programs that use only up to ten qubits. Thus, in reality, most error mitigation schemes lead to an increase in the total execution time of the quantum application because of (1) increased compilation time, (2) additional circuit runs, (3) long cloud latencies, and (4) classical processing time for combining the results. Thus, for practical adoption, software error mitigation techniques must optimize across two vectors: (1) they must improve the application fidelity while (2) remaining scalable and without increasing the overall execution time on the quantum hardware significantly, or else the quantum advantages might be diminished (or lost), especially in the context of NISQ applications. An orthogonal technique is to parallelize the executions on the quantum and classical resources as much as possible.

1.10 Thesis: Leverage Software and Architecture Solutions to Overcome Errors

The goal of this dissertation is to close the gap between quantum applications and noisy quantum devices by overcoming the impact of errors using software and architecture solutions. In particular, this dissertation makes the following contributions:

Software techniques to reduce the impact of measurement and idle errors: In this space, the dissertation presents two solutions: *JigSaw* and *ADAPT*. On near-term quantum systems, qubit measurements tend to be the most error-prone operations (with an average error-rate of 4%) and often limit the size of quantum programs that can be run reliably on these machines. As quantum programs create and manipulate correlated states, all the program qubits are measured in each trial and thus, the severity of measurement errors increases with the program size. Prior works mainly focus on reducing the errors in gate operations, whereas solutions for mitigating measurement errors are either not scalable or inefficient (because they are still limited by measurement of all program qubits). This dissertation presents *JigSaw* in Chapter 3, a software framework that reduces the impact of measurement errors by running a program in two modes. First, running the entire program and measuring all the qubits for half of the trials to produce a global (albeit noisy) histogram. Second, running additional copies of the program and measuring only a subset of qubits in each copy, for the remaining trials, to produce marginal or localized (higher fidelity) histograms over the measured qubits. *JigSaw* then employs a Bayesian post-processing step, whereby the histograms produced by the subset measurements are used to update the global histogram. *JigSaw* performs the subset measurements on qubits with the least measurement error-rates via recompilation. Evaluations on three different IBM quantum computers with 27 and 65 qubits show that *JigSaw* improves the success rate of programs on average by 3.6x and up-to 8.4x in the best-case. Unlike prior solutions such as matrix-based measurement error mitigation, the complexity of *JigSaw* is linear in the number of qubits and the number of trials performed.

In addition to software solutions for reducing active errors, such as measurement errors, this thesis also presents techniques for mitigating idling errors that occur when qubits remain idle. Prior works on mitigating idling errors have been mainly limited to the devices community that have proposed *Dynamical Decoupling (DD)* to reduce stray noise on idle qubits by continuously executing a specific sequence of single-qubit operations that effectively behave as an identity gate. Thus, the qubits do not remain idle anymore while their state remains effectively unchanged. Unfortunately, existing DD protocols have been primarily studied at the scale of characterization of individual qubits and their efficacy at the application level is not fully understood.

Our experiments show that naively enabling DD for every idle qubit does not necessarily improve fidelity. While DD reduces the idling error-rates for some qubits, it increases the overall error-rate for others due to the additional operations of the DD protocol. Furthermore, idling errors are program-specific and the set of qubits that benefit from DD changes with each program. To enable robust use of DD, this thesis presents *Adaptive Dynamical Decoupling (ADAPT)* in Chapter 4, a software framework that estimates the efficacy of DD for each qubit combination and judiciously applies DD only to the subset of qubits that provide the most benefit. ADAPT employs a *Decoy Circuit*, which is structurally similar to the original program but with a known solution, to identify the DD sequence that maximizes the fidelity. For scalability of the search, ADAPT employs a localized algorithm that has linear complexity in the number of qubits. Experiments on IBM quantum machines (with 16-27 qubits) show that ADAPT improves the application success-rate by 1.86x on average and up to 5.73x compared to no DD and by 1.2x compared to DD on all qubits.

Cloud management techniques to maximize throughput of quantum computers: While error mitigation solutions, such as JigSaw and ADAPT, present exciting opportunities to improve the application fidelity on emerging quantum computers, they also require extra quantum circuit runs and classical resources for the generation of decoy circuits (in ADAPT) and post-processing (in JigSaw). Both schemes require an additional number of

circuits that scale linearly with the number of qubits in the program. While classical resources are more readily available in the cloud, quantum resources are relatively scarce. In most cases, quantum jobs are scheduled using queues on the limited number of available quantum computers, leading to increased wait times as the number of quantum circuit runs increases. In the worst case, if the circuits stay too long in the queue during which the error rates of the hardware change (due to device drifts and re-calibrations), the performance of the error mitigation techniques may go down. To address this bottleneck, we leverage the insight that owing to high error rates on emerging quantum computers, programs with only a few qubits can be executed reliably, resulting in the underutilization of resources. This thesis proposes *Multi-programming Quantum Computers* in Chapter 5 to improve the throughput of emerging quantum computers in the cloud by concurrently executing multiple workloads on the same machine.

However, multi-programming quantum computers may adversely impact the fidelity of individual applications. To enable multi-programming in a robust manner, this dissertation proposes three solutions. First, we develop methods to partition the qubits of a device into multiple reliable regions using error information from the machine calibrations so that each program can have a fair allocation of reliable qubits. Second, we observe that when two programs are of unequal lengths, measurement operations may impact the fidelity of the co-running programs due to crosstalk. To reduce this interference, we propose a context-aware *Delayed Instruction Scheduling* policy that delays the start of the shorter program such that all the measurement operations from all the co-running applications can be performed together at the end. Third, we develop an *Adaptive Multi-Programming* design that monitors the fidelity of each application in the shared environment at runtime and reverts to isolated program execution mode if the impact of multi-programming is greater than a predefined fidelity difference threshold. Our evaluations on a 14-qubit quantum computer from IBM show that the proposed solution can improve resource utilization and throughput by up to 2x while limiting the impact on application fidelity.

Hardware solutions to study QEC codes on emerging quantum hardware: An important use-case of emerging quantum hardware is studying small QEC codes so that large fault-tolerant systems can be built in the future. QEC codes encode logical qubits and distribute information using several physical qubits. By periodically executing a syndrome extraction circuit on the logical qubits, information about errors (called *syndrome*) is extracted while running programs. A decoder uses the syndromes to identify errors in real-time (typically in the order of a micro-second on existing superconducting quantum computers), which is necessary to prevent the accumulation of errors. Unfortunately, although there exist several decoding algorithms in practice, they typically rely on software implementations that are too slow to enable real-time decoding.

To enable real-time decoding for small QEC codes on emerging quantum computers, this dissertation presents ***LILLIPUT***— a Lightweight Low Latency Look-Up Table decoder, in Chapter 6. LILLIPUT consists of two parts— First, it translates syndromes into error detection events that index into a Look-Up Table (LUT) whose entry provides the error information in real-time. Second, it programs the LUTs with error assignments for all possible error events by running a software decoder offline. LILLIPUT tolerates an error on any operation in the quantum hardware, including gates and measurements, and the number of tolerated errors grows with the size of the code. LILLIPUT utilizes less than 7% logic on off-the-shelf FPGAs enabling practical adoption, as FPGAs are already used to design the control and readout circuits in existing systems. LILLIPUT incurs a latency of up to 42 nanoseconds and enables real-time decoding. A key limitation of LILLIPUT is that its memory complexity grows exponentially with the size or redundancy of the code. We propose Compressed LUTs (CLUTs) to reduce the memory required by LILLIPUT. By exploiting the fact that not all error events are equally likely and only storing data for the most probable error events, CLUTs reduce the memory needed by up-to 107x (from 148 MB to 1.38 MB) without degrading the accuracy of the decoder.

CHAPTER 2

BACKGROUND

2.1 Basics of Quantum Computing: A Primer on Quantum Information Theory

A **qubit** is the fundamental unit of information on a quantum computer and may be represented mathematically using a complex vector $|\Psi\rangle$ that denotes a *superposition* of the basis states $|0\rangle$ and $|1\rangle$ as shown in Equation 2.1, such that α and β are complex probability amplitudes with $|\alpha|^2 + |\beta|^2 = 1$.

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (2.1)$$

To visually denote a qubit, we may also use the Bloch sphere representation where a qubit is any point on the unit sphere [51], as shown in Figure 2.1(a). Similarly, an n -qubit system simultaneously exists in a superposition of 2^n basis states. This exponential scaling in state space with a linear increase in the number of qubits enables quantum advantage.

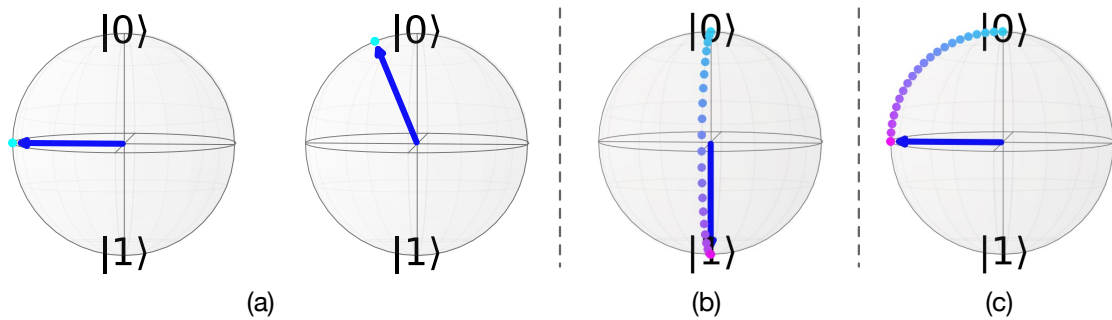


Figure 2.1: (a) Bloch sphere representation of a qubit. (b) X gate performs an inverse or bit-flip operation. (c) H gate creates an equal superposition state.

Quantum gates are operations that enable us to manipulate the probability amplitudes of the qubits during computations. We can think of quantum gates as operations that rotate a qubit on the Bloch sphere. For example, Figure 2.1(b) shows how an X gate takes a qubit

from state $|0\rangle$ to $|1\rangle$. For a qubit in superposition, the X gate flips the probability amplitudes corresponding to the two basis states, as denoted in Equation 2.3. We can abstract the X gate operation as an inverse or *bit-flip* operation.

$$|\psi\rangle \xrightarrow{X} \beta |0\rangle + \alpha |1\rangle \quad (2.2)$$

Similarly, a different gate enables a different rotation. For example, the Y gate introduces a relative phase between the two basis states of a qubit. So, the Y gate is equivalent to a *phase-flip* operation on the qubit ψ that changes its state as described in Equation 2.2.

$$|\psi\rangle \xrightarrow{Y} \alpha |0\rangle - \beta |1\rangle \quad (2.3)$$

The Z gate performs both bit-flip and phase-flip operations. So applying the Z gate on the qubit ψ changes its states as described by Equation 2.4.

$$|\psi\rangle \xrightarrow{Z} \beta |0\rangle - \alpha |1\rangle \quad (2.4)$$

The X , Y , Z , and I (identity) operations together are also called the *Pauli* operations [52]. The *Hadamard* or H gate creates an equal superposition of $|0\rangle$ and $|1\rangle$, as shown in Figure 2.1(c). The X , Y , Z , and H gates are **single-qubit gates** that only manipulate the state of one qubit. Similarly, we can have multi-qubit operations. For example, the *Controlled NOT* or $CNOT$ operation is a **two-qubit gate** that operates on two qubits (control and target) and *entangles* them. If the control and target qubits are both in state $|0\rangle$, their states after performing the $CNOT$ remain unchanged. However, if the control qubit is in state $|1\rangle$, it flips the target qubit from $|0\rangle$ to $|1\rangle$ or $|1\rangle$ to $|0\rangle$ depending on its state.

Quantum states and gates may also be denoted using a matrix representation. For example, Equation 2.5 shows the matrix representation of the basis states and an arbitrary quantum state, whereas Equation 2.6 shows the representation of different operations.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \quad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \quad |\psi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} \quad (2.5)$$

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (2.6)$$

Quantum operations may be denoted as matrix operations. For example, Equation 2.7 shows the transformation caused by an X gate.

$$X |0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \times 1 + 1 \times 0 \\ 1 \times 1 + 0 \times 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \quad (2.7)$$

Quantum measurement or **readout** operation destroys the quantum superposition and collapses a qubit into one of its basis states depending upon the probability amplitudes associated with each of them. For example, the qubit $|\psi\rangle$ collapses to a binary “0” or “1” with probabilities $|\alpha|^2$ and $|\beta|^2$ respectively upon measurement. Similarly, measuring n qubits yields an n -bit classical outcome or bitstring. For a comprehensive understanding of quantum information theory, please refer to the book by Nielsen and Chuang [53].

2.2 Quantum Algorithms

Quantum algorithms use quantum gates to manipulate the state space of the qubits such that the resultant quantum **program** or **circuit** produces the desired outcome. In most quantum algorithms, the qubits are initialized in a superposition state (when all possible states are equally likely), and manipulated through a sequence of quantum gates to adjust the probability amplitudes of the basis states such that the probabilities of the correct outputs of the program are amplified so that the qubit collapse to those desired outcomes when measured at the end of the program. Let us take the example of a quantum circuit, called the *Green-*

berger–Horne–Zeilinger or the *GHZ* circuit [54], shown in Figure 2.2(a). It is a 2-qubit circuit where the qubits $q[0]$ and $q[1]$ are initialized to $|0\rangle$. The H gate creates an equal superposition on $q[0]$ and the $CNOT$ operation controls the state of $q[1]$ depending upon the state of $q[0]$. For 50% of the times when $q[0]$ is in state $|0\rangle$, the state of $q[1]$ remains unchanged. For the remaining 50% of the times, when $q[0]$ is in state $|1\rangle$, the state of $q[1]$ changes to $|1\rangle$. Measuring the two qubits into the 2-bit classical register c produces the outcomes “00” or “11” with equal probabilities, as shown in Figure 2.2(b). In fact, even if we were to measure only one of the qubits, we would exactly know the outcome of the measurement of the other qubit due to the properties of the entangled state.

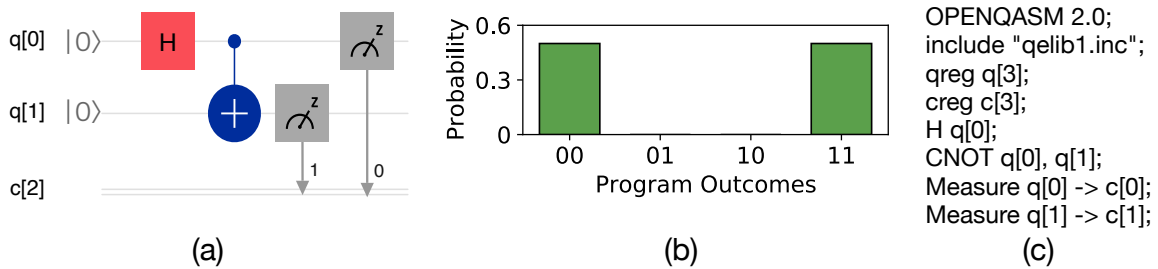


Figure 2.2: (a) A GHZ circuit and its (b) output distribution. (b) The equivalent quantum program is written in IBM’s OPENQASM language.

The circuit transformation can be mathematically denoted as shown in Equation 2.8.

$$|00\rangle \xrightarrow{H} \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) \xrightarrow{CNOT} \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad (2.8)$$

We can create a GHZ state with any arbitrary number of qubits. For example, although there are 2^{100} possible states with a hundred qubits, the GHZ circuit reduces the state space to only two possible outcomes (*all-zeros* and *all-ones*) with just a hundred operations. Just like traditional computing, quantum applications are mapped to quantum algorithms and written as quantum programs in high-level quantum programming languages. Figure 2.2(c) shows the equivalent quantum program written in IBM’s OPENQASM for the two-qubit GHZ circuit, Although, this program may still seem very low-level, quantum programming languages in reality offer high-level semantics and programming primitives.

2.3 Quantum Hardware: Qubit Devices and Control Processor

A quantum computer comprises of two key hardware components- qubit devices and the control processor. A (physical) qubit is any two-level system that can exist in any quantum superposition of two independent (physically distinguishable) quantum states. The qubits are controlled by a classical control processor that sends the appropriate signals to perform quantum operations (both gate and measurement). The control processor also comprises the readout logic that transforms the measurement output signals of qubits into 0s and 1s.

There are several possible implementations of physical qubits namely superconducting qubits, trapped ions, photonic qubits, neutral atom-based qubits, silicon spin/quantum dots, and topological qubits which are briefly described next.

1. **Superconducting qubits:** are made up of Josephson junctions that exhibit quantized energy levels when cooled to milli-Kelvin temperatures. Also known as artificial atoms, superconducting qubits are controlled using microwave signals.
2. **Trapped ions:** use ions as qubits and a trap that holds them in specific locations. They are controlled using a laser (or microwave) and measured using photon detectors that detect the number of photons scattered during the readout operation.
3. **Photonic qubits:** use photons for implementing qubits that are optically controlled using components such as phase shifters and beam-splitters.
4. **Neutral atoms:** use atoms trapped together in an array cooled to ultra-low temperatures and controlled using the precise application of lasers.
5. **Silicon spin/quantum dots:** are artificial atoms created by adding an electron to semiconductor materials (such as silicon atoms) and then the spin of the electron is controlled via microwave signals.
6. **Topological qubits:** use braids or knots in trapped quasiparticles or anyons to implement the qubits. They are extremely hard to build and have not been demonstrated yet.

Figure 2.3 shows the historical progress made towards building quantum computers of different sizes across various device technologies as well as the system sizes that are currently planned for the future (derived from the market and technology report by Yole [55] and other publicly available data, press releases, and corporate roadmaps).

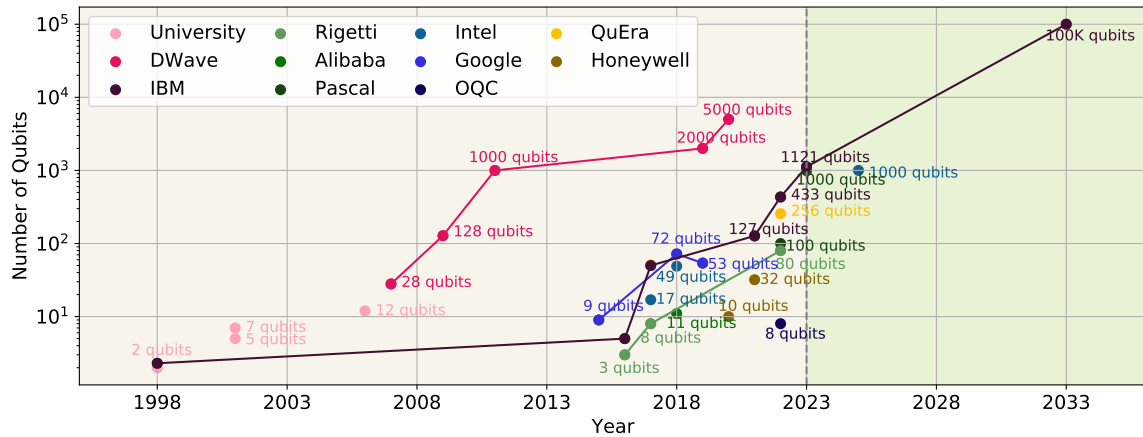


Figure 2.3: Physical qubits roadmap of quantum computers (the past and promised future)

Now a natural question probably arises in the mind of a reader as to why there exist so many independent approaches to building qubits. In reality, there are several factors that determine the "performance" of a qubit are described next.

1. **Qubit lifetimes:** Qubits must retain their information for prolonged periods of time.
2. **Operation fidelities:** Quantum operations must be supported with high fidelities.
3. **Connectivity:** Connected qubits enable us to perform two-qubit operations directly whereas unconnected qubits require additional operations to move them to adjacent locations. Therefore, qubit technologies that offer higher connectivity are desirable.
4. **Scalability:** We must be able to build and control up to thousands and millions of qubits without increasing their error rates.

Each qubit technology presents a unique set of trade-offs, as summarized in Table 2.1. For example, trapped-ion systems offer higher qubit connectivity and gate fidelities but it is harder to scale their laser control electronics.

Table 2.1: Comparison of different qubit technologies

Qubit Technology	Advantages	Challenges	Example
Superconducting	High gate fidelities	Cryogenic operations Short qubit lifetimes	Google, IBM Rigetti
Trapped ions	Extremely high gate fidelities Very long qubit lifetimes	Poor laser scalability Ultra-high vacuums	Honeywell IonQ
Photonic	Promising gate fidelities No cryogenics or vacuums	Two-qubit gates hard Short qubit lifetimes	PsiQuantum Xanadu
Neutral atoms	Promising gate fidelities Long qubit lifetimes	Poor laser scalability Ultra-high vacuums	QuEra, Pascal ColdQuanta
Silicon/Spin	Promising gate fidelities Good qubit lifetimes	Cryogenic operations High interference/crosstalk	Intel
Topological	Extremely high gate fidelities Extremely long qubit lifetimes	Extremely hard to build * No demonstration yet	Microsoft

Although superconducting quantum systems had a head-start, other technologies are now catching up as well. Scaling the qubit devices as well as the related control architectures remains a grand engineering challenge, and both remain active areas of research. The control processors are mostly designed as custom accelerators that are specially tailored for the purpose of controlling the qubits as opposed to general-purpose hardware such as conventional CPUs. Currently, FPGA-based designs are used as they are cost-efficient [28, 56, 57, 58, 59]. Also, their reconfigurability allows us to modify the designs more conveniently compared to silicon ASICs. However, these designs are only preliminary steps, and we need to move towards more efficient ASIC-type designs as the number of qubits scales. For the same, currently, both CMOS [60, 61] and superconducting device technologies [62] are being considered as potential choices for implementations. CMOS and superconducting devices offer unique trade-offs. While superconducting designs are more energy-efficient compared to CMOS, the technology lacks large device densities and dense memory solutions. Thus, in addition to qubits, substantial research efforts are needed to build better control processors and system-level solutions for quantum computing.

2.4 Organization of a Quantum Computer

Quantum computers require sophisticated infrastructure for qubits and control electronics and incur huge deployment and operational costs. The infrastructure required depends on the qubit device technology. For example, as explained earlier, superconducting qubits require cryogenic coolers and microwave devices, whereas trapped-ion quantum computers require ultra-high vacuum chambers and lasers. Hence, they are accessible to users via cloud services. Cloud services ensure ease of use where users can take their hands off actual resource management tasks. User programs, written in high-level languages, are translated into executables by decomposing each high-level instruction into machine-compliant low-level native gates using a compiler. A host processor, which is a classical system, provides the software development tools and services as well as the networking and storage support required to allow users to interact with a quantum computer. The classical control processor uses the compiled code to schedule pulses and manages the precise control necessary to perform operations on the qubit devices. The program solution is inferred from the output distribution obtained by running the executable for thousands of trials. Note that VQAs are executed for thousands of iterations where each such iteration consists of several trials. Figure 2.4 shows the organization of a quantum computer.

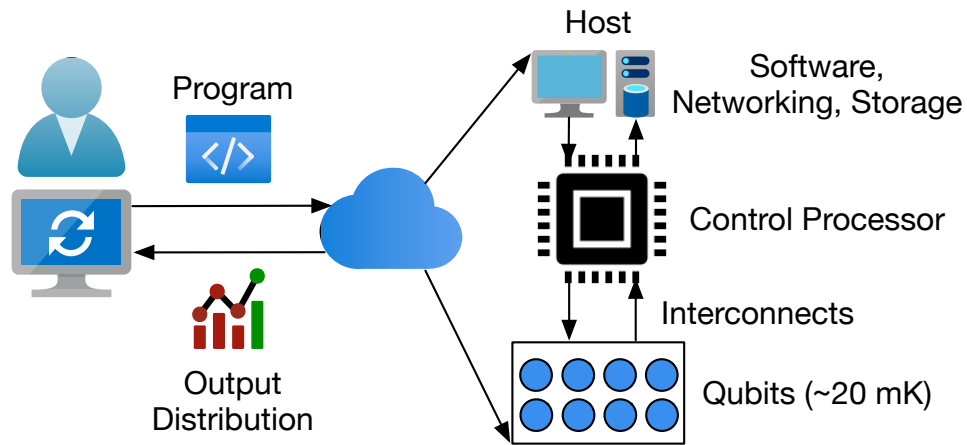


Figure 2.4: Organization of a quantum computer.

2.5 From Quantum Application to Qubit Devices

Quantum programs are written in high-level languages, such as OpenQASM [63] or Circ, which must be translated into a sequence of device-specific low-level assembly instructions for execution on the quantum hardware.¹ The set of low-level instructions that are supported on a quantum device (also known as native or basis gates) or the Instruction Set Architecture (ISA) depends on the machine. For example, although both IBM and Google quantum computers use superconducting transmon devices, they use different low-level instructions to perform the actual gate operations. The role of a NISQ compiler is to transform a high-level program into a machine executable format or *quantum object code*. Figure 2.5 shows an overview of the key steps involved in NISQ compilation.

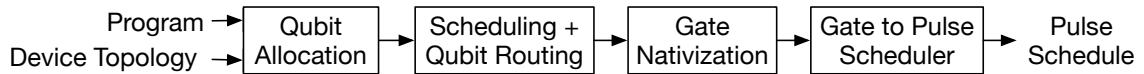


Figure 2.5: Overview of NISQ compilation.

The first step to compile a program is to allocate the physical qubits of the device to the program qubits, a process called *qubit allocation* or *qubit mapping*. For example, qubits Q_0 to Q_3 of the Bernstein Vazirani benchmark are mapped to physical qubits A to D , as shown in Figure 2.6(a-b). Additionally, compilers perform the task of *qubit routing* to overcome the limited connectivity of NISQ devices. The qubit connectivity on most near-term quantum computers is sparse and qubits generally do not have all-to-all connectivity. On the other hand, two-qubit gate or CNOT operations can only be performed between two physical qubits that are connected via a coupler or link. To perform a CNOT gate between two unconnected qubits, compilers introduce SWAP instructions that relocate the qubits to two adjacent locations that are physically connected. For example, the CNOT Q_2, Q_3 operation in Figure 2.6(a) cannot be performed because physical qubits B and D are not connected. To perform this operation, the compiler introduces a SWAP instruction,

¹The compilation flow for fault-tolerant program execution is different from the NISQ execution model and its discussion is beyond the scope of this thesis.

as shown in Figure 2.6(c). Note that the compiler may also SWAP qubits Q_0 and Q_3 or qubits Q_1 and Q_2 , instead of qubits Q_1 and Q_2 . This process of SWAP insertion is called *qubit movement* or *qubit routing*. Each SWAP is a sequence of three CNOT gates and takes about 1.2 microseconds on current generations of IBM quantum computers. Thus, SWAP operations incur the overheads of increased gate operations as well as circuit depth. Relocating two non-adjacent qubits on larger devices may require several SWAPs depending on the distance between them. Overall, the SWAP overheads depend on the size and characteristics of a program and the sparsity of the qubit connectivity topology.

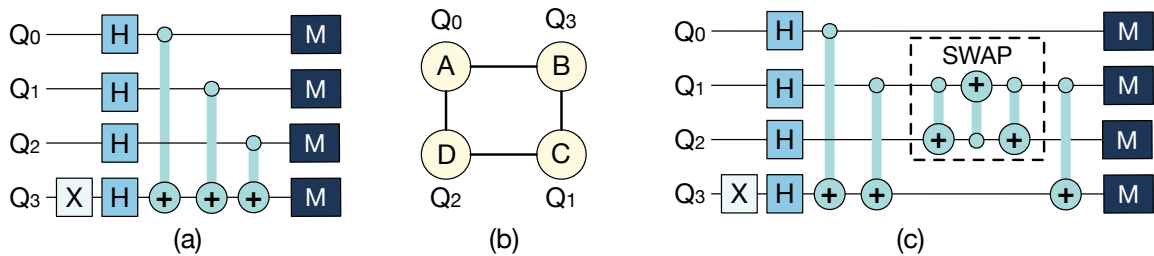


Figure 2.6: (a) A 4-qubit Bernstein Vazirani circuit (b) Mapping of program qubits to the physical qubits of an example NISQ hardware (c) Compiler SWAPs qubits Q_1 and Q_2 to enable the CNOT operation between qubits Q_2 and Q_3 as they are not physically connected.

The next step is *gate nativization*, during which the operations of the scheduled and routed program are decomposed into native gates supported by the device. It is possible to decompose the same high-level operation using more than one combination of native gates. Moreover, many recent qubit machines support more than a single two-qubit native gate, and a high-level CNOT operation can be decomposed using either of them on these machines. For example, Rigetti systems allow up to three 2-qubit native gates, namely CZ, CPHASE, and XY, each of which differ in how they are actually implemented in the hardware. This gives additional flexibility to the compiler to express high-level operations.

Compilation for superconducting quantum systems involves an additional step after a high-level quantum program is translated to a machine-compliant assembly code because, in reality, qubits devices are controlled by analog signals sent using microwave waveform

generators.² Superconducting qubits encode information in a non-linear oscillator formed by connecting a Josephson junction and a capacitor in parallel. To perform operations, these devices are controlled by applying analog pulses of microwave frequencies. Thus, the low-level assembly code is further translated to a sequence of control pulses that are actually used to execute the program on the real hardware. It is the pulse level implementations that differentiate two native gates even if they have identical functionality. For example, the pulses used to implement CZ, CPHASE, and XY gates even on the same qubit are different on Rigetti systems. In the most naive form of execution, a user can simply send a program to the cloud service provider that hosts the quantum computer where the backend tools perform the necessary compilation steps. Alternatively, advanced users may also choose to compile a program on their end locally and send the compiled quantum object code to the cloud provider. Thus, in reality, the software stack must be capable of addressing the requirements of a broad set of users ranging from quantum enthusiasts to experts.

As existing quantum ISAs consist of only a limited set of single-qubit and two-qubit instructions, the two-step approach is inefficient and may result in long latency object code with greater vulnerability to errors. To overcome this limitation, some quantum systems break the ISA abstraction and expose a much lower level of control to the software. Instead of translating programs to a quantum object code with machine-compatible low-level instructions and then generating the control pulses, *pulse-level compilers* can directly translate programs into pulse-level instructions producing a time-based *schedule*. Currently, this feature is supported mainly on IBMQ systems and has been shown to be effective. Compiling a program to a pulse-level schedule requires decomposing the circuit into an accurate sequence of pulses scheduled in time with proper synchronization. While pulse-level compilers allow fine-grained control to programmers and support the addition of custom pulses and new gate definitions, they must ensure accurate timing and synchronization between the instructions to ensure functional correctness.

²Other systems translate into equivalent control sequences such as laser pulses. We limit our discussion to superconducting devices for the scope of this thesis.

2.6 Errors in Quantum Hardware

Qubits are extremely sensitive to noise and prone to errors. These errors can produce incorrect outputs during computations. For example, the circuit shown in Figure 2.7 is ideally expected to produce the bitstrings “00” and “11” with 50% probability each. However, in the presence of errors, the circuit may produce these outcomes with incorrect probabilities. Noise can also lead to erroneous outcomes (such as “01” and “10” in this example).

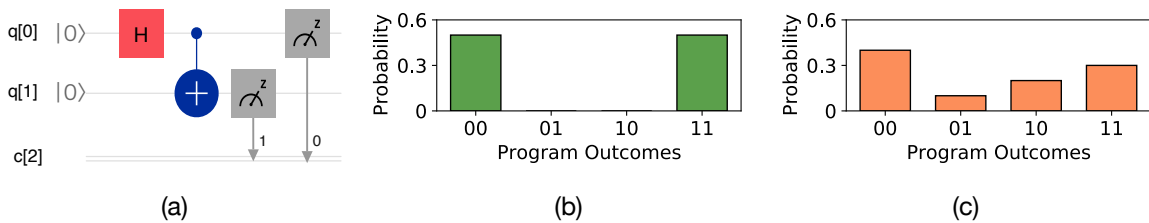


Figure 2.7: (a) A two-qubit GHZ circuit and its (b) ideal output distribution on a noise-free quantum computer. (c) The output distribution on a real noisy quantum computer.

Qubit errors can be broadly classified into two categories:

1. **Idling errors:** Qubit devices offer limited lifetimes when it comes to storing quantum information as they accumulate errors from their unwarranted interactions with their environments. Qubits can lose their information even when they are left idle and no operations are performed on them. These errors are called *decoherence* errors and the probability of a qubit encountering this type of error increases exponentially with time. This limits the depth of circuits or the longest sequence of serial operations that can be reliably run on a quantum machine. For example, coherence errors cause qubits to naturally decay to the lowest energy state ($|0\rangle$) within a few microseconds on existing Google and IBM hardware. Moreover, qubits can also lose their phase information due to their unwarranted interactions with environmental noise channels (known as *dephasing*). In addition to decoherence errors, undesirable crosstalk from ongoing operations during program execution can affect the state of their neighboring (*spectator*) idle qubits [64]. These types of errors are also often referred to as *idling* errors as they impact a qubit while it is idle.

2. Operational Errors: These errors are caused when a qubit is actively performing any gate or measurement operations. On superconducting quantum computers such as those from IBM and Google, quantum instructions are performed by sending analog voltage and current pulses of microwave frequencies to the qubits, couplers connecting the qubits, and resonators. However, these operations are imperfect and cannot be applied with precise accuracy. Errors in single-qubit and two-qubit operations result in *gate errors*. The average error-rate is 0.1% for single-qubit gates and about 1% for two-qubit gates on IBMQ systems. Qubit measurement operations performed at the end of a program are vulnerable to errors too. *Measurement errors* lead to incorrect results even if a program does not encounter any other errors. On existing IBMQ and Google hardware, measurement operations tend to be the dominant sources of errors, with average error-rates between 1-4%. Moreover, the error-rate of these operations depends on the number of other ongoing operations. For example, simultaneous two-qubit or measurement operations result in an increase in the effective operational error-rate due to an increase in the number of sources of unwanted couplings and crosstalk.

Performing a quantum operation requires very precise control pulses which requires determining the optimal control parameters for the on-chip device components. This process is known as *calibration* and is a crucial device management operation. Calibration is necessary to perform high-fidelity gate and measurement operations and quantum computers require frequent and accurate system-level calibration (in the order of a few hours). It requires the execution of several randomized benchmarking circuits to adequately capture the error trends. However, it also means that a machine could become unusable when calibration operations are ongoing. To reduce the system downtimes, most device providers or cloud providers often resort to localized infrequent calibrations and often compromise on operational fidelities. Typically, with re-calibrations, the device error-rates change and this information is periodically exposed to the software.

2.7 Software Error Mitigation: Prospects, Challenges, and Related Works

The objective of software error mitigation is to reduce the impact of hardware errors using the software. These policies are developed to be used at the compilation stage, output post-processing stage, or a combination of both. Compilation-level error mitigation policies attempt to generate highly optimized machine code such that the compiled executable is more resilient to hardware errors. Output post-processing schemes alter the noisy output distributions obtained from the NISQ hardware to boost the probabilities of the correct outcomes while reducing those of the incorrect outcomes. Some of the error mitigation techniques use the compilation tool-chain to generate additional characterization circuits or functionally identical copies of the same circuit (that generate different program schedules due to differences in the scheduling) to gather more accurate device and/or application-specific error information. The output distributions of these additional circuits are used to rectify the output distribution of the original program.

Software error mitigation techniques promise to bridge the gap between algorithms and devices in the NISQ-Era and have the potential to accelerate the field by 10-20 years [49]. often incur overheads in terms of the number of quantum circuit executions, classical resources required for compilation as well as post-processing. In most cases, if the overheads for these operations are prohibitively large, then the quantum speed-up may not be very attractive. For example, solver-based compilers lead to several days of compilation time even for moderately large programs with few tens of qubits and therefore, the overall latency of the quantum task will be dictated by the compilation time. On the other hand, if a compilation strategy increases the number of circuit executions by 10x, then running them on real quantum computers via cloud services will be heavily bottlenecked by the long cloud waiting latencies. Thus, for software error mitigation techniques to be beneficial and practical in improving the overall reliability of emerging quantum systems, we must be able to parallelize the classical as well as the quantum computations as much as possible.

Some of the key error-mitigation methods proposed in recent years are discussed next.

2.7.1 Qubit Allocation and Routing Policies for NISQ Applications

SWAP minimization: Compilers introduce SWAP instructions to overcome the limited connectivity on NISQ hardware which increases the effective error-rate. To minimize the impact of SWAPs, Zulhener et al. proposed an A* search algorithm that searches for all possible SWAP combinations between two qubits and selects the routing path which minimizes the number of SWAPs [65]. Further, this study shows that a local A* search for each pair of unconnected qubits may be sub-optimal at the program-level because SWAPs relocate qubits that affect future operations. Instead, the proposed SWAP insertion policy uses a look-ahead scheme where the circuit is decomposed into layers and SWAPs are introduced in a given layer such that the relocations caused by these SWAPs aid operations in future layers. This approach optimizes for reduced circuit depth and gate counts. SWAP minimization is NP-Complete [66]. In recent years several other compilers have been proposed that use solvers [67, 68, 69, 70, 71, 72, 73, 74, 75, 76], graph-partitioning [77], and dynamic programming [66]. However, they incur long latencies that make them hard to scale and limit their adoption for practical applications with hundreds of operations. More recently, industrial tool-chains have relied on heuristic compilers [78, 79, 80, 81].

Low-latency SWAP minimization: The A* algorithm is hard to scale due to its poor complexity. SABRE reduces the routing complexity by exploiting the insight that a SWAP can only be performed between two adjacent qubits and therefore, the search space can be significantly reduced [82] by searching for SWAP candidates locally. SABRE uses a heuristic approach that exploits the reversibility property of quantum circuits to compile both forward and reverse circuits to eventually converge on a good initial mapping. SABRE reduces the compilation latency while still minimizing the SWAP overheads during routing. An orthogonal method, known as the time optimal qubit mapping or TOQM compiler trades off SWAP cost for lower depth [83].

2.7.2 Noise-Aware Compiler Policies for NISQ Applications

Success rate maximization: The error-rates of qubits devices and links are non-uniform across the entire system (spatial variation). For example, the error-rate of the most vulnerable link on 20-qubit IBMQ-Tokyo is up-to 7.5x the error-rate of the link with the lowest error-rate [47]. Furthermore, the error-rates of each qubit and link does not remain constant but vary with time (temporal variation). Therefore, the probability of successfully executing an operation depends on the individual error-rates of the qubit and link at any given instant of time. Recent studies have proposed noise-aware qubit allocation and routing policies [47, 48]. The objective of these optimizations is to map program qubits to less erroneous physical qubits and choose SWAP paths that maximize the overall probability of successfully executing a program as opposed to choosing the minimal number of SWAPs.

Murali et al. proposed a generic noise-aware compiler that translates a program into machine-compatible object code for multiple quantum computers [84]. This framework allows programmers and device experts to conduct real-system studies on different hardware platforms and propose directions for future hardware/software enhancements.

Exploiting reversibility of quantum circuits to learn noise: Quantum circuits are reversible which means if the true (or forward) circuit and reverse circuit are executed together, the output of the circuit will be the same as the input in the absence of noise. QRAFT leverages this insight to run (1) the given circuit and (2) the given circuit with the reverse circuit [85]. At the end of the execution, QRAFT uses the outputs of (2) to learn about the state errors of individual output states of the given circuit. To accomplish this QRAFT uses a “true state probability prediction model”. The role of this model is to predict the output of the forward circuit, given an output observed in the forward plus reverse circuit. The model is trained using a large variety of random circuits with different types of static and dynamic features and tuned using Bayesian Optimization. The trained model is then used to predict the true state probabilities of any quantum algorithm.

2.7.3 Circuit Decomposition Techniques

Peng et al. [86] proposed a theoretical approach for decomposing a larger program into smaller sub-circuits for execution on smaller quantum machines. In CutQC, Tang et al. introduced a scalable approach for efficient circuit cutting using this insight to improve the fidelity of large programs [87]. This tool decomposes a program into multiple smaller sub-circuits which are executed independently and the individual results are used to reconstruct the output distribution of the original program using a classical machine. As the sub-circuits make fewer gate operations and measurements, their vulnerability to errors is reduced. Unfortunately, this scheme reconstructs the output of the probability of each of the 2^n outcomes of an n -qubit program and therefore, incurs exponential complexity.

2.7.4 Error-Specific Mitigation Policies

In addition to software optimizations that address errors in general, various other proposals focus only on mitigating specific types of errors, which are discussed next.

Policies for mitigating correlated errors

Error diversification: The incorrect outcomes produced on a NISQ computer are not fully arbitrary and do not appear with a uniform probability. Instead, some of the outcomes are produced with much higher probability due to *correlated* errors. For example, if the correct answer of a 4-qubit program is “1111”, the erroneous outcome “1110” often occurs with a much higher probability than “1000”. Executing all the trials of a program on the same physical qubits using the same compiled quantum object makes the program vulnerable to the same correlated errors. The Ensemble of Diverse Mappings (EDM) policy executes multiple copies of a program on different initial mappings (which eventually results in different SWAP routes) as opposed to the best mapping [88]. This steers the program to make dissimilar mistakes and therefore, encounter different correlated errors. When the results from individual executions are added, the impact of correlated errors is minimized.

Machine diversification: Patel et al. improved the effectiveness of EDM in VERITAS by selecting dissimilar initial qubit allocations on different machines to improve the diversity introduced by each mapping [89]. This allows the compiler to choose the best mapping on each device as opposed to sub-optimal mappings on the same device. VERITAS also post-processes the output distribution by selectively adjusting the probabilities of each outcome.

Policies for mitigating crosstalk

Crosstalk-aware scheduling: Operational crosstalk affects the fidelity of an operation if it is scheduled concurrently with other operations. For example, CNOT-CNOT crosstalk deteriorates the fidelity of CNOT operations when multiple CNOTs are scheduled in parallel. To reduce the impact of CNOT-CNOT crosstalk, the gates must be scheduled serially. This introduces a trade-off because although concurrent CNOTs can reduce the circuit-depth, they are more vulnerable to crosstalk. Instead of scheduling all CNOT gates serially, Murali et al. describe an optimal scheduling policy that schedules only those CNOTs serially that are most susceptible to crosstalk while maximizing the concurrency for the remaining CNOTs [90]. To be effective, this scheme must accurately identify CNOT pairs that are most susceptible to crosstalk. Instead of extensive device characterization, this scheme uses localized characterization as crosstalk mostly affects nearest neighbors.

Crosstalk-aware frequency allocation: Frequency crowding leads to crosstalk in superconducting systems because frequencies allocated to qubits must satisfy several constraints for high-fidelity operations but can only be chosen from a limited range. For example, while the frequency of a two-qubit gate must be far from its neighboring gates, only a few options are available in practice due to range limitations. This results in crosstalk as frequencies allocated on real devices are close to each other. Shi et al. proposed a compilation algorithm that mitigates the impact of crosstalk via program-specific frequency tuning and instruction scheduling [91]. This work exploits the fact that crosstalk between two parallel gates can be avoided by creating sufficient separation: (a) either in frequency, (b) or in time.

Policies for mitigating measurement errors

Measurement errors are the dominant source of errors on most superconducting quantum computers and several policies have been investigated to reduce the impact of these errors.

Noise matrix based post-processing: Matrix-based methods post-process the noisy output distribution of a program using an inverse noise matrix [92, 93, 94]. IBM’s complete error mitigation uses a $2^n \times 2^n$ matrix for an n -qubit program, where the matrix captures the probability of measuring each of the 2^n states as one of the 2^n states. For example, the noise matrix of a 2-qubit program is a 4x4 matrix where the entry corresponding to “00” comprises of the probabilities of measuring this state as “00”, “01”, “10”, and “11”.

Preparing the full inverse noise-matrix and post-processing incurs exponential complexity and is hard to scale. IBM’s tensored error mitigation overcomes this limitation by assuming independent measurement errors. Bravyi et al. propose a scheme whose complexity is between the complete and tensored error mitigation schemes. It assumes independent and locally correlated errors but can only be applied to specific applications [95].

State-Based Error Mitigation: The vulnerability of a qubit to measurement errors depends on its state and the probability of successfully measuring state “0” is higher than state “1”. Few prior schemes exploit this variability and transform a more error-prone quantum state to another one less susceptible to errors [93, 96] using single-qubit gates. For example, if the state of a qubit is “1”, the compiler transforms it to state “0” using an X gate before measuring it. As the error-rates of X gates are much lower than measurement error-rates, introducing an X gate per qubit does not increase the effective error-rate. However, the output of a program is not known a-priori and determining which qubits to flip is a challenging problem. For example, if a qubit that is supposed to output a “0” is flipped and measured, its effective error-rate may increase. To tackle this challenge, Tannu et al. introduced static and dynamic policies to selectively flip qubits so that the overall probability of success is maximized.

2.7.5 Application-specific Error Mitigation Policies

Quantum Approximate Optimization Algorithms (QAOA) [38] is considered to be one of the most promising NISQ applications and is primarily used for hard combinatorial optimization problems in various application domains. QAOA circuits consist of several multi-qubit CPHASE operations which depends on the problem being solved. In reality, these CPHASE operations commute which means the order of the gates can be altered without changing the output of the circuit (theoretically). Alam et al. exploit this insight and propose a compiler specifically for QAOA circuits that re-orders the CPHASE gates in the circuit such that a maximum number of these gates can be scheduled in parallel. This reordering results in increased concurrency which effectively reduces the circuit depth [97]. This study also discusses efficient qubit allocation and routing policies so that the overall number of SWAPs is minimized while improving concurrency. Ayanzadeh et al. show that most real-world applications cater to irregular graphs where some nodes have a larger number of edges or connections compared to others, also known as hotspots. Moreover, these nodes contribute to a larger number of operations in the QAOA circuits (as the number of operations contributed by a node is directly proportional to the number of edges connected to it). By leveraging this insight, the authors propose to split a large QAOA circuit into smaller sub-circuits with fewer operations by dropping a limited number of hotspots, running them independently, and computing the final solution at the end.

Application-specific circuit design techniques exist for Variational Quantum Eigensolver (VQE) applications in quantum chemistry for which the VQE circuit (or ansatz) is prepared by tailoring the qubit-qubit interactions or two-qubit operations specifically based on the connectivity of the device [98]. A similar strategy can also be adapted for (variational) quantum machine learning (QML) applications on NISQ machines. However, some recent studies suggest that although hardware-efficient circuit design approaches are more resilient to noise, they may affect the overall trainability of the circuit parameters for variational algorithms [99].

2.7.6 Pulse-level Compilation Policies

These policies exploit the insight that existing basis gate sets or the assembly-level instruction sets to which programs are compiled are often too far from the actual hardware primitives at the pulse-level. Instead of compiling to the ISA, pulse-level compilers directly compile a program to time accurate pulses with reduced vulnerability to errors. However, most of these approaches require additional support for determining and calibrating custom pulses on each specific quantum hardware.

Optimal pulse schedule generation: Gokhale et al. have proposed pulse-level optimizations to achieve greater fidelity [100]. These optimizations perform any single-qubit gate using only one pulse instead of using multiple pulses and thus, reduce the latency and possibility of errors. This compiler also discusses optimizations to perform more efficient two-qubit gates. Lastly, this compiler leverages additional energy levels of a qubit beyond the conventional two levels to improve the fidelity of the readout operations. A similar approach was used to improve the Quantum Volume on IBMQ systems [101], which is a hardware and software agnostic metric used to benchmark quantum computers [102].

Instruction aggregation: Shi et al. discuss optimizations to generate custom pulses for an aggregated block of instructions [103]. This reduces the program latency as the optimized pulse schedule takes lesser time to execute. For example, the latency of a SWAP is reduced by a factor of 3.65x if an optimized pulse is used instead of combining the individual pulses for the block of three sequential CNOT gates. As the probability of decoherence increases exponentially with time, reducing the pulse duration significantly improves the fidelity of quantum circuits. This approach (1) translates a circuit into low-level assembly instructions, (2) reduces the circuit depth by instruction reordering generates, (3) generates an optimized pulse-schedule for the aggregated instructions block.

2.8 Quantum Error Correction using Surface Codes

QEC codes enable us to close the gap between the error rates of the quantum hardware and the error rates that can be tolerated at the application level. QEC codes encode a logical qubit using a set of *data* and *parity* qubits. The data qubits collectively store the quantum information, whereas the parity qubits periodically extract information about errors on the data qubits by executing a *syndrome extraction circuit*. Errors on the data qubits are projected as failed parity checks on the parity qubits. By periodically executing a syndrome extraction circuit on the logical qubits, information about errors is extracted. The error-rates are reduced by identifying and correcting errors in real-time using a classical *decoder*. Once the decoder identifies the optimal correction, it sends the information to the control processor so that future operations can be updated to reverse the impact of errors. The logical error rate reduces exponentially with the increasing redundancy or *distance* of the QEC code if the physical error-rate is lower than a threshold.

Surface codes [104, 105, 106, 107] are regarded as the most promising QEC code owing to the requirement of grid or nearest-neighbor connectivity, which is relatively easier to build in hardware, and high thresholds [108]. It encodes a logical qubit into a 2-dimensional lattice of alternating data and parity qubits. The size and layout of the lattice depend on the code distance d which determines the code redundancy and the length of the shortest error chain ($\frac{d+1}{2}$) that cannot be corrected. An error on a data qubit is detected by the adjacent parity qubits by executing a syndrome extraction or stabilizer circuit, where each parity qubit measures a four-qubit operator called a *stabilizer*. X errors are detected by the Z stabilizers, whereas Z errors are detected by the X stabilizers [109]. For example, Figure 2.8(a) shows the layout of distance 3 regular surface code that can correct error chains of length 1. It consists of 13 data qubits (qubits A to M) and 12 parity qubits (qubits Z_0 to Z_6 and X_0 to X_6). An X error on data qubit D is detected by the Z stabilizers Z_0 and Z_2 , whereas a Z error on this qubit is detected by the X stabilizers X_0 and X_1 .

Figure 2.8(b) shows the layout of distance 3 *rotated surface code* that is obtained by rotating the lattice of a regular surface code by 45 degrees and removing some of the data and parity qubits. The rotated code requires fewer qubits and gate operations to extract a syndrome. Therefore, the rotated surface code is preferred over regular lattices. Figure 2.8(c) shows the Z stabilizer circuit. In the event of an error, the stabilizers are measured to be non-zero. More errors increase the length of the non-zero syndrome chain. The syndrome extraction latency depends on the speed of each operation. For example, on Google Sycamore, it takes about 1 microsecond to execute this circuit [28], which also determines the minimum duration of a *syndrome extraction cycle*. The latency is projected to reduce in the future [110]

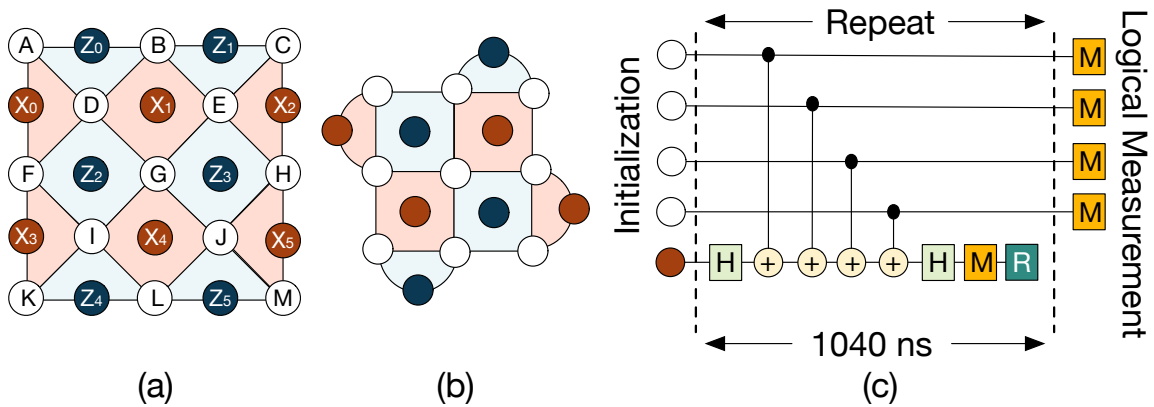


Figure 2.8: Distance 3 (a) regular and (b) rotated surface code. Z stabilizers (in blue) detect X errors and X stabilizers (in red) detect Z errors. (c) Z stabilizer circuit.

Note that the resource overheads of surface codes grow quadratic with the code redundancy or distance. To overcome this limitation, codes such as subsystem codes, Floquet codes [111, 112], and QLDPC codes have been proposed that offer asymptotically lower overheads than surface codes at the expense of more irregular and denser device topology. Mostly in theory, these codes still have a long way to go as we must figure out methodologies to construct quantum architectures with such irregular topologies while maintaining good device qualities and determine accurate algorithms to decode errors, both of which are currently open problems as of now.

2.9 Error Decoding: Prospects, Challenges, and Related Works

Figure 2.9 shows an overview of Fault-tolerant Quantum Computers (FTQCs) that **1** use a control processor to send instructions to the qubits to execute syndrome extraction circuits, **2** measure the parity qubits to obtain the syndrome bitstring, **3** use decoders to analyze syndromes for identifying errors, and **4** send the correction to the *control processor* so that errors are corrected in real-time and the future operations are updated. Thus, the overall performance of a QEC code is a function of the ability of the decoder to accurately correct errors in real-time.

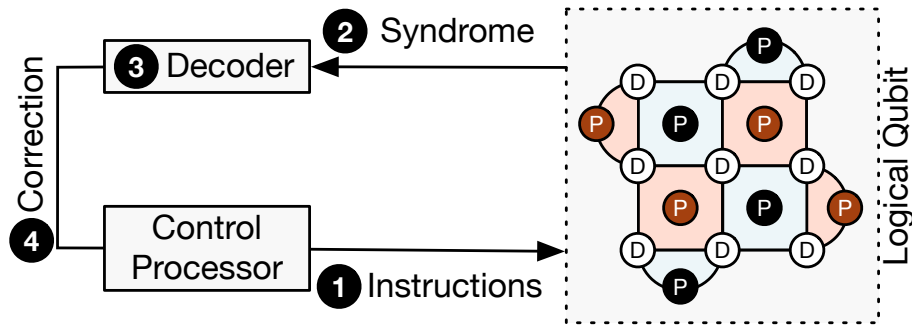


Figure 2.9: Overview of steps required for QEC.

A decoder uses the output of stabilizer measurements, the syndrome, to determine a set of corrections that must be applied to the data qubits. The length of an error-chain depends on the error-rates as well as the distance of the code. X-type and Z-type errors are corrected independently (then Y-errors are automatically corrected). Decoding may be treated as a *matching* problem on a square grid (2-dimensional space) in which the non-zero syndrome bits are matched and in the process, errors are assigned to the data qubits along the matching path. The code lattice is transformed into a *decoding graph* comprising nodes corresponding to the parity qubits and edges connecting the nodes corresponding to the data qubits. However, in reality, syndrome extraction circuits include gate and measurement operations which frequently encounter errors, resulting in erroneous or imperfect syndromes. For example, even if there is no error on a data qubit, its adjacent parity qubits

may be measured as non-zero due to a measurement error during syndrome extraction. To tolerate operational errors during syndrome extraction, a decoder analyzes at least d consecutive rounds of syndromes, where d is the code distance used. This approach translates decoding into a matching problem on a 3-dimensional graph spanning space and time. Errors on data qubits result in parity qubit flips in the same round, representing *space-like* error events. Measurement errors cause parity qubits flips across consecutive rounds, representing *time-like* error events. Gate errors during syndrome extraction result in flips on diagonally located parity qubits across two consecutive rounds, resulting in *space-time-like* error events. Thus, the complexity of decoding grows with the size of the syndrome or distance of the code. Estimates from Google suggest that distances as large as 25 may be required to run some of the practical quantum applications.

A decoder must satisfy three constraints: *accuracy*, *latency*, and *scalability* for adoption in large FTQCs. To satisfy the accuracy constraint a decoder must correctly identify the errors with high probability. The accuracy depends on the algorithm used and the QEC code. For example, the Minimum Weight Perfect Matching algorithm is widely recognized as the gold-standard for decoding surface codes, whereas similar algorithms are non-existent for color codes or emerging QEC codes. A decoder satisfies the latency constraint if it can be executed within one round of syndrome extraction. This latency depends on the qubit device technology and the operational latencies that can be supported. For example, trapped ion systems can tolerate longer decoding latencies compared to superconducting machines. Failure to achieve the latency constraint causes errors to accumulate and may lead to a backlog problem [113]. The scope of decoders in the context of emerging quantum systems typically spans the study and demonstration of QEC codes and limited forms of logical operations that are feasible on quantum hardware with a few hundred to thousands of physical qubits. However, in the long run, decoders must be implemented in a manner such that they can be scaled to hundreds and thousands of logical qubits. Therefore, the study of QEC codes and solutions for real-time decoding remains an active area of research.

2.9.1 Software Approaches to Decoding

Several decoding approaches have been proposed in the literature. However, most studies focus primarily on accuracy and typically use software implementations. For the scope of this thesis, we limit our discussions to decoding surface codes, although some of these approaches are also applicable to other codes.

Lookup Table (LUT) Decoder [114]: Tomita et al. first investigated software LUT decoders using a fixed set of rules to determine the error assignments. It uses a lookup table (LUT) to store corrections for every possible syndrome. The LUT is indexed by the syndrome bits and the corresponding entry stores the correction. However, this design is only limited to distance 3 surface codes and 3 cycles.

Minimum Weight Perfect Matching (MWPM) [106]: This decoder uses a graph pairing algorithm and is considered to be one of the most effective in terms of accuracy. However, its time complexity ranges from $O(n^3)$ to $O(n^7)$ and implementing it in less than 800 ns is an open problem [115]. Fowler proposed a parallel implementation of this decoder that reduces the average time complexity to $O(1)$, although the worst-case complexity remains significant and no practical implementation of this approach has been demonstrated [116].

Machine Learning (ML) Decoders: They train neural networks with the underlying error probability distribution. During decoding, the syndrome data is an input to the neural network that infers the correction [117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133]. They require substantial computational and memory resources to store and process large amounts of training data (in the order of GBs) depending on the code distance. However, many of the design principles described in our paper can be applied to ML-Decoders to improve their performance and scalability.

Union-Find (UF) Decoder [134, 135]: UFD is a graph-based algorithm that processes the syndrome in almost-linear time by growing clusters around non-zero syndrome bits until they have an even parity. However, its accuracy is lower than MWPM.

2.9.2 Paradigm Shift Towards Hardware Decoders

Software implementations are typically slow [136], especially in the context of superconducting quantum computers. As qubits are typically maintained at a few milli-Kelvins inside a dilution refrigerator, software decoders incur significant communication overheads in transmitting syndromes to the external CPUs running at room temperatures. Hence, more recently, there has been a paradigm shift towards hardware decoders that can be co-located on the FPGAs hosting the control processor, designed as fixed-function accelerators, or placed at 4 Kelvin inside dilution refrigerators [115, 137, 138].

Superconducting Decoders: SFQ-based decoders [115, 138] represent significant milestones in the field of fast decoders. They rely on the computational capabilities of the emerging superconducting technology to enable high-speed computations. However, they trade off accuracy for lower decoding latency and are reliant on the progress of the superconducting devices, which limits their applicability, particularly in the near-term.

Hardware Union-Find Decoders: The Union-Find decoder is extremely attractive for hardware adoption owing to its simplicity. Until now, two implementations of the algorithm have been proposed in hardware- the first one uses a custom accelerator approach [137], whereas the second one targets low-cost FPGA-centric implementation [139].

Hardware MWPM Decoders: The MWPM algorithm is widely regarded as the gold standard for decoding surface codes and an FPGA implementation of the same has been proposed to decode small distance surface codes [140].

Hierarchical Decoders: Delfosse et al. proposed a hierarchical decoder [141] that uses low-cost decoders for the average-case errors and sophisticated decoders for worst-case errors or harder to decode error events. Recently, a superconducting implementation of a lightweight low-cost decoder specifically suited for hierarchical decoder designs has been introduced [142]. Chamberland et al. proposed a different hierarchical design that uses more accurate local neural network decoders [143].

CHAPTER 3

EFFICIENT TECHNIQUES FOR MITIGATING MEASUREMENT ERRORS ON EMERGING QUANTUM COMPUTERS

This chapter of the dissertation presents *JigSaw* [144], a software error mitigation technique to reduce the impact of measurement errors on the fidelity of applications on emerging quantum computers. Qubit measurements tend to be the most error-prone operations on these systems (with an average error-rate of 4%) and often limit the size of quantum programs that can be run reliably on these systems. JigSaw reduces the impact of these errors by using additional copies of the programs that measure only a subset of qubits with higher fidelity and using this information to boost the probabilities of the correct outcomes in the output distribution of the original program.

3.1 Motivation

Qubit measurement error rates can constrain the size of the largest program (in terms of the number of qubits) that can be run reliably on a near-term quantum computer [145]. To understand why qubit measurements are error-prone, one must understand the underlying process used to determine the state of a qubit. Superconducting qubits, similar to the devices from IBM and Google, are measured using a *dispersive qubit readout protocol*. In this protocol, a qubit is coupled to a measurement resonator whose resonance frequency experiences a shift depending on the state of the qubit and by measuring this shift, the state of the qubit is determined [146, 147, 148]. For superconducting devices, a signal corresponding to the state of a qubit is obtained when a readout pulse is applied to the qubit [149]. This signal is translated into a single-valued complex number to classify the state of the qubit as “0” or “1” using a measurement *discriminator*. Measurement errors can be attributed to a variety of factors.

1. The shift in resonance frequency is sensitive to noise and drifts in time.
2. Qubit measurement involves many complex instruments operating across multiple thermal domains that introduce errors due to *crosstalk* and unwanted couplings. The impact of crosstalk is not fully understood and is hard to minimize at the device level [150, 149].
3. Existing discriminators are inefficient and perform poorly for several quantum states [151].
4. Measurement operations are slow (takes about 4-5 μs on IBMQ hardware and 800 ns on Google devices [43]) and qubits often decay to the ground state during the readout process.

These factors limit the ability of existing quantum systems to perform fast and accurate qubit measurements at scale. On existing IBMQ and Google hardware, measurement operations tend to be the dominant sources of errors, with median error rates between 2.76% to 7.1%, and worst-case error rates as high as 11.7% to 22.2%.

Measurement operations limit the fidelity of large programs mainly due to two reasons:

3.1.1 Measurement Crosstalk

Measurement operations are more vulnerable to errors when a larger number of qubits are simultaneously measured due to measurement crosstalk. For example, the average error rate of simultaneous measurements is 1.26x higher than isolated measurements on Google Sycamore, as shown in Table 3.1. Our experiments on IBMQ hardware show similar trends where simultaneously measuring a larger number of qubits can significantly increase the measurement error-rates. Furthermore, our experiments show that the impact of such crosstalk depends on the physical qubit and its state and thus, is hard to characterize.

Table 3.1: Measurement Errors on Google Sycamore [152]

Measurement Mode	Measurement Error Rates (%)			
	Min	Average	Median	Max
Isolated	2.60	6.14	5.70	11.7
Simultaneous	3.30	7.73	7.10	20.9

3.1.2 Impact of Spatial Variation

Noise-aware compilers [48, 47] account for the error characteristics of the underlying quantum hardware and avoid mapping program qubits onto hardware qubits with worst-case errors. However, while this works very efficiently for small programs, spatial variation in measurement error rates often force compilers to map program qubits on unreliable physical qubits, as programs grow in size, because the qubits with the lowest error rates are not spatial neighbors, as shown in Figure 3.1. For example, it is not possible to map any program with more than six qubits on the 27-qubit IBMQ-Toronto without using a physical qubit with more than 2.7% measurement error-rate (the median error-rate). Further, the compiler is forced to use physical qubits A and B with more than 20% measurement error rate for programs with sixteen and twenty-one qubits or more respectively.

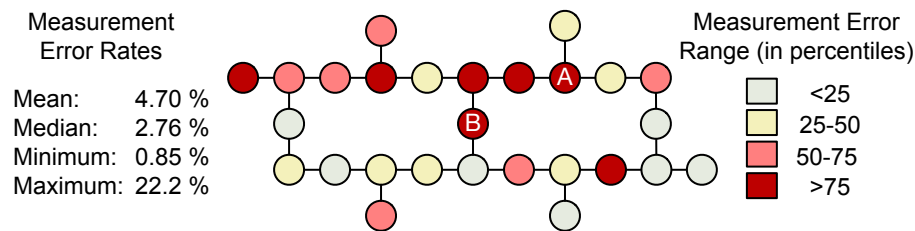


Figure 3.1: Spatial variation in measurement error rates of qubits on IBMQ-Toronto.

3.2 Key Insights of Proposed Solution

By default, quantum programs measure all the qubits in each trial, subjecting all the trials to the accumulated measurement errors from all the qubits. JigSaw reduces the impact of measurement errors on the fidelity of NISQ programs by:

1. Reducing the number of measurement operations by performing some trials with measurements only on a subset of qubits and effectively lowering the impact of crosstalk.
2. Remapping to ensure that the subset measurements are performed on the qubits with the lowest error rates. This gets us an effective measurement error rate closer to the minimum.

3.3 Challenges in Measurement Subsetting: Correlation versus Fidelity Trade-off

: Quantum computers obtain their exponential power by creating and manipulating highly correlated states. To measure this correlated state, a quantum program measures all the qubits in each trial. If there was no correlation, and each qubit had an independent probability of being in the 0 or 1 state, then one could simply split the trials into N groups (one group for each qubit), obtain the independent probability of being in 0 or 1 state for each qubit, and then obtain the probability distribution over all the qubits through multiplication. While this approach has high fidelity for each trial (only one measurement), it captures zero correlation between the qubits. Measuring a subset of qubits (larger than a single qubit but not all qubits) captures some correlation (within the measured qubits) but the correlations between these marginal distributions remain unknown, and therefore, multiplication or a tensor product may not yield the correct output distribution.

Thus, measuring all the qubits provides full correlation (but low fidelity) and fewer measurements provide higher fidelity (but weaker correlation). Ideally, we want both full correlation and high fidelity. Our proposed design, JigSaw improves application fidelity by retaining the global correlation of the original program, while simultaneously benefiting from the higher fidelity obtained from fewer measurements.

3.4 Overview of Proposed Design: JigSaw

Figure 3.2 shows the overview of JigSaw. JigSaw executes a program in two modes. First, JigSaw executes half of the trials in the *global-mode* in which a program is executed in its entirety and all the qubits are measured to obtain the global probability mass function (PMF).¹ Second, for the remaining trials, JigSaw runs additional copies of the program or *Circuits with Partial Measurements (CPM)* that measure fewer qubits in the *subset-mode*, to obtain more accurate marginal or local-PMFs over the *subset* of qubits being measured.

¹We use Probability Mass Function (PMF) for the results of program execution because these include discrete and not continuous values.

However, CPM alone cannot be used to infer the output PMF of a program without information about the correlations between these local-PMFs. To address this challenge, JigSaw employs a post-processing or *reconstruction* step that updates the global-PMF using the local-PMFs. This enables JigSaw to improve the application fidelity while simultaneously retaining the global correlation without requiring any additional trials.

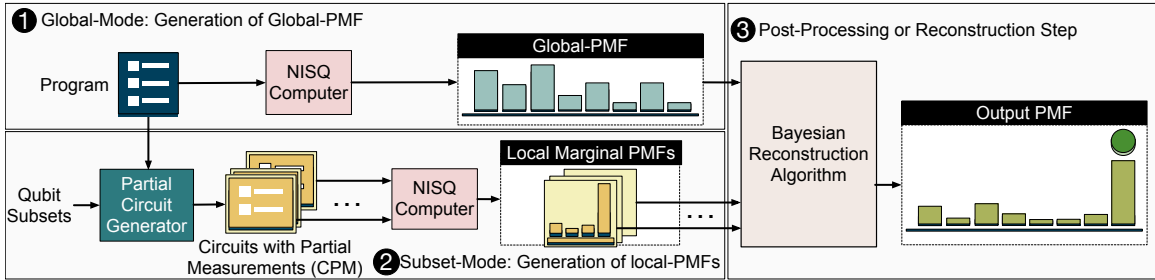


Figure 3.2: Overview of JigSaw. JigSaw runs the program in two modes: global-mode (all program qubits are measured) and subset-mode (only a subset of qubits are measured), and then uses the local-PMFs generated in subset-mode to update the global-PMF obtained during the execution in global-mode.

3.5 Global-Mode: Generation of Global-PMF

In this mode, JigSaw executes the program and measures all the qubits to produce a global-PMF. This is identical to the baseline and is done for half of the trials. We use Noise-Aware SABRE [82] for compilation to obtain a global-PMF with high fidelity. A NISQ compiler maps the logical qubits of a program onto the physical qubits of the hardware and generates a schedule by translating the high-level instructions into low-level machine-specific operations. To overcome limited connectivity on NISQ hardware, compilers also insert SWAP instructions to bring two non-adjacent qubits next to each other, so that CNOT operations can be performed between them. Noise-Aware SABRE accounts for the hardware error characteristics and generates a schedule that maximizes the *Expected Probability of Success* (EPS) [153]. EPS is the expected probability of successfully executing each gate and measurement operation in a schedule and is computed at compilation time by using the error rates obtained from the daily calibration report of the quantum hardware.

3.6 Subset-Mode: Generation of Local-PMFs

In the subset-mode, JigSaw runs several Circuits with Partial Measurements (CPM), for the remaining half of the trials which are equally distributed between the CPM. The next subsection discusses how CPM are generated and optimized for greater fidelity.

3.6.1 Circuits with Partial Measurements

A CPM is identical to the original program, except that it measures only a subset of qubits. CPM produces high-fidelity local-PMFs over the qubits measured. For example, Figure 3.3(a) shows a CPM of a 4-qubit Bernstein Vazirani program that measures 2 qubits, Q_0 and Q_1 . The key parameter in JigSaw is the number of qubits measured in a CPM and is called the *subset size*. The default JigSaw design uses CPM that measure 2 qubits. This is the smallest possible subset size that captures some correlation while performing the fewest possible measurements. Measuring only one qubit in a CPM captures zero correlation and therefore, is not used. By default, a sliding window method is used to generate the CPM so that there are N unique CPM for an N -qubit program. For example, for a 4-qubit program with qubits q_0, q_1, q_2, q_3 , four CPM are generated, measuring the qubit subsets $(q_0, q_1), (q_1, q_2), (q_2, q_3),$ and (q_0, q_3) . Therefore, the number of CPM is the same as the number of qubits.

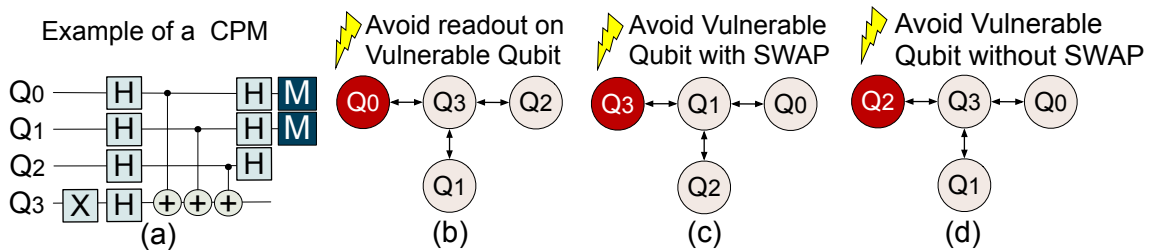


Figure 3.3: (a) Example of a CPM (b) Compiler avoids vulnerable qubit. Mapping that avoids vulnerable qubit (c) with extra SWAP (d) without incurring extra SWAP.

3.6.2 Optimizations to Improve Fidelity of the CPM

As JigSaw heavily relies on the accuracy of the local-PMFs, the compiler recompiles each CPM to maximize their fidelity.

Optimizing for Measurement Errors

JigSaw recompiles each CPM to exploit variability in measurement errors and ensure that measurements of desired program qubits are performed on the strongest physical qubits. The compiler avoids measurements on the physical qubit(s) with the highest readout error rate, referred to as *vulnerable qubit(s)*, in an allocation. For example, the compiler eliminates the allocation in Figure 3.3(b) for the CPM in Figure 3.3(a) to avoid readout of Q_0 on the vulnerable qubit and selects an alternate allocation.

Avoiding Extra SWAPs

To avoid measurement on the vulnerable qubit(s), the compiler often uses alternate qubit allocations, some of which may need extra SWAP instructions. However, JigSaw avoids such allocations that require extra SWAPs to avoid additional gate errors. For example, the compiler selects the allocation shown in Figure 3.3(d) over Figure 3.3(c) because the latter requires an extra SWAP (SWAP Q_0, Q_1). While in most cases, the compiler finds alternate mappings without incurring extra SWAPs, for cases where the compiler cannot find a mapping without inserting an extra SWAP, it picks the mapping that maximizes the expected probability of success.

The ability to find alternative mappings depends on device connectivity, spatial location of good qubits, and program characteristics. Our studies show most CPM can reuse qubit allocations chosen by the Ensemble of Diverse Mappings [88] policy. Also, we use SABRE, which has low latency.

3.7 Post-processing via Bayesian Method

JigSaw produces $(N+1)$ PMFs for a program with N qubits: one global-PMF and one local-PMF for each of the N CPMs. The post-processing step combines the higher fidelity local-PMFs from each CPM into the global-PMF. JigSaw uses a *Bayesian Reconstruction* algorithm, which is inspired by Bayesian updating in statistics, whereby a prior probability estimate is updated using additional information [154]. For JigSaw, the global-PMF (P) offers the prior probability estimate, whereas the set of marginals (M) obtained from the CPM provides the additional information. The term *marginal* is used to denote a set comprising of a subset of qubits measured in a CPM and the local-PMF produced by the CPM.

Consider an example of a 3-qubit program. Let $P = \{000 : a, 010 : b, 010 : c, 011 : d, 100 : e, 101 : f, 110 : g, 111 : h\}$ represent a generic global-PMF of a 3-qubit program (with qubits Q_2, Q_1, Q_0), where a to h are the probabilities of observing outcomes 000 to 111 respectively. Similarly, $m_0 = [\{00 : \alpha, 01 : \beta, 10 : \gamma, 11 : \delta\}, [1, 0]]$ refers to a generic marginal for a CPM measuring qubits $[Q_1, Q_0]$.

The steps for the Bayesian Reconstruction algorithm are described in ???. The algorithm uses each marginal to update the probabilities of each outcome in the global-PMF (P). It starts by searching for all outcomes in P associated with each outcome in a marginal, by evaluating the bits at the corresponding qubit positions. For example, for $(Q_1, Q_0) = 00$ in the marginal m_0 , the candidates in P are $(Q_2, Q_1, Q_0) = 000$ and 100 . For each marginal in M , the Bayesian Update function generates an updated PMF (P_{post}) by updating the probabilities of each outcome in P using the associated probabilities in the marginals. For example, the probabilities of 000 and 100, i.e., a and d respectively, are updated in proportion with α (corresponding to 00 in m_0). The algorithm produces a posterior output PMF (P_{out}) by adding all the intermediate PMFs (P_{post}) to the global-PMF (P). The algorithm is recursively called and terminates when the Hellinger Distance [155] between the output PMF prior to and post the function call does not change, implying the PMFs are similar.

Algorithm 1 Bayesian Reconstruction Algorithm

Input: (1) Global-PMF $P = \{B_x : pr_x\}$ where B_x is a n -bit outcome (2) Set of j Marginals $M = \{m_j\}$ where $m_j = [\{B_y : pr_y\}, \{i_0 \dots i_k\}]$ for k -bit outcome B_y

Output: PMF $P_{out} = \{B_x : P_x\}$, $P_x \in [0, 1]$

```
1: function BAYESIAN_UPDATE(P,m)
2:    $P_o = P$ 
3:   for each (entry  $B_y : pr_y$ ) in  $m$  do
4:     candidate = []
5:     for each  $B_x$  in  $P$  do
6:       // Obtain list of outcomes in  $P$ 
7:       outcome  $\leftarrow$  bits in  $B_x$  corresponding to qubits  $\{i_0 \dots i_k\}$ 
8:       candidate.append((outcome,pr_x))
9:       // Obtain Update Coefficients
10:      normalize candidate
11:    end for
12:    for each outcome in candidate do
13:      // Obtain posterior probabilities
14:       $P_o[outcome] = \frac{candidate[outcome] \times pr_y}{(1-pr_y)}$ 
15:    end for
16:  end for
17:  normalize  $P_o$ 
18:  return  $P_o$ 
19: end function
20: // Perform Bayesian Reconstruction
21: function BAYESIAN_RECONSTRUCTION(P,M)
22:    $P_{out} = P$ 
23:   for each  $m_j$  in  $M$  do
24:      $P_{post} = \text{Bayesian\_Update}(P, m_j)$ 
25:      $P_{out} = P_{post} + P_{out}$ 
26:   end for
27:   normalize  $P_{out}$ 
28:   return  $P_{out}$ 
29: end function
```

The steps involved in the Bayesian update process is explained next using a quantitative example. Figure 3.4 shows the update sequence using experimental data for a 3-qubit program whose global-PMF is denoted by P (in the order Q_2, Q_1, Q_0) using a marginal m_0 from a CPM that measures the qubits (Q_1, Q_0) . Note that for readability, Figure 3.4 only shows the steps for marginal $[Q_1, Q_0]$, whereas the output PMF shown is obtained from recursive updates with additional marginals.

Below is an outline of the steps involved in the reconstruction process.

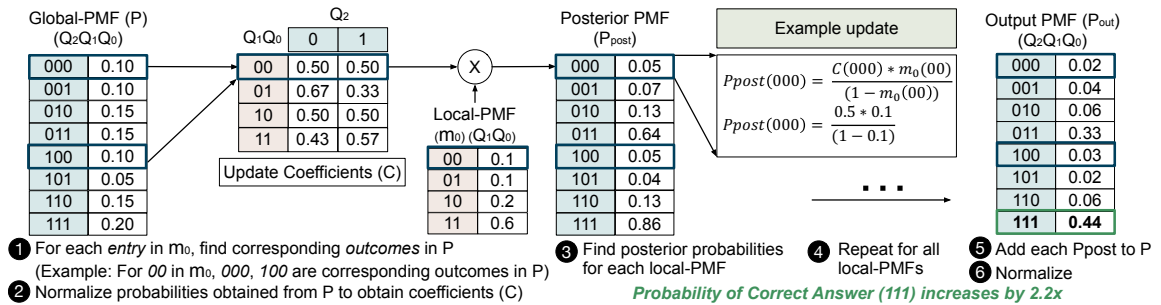


Figure 3.4: Steps in Bayesian Reconstruction for a 3-qubit program (qubits Q₂, Q₁, Q₀) using a CPM measuring Q₁, Q₀.

- **Step 1** : The algorithm searches for all candidate outcomes in P for each entry in marginal m₀ by evaluating the bits at the corresponding qubit positions. For example, 000 and 100 are the candidate outcomes in P for 00 in m₀, obtained by matching the values for (Q₁, Q₀) in P and m₀.
- **Step 2** : Next, the function computes the Update Coefficients C for each of the outcomes in P by normalizing their respective probabilities of occurrence in P.
- **Step 3** : Next, the algorithm computes the posterior probabilities for each observed outcome in P by scaling them using the corresponding probabilities observed in m₀. Figure Figure 3.4 explicitly shows the computation for obtaining the posterior probability of outcome 000 (P_{post}[000]) using the Update Coefficient C[000] and marginal information m₀[00].
- **Step 4** : The algorithm repeats Steps 1-3 to generate an intermediate posterior PMF (P_{post}) for each marginal.
- **Step 5** : Each P_{post} is added to the global-PMF (P).
- **Step 6** : The final output PMF (P_{out}) is obtained by normalizing the probabilities.

As the Bayesian updates for each CPM are performed independently and the intermediate PMFs are added in the final step, the order of updates does not matter.

3.8 Multi-Layer JigSaw (JigSaw-M)

The performance of JigSaw saturates when CPM of the same subset size that does not offer any incremental information is used. More unique CPM can be designed using different subset sizes and JigSaw can be enhanced even further to improve application fidelity.

3.8.1 Global and Subset-Modes for JigSaw-M

The global mode for JigSaw-M is identical to JigSaw which generates the global-PMF by executing the program and measuring all the qubits. The subset-mode for JigSaw-M executes CPM of non-uniform subset sizes s , such that $s_{\min} \leq s \leq s_{\max}$, where s_{\max} and s_{\min} are the maximum and minimum subset sizes respectively. The default implementation uses a sliding window method to generate unique CPM for each subset size, similar to JigSaw, but other methods may be used as well. If CPM of \mathbb{S} different sizes are used, JigSaw-M produces $(\mathbb{S}N+1)$ PMFs for an N -qubit program, one global-PMF and N local-PMFs for each subset size. By default, the design uses CPM of sizes 2 to 5.

3.8.2 Adapting Reconstruction for JigSaw-M

JigSaw-M comprises of CPM of \mathbb{S} different sizes. There is a choice between which CPM must be used first to update the global-PMF. However, there exists a *trade-off* between the fidelity of a CPM and the correlation it can capture depending on the number of qubits it measures. A smaller CPM offers higher fidelity due to fewer measurement errors but captures lower correlation, whereas a larger CPM offers higher correlation, but has lower fidelity as it is more prone to measurement errors. Thus, for JigSaw-M, the reconstruction algorithm first updates the global-PMF using the CPM of the highest size (s_{\max}), limiting the loss of global correlation. The updated PMF (P_s) is then further enhanced using CPM of the next higher size. The process is repeated until the smallest CPM are used. This top-down ordering maximally preserves the global correlation, while improving fidelity.

3.9 Evaluation Methodology

3.9.1 Quantum Hardware Platforms

For all evaluations, three different quantum computers from IBM: 27-qubit *IBMQ-Toronto*, 27-qubit *IBMQ-Paris*, and 65-qubit *IBMQ-Manhattan* are used.

3.9.2 Baseline Compiler

For the baseline, Noise-Aware SABRE [82] is used to compile and map the program onto the physical qubits with the lowest error rates. JigSaw is also evaluated against the Ensemble of Diverse Mappings (EDM) policy that runs independent copies of the program on different groups of physical qubits [88] and improves the ability to infer the correct answer. Note that while Noise-Aware SABRE is used for the results presented in this dissertation, other noise-adaptive compilers [48, 47] may be used too.

3.9.3 Benchmarks

The benchmarks used for evaluations are described in Table 5.1. The type and size of benchmarks are derived from prior works [82, 47, 48, 88]. Note that the IBMQ machines used for evaluations have a Quantum Volume [102] of 32, which means that square circuits of only up to size 5 can be run reliably and therefore, the size of the benchmarks used is already much larger even if they do not use all the qubits that are present on the machine. To generate the *Quantum Approximate Optimization Algorithm* circuits, IBM's Qiskit AQUA [156] VQE tool-chain is used with the optimal parameters for each circuit.

3.9.4 Experimental Setup: Number of Trials

For each program, between 32K to 256K trials for the baseline are used depending on the program size. This represents the highest fidelity that can be obtained by increasing trials alone and serves as a strong baseline as more trials do not improve fidelity (mainly

Table 3.2: Details of NISQ benchmarks

Name	Algorithm	#Qubits (n)	1Q Gates	2Q Gates
BV-n	Bernstein-Vazirani [157]	6	$2(n+1)$	n
Graycode-n	Graycode Decoder	18	$n/2$	$(n-1)$
QAOA-n (p=1)	Maxcut with p=1 [38]	8	4n	$(n-1)$
QAOA-n (p=2)	Maxcut with p=2	10, 14	6n	$2(n-1)$
QAOA-n (p=4)	Maxcut with p=4	10, 12	10n	$4(n-1)$
GHZ-n	Greenberger-Horne-Zeilinger [54]	14	1	$(n-1)$
Ising-n	Ising model [158]	10	$n(4.5n-2)$	$n(n-1)$

due to correlated errors). Figure 3.5 shows the Probability of Successful Trial (PST) of several GHZ and QAOA benchmarks executed on IBMQ-Paris for up to 4 million trials. We observe similar behavior for other workloads and machines.

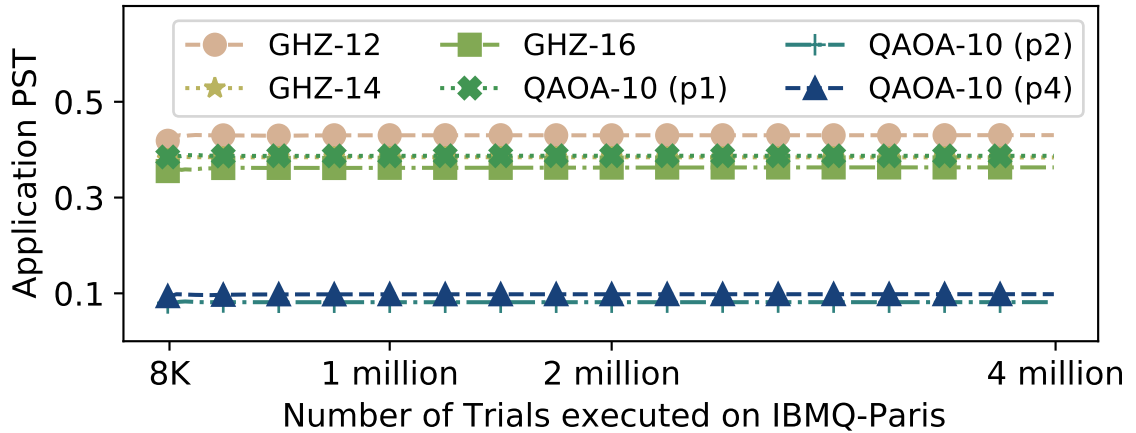


Figure 3.5: Impact of Number of Trials on Probability of Successful Trial of Applications.

For EDM [88], an ensemble of four mappings is used with the trials equally divided among the mappings. For JigSaw and JigSaw-M, the trials are equally split between the global-mode and the subset-mode. In the subset-mode the trials are equally split between all the CPM for both JigSaw and JigSaw-M. Therefore, JigSaw, JigSaw-M, and EDM all use the same number of trials as the baseline. All experiments are performed within the

same calibration cycle but similar results are observed across different calibration cycles. The equal split for trials is used for the sake of simplicity because the fidelity saturates for the number of trials used in our evaluation. If the number of trials is limited, the split between global-mode and subset-mode can be tuned to possibly obtain even larger gains.

3.9.5 Figure-of-Merit

For evaluations, metrics from prior works are used whose details are discussed below:

Probability of Successful Trial (PST)

The *Probability of Successful Trial (PST)* [47, 88, 48] is the ratio of the number of trials with the correct output to the total number of trials, as described in (Equation 5.6).

$$PST = \frac{\text{Number of trials with the correct output}}{\text{Total number of trials}} \quad (3.1)$$

Fidelity

The *Fidelity* of a program is obtained by measuring the Total Variation Distance (TVD) [159] between the output distributions on a noise-free quantum computer (P) and real hardware (Q). TVD allows us to measure the fidelity of quantum programs [160] whose output can be a probability distribution with more than one correct answer. The Fidelity ranges between 0 and 1, where 1 represents two identical distributions and 0 means completely dissimilar distributions, as shown in (Equation 5.4). While TVD is used here, other metrics such as Hellinger Distance [155] or Kullback–Leibler divergence [161] may be used too.

$$TVD(P, Q) = \sum_{i=1}^k || P_i - Q_i || \quad (3.2)$$

$$Fidelity(P, Q) = 1 - TVD(P, Q)$$

A higher Probability of Successful Trial (PST) and Fidelity are desirable.

3.10 Final Evaluations

In this Section, the impact of JigSaw on application fidelity is discussed.

3.10.1 Results for Probability of Successful Trial

Figure 3.6 shows the improvement in the Probability of Successful Trial (PST) using JigSaw and JigSaw-M. Evaluations using three different quantum hardware from IBM and tens of quantum benchmarks show that JigSaw the PST by 2.91x on average and by up-to 7.87x compared to the baseline. JigSaw-M improves PST by 3.65x on average and up-to 8.42x compared to the baseline. Compared to JigSaw, JigSaw-M improves the PST of applications by 1.26x on average and up-to 1.72x.

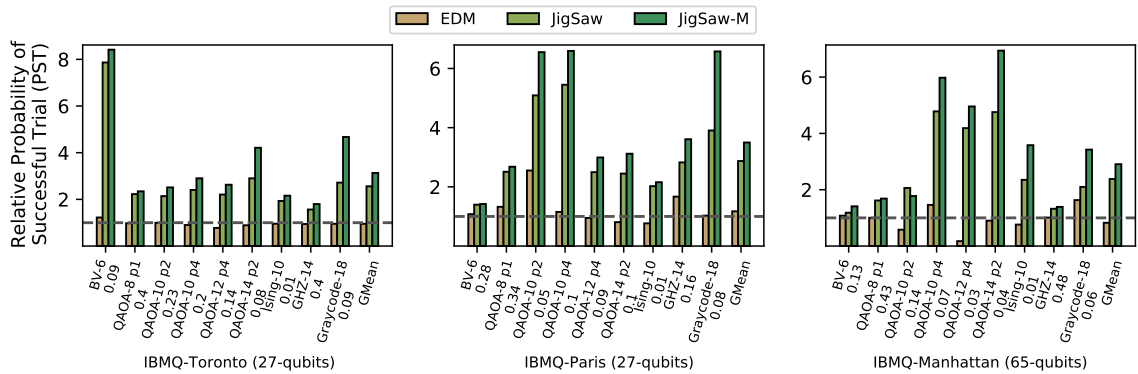


Figure 3.6: Probability of Successful Trial (PST) from JigSaw and JigSaw-M relative to baseline and comparison with prior work Ensemble of Diverse Mappings (EDM) [88]. Number below the label shows absolute PST for the benchmark.

3.10.2 Results for Fidelity

Table 3.3 compares the Fidelity for EDM, JigSaw, and JigSaw-M relative to the baseline. For instance, on IBMQ-Toronto, EDM degrades Fidelity by 0.96x on an average, whereas JigSaw and JigSaw-M improve it by 2.17x and 2.54x respectively. Overall, JigSaw improves the Fidelity on average by 2.12x, whereas JigSaw-M improves the Fidelity on average by 2.47x and by up-to 8.41x compared to the baseline. Therefore, the output

distributions of programs obtained from JigSaw and JigSaw-M have higher Fidelity and are significantly more similar to the distributions obtained on a noise-free quantum computer.

Table 3.3: Fidelity obtained from EDM, JigSaw, and JigSaw-M relative to the baseline

IBMQ (Hardware)	EDM			JigSaw			JigSaw-M		
	Min	Max	Avg	Min	Max	Avg	Min	Max	Avg
Toronto	0.78	1.22	0.96	1.07	7.86	2.17	1.07	8.41	2.54
Paris	0.77	2.54	1.19	1.09	5.07	2.33	1.11	6.52	2.77
Manhattan	0.43	1.62	0.93	1.18	3.26	1.84	1.28	4.43	2.10

3.10.3 Impact of Number of Circuits with Partial Measurements and Selection Method

The default JigSaw design uses a sliding window method to generate a handle of unique CPM. To understand the impact of the number of CPM and selection method, we perform an empirical study using a 12-qubit QAOA program on IBMQ-Paris.

Sensitivity to Number of CPM

The total number of possible CPM (measuring two qubits) for a Q qubit program is ${}^Q C_2$. To understand the impact of the number of CPM (N) on its effectiveness, JigSaw randomly generates N circuits with partial measurements of subset size 2 out of all the 66 possibilities (${}^{12} C_2 = 66$) and uses these N local-PMFs to update the global-PMF. The process is repeated hundreds of times for each N and Figure 3.7(a) shows the average improvement in Application PST from JigSaw as N is increased. The performance of JigSaw saturates when additional CPM do not offer incremental information. Thus, only a few unique CPM are sufficient for JigSaw to be effective. Further, to obtain a greater number of unique CPM, JigSaw-M generates CPM of non-uniform subset sizes.

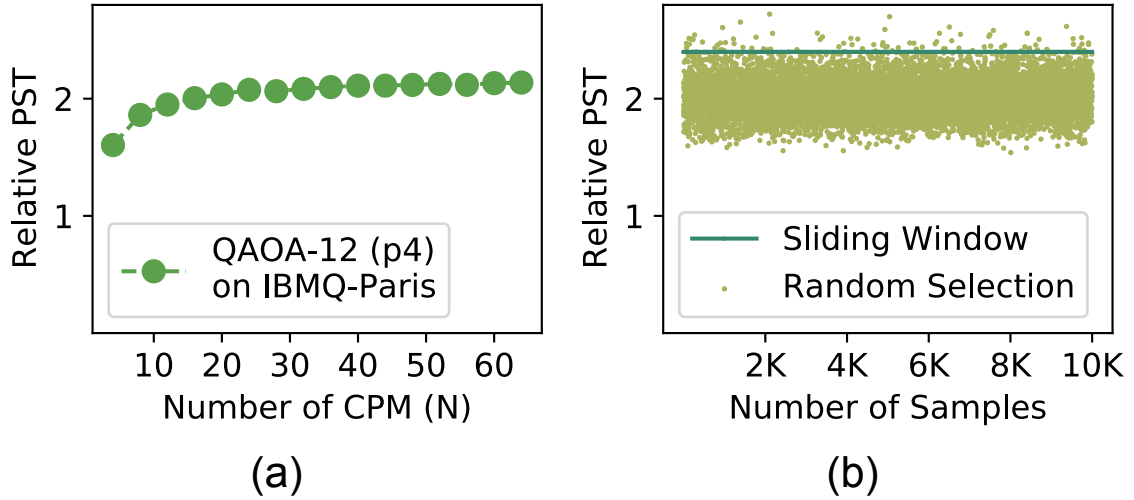


Figure 3.7: Impact of (a) Number of CPM and (b) CPM Selection Method on the Performance of JigSaw.

Sensitivity to CPM Selection Method

To study the impact of the CPM selection method, JigSaw randomly selects a group of CPM of subset size 2 from all the 66 possibilities while ensuring that each program qubit is measured in a CPM at least once. As there are 12 qubits in the program JigSaw selects 12 random CPM each time and the process is repeated 10,000 times. Figure 3.7(b) shows the relative improvement in PST for this study, and we observe that we get similar results irrespective of the CPM. Thus, although our default design uses a sliding window method, JigSaw is equally effective even if any other technique to generate the CPM is used.

3.10.4 Impact of Recompilation

JigSaw mainly benefits from measurement subsetting and recompilation. By recompiling each CPM, the effective measurement error-rates for CPM are close to the best-case qubits rather than close to the average-case qubits (which is the case for the global-mode and the baseline). For example, Figure 3.8 shows that the probability of correctly measuring a qubit in a CPM increases by up-to 3.25x compared to the baseline for a BV-6 benchmark on IBMQ-Toronto. Note that the probability of correctly measuring a qubit is computed from

the set of outcomes where the particular qubit is correctly measured, even if the overall outcome is erroneous and does not represent the correct answer. Thus, recompiling CPM can significantly enhance the effectiveness of JigSaw.

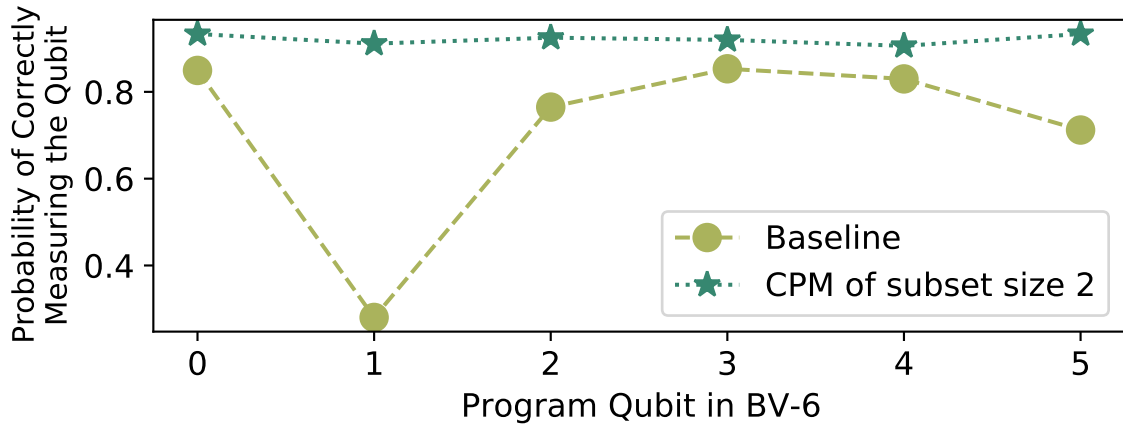


Figure 3.8: Probability of successfully measuring a qubit of a 6-qubit BV program on IBMQ-Toronto in the (a) baseline (b) in each CPM after recompilation.

Figure 3.9 shows the Mean PST from JigSaw without recompilation (subsetting only), JigSaw with recompilation, and JigSaw-M relative to the baseline. Without recompilation, JigSaw improves the PST by 1.92x on average and up-to 3.26x, whereas with recompilation JigSaw improves the PST by 2.91x on average and up-to 7.8x compared to the baseline. With recompilation, JigSaw-M improves the PST by 3.65x on average and by up-to 8.4x.

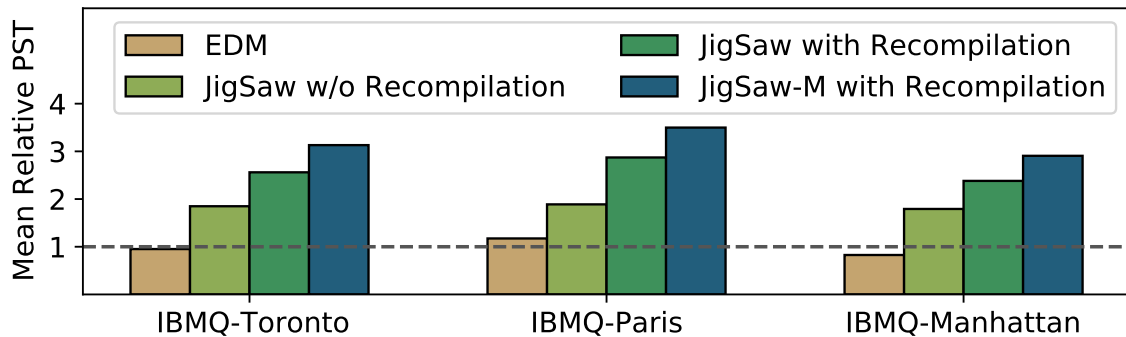


Figure 3.9: Comparison of Mean PST relative to the Baseline.

3.11 Scalability of Bayesian Reconstruction Algorithm

The applicability of reconstruction algorithms is often limited by their memory and time complexity. For example, tensor-product based algorithms such as those used in typical circuit-cutting frameworks are extremely hard to scale due to their exponential complexity. Therefore, the scalability of our proposed Bayesian Reconstruction using an analytical model is studied, which is described next.

3.11.1 Insight: Bounded by Observations

JigSaw limits the complexity of the reconstruction algorithm by storing and updating only the non-zero entries generated in the global-mode. Although the number of possible non-zero entries in the global-PMF can scale exponentially with the program size, the actual number of entries observed is much lower and is *limited by the number of trials*, particularly for large programs. For example, there are 2^{100} possible outcomes for a 100-qubit program, and assuming the circuit produces a uniform distribution, it would require a minimum of 2^{100} or 10^{30} trials to observe each of these outcomes at least once, which is impractical. In practice, we may be able to execute at-most a few million trials, because the time to execute the trials still increases linearly with the number of trials. Moreover, practical quantum algorithms are designed to produce output distributions with relatively low variance and bounded possible outcomes. For example, Table 3.4 shows that a Graycode-18 benchmark produces only up to 18.5K unique outcomes when executed for 512K trials, even though 256K ($=2^{18}$) outcomes are possible. Note that the total number of unique outcomes depends on the error characteristics of the quantum hardware. A machine with very high error rates tends to produce more unique outcomes compared to a machine with lower error-rates. It also depends on the size of the program and number of trials executed. In the worst-case, each trial produces a unique outcome. We bound the complexity of JigSaw reconstruction by focusing only on the outcomes observed rather than all the possible outcomes.

Table 3.4: Number of Outcomes in the Global-PMF for a Graycode-18 Benchmark on IBMQ Hardware

Outcomes	IBMQ-Toronto	IBMQ-Paris	IBMQ-Manhattan
Observed (Obs)	17.0 K	17.3 K	18.5 K
Maximum (Max)	256 K	256 K	256 K
Ratio (Obs/Max)	6.6 %	6.8 %	7.2 %

3.11.2 Memory Complexity

JigSaw only stores non-zero entries in the global, output, local PMFs, and an intermediate PMF for each CPM. We assume N is the number of CPM. For simplicity, the analysis assumes the global-mode and each CPM in subset-mode uses T trials.²

Global, Intermediate, and Output PMFs

Each global-PMF entry comprises of an n -bit string outcome and its probability of occurrence, as shown in Figure 3.10(a). Assume that there are ϵT entries in the global-PMF, where $0 < \epsilon \leq 1$. As JigSaw only updates the probabilities of the global-PMF entries, the intermediate and output PMFs are only required to store the updated probabilities, as shown in Figure 3.10(b). Hence, the global-PMF requires $(n + 8)$ bytes per entry, whereas the intermediate and output PMFs each require 8 bytes per entry.

Experiments on IBMQ systems show that $\epsilon \ll 1$ does not change rapidly with increasing trials. For example, Figure 3.11 shows the number of unique outcomes and ϵ when some GHZ and QAOA benchmarks are executed for 4 million trials on IBMQ-Paris. Note that the value of ϵ depends on the error-rates of the machine, the size of the program (determines the state space of the outcomes), and the number of trials. In the worst-case, when each trial produces a unique outcome, $\epsilon = 1$.

²For simplicity, we assume up-to 1 million trials each for the global-mode and each CPM, which is a severely pessimistic assumption. In practice, the trials are split between a large number of CPM, so the storage and timing complexity gets reduced further.

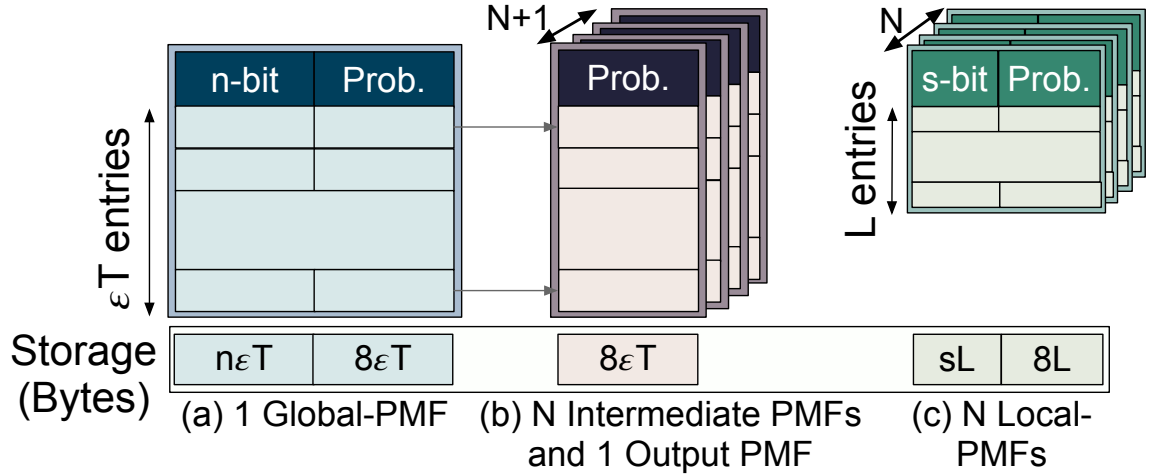


Figure 3.10: Memory required to store (a) the global, (b) N intermediate and 1 output PMF, and (c) N local-PMFs.

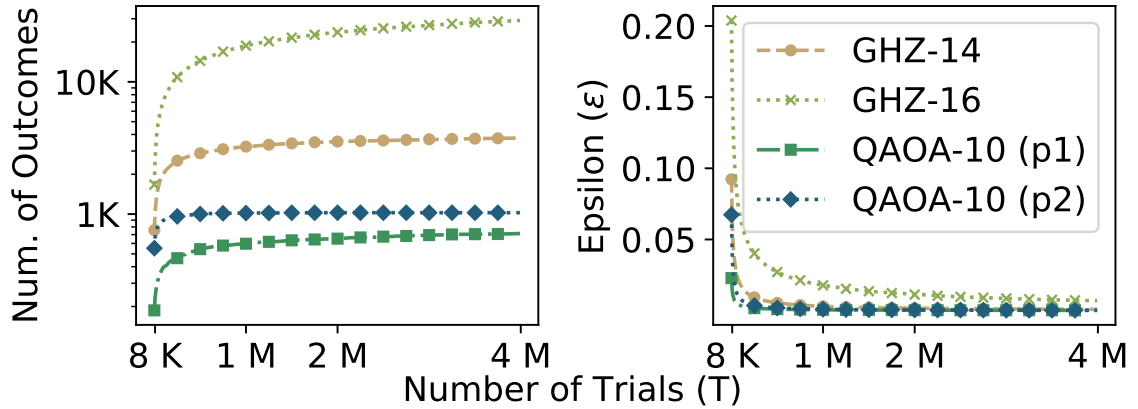


Figure 3.11: (a) Number of Global-PMF entries and (b) Epsilon (ϵ) with increasing trials (T) on IBMQ-Paris.

Local-PMFs

A local-PMF for a CPM of subset size \sim consists of $L = \min(2^\sim, \delta T)$ entries, where $0 < \delta \leq 1$, and requires $L(\sim + 8)$ bytes, as shown in Figure 3.10(c). To minimize errors on CPM, s must be small such as 2 in the default JigSaw design. For such small \sim , a local-PMF consists of all possible 2^\sim entries. But for large \sim , $L \ll 2^\sim$ and is denoted by δT . For example, local-PMFs of size 2 and 10 for a GHZ-14 program on IBMQ-Toronto contain 4 and 297 entries respectively, even though 1024 entries are possible for $\sim = 10$.

JigSaw stores one Global-PMF, N intermediate PMFs, one output PMF and N local-PMFs. JigSaw-M stores one Global-PMF, N intermediate PMFs, one output PMF and $\mathbb{S}N$ local-PMFs where \mathbb{S} subset sizes are used. Although JigSaw-M uses more CPM, since it employs hierarchical reconstruction from the highest to the lowest subset size, only N intermediate PMFs are required which are reused across reconstruction rounds. Thus, the total memory capacity (in bytes) is given by (Equation 3.3). The memory complexity for JigSaw is obtained for $\mathbb{S} = 1$ as it uses CPM of a single subset size only.

$$\text{Memory} = \{n + 8(2 + N)\}\epsilon T + L(\sim + 8)\mathbb{S}N \quad (3.3)$$

3.11.3 Time Complexity

JigSaw updates each Global-PMF entry for each entry in a local-PMF. Obtaining the update coefficients require ϵT operations and the update itself requires $3\epsilon T$ operations per local-PMF. Assuming JigSaw uses N CPM, it requires $4\epsilon NT$ operations. Similarly, JigSaw-M requires $4\epsilon \mathbb{S}NT$ operations. As JigSaw only stores and updates non-zero PMF entries, which is much lower than the maximum possible, and is limited by the number of trials, the time-complexity increases linearly with the number of trials and qubits.

3.11.4 Results for Scalability Analysis

Table 3.5 shows the memory and number of operations required for programs of different input sizes n , values of ϵ , δ , and number of trials T . To obtain the typical complexity, we use $T = 1$ million and $\epsilon = \delta = 0.05$ (from Figure 3.11), whereas we use $\epsilon = \delta = 1$ to obtain the upper bound. For JigSaw, we assume CPM of subset size 5 and the number of CPM (N) to be the same as the number of qubits in the program (as our default design). For JigSaw-M, we assume sizes 5,10,15, and 20. We observe that the storage and time complexity is linear with the number of trials and qubits in the program, making JigSaw applicable to programs with hundreds of qubits.

Table 3.5: Scalability Analysis of JigSaw and JigSaw-M: Memory (in GB) and Number of Operations (in million)

Qubits (n)	$\epsilon = \delta$	Trials (T)	JigSaw		JigSaw-M	
			Mem	OPs	Mem	OPs
100	0.05	32K	0.01	0.66	0.02	2.64
		1024K	0.05	21.0	0.42	83.9
	1.0	32K	0.03	13.1	0.20	52.4
		1024K	0.96	419	3.97	1677
500	0.05	32K	0.01	3.28	0.1	13.12
		1024K	0.24	105	2.09	419
	1.0	32K	0.15	65.5	0.99	262
		1024K	4.74	2097	19.8	8388

3.12 A First Order Estimation of Number of Trials

For simplicity, by default, we execute the global-mode for half of the trials and the subset-mode for the remaining half. We also equally distribute the trials in the subset-mode between the CPM for both JigSaw and JigSaw-M. However, if the trials are severely limited, (1) the distribution of trials may be fine-tuned or (2) the subset-mode may be executed for a few thousands of extra trials (global-mode corresponds to the baseline). We perform an analysis of how the trials may be allocated for each CPM.

Let there be $N(= 2^n)$ possible outcomes for a program that measures n qubits. If p is the probability of observing an outcome and each of the N outcomes is equally likely to appear at the end of a trial, then $p = 1/N$. The probability that a given outcome has appeared at least once after t trials is then given by (Equation 3.4).

$$\mathbb{P} = [1 - (1 - p)^t] \tag{3.4}$$

If $t \approx \alpha N$ and N is large, \mathbb{P} may be approximated as (Equation 3.5).

$$\mathbb{P} = 1 - e^{-\alpha} \quad (3.5)$$

Thus, in order to obtain the given outcome at least once with probability \mathbb{P} , the number of trials required is given by (Equation 3.6).

$$t = -\ln(1 - \mathbb{P})N \quad (3.6)$$

Hence, the total number of trials required to observe every possible outcome at least once with probability \mathbb{P} is given by (Equation 3.7).

$$\text{Total number of Trials} = -\ln(1 - \mathbb{P})N^2 \quad (3.7)$$

We measure only 2 qubits in each CPM in the default JigSaw design and thus, only about 150 trials are required to ensure (with 99.99% probability) that we obtain each possible answer at-least one time. Similarly, as JigSaw-M uses CPM of different sizes, the estimated number of trials would still range within a few thousands.

Key Takeaways

- Measurement errors limit the fidelity of quantum programs.
- JigSaw reduces the impact of measurement errors by running a program in two modes—the global mode that measures all the qubits and the subset mode that measures subsets of qubits using additional copies of the program.
- JigSaw uses a Bayesian reconstruction algorithm to combine the high fidelity marginal information from the subset mode into the output distribution from the global mode.
- JigSaw further improves the fidelity by performing the subset measurements on qubits with the lowest measurement errors via recompilation.

3.13 Evolution of the Measurement Error Mitigation Landscape and Follow-up Works

Software-based measurement error mitigation is a promising technique to reduce the impact of readout errors, which tend to be the dominant sources of errors in emerging quantum systems. Most importantly, although error-rates are overall decreasing across different generations of quantum computers, measurement errors are reducing at a much slower rate than gate errors. Thus, measurement error mitigation remains crucial to improve the fidelity of quantum programs. IBM’s matrix-based method is one of the earliest methods in this space [50]. Although very effective, its applicability is limited by the exponential complexity of the post-processing step. The Flip-and-Measure method [96] on the other hand is a relatively simpler technique that transforms a more error-prone quantum state to a less susceptible one [93, 96] using single qubit gates and does not require any overheads for post-processing. However, its overall performance is still bottlenecked by the effective measurement error-rate of all the qubits. JigSaw overcomes this bottleneck by performing subset measurements. Moreover, our evaluations show that JigSaw can be combined with other error mitigation schemes for enhanced performance. For example, the subset mode output distributions can be first rectified using IBM’s matrix-based method and then used for JigSaw’s Bayesian Reconstruction algorithm. One of the key insights of JigSaw is using the quantum hardware to generate a global output distribution which guides the post-processing step and reduces its complexity. Similar insights later appeared for the recent matrix-based error mitigation schemes from IBM [162] and are currently integrated in IBM’s Qiskit software tool-chain [163]. Recently, Dangwal et al. proposed VarSaw [164], which tailors JigSaw for variational quantum algorithms by reducing the total number of subset measurements required to be performed across the iterations in the variational loop. In the future, we expect more effective, low-overhead, and scalable solutions to be developed to further reduce the impact of measurement errors on the fidelity of applications executed on emerging quantum hardware.

CHAPTER 4

EFFICIENT TECHNIQUES FOR MITIGATING IDLING ERRORS ON EMERGING QUANTUM COMPUTERS

This chapter of the dissertation presents *Adaptive Dynamical Decoupling (ADAPT)* [165], a robust error mitigation technique to reduce the impact of idling errors on the fidelity of applications of emerging quantum computers. *Dynamical decoupling (DD)* is a well-known device-level technique to reduce idling errors. ADAPT presents solutions that enable robust usage of DD at the application level.

4.1 Motivation

Idling errors are common in quantum hardware. For example, qubits naturally decohere even if they are not undergoing any operations. Moreover, undesirable crosstalk can cause ongoing to affect the state of neighboring (spectator) idle qubits. The characteristics of quantum programs cause many of the qubits to remain idle for a significant period of time during program execution. There are three key reasons for qubits to remain idle: (1) limited parallelism, as quantum programs get decomposed into a sequence of instructions with data dependency (2) high latency of two-qubit gates compared to single-qubit gates, and (3) additional data movement or SWAP operations. For example, let us take a 4-qubit Bernstein-Vazirani (BV) circuit shown in Figure 4.1(a). Due to low parallelism, the CNOT gates of this circuit must be scheduled serially. Consequently, qubit Q0 remains idle when CNOTs \textcircled{B} and \textcircled{C} are executed. Although existing compilers minimize idle times by scheduling instructions *as late as possible* [90], this optimization is not feasible for all qubits as the computation must make forward progress. For example, late initialization causes Q2 to not experience any idle time but cannot optimize Q1 from remaining idle during the execution of CNOT \textcircled{C} . The long latency of the CNOT gates exacerbates the

idle times. Even if a program can orchestrate parallel operations on different qubits, qubits with single-qubit operations (10x faster) finish execution earlier than qubits with two-qubit gate operations and remain idle. Furthermore, CNOT gates on the same hardware incur different latencies. For example, the worst-case CNOT gate latency on IBMQ-Toronto is 1.95x the average latency. Consequently, parallel CNOT gates with variable latencies finish at different times. Finally, compilers insert SWAP instructions to overcome limited device connectivity causing serialization and long idle periods. For example, Figure 4.1(b) shows that the idle time of qubit Q0 for different BV circuits is about an order of magnitude higher on IBMQ-Toronto compared to a machine with similar error rates but all-to-all connectivity.

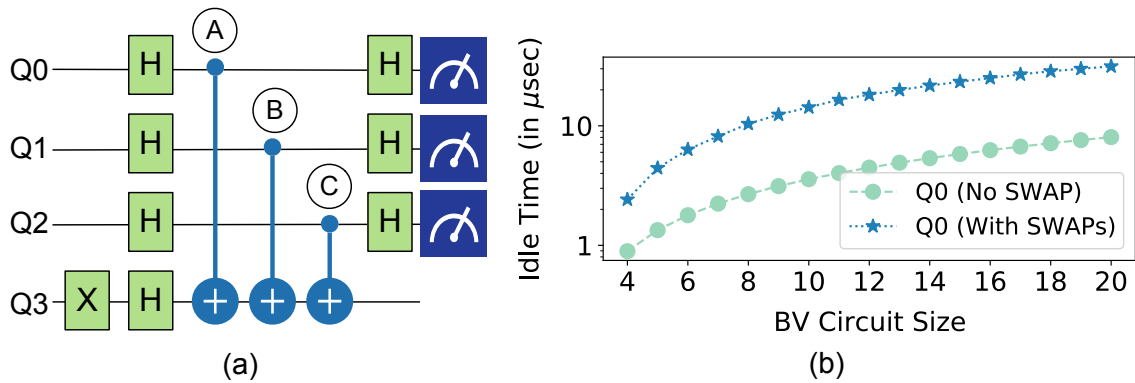


Figure 4.1: (a) 4-qubit Bernstein-Vazirani circuit (b) Impact of SWAPs on the idle time of Q0 when BV circuits with increasing sizes are executed on IBMQ-Toronto.

Table 4.1: Idling Times for Programs on IBMQ-Rome

Workload Name	Program Latency	Idle Fraction (%)					Fidelity	
		Q ₀	Q ₁	Q ₂	Q ₃	Q ₄	No DD	DD on all
QFT-5	13.1 μseconds	92	38	17	33	63	0.18	0.41
QAOA-5	2.50 μseconds	82	37	35	63	79	0.64	0.80
Adder	9.90 μseconds	41	42	9	42	75	0.40	0.37

Table 4.1 shows the program latency (post-compilation) and the percentage of time for which each qubit in three different five-qubit programs remains idle on IBMQ-Rome. Note that across these workloads, the qubits remain idle on an average more than 50% of the time, and as much as 92%.

4.2 Challenges in Reducing Idling Errors

To minimize the impact of idling errors, experimentalists have proposed *Dynamical Decoupling (DD)* [166, 167, 168]. As shown in Figure 4.2(b), DD keeps an idle qubit active by continuously rotating its state using single-qubit operations which suppresses the coupling between environmental noise and the qubit. DD is implemented by repeated execution of a sequence of single-qubit operations that returns the qubit to its original state (for example, a sequence of XYXY operations, as shown in Figure 4.2(b)). Thus, DD operations do not change the qubit’s overall state as they collectively behave as an identity gate that suppresses the noise. DD is widely used in qubit characterization and has been shown to be effective on IBM [101], Rigetti [169], and Google quantum hardware [170].

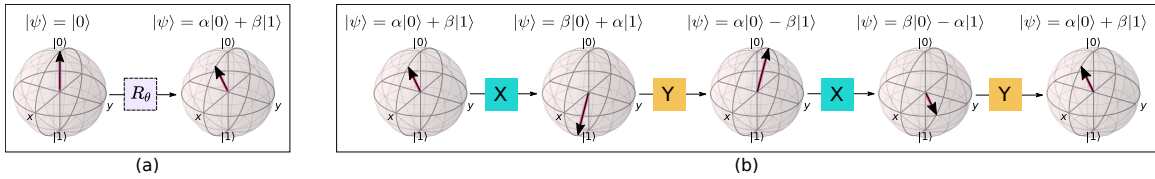


Figure 4.2: (a) Qubit-rotation operation on a single qubit (b) Dynamical Decoupling (DD) using XYXY (XY4) sequence.

While DD can reduce idling errors, the extra operations introduced by it can cause gate errors. If the error-rate of these extra operations exceeds the reduction in idling errors, then employing DD can reduce fidelity. Thus far, DD has been primarily limited to device-level studies for characterizing single qubits and its efficacy in reducing idling errors at the application level is not yet fully understood.

Table 4.1 shows the application fidelity for the two cases (a) when the program does not employ DD and (b) when all qubits employ DD during the idle periods. We observe that applying DD on all the qubits can improve fidelity. However, our characterization studies show that even when DD improves application fidelity, we may obtain even higher fidelity by applying DD to only a select subset of qubits. While DD has been effective at the single qubit level, it is unclear how to apply robustly DD at the application level.

4.3 Characterizing the Effectiveness of Dynamical Decoupling

4.3.1 Mitigating Idling Errors using DD

We quantify idling errors and the impact of DD in mitigating these errors using the characterization circuits shown in Figure 4.3(a) and (b). In the first circuit, shown in Figure 4.3(a), we initialize the qubit $q[0]$ in an arbitrary state by rotating along the Y-axis, using an $R_y(\theta)$ gate, and allow it to evolve freely over the idle period. This is achieved on IBMQ systems by inserting Delay or Identity gates. In the end, we bring the qubit back to state $|0\rangle$ by performing an inverse rotation, $R_y^{-1}(\theta)$, and measure it. To understand if the errors are more than just natural decoherence, we perform a similar experiment. However, now we use XY pulses, typically used for DD, throughout the idle period, as shown in Figure 4.3(b). Consequently, qubit $q[0]$ does not remain idle in this circuit. For effective characterization, multiple initial states are prepared by choosing different values of θ and studying the circuits for $1.2 \mu\text{s}$ (typical latency of 3 CNOTs). Figure 4.3(c) shows the fidelity of this circuit for a qubit on IBMQ-London. The fidelity of qubit $q[0]$ improves significantly when DD is applied as shown in Figure 4.3(c).

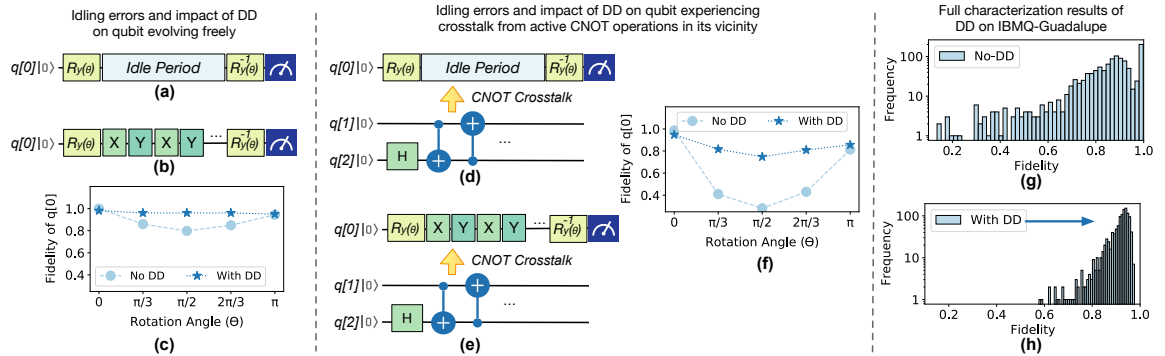


Figure 4.3: Circuit to evolve qubit $q[0]$ (a) freely (b) with DD. (c) Fidelity of $q[0]$ with free evolution and with DD. Circuit to evolve $q[0]$ (d) freely (e) with DD in the presence of crosstalk from ongoing CNOT operations. (f) Fidelity of $q[0]$ in the presence of crosstalk, with and without DD. Distribution of fidelity of the idle qubit (g) without and (h) with DD when circuits (d-e) are executed on every qubit-link combination on IBMQ-Guadalupe.

4.3.2 Effectiveness of DD under Crosstalk

In quantum programs, a qubit typically remains idle when other qubits are actively performing gate operations. Prior studies show that ongoing two-qubit CNOT operations generate crosstalk that can significantly lower the fidelity of concurrent CNOT operations [90, 171]. To study the impact of crosstalk from concurrent CNOT operations on idling errors, the fidelity of an idle qubit is measured by executing the circuits shown in Figure 4.3(d-e). In the first circuit, qubit $q[0]$ evolves freely in the presence of CNOT operations on neighboring qubits, whereas in the second circuit, the qubit $q[0]$ evolves in the presence of the XY DD gate sequence. Running these circuits on IBMQ-London for five different quantum states of qubit $q[0]$ and an idle time period of $2.4 \mu s$, we observe that the fidelity of the idle qubit $q[0]$ drops up-to 34% in the presence of concurrent CNOT operations, as shown in Figure 4.3(f). Thus, idling errors get amplified significantly in the presence of crosstalk, making quantum programs extremely vulnerable to these errors. However, DD continues to remain effective even in the presence of crosstalk from ongoing operations, as the fidelity improves from 34% to 75%.

We also characterize idling errors on 16-qubit IBMQ-Guadalupe. The idle qubit $q[0]$ is mapped to every physical qubit. Further, for each idle qubit, the active qubits – $q[1]$ and $q[2]$, are mapped to any of the remaining fifteen qubits that are physically connected. On IBMQ-Guadalupe, there are 224 such possible combinations and for each qubit-link combination, the two circuits (without and with DD) are executed for five different theta values (initialized using θ in $[0, \pi]$). The idle time is increased to $8 \mu s$ to understand idling errors and the effectiveness of DD in the context of large programs. Note that $8 \mu s$ is reasonable, as even a small program with 6-8 qubits and 20 serial CNOTs can experience such large idle periods. Figure 4.3(g-h) shows the distribution of the fidelity of these 2240 ($= 224 \times 5 \times 2$) circuits, without and with DD respectively. The fidelity of the idle qubit drops to 84.5% on average and up-to 13.6% in the worst case, when it evolves freely. However, with DD, the average fidelity improves to 91.3% and 57.7% in the worst case.

4.3.3 Factors Influencing Idling Errors and DD

To understand factors influencing idling errors and the effectiveness of DD, the characterization experiments discussed in subsection 4.3.2 are repeated on other IBMQ systems across multiple calibration cycles. For example, on 27-qubit IBMQ-Toronto, there are 700 qubit-link combinations and each of them are characterized using a total of 7000 circuits. The following key observations are made:

While DD generally improves the fidelity of the idle qubit $q[0]$, there are many instances where DD worsens the fidelity. For example, Figure 4.4 shows the histogram of the relative fidelity of qubit $q[0]$ in the presence of DD. While DD improves the fidelity up-to 3.95x in the best case, it lowers the fidelity to 0.21x in the worst case.

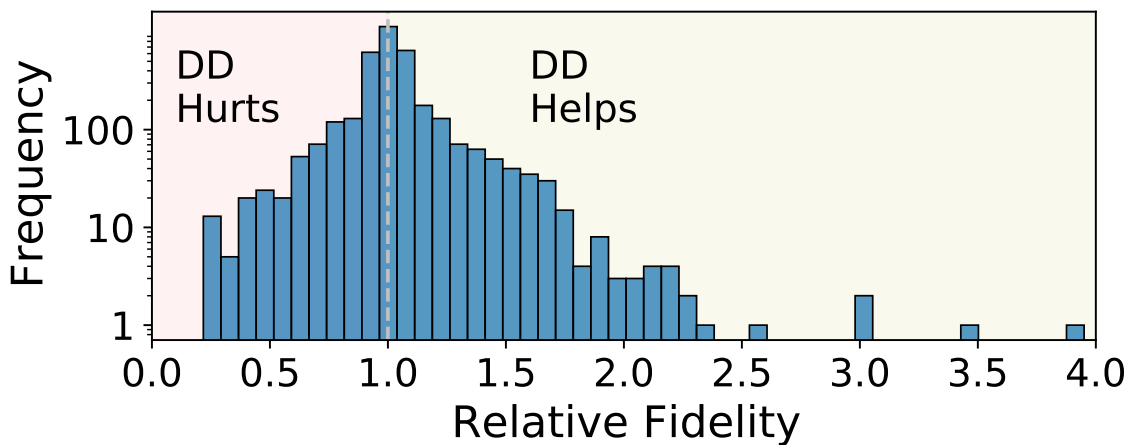


Figure 4.4: Distribution of Relative Fidelity of q_0 in the presence of DD for 700 qubit-link combinations on IBMQ Toronto.

Applying DD pulses to every qubit during each idle time window can adversely impact the fidelity of a program.

Idling errors depend on the state of qubits and concurrent CNOTs. Further, the effectiveness of DD changes across calibration cycles. For example, Figure 4.5 shows the relative fidelity of Qubit-12 when CNOT operations are performed on the Link:17-18 for two different calibration cycles. In the first cycle, DD improves the fidelity up-to 1.27x, whereas

DD degrades the fidelity up-to 0.35x in the second cycle. Our experiments also show that idling errors exist between qubit-link pairs that may not be present in the same on-chip neighborhood, making localized characterization approaches [90] inadequate. Similar observations are made on (a) systems such as IBMQ-Paris and IBMQ-Casablanca and using other (b) DD pulses [101].

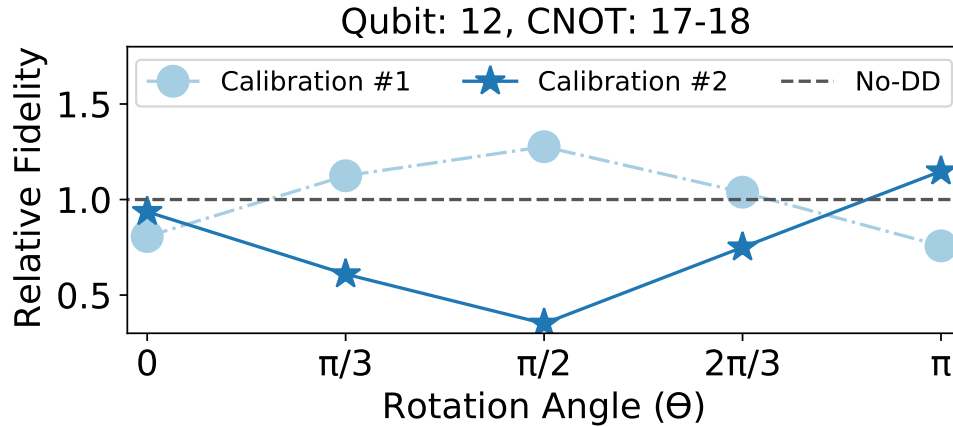


Figure 4.5: Relative Fidelity of Qubit-12 in the presence of CNOTs on Link:17-18 for different calibration cycles.

It is impractical to estimate the effectiveness of DD for all possible quantum states with respect to each qubit-link combination in large systems every calibration cycle.

The characterization complexity increases further when simultaneous CNOT operations are considered on multiple links. Our experiments show that although CNOT operations on multiple links can result in higher idling errors in general, such an additive effect does not exist always. For some of the cases, one of the active links dictates the overall idling error. In rare situations, multiple CNOTs can reduce idling errors.

To summarize, our evaluations using IBMQ systems show complex trends in idling errors and the effectiveness of DD. We confirm that (1) ongoing CNOTs increase idling errors, (2) the idling error-rate and effectiveness of DD depends on the CNOT patterns and combination of the idle and active qubits, and (3) the idle duration.

4.3.4 Impact of DD on Application Fidelity

The most straightforward method to enable dynamical decoupling at the application level is to insert DD pulses wherever feasible. To implement this design, all program regions where each qubit is idle can be identified and DD sequences may be inserted. However, naively inserting DD sequences for all qubits may not always be beneficial, as already in the characterization experiments. To demonstrate this effect at the application level, two 6-qubit benchmarks, *Quantum Fourier Transform (QFT)* and *Bernstein Vazirani (BV)*, are run on 27-qubit IBMQ-Toronto, with DD applied on all 64 (2^6) possible qubit combinations.

Figure 4.6 shows the fidelity (likelihood of getting the correct answer) for all 64 DD combinations, with 0 (000000) being the baseline when no DD is applied, and 63 (111111) being the case when DD is applied on all six qubits. The study shows that both benchmarks show significant variation in fidelity for different DD sequences. For *QFT*, enabling DD for all qubits increases the fidelity by 2.6x, however, the fidelity can be improved up-to 6.6x by choosing sequence "010100". For *BV*, applying DD on all qubits degrades fidelity to 0.88x, and we may deem DD to be counter-effective for this benchmark. However, using the DD sequence "010100" can improve the fidelity by 1.1x compared to no-DD and up-to 1.26x compared to DD-for-all. Note that the best DD sequence depends on the workload characteristics and the physical qubits used to run the program.

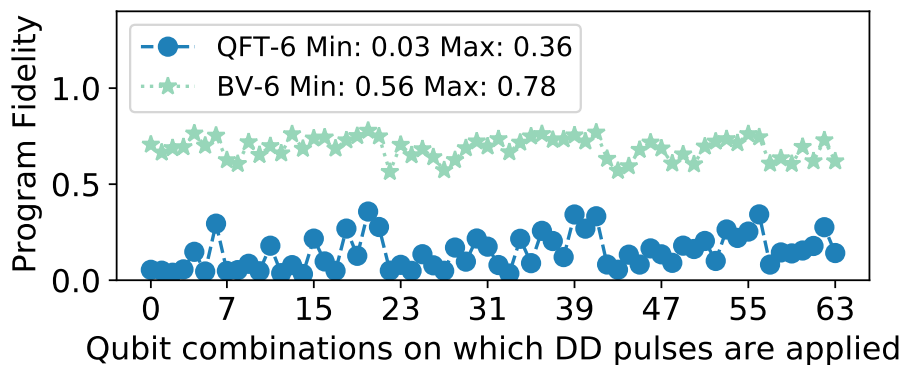


Figure 4.6: Fidelity of QFT and BV benchmarks with all possible DD sequences on IBMQ-Toronto. The sequence 0 (000000) denotes DD on none of the qubits and sequence 63 (111111) denotes DD on all qubits.

4.4 Overview of Proposed Design: Adaptive Dynamical Decoupling

To enable the robust use of DD at the application level, this dissertation presents *Adaptive Dynamical Decoupling (ADAPT)*. ADAPT identifies the combination of DD sequences that suppress idling errors and maximizes the application fidelity. ADAPT is implemented as a compiler pass that can be easily integrated with existing and future quantum compiler tool flows. This section provides an overview of ADAPT, discusses the design issues in estimating the optimal DD sequence, and proposes scalable search algorithms for the same.

ADAPT identifies all idle qubit slots in a quantum circuit and applies DD gate sequences during these idle periods. However, the optimal subset of qubits on which DD must be applied is neither known a-priori to program execution nor practically feasible to obtain through extensive device characterization. To overcome this challenge, ADAPT relies on a *Decoy Circuit* which is structurally similar to the input program, but with a known solution. Furthermore, to limit the complexity of the search for the optimal DD sequence, ADAPT employs a localized algorithm. Figure 6.2 shows an overview of ADAPT which accepts a quantum circuit as the input and outputs the circuit with the most optimal DD sequence. We discuss the specific design details of ADAPT next.

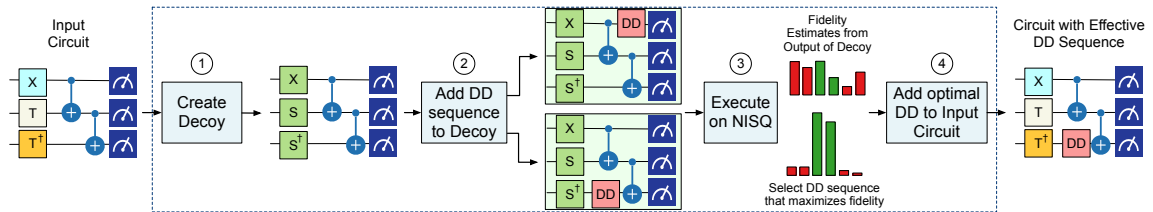


Figure 4.7: Overview of ADAPT with key building blocks.

4.5 Clifford Decoy Circuits (CDC)

If the outcome of a quantum circuit is known, we can apply DD on different subsets of qubits and assess their effectiveness in improving fidelity. Unfortunately, practical applications are hard to simulate using conventional computers, and their correct outcome is

unknown. To overcome this challenge, we leverage the following insights.

Insight #1: Not all quantum circuits are hard to simulate and circuits comprising of Clifford gates only can be simulated efficiently on conventional computers [172, 173].

Insight #2: Our characterization experiments show that crosstalk from CNOT operations is a dominant source of idling errors. Thus, two circuits with similar CNOT structures encounter similar idling errors.

ADAPT uses these two insights ① to generate an efficient *Clifford Decoy Circuit (CDC)* that preserves the structure of the input program. ② Next, ADAPT applies different DD combinations to this decoy circuit, and ③ selects the DD sequence that maximizes the fidelity of the decoy circuit. ④ Finally, ADAPT applies this optimal DD sequence to the input circuit and executes it.

4.5.1 Design of Clifford Decoy Circuits (CDC)

ADAPT relies on Clifford Decoy Circuits generated using gates from the Clifford group – *CNOT, X, Y, Z, H, S*. To create the CDC of a circuit, ADAPT replaces the non-Clifford gates of the circuit using the closest Clifford gates. To measure the closeness of a non-Clifford gate in the program with a Clifford gate, operator norm is used which is a distance measure used in the literature for approximating one unitary with another.

$$\|U - V\|_\infty := \max_{|\psi\rangle \neq 0} \frac{\|(U - V)|\psi\rangle\|_2}{\|\psi\rangle\|_2} \quad (4.1)$$

For example, by using the operator norm, the U1 gate is either replaced by Z or S gates, whereas U2 and U3 gates are replaced by the closest Clifford gates depending on the Euler angles associated with the gates. As CNOT is a Clifford gate, the structure and usage of these gates are identical between the CDC and the input program and therefore, the CDC encounters similar crosstalk from CNOT operations. This also ensures that the qubits in the CDC experience similar idle times as the original circuit.

4.5.2 Effectiveness of Clifford Decoy Circuits

To test the effectiveness of decoy circuits, we compare the fidelity of a 4-qubit quantum ADDER benchmark and its corresponding CDC for all possible DD sequence combinations. For example, this five qubit program has 16 (2^4) possible DD sequence combinations where the combination “0 (0000)” indicates that DD is not applied on any of the qubits, whereas the combination “15 (1111)” indicates that DD is applied to all of the qubits. The Spearman’s Correlation Coefficient is used to quantify the agreement between the input program and the CDC. Figure 4.8 shows the trend in program fidelity for the actual circuit and the CDC and we observe that the program fidelity is strongly correlated to the fidelity of the CDC (Spearman’s Correlation Coefficient = 0.78)

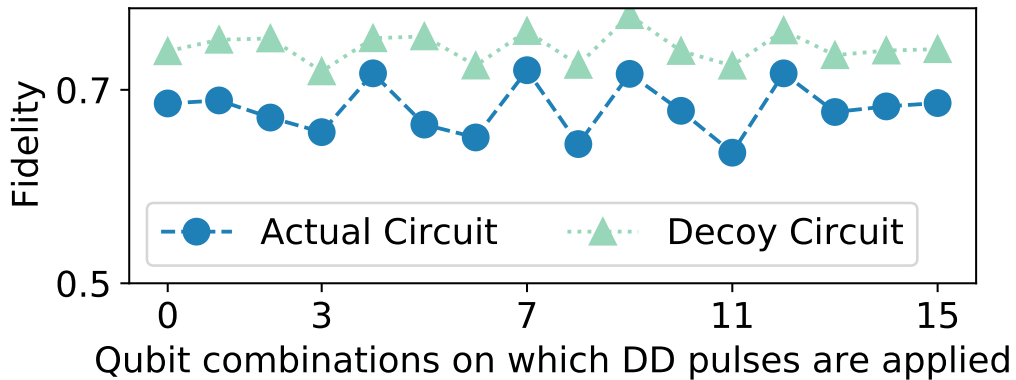


Figure 4.8: Correlation between the Fidelity of a 4-qubit Adder circuit on IBMQ-Guadalupe and the corresponding Clifford decoy circuit. We observe a strong correlation.

4.6 Overcoming the Limitations of CDCs: Seeded Decoy Circuits

ADAPT generates sub-optimal DD sequences when there is a mismatch between the fidelity trend of the input program and its CDC. Our experiments show that a CDC with high variance in the output distribution can be insensitive to changes in idling errors and relying on them may result in sub-optimal DD sequences. For example, if a CDC produces a uniform distribution, executing this CDC with different DD sequences does not significantly change the output distribution in the presence of idling errors.

To tackle this problem, ADAPT uses *Seeded Clifford Decoy Circuits (SDC)* that generate output distributions with low entropy, thus making them sensitive to idling errors. While simply removing all the single qubit gates from an input program and preserving the CNOT structure only, as shown in Figure 4.9, can generate a decoy circuit, it does not truly mirror the fidelity trends of the input circuit because it does not capture the phase errors. To ensure that the output entropy is reduced while still being representative of the input circuit, SDCs use a very limited number of non-Clifford gates. SDCs apply an initial layer of non-Clifford gates on a few qubits and replace the remaining non-Clifford gates with Clifford gates. For example, unlike the CDC shown in Figure 4.9(c) that uses all Clifford gates, the SDC shown in Figure 4.9(d) uses non-Clifford gates in the first layer of the circuit and Clifford gates in the later circuit layers.

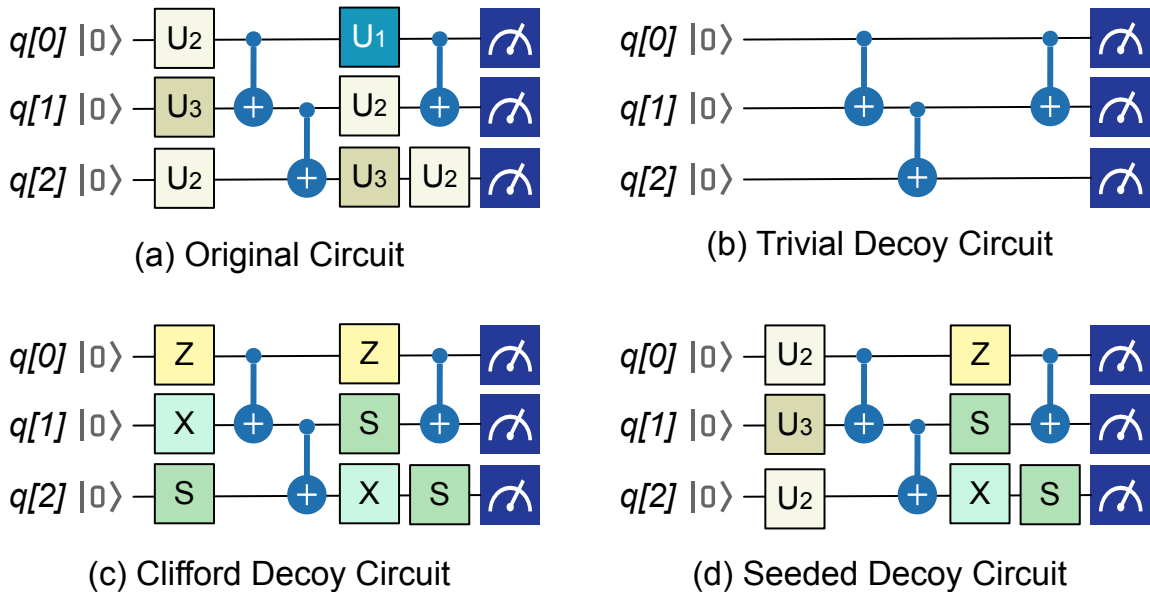


Figure 4.9: (a) Example quantum circuit. Decoy circuit construction using (b) Clifford gates (c) only CNOT (d) Mostly Clifford and few non-Clifford gates.

Our experiments show that SDCs can produce a rich state evolution while generating low entropy outputs. Although the simulation cost of SDC is slightly higher than CDC, it is not as expensive as running a full quantum simulation, which requires exponential cost. Moreover, SDCs produce low entropy outputs, and thus, further optimizations to reduce

the sampling cost can be deployed as well. For simulations, Qiskit Extended Stabilizer Simulator (based on [174]) is used. Table 4.2 shows the effectiveness of SDCs in improving the correlation between the decoy and original circuits and the time required to simulate the SDCs for 64,000 shots. To test the scalability, a 100-qubit QAOA SDC is simulated, which requires 330 seconds for 100,000 shots. Note that CDCs or SDCs require simulation only once because applying different DD sequences does not alter the output on a simulator.

Table 4.2: Correlation between Decoy and Input Circuits (higher is better)

Benchmark	Adder	QFT-6	QAOA-8	QAOA-10
Platform	IBMQ-Rome	IBMQ-Paris	IBMQ-Paris	IBMQ-Paris
CDC-Correlation	0.76	0.53	0.22	-0.012
SDC-Correlation	0.81	0.68	0.74	0.62
SDC-SimTime	1.2 Sec	5.4 Sec	12.8 Sec	22.2 Sec

4.7 Managing Search Complexity

The state space for all possible DD sequences scales exponentially with the problem size. For a program with N qubits, there are 2^N possible combinations of qubits on which DD pulses can be applied. The combination “000..0 _{N} ” represents DD applied on none of the qubits, whereas the combination “111..1 _{N} ” represents DD applied on all the qubits, whenever the qubit is idle. Unfortunately, one cannot search this design space exhaustively for large programs even with CDCs. Instead, ADAPT performs a localized search, whereby it first tries to find the best subset of qubits in a neighborhood of 4-qubits (searching for all 16 combinations) before moving to the next neighborhood of 4-qubits. Therefore, for a circuit with N qubits, it uses at most $4 \times N$ decoy circuits to estimate the best DD sequence. Thus, the search complexity increases linearly in the number of qubits. To accommodate the limitations of decoy circuits and obtain the optimal DD sequence, we take a conservative estimate from the top two predicted sequences from ADAPT. For example, if the two best predictions are ”1001” and ”1011”, the chosen sequence is ”1011”.

4.8 ADAPT: Design Implementation

Figure 4.10 shows the workflow of ADAPT and how it fits into existing execution models.

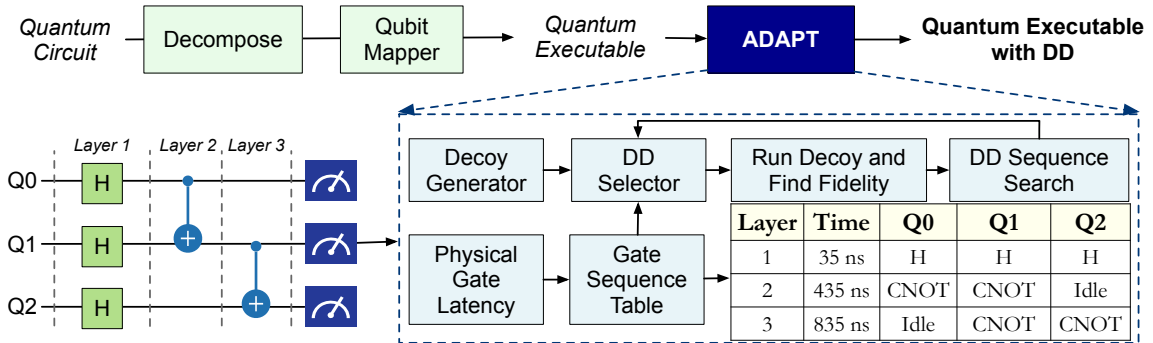


Figure 4.10: Overall Workflow of ADAPT.

4.8.1 Integration in the Compiler Tool-Flow

ADAPT is applied at the end of existing compiler passes after a program has been compiled into the machine-specific instructions of a given hardware. ADAPT is orthogonal to existing optimizations, and it is applied after all other compiler passes. Therefore, it can be seamlessly integrated with any existing compiler.

4.8.2 Finding Idle Qubits

To find idle qubits in a program, ADAPT translated the executable obtained from the compiler into an intermediate representation termed as the *Gate Sequence Table (GST)*, as shown in Figure 4.10. The GST slices the compiled circuit into layers and captures the data dependencies between the qubits in time. Note that the typical circuit representation used by the decomposition and mapping passes does not capture the idle cycles as gate latencies are not embedded in circuit representations. Whereas, GST uses timestamps generated by using physical gate latencies available from the machine calibration data to indicate the start and end times of each gate. By querying the GST, ADAPT identifies the exact idle period for any qubit in a program and inserts the DD gate sequences accordingly.

4.8.3 Pulses for Implementing DD

In theory, any sequence of instructions that are effectively identity gates can be used to implement DD. For example, one could simply use XX and YY pulses as the pairs result in identity operations. In this dissertation, two different DD protocols are studied: the XY-4 sequence and the IBMQ-DD sequence as both have been shown to be effective for superconducting qubit devices in general [169, 170], and particularly on IBMQ systems [169, 101]. The XY-4 sequence, shown in Figure 4.11(a), is repeated execution of "X-Y-X-Y" gates, which takes about 210 ns on most IBMQ machines as per its most optimal decomposition shown in Figure 4.11(b). Note that as ADAPT adds DD sequences to the already compiled executable, it is necessary to add DD gates in the machine-compliant instruction format. Each "X" and "SX" gate takes about 35 ns and the "RZ" gate is performed in software [175]. For this protocol, ADAPT inserts the DD gates for any idle windows with a duration larger than 210 ns. Also, ADAPT continuously inserts XY-4 DD sequences for larger idle windows. For the IBMQ-DD sequence, ADAPT uses an approach similar to prior work [101] and inserts the decomposition of " $X(\pi)$ " and " $X(-\pi)$ " gates evenly during the idle period, as shown in Figure Figure 4.11. ADAPT uses a 10 nanosecond free evaluation buffer after each X and Y gate, consistent with the prior study on IBM systems [169].

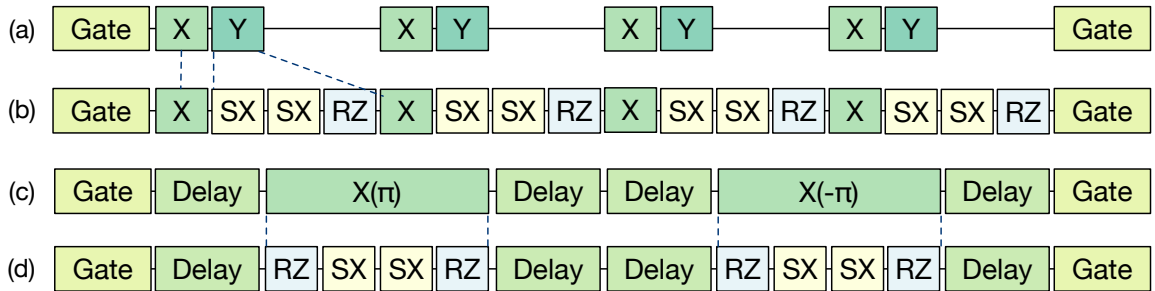


Figure 4.11: (a) XY-4 DD sequence, (b) decomposition of XY-4 sequence on IBMQ systems, (c) IBMQ-DD using $X(\pi)$ - $X(-\pi)$ gates, and (d) decomposition of IBMQ-DD sequence inserted in the idle window between two gates.

4.9 Evaluation Methodology

In this section, the evaluation infrastructure used to assess ADAPT is discussed.

4.9.1 Compiler

IBM’s Qiskit tool-chain is used to compile the benchmarks [176]. The Qiskit transpiler with ”*noise adaptive*” mapping [48, 47], ”*sabre routing*” policy [82], and optimization level 3 flags. Identical mappings and sequences of CNOT gate operations are ensured across all the policies evaluated for each benchmark. While IBM’s compiler tool-chain is used for the results presented in this dissertation, ADAPT is independent of the compiler being used, and therefore, any other compiler may be used as well. As ADAPT integrates with existing compilers as a post-compile step, the optimal decomposition of the DD gates is added to obtain the final compiled quantum object. For the implementation of the DD pulses, both XY-4 and IBMQ-DD (XX) sequences are used because they have been previously studied for IBMQ systems and summarize the broad category of DD protocols available. Although beyond the scope of this dissertation, ADAPT can be implemented using other DD sequences and specialized gate pulses as well.

4.9.2 Quantum Hardware Platforms

Three different quantum hardware from IBM are used for the evaluations. The details of each hardware and their average error trends are listed in Table 4.3.

Table 4.3: Error Characteristics of IBMQ Hardware.

Machine Name	Num. of Qubits	Error Rate (in %)		T1	T2
		CNOT	Measurement	(μs)	(μs)
IBMQ-Guadalupe	16	1.27	1.86	71.7	85.5
IBMQ-Paris	27	1.28	2.47	80.8	83.4
IBMQ-Toronto	27	1.52	4.42	105	114

4.9.3 Benchmarks

Benchmarks of different sizes and structures are used to test the effectiveness of ADAPT. Table 5.1 summarizes the benchmarks used. The size and type of benchmarks are derived from prior works on software mitigation of hardware errors [82, 47, 48, 88, 153, 177]. Additionally, some of these benchmarks are run with different initial conditions. For example, QFT-7A and QFT-7B have identical structures, but compute the Fourier transform of two different quantum states. This tests the effectiveness of decoy circuits for the evolution of different quantum states.

Table 4.4: Quantum Benchmark Characteristics

Benchmark Description	Benchmark Name	Num Qubits	Total Gates	Circ Depth	Avg. Idle Time (μs)
Bernstein Vazirani	BV-7	7	95	30	8.8
	BV-8	8	132	55	15.2
Fourier Transform	QFT-6A	6	102	47	14.6
	QFT-6B	6	228	148	27.9
	QFT-7A	7	360	202	50.0
	QFT-7B	7	354	207	49.7
Approx. Optimization	QAOA-8A	8	32	19	4.6
	QAOA-8B	8	60	30	8.1
	QAOA-10A	10	77	37	13.5
	QAOA-10B	10	180	50	7.7
Phase Estimation	QPEA-5	5	175	94	11.9

4.9.4 Figure-of-Merit

To quantify the program fidelity, the output distribution obtained on a real machine is compared with respect to an idealized (error-free) machine (results obtained on an error-free simulator). The program fidelity is quantified by computing the distance between the two

probability distributions. *Total Variation Distance (TVD)* [159] is used to evaluate the distance between ideal output probability (P) distribution and the real experiment’s output (Q), as shown in (Equation 4.2). A Fidelity of 1 means identical distributions, whereas 0 means completely different distributions. Therefore, a higher Fidelity is desirable.

$$TVD(P, Q) = \frac{1}{2} \sum \|P_i - Q_i\| \tag{4.2}$$

$$Fidelity = 1 - TVD(P, Q)$$

Prior works including JigSaw have used similar metrics such as Success Probability [47, 48, 153, 178]. A distance-based metric is used for evaluating ADAPT because the output of the quantum program can be a probability distribution with multiple correct answers. Also, TVD closely matches prior metrics.

4.9.5 Number of Trials

Each experiment used up-to 32,000 shots, depending on the program size, to obtain the output probability distributions. The largest benchmark in our study uses ten qubits (QAOA-10), so it can produce a maximum of 1024 (2^{10}) unique solutions and therefore, the number of samples used in our study is sufficiently large for the workloads.

4.9.6 Competing Policies

For our evaluations, four competing policies are used which are described next:

1. **No DD (Baseline):** DD is not applied to any idle qubit.
2. **All-DD:** DD is applied on all program qubits during any time period when they are idle.
3. **ADAPT:** The optimal DD sequence is obtained from the proposed framework.
4. **Runtime Best:** Evaluates the program with all possible DD sequences (2^N for an N -qubit program) and the sequence with the highest fidelity at runtime is selected.

4.10 Final Evaluations

This section provides the evaluation results for ADAPT across three different quantum computers: 27-qubit IBMQ-Paris, 27-qubit IBMQ-Toronto, and 16-qubit IBMQ-Guadalupe.

4.10.1 Results for IBMQ-Paris

Figure 4.12 shows the Fidelity of four benchmarks for the XY4 protocol compared to the baseline (without DD). The number below each benchmark label specifies the baseline fidelity. On average DD improves Fidelity. Applying DD on all qubits improves the fidelity by 1.97x on average and up to 2.89x. However, ADAPT improves the application fidelity by 3.27x and by up-to 5.7x. The effectiveness of DD increases with increasing program size, which is expected because larger programs have more operations and depth, leaving room for longer idle time windows in the program.

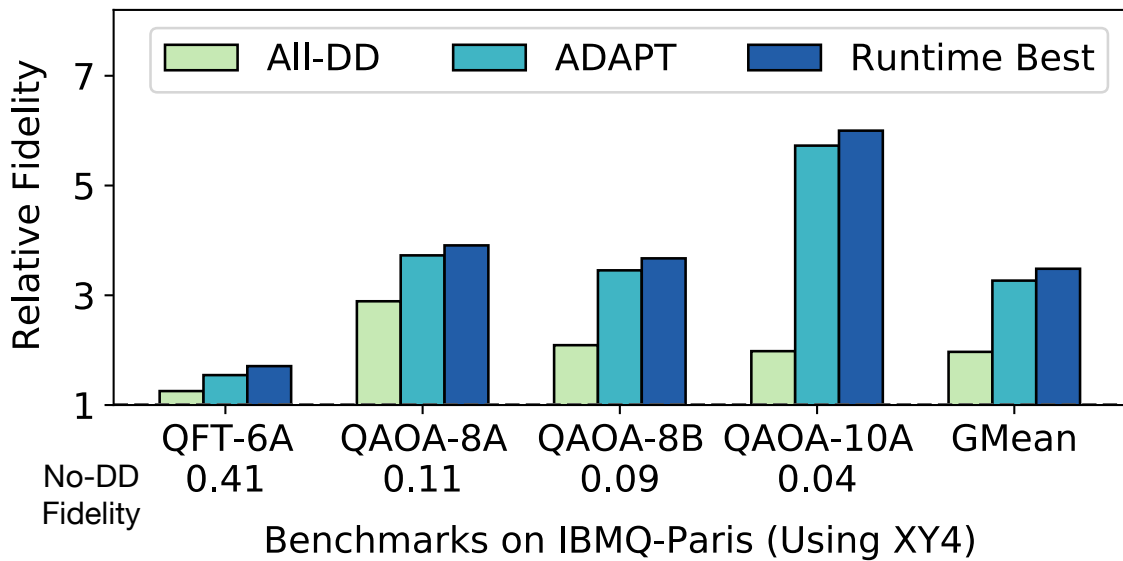


Figure 4.12: Relative fidelity on 27-qubit IBMQ-Paris using XY4 DD sequence.

The optimal sequence at run-time outperforms both ADAPT and All-DD. This is due to the limitations of the decoy circuits and the limited search space explored by ADAPT. The default ADAPT design searches the best sequence in a neighborhood of up-to four

qubits at a time. For example, for QAOA-10, ADAPT uses only 36 decoy circuits, unlike the entire 1024 possible decoy circuits space. Nonetheless, the fidelity improvement of ADAPT is close to the runtime best and higher than All-DD for these workloads. Note that experiments could not be performed using the IBMQ-DD protocol for IBMQ-Paris because of changes in the basis gates and subsequent retirement of the machine.

4.10.2 Results for IBMQ-Toronto

Figure 4.13 shows the Fidelity of All-DD, ADAPT, and the Runtime-Best policies for 27-qubit IBMQ-Toronto machine, relative to the baseline for two different DD protocols. The number below each benchmark label specifies the baseline fidelity of the application. The structure of the QFT circuit cause qubits to remain idle for substantial time periods. For example, in QFT-6B, Qubit-0 is idle for 90% of the total time taken for the overall execution. Although a long sequence of DD gates adds a significant amount of single-qubit gate errors, it is still effective in improving the overall fidelity. Overall, ADAPT outperforms the baseline and improves the Fidelity by 1.52x and by up-to 3.1x for the XY4 protocol. Compared to All-DD, ADAPT improves the Fidelity on average by 1.3x and by up-to 1.89x. For the IBMQ-DD scheme, ADAPT improves the Fidelity by 1.47x and up-to 2.67x compared to the baseline. Thus, ADAPT is a generalized technique to identify qubits most vulnerable to idling errors at runtime and has applicability irrespective of the DD protocol.

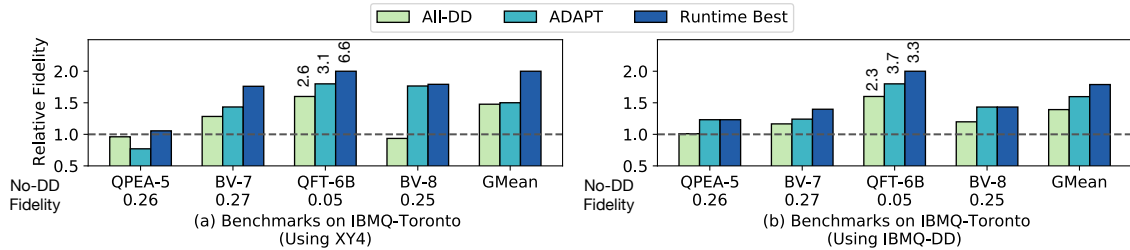


Figure 4.13: Relative Fidelity of different policies on 27-qubit IBMQ-Toronto using (a) XY4 and (b) IBMQ-DD sequences.

4.10.3 Results for IBMQ-Guadalupe

Figure 4.14 shows the Fidelity of the three different DD policies for the 16-qubit IBMQ-Guadalupe machine, normalized to the No-DD baseline. Note that this is one of the most recently released IBMQ systems with significantly reduced gate latencies and error-rates and improved coherence times. So, to test the robustness of ADAPT, we run slightly larger workloads (in terms of the number of qubits, two-qubit operations, and circuit depth) on this machine. Here too, the number below each benchmark label specifies the baseline fidelity of the application. We observe that in general applying DD on all idle qubits for such large programs slightly degrades the fidelity in specific cases (QFT-7A for example). Note that the dominant source of errors in these circuits is not idling errors but gate and measurement errors. However, ADAPT is more robust and generally outperforms the All-DD policy. For example, the fidelity of the QAOA-10B benchmark improves by 3.1x compared to the baseline (without DD) and 4.65x compared to applying all-DD. However, we observe that in all these cases, the most optimal DD sequence at runtime outperforms All-DD.

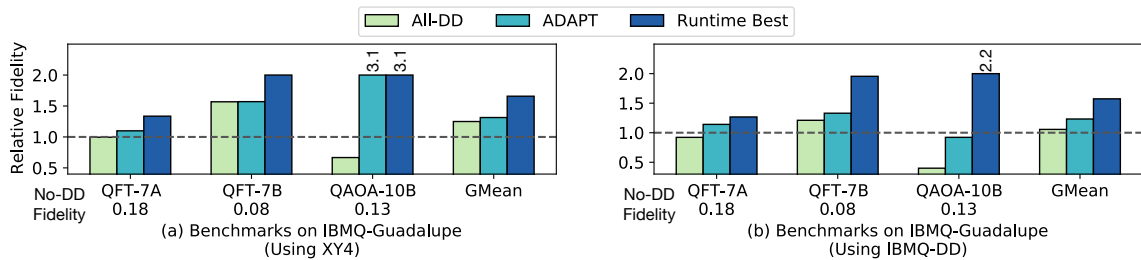


Figure 4.14: Relative Fidelity of different policies on 16-qubit IBMQ-Guadalupe using (a) XY4 and (b) IBMQ-DD sequences.

Table 4.5 summarizes the minimum, average, and maximum Fidelity for the three different dynamical policies normalized to the baseline on three IBMQ machines ranging from 16 to 27 qubits. Overall, ADAPT improves application Fidelity by 1.7x on average and up-to 5.73x compared to No-DD.

Table 4.5: Summary of Results

Machine	All-DD/ XY4			ADAPT/ XY-4			ADAPT/ IBMQ-DD		
	Min	GMean	Max	Min	GMean	Max	Min	GMean	Max
Paris	1.25	1.97	2.89	1.55	3.27	5.73	–	–	–
Toronto	0.58	1.17	2.61	0.68	1.23	3.06	0.99	1.42	2.67
Guadalupe	0.67	1.10	1.57	1.1	1.31	3.10	0.92	1.23	1.33

4.10.4 Impact of DD Pulse Type

We compare the effectiveness of the two protocols studied in this paper standalone using some additional characterization experiments: the XY-4 sequence and the state-of-the-art XX sequence recently proven to be effective on IBMQ systems [101]. In the experiment, three different circuits are prepared for variable idle times (T), as shown in Figure 4.15. In the circuit, a quantum state is prepared by performing a single qubit rotation and the associated qubit is kept idle. Throughout the idle time, CNOT operations are repeatedly performed on a specific physical link of the device that is not connected to the qubit under study to induce crosstalk while the qubit under study remains idle. Finally, the qubit under study is brought back to the ground state by performing the inverse rotation. In the first circuit, no DD pulses are inserted, whereas in the second circuit the XY4 DD pulses are inserted throughout the idle period. The third circuit uses IBM’s DD sequence in which $X(\pi)$ and $X(-\pi)$ are evenly placed by waiting for specific delay slots, as shown in Figure 4.15(c). The time period for each individual delay slot is computed from the difference of idle time and length of the two X rotations, as described in (Equation 4.3). Note that the optimal decomposition is used for both the DD protocols to ensure a fair comparison in our evaluations.

$$\text{Delay} \left(\frac{\tau}{4} \right) = \frac{T - \text{length of } X(\pi) - \text{length of } X(-\pi)}{4} \quad (4.3)$$

These circuits are run for each qubit and physical link combination (total of 224 com-

binations possible) on 16-qubit IBMQ-Guadalupe and Figure 4.15(d) shows the average fidelity of each circuit as the idle time is increased. The study shows that on average XY4 sequence outperforms the IBMQ DD sequence with increasing idle time. Similar results are reported for the Google Sycamore hardware [170, 43] as well as other works [179]. This is because when idle periods are longer, there is still sufficient delay between the two X rotations during which errors can accumulate for the IBMQ-DD sequence. While such long idle periods may not be observed for random circuits used in Quantum Volume experiments as in the study [101], they exist in many other practical quantum applications (QFT for example). For circuits with long idling periods, the IBMQ DD protocol often performs worse than applying XY4 continuously because the latter does not experience large delays between DD pulses. To account for this in our evaluations at the application level, the IBMQ-DD sequence is used in a more conservative manner by inserting the DD gate sequence multiple times for large idle periods. This is similar to the policy used for the XY4 protocol and enables a fair assessment.

Key Takeaways

- Qubits encounter idling errors even when they are not actively performing any operations. Qubits typically remain idle for long periods of time in most quantum programs.
- Dynamical decoupling (DD) is a device-level technique for mitigating idling errors, but its efficacy at the application level is not yet fully understood.
- *Adaptive Dynamical Decoupling (ADAPT)* is a software framework that estimates the efficacy of DD for each qubit combination and judiciously applies DD only to the subset of qubits that provide the most benefit.
- ADAPT employs a *Decoy Circuit*, which is structurally similar to the original program but with a known solution, to identify the DD sequence that maximizes the fidelity.
- To avoid the exponential search of all possible DD combinations, ADAPT employs a localized algorithm that has linear complexity in the number of qubits.

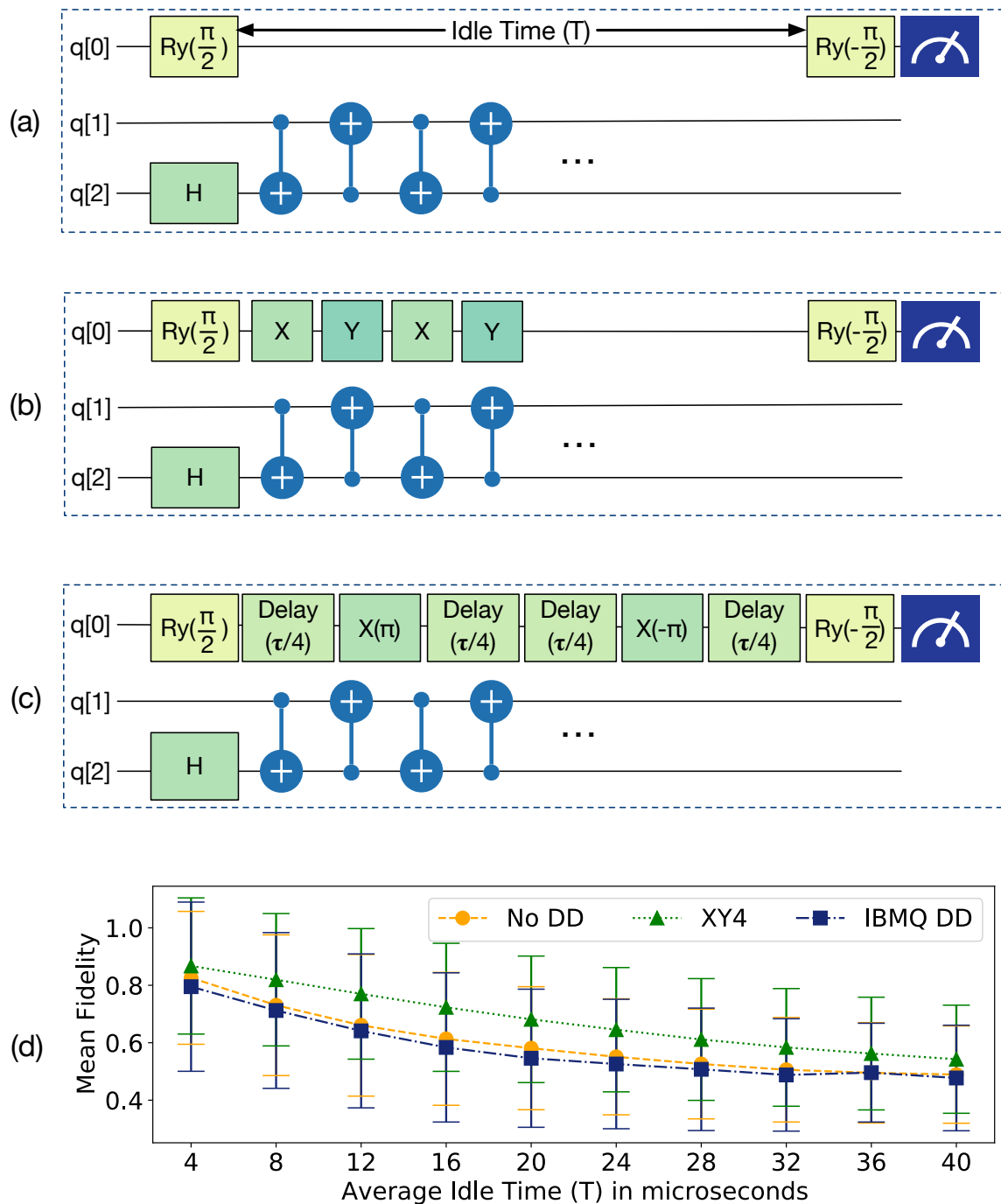


Figure 4.15: A characterization circuit (a) without any DD (b) with the XY4 DD sequence (c) IBMQ DD sequence. (d) Mean fidelity from individual DD sequences on 16-qubit IBMQ Guadalupe when the idle time is increased.

4.11 Evolution of the Idling Error Mitigation Landscape and Follow-up Works

Dynamical decoupling was proposed as early as the 1990s to overcome idle errors in quantum devices [180, 181]. In subsequent years, different protocols were introduced to adequately suppress idling errors on different qubit technologies [167, 168, 166, 182, 183, 184]. However, most of these studies were limited to the device level and the usage of DD was primarily relevant in the context of device characterization. In 2018, Pokharel et al. demonstrated the suitability of DD on relatively noisy and small-scale cloud-based quantum computers [169]. Strikis et al. [185] proposed a strategy that inserts an extra gate before and after each operation to reduce both active and idle errors by using a learning scheme to identify the type of extra gates that must be used. Similarly, Zlokapa et al. [186] proposed to train a deep neural network to learn the noise characteristics of a 5-qubit machine and use this network to identify the best DD pulses. A year later, IBM used a custom DD protocol, wherein the compiler inserts DD custom pulses on all qubits wherever possible and used it to demonstrate a milestone achievement of reaching a quantum volume of 64 [101] (subsequently 128) for their superconducting machine. Unlike these studies, ADAPT is the first proposal that assesses the impact of *always* applying DD sequences at the application level. In the same year, IBM released software changes to their Qiskit framework that enables a programmer to insert different DD sequences, of variable lengths, on program qubits of their choice, re-emphasizing some of the key insights of ADAPT [187]. DD continues to be an important software error mitigation technique as a recent study shows that quantum speedup could not be achieved on some of the existing quantum systems without it [188]. While the underlying principle of using decoys have been relevant in various classical computing problems and other real-world situations [189], its applicability in the quantum computing domain has only increased since the introduction of ADAPT. The insight of using Clifford-based decoy circuits from ADAPT was later used in other areas such as native gate selection on emerging quantum platforms [190] and circuit-cutting [191].

CHAPTER 5

EFFICIENT SOLUTIONS FOR IMPROVING THE THROUGHPUT OF EMERGING QUANTUM COMPUTERS VIA MULTI-PROGRAMMING

This chapter of the dissertation presents *Multi-programming Quantum Computers (MQC)* [144], a quantum cloud resource management solution to improve the throughput of emerging quantum computers by concurrently running multiple applications on the same quantum hardware. However, a multi-programmed environment can adversely impact individual application fidelity due to interference between co-running applications. This dissertation presents solutions that can improve resource utilization and throughput by up to 2x while limiting the impact on application fidelity.

5.1 Motivation

Quantum computing research has accelerated in recent years leading to an increase in the number of active quantum users from a wide spectrum of scientific domains. Simultaneously, there is growing interest in leveraging software error mitigation to improve the application fidelity on emerging quantum computers. However, these techniques typically require the execution of additional characterization circuits. Thus, the effective number of circuits executed by a user is typically much higher than the actual program, which increases the total execution time on the quantum hardware. Most near-term quantum applications use variational algorithms that execute such a large number of circuits for several iterations. Moreover, these circuits are typically executed on quantum computers by accessing them via cloud services. Programs from multiple users are scheduled using job queues which leads to long waiting times. In the worst case, if the latencies are too long and the error characteristics of the hardware change due to device drifts, the efficacy of the software optimizations used in the compiled program (waiting in the queue) may reduce.

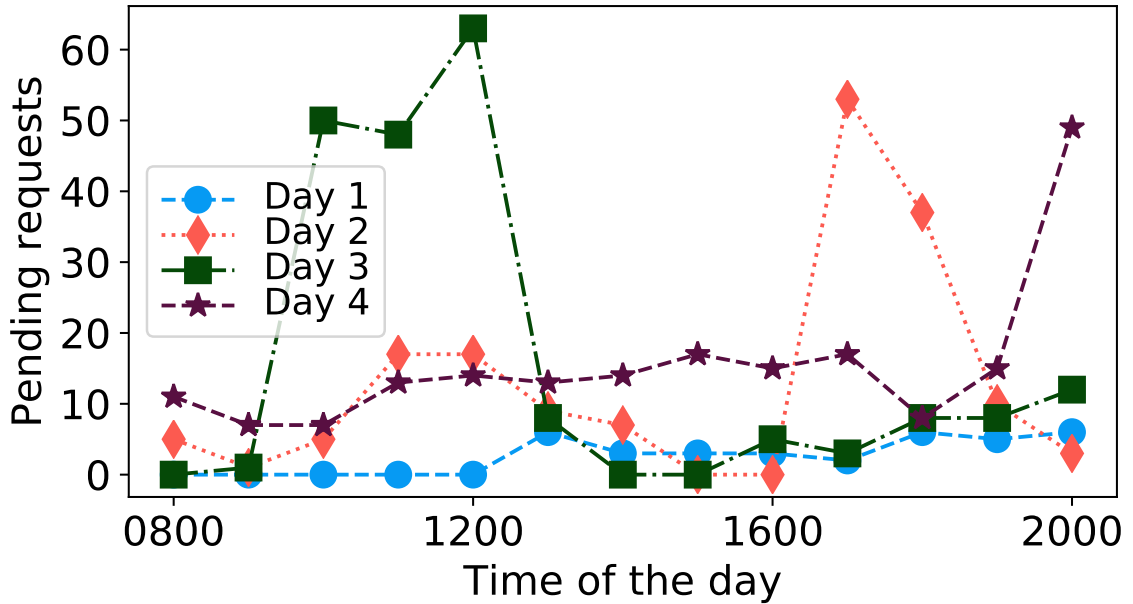


Figure 5.1: Pending traffic for IBM Q16 on different days.

Unfortunately, the rate at which quantum computers are being developed lags the rate of increase in the number of users and software error mitigation schemes. The limited number of systems leads to increasing contention for access to quantum resources and wait times, as shown in Figure 5.1. This data is obtained by monitoring the activity on one of the IBM quantum computers in the year 2018. Quantum computers today have up to thousands of pending jobs causing wait times for execution to be in the order of a few days. More recently, reservation-based policies such as Qiskit Runtime [192] have been suggested whereby a user can request dedicated access to the quantum computers. However, quantum resources may remain underutilized in such execution environments when the quantum machine waits for the classical computations to finish before it can perform its next step (for example, waiting for the updated circuit during the training phase of variational quantum algorithms). Instead, an orthogonal approach is to increase the rate at which a quantum computer services the requests (i.e. increase the throughput of the system). Increased throughput allows the system to handle a larger number of requests in the same amount of time and therefore, is desirable.

5.2 Challenges in Increasing Throughput of Quantum Computers

The throughput of quantum computers can be increased by building more quantum computers or by using the same quantum computer to run more programs per second (in other words, increasing its throughput). Building more quantum computers is non-trivial given the costs involved. The throughput of quantum computers can be increased by sharing quantum resources between multiple users. The high error-rates of quantum hardware limit most users from using all the available qubit devices. For example, the quantum volume of the 27-qubit IBMQ-Kolkata is only 128, which means that typically square circuits using only up to 7 qubits can be reliably run on this machine. This leads to under-utilization of quantum resources in the cloud. While this looks enticing from the perspective of multi-programming, there are several challenges. *First*, as the fidelity of applications depends on the physical qubits allocated to the program due to device-level variabilities, it is important to ensure fairness while allocating qubits to multiple programs in a shared environment. Partitioning a quantum computer to share resources may eventually allocate weaker qubits to an application and restrict a compiler's optimization capabilities for choosing reliable qubit movement paths. *Second*, qubits are extremely fragile and signals applied to one qubit can leak onto the other qubits causing unwarranted fluctuations in their quantum states. Interference may occur due to the crosstalk introduced by additional operations and qubit measurements. Such interference lowers the fidelity of the co-running application(s).

The challenges in fair resource allocation arise from the uniqueness of each physical qubit that is exhibited in the non-uniformity in coherence times, gate, and measurement error rates. Furthermore, these error rates vary over time. Thus, the physical qubits allocated to a program directly impact its reliability [47, 48]. Compilers account for this variation to perform qubit allocation and select qubit movement paths to enable *SWAP* operations. Multi-programming constraints the compiler to use a restricted set of physical qubits, limiting its capability to optimize for greater reliability. In order to understand the

restrictions imposed on qubit allocation, we look at the allocations of a 4-qubit program P1 and a 5-qubit program P2 on a hypothetical NISQ architecture. As shown in Figure 5.2(a), when mapped independently, P1 is allocated physical qubits A, B, I, and J whereas, P2 is allocated physical qubits A, B, C, I, and J. Figure 5.2(b) shows a qubit allocation for both programs together. The average link error rate of the regions allocated to P2 for independent execution and in the shared environment are 2.2 and 2.6 respectively. The allocation in the shared environment is 18% weaker.

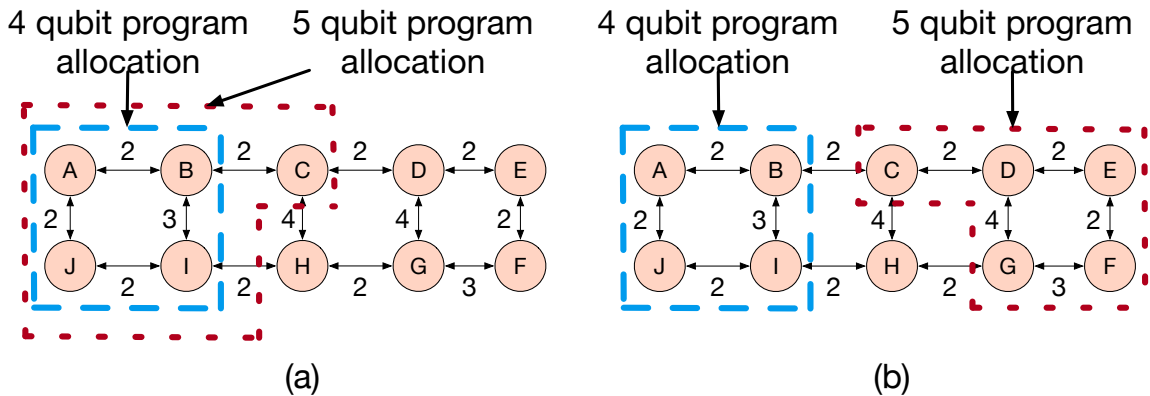
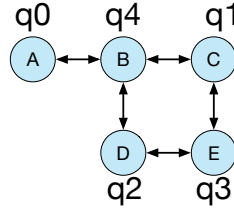


Figure 5.2: (a) Qubit allocation of a 4-qubit program P1 and a 5-qubit program P2. (b) Qubit allocations of P1 and P2 on a multi-programmed NISQ computer. Each node represents a qubit and the label on each edge represents the link error rate.

Application fidelity not only depends on qubit allocation but also depends on program characteristics and network topology of the allocated region. A well-connected region can minimize the total cost of SWAPs inserted to bring two non-adjacent qubits physically next to each other so that a CNOT gate can be executed. For instance, Figure 5.3 shows a program that executes 4 CNOT instructions and two possible network topologies. In the partition shown in Figure 5.3(b), the compiler needs to insert a SWAP operation in order to perform the 4th CNOT instruction. However, a better connected region as shown in Figure 5.3(c) requires a lesser number of SWAPs (in this case 0). When a quantum computer is partitioned for multi-programming, application reliability can vary based upon the number of SWAPs inserted. This depends on the network topology of the assigned partition.

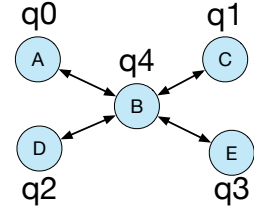
Example Program

1. cnot q0, q4
2. cnot q1, q4
3. cnot q2, q4
4. cnot q3, q4



Number of SWAPs : 1

(a)



Number of SWAPs : 0

(c)

Figure 5.3: (a) An example NISQ program (b) this topology requires 1 SWAP to perform Instruction 4 (c) this topology does not require any extra SWAP to execute the program.

5.3 Overview of Proposed Solution: Multi-Programming

This dissertation presents solutions for multi-programming emerging quantum computers while minimizing the impact on application fidelity. It relies on three key features: (1) *Fair and Reliable Partitioning (FRP)* algorithm that splits the qubit resources into multiple groups in a fair manner while avoiding the qubits/links that have extremely high error rates, (2) *Delayed Instruction Scheduling (DIS)* policy that mitigates the interference of between co-running programs, and (3) an *Adaptive Multi-Programming (AMP)* design that monitors the impact of multi-programming on fidelity at runtime and reverts the system to isolated execution mode if it reduces beyond an acceptable threshold. The details of each of these features are described in the next sections.

5.4 Fair and Reliable Resource Allocation Policies

Multi-programming quantum computers is feasible with negligible impact on application fidelity if each of the co-running applications in the shared environment is allocated qubit resources that are similar to the resources allocated if the program was to be run in an isolated environment. This enables us to run each application on qubits with low error-rates and ensures similar SWAP overheads. This section describes the details of the *Fair and Reliable Partitioning (FRP)* algorithms that enable us to achieve this goal.

5.4.1 Qubit Allocation for Multi-programs: Key Insights

To enable multi-programming on a quantum computer, two or more isolated regions must be located where multiple programs can be mapped and executed simultaneously. Physical qubits exhibit dissimilarities in error rates and coherence times on the same quantum architecture. For example, the average 2-qubit gate error rate on each physical link and measurement error rates for each qubit of IBM Q16 as shown in Figure 5.4.¹ We make two key observations from the error characteristics:

1. Not all good links are spatially co-located. A region with good links has weak links as well. For example, qubits $Q2$ and $Q12$ have two links each with error rates of 4%, but the link that connects them physically has an error rate of 17%.
2. Qubits with good connectivity and reliable links can still suffer from high measurement error rates (for example: $Q3$, $Q11$).
3. Qubits connected to link(s) with low gate error rate(s) do not necessarily have a large degree of freedom (number of links) and may also suffer from high measurement errors (for example: $Q7$).

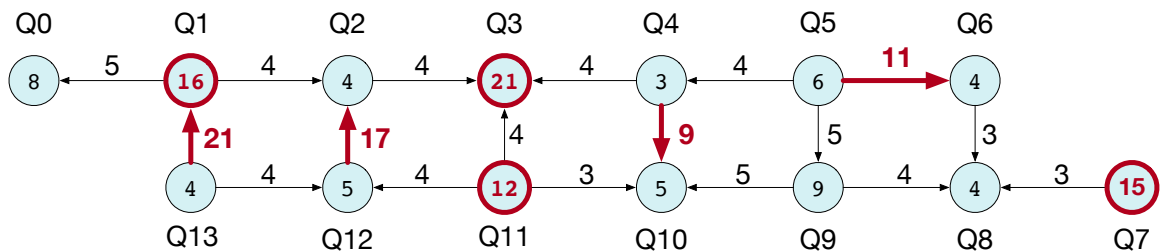


Figure 5.4: Error rates on IBM Q16. A node represents a qubit and the label on each edge is the 2-qubit gate error rate for that link. Links marked in bold have above mean 2-qubit gate error rate whereas qubits circled bold have greater than mean measurement error rate.

Similar trends are observed on IBM Q20 based on previously reported numbers [47]. The machine is currently retired. The trends hold true even on the most recent quantum computers from IBM and other device providers such as Rigetti and Google.

¹Error rates in this Figure are based on calibration data collected on 03.14.2018

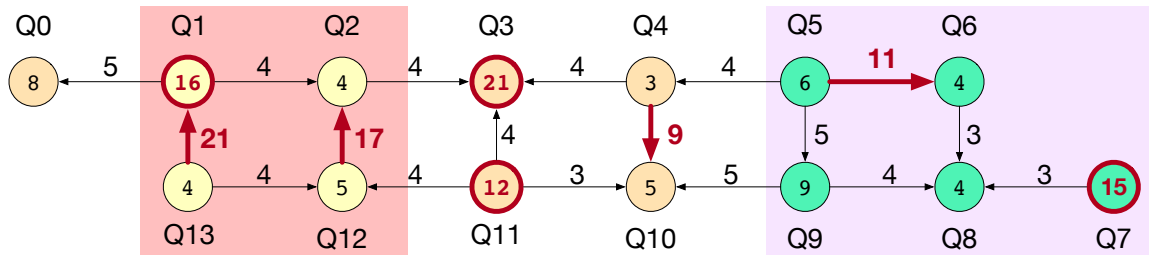
High-fidelity qubits are thus usually distributed spatially across the entire architecture, rather than being situated next to each other. Noise-adaptive compilers try to find a sweet spot by locating reliable qubits as well as allocating program qubits to physically close and well-connected qubits.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

5.4.2 Fairness in Qubit Allocation:

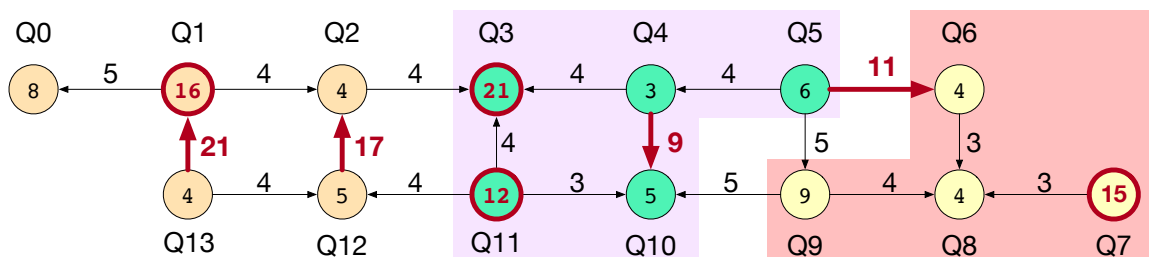
The qubit allocation Algorithm 2 analyzes the underlying architecture and ranks links and qubits depending upon their *utility*, and classifies the physical qubits into 3 groups of high, medium, and low utility. *Utility* of a physical qubit is defined as the ratio of the number of links (degree of freedom) to the sum of link error rates. The algorithm chooses a high-utility qubit with good neighboring qubits as the root and grows a graph by adding nodes along the boundary. Parameters α and β assists in choosing a high-quality root node in Algorithm 2.

Accounting for CNOT error rates

The algorithm locates qubit clusters such that most of the bad links or links with the highest error-rates are avoided by the programs together. For example, Figure 5.5 represents two potential partitions while allocating qubits for two programs with 4 and 5 qubits each. The partition in Figure 5.5(b) is considered better than Figure 5.5(a) since it avoids 75% of the weak links. The algorithm achieves this by choosing a different starting rank for generating sub-graphs and observing the total number of weak links in both regions.



(a)



(b)

Figure 5.5: Two possible partitions (a) and (b) for a 4-qubit program and a 5-qubit program that are scheduled to be run concurrently on the NISQ machine. Partition (b) is more reliable compared to partition (a).

Accounting for measurement errors

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

Accounting for program characteristics

Each quantum program has its own characteristics and uses individual program qubits differently. Our proposed partitioning algorithm accounts for these characteristics while allocating qubits by profiling the *usage* and *interaction graph* of each program qubit. *Usage*

of a program qubit is defined as the number of operations performed on it and its *interaction graph* is the set of other program qubits it interacts with. Few quantum algorithms use ancilla qubits in the program that are not measured (for example, the target qubit in Bernstein-Vazirani algorithm). These ancilla qubits are mapped to physical qubits with higher measurement error rates. Program qubits with high usage are mapped to physical qubits with higher utility followed by mapping its neighbors to program qubits from its interaction graph. The process is repeated until all program qubits are allocated.

The qubit allocation from Algorithm 2 is used as the initial mapping by the SABRE compiler [82]. We enhance SABRE (called *Variation Aware SABRE*) to use error data instead of Dijkstra’s distance that is used in SABRE originally to perform qubit movement operations. We also use the reverse circuit of the program to aid in mapping ancillae qubits on qubits with high measurement errors. It also assists in scheduling two or more programs under the delayed instruction scheduling described later in Section Section 5.5.

5.4.3 To partition or not to partition?

After selecting qubits for each application in the shared environment, the compiler analyzes the fidelity of each partition and flags a warning if at least one of the programs is allocated qubits with lower fidelity as compared to its qubit allocation in an isolated environment. The compiler uses the *tolerance factor* to decide the difference in fidelity it can tolerate. A user or the cloud provider may choose not to run a program on the shared environment based on the warning. The default implementation turns off multi-programming in the presence of the warning. This *tolerance factor* can be treated as a software knob available to the programmer. The ability to partition a quantum computer into regions where programs can be concurrently executed not only depends on the size of the quantum computer, the size of the programs in context, and the distribution of error rates but also on the daily variation in error rates. Regions on a quantum computer exhibiting completely different characteristics on different days can significantly limit the opportunities for multi-programming.

5.4.4 Algorithm Implementation

The detailed implementation of the FRP algorithm is described next.

Algorithm 2 Fair and Reliable Partitioning

```
1: // Locate a reliable cluster on the chip
2: function CREATE_SUB_GRAPH(Program, utility, CMR)
3:    $rank \leftarrow$  starting rank ;  $root \leftarrow node[rank]$ ;
4:   while root node not found do
5:     if  $\alpha\%$  of root's neighbors have high utility and
6:        $\beta\%$  nodes including root have measurement
7:       errors ; mean measurement error then
8:         root node found
9:       else
10:         $rank \leftarrow rank + 1$ 
11:      end if
12:    end while
13:    Grow graph from root by adding neighbors along
14:    the boundary until program can fit and If CMR is
15:    low, exclude nodes with large measurement errors.
16:    return sub graph
17: end function
18: // Perform qubit allocation for Fair and Reliable Partitioning
19: function FAIR_AND_RELIABLE_PARTITION(graph,usage, interaction)
20:   if ancillae in program then
21:     while all ancillae not mapped do
22:       Allocate ancilla ( $AnQ$ ) to qubit ( $PhyQ$ ) in SG
23:       with high measurement errors. Allocate
24:       non-allocated neighbors of  $PhyQ$  to program
25:       qubits that interact with  $AnQ$ .
26:     end while
27:   end if
28:   while all program qubits not mapped do
29:     Allocate program qubit  $PrgQ$  in order of usage
30:     to high utility qubit  $PhyQ$  in SG. Allocate non-
31:     allocated neighbors of  $PhyQ$  to unmapped
32:     program qubits that interact with  $PrgQ$ .
33:   end while
34:   return qubit allocation
35: end function
```

5.5 Delayed Instruction Scheduling Policy in Shared Resource Environments

Interference from measurement is one of the potential challenges in multi-programming a quantum computer, as will be discussed in this section. Crosstalk from additional operations can cause unwarranted fluctuations in the states of qubits. Interference from measurement operations can significantly limit our ability to multi-program quantum computers and this dissertation presents the *Delayed Instruction Scheduling (DIS)* policy to minimize the impact of this interference.

5.5.1 When Should Qubits be Measured?

When two programs are executed in parallel, conventional wisdom suggests that they be launched as decoupled independent threads as shown in Figure 5.6(a). In this scheduling, the qubits of each program are measured as soon as all its gate operations are complete. Unfortunately, on a quantum computer, if the two programs are decoupled, the measurements corresponding to the shorter program can inject noise channels into the operating environment of the co-running program, lowering its fidelity. Another possible approach is to schedule measurements after all gate operations corresponding to all the co-running applications are executed, as shown in Figure 5.6(b). This approach is used by the IBM *Qiskit* compiler as the default. Note that the DIS policy was developed based on the hardware constraints in the year 2018. More recently, mid-circuit measurements allow measurement operations to be performed with high fidelity even when other operations are ongoing [193]. However, they may still cause interference and IBM's compiler still defaults to measurements at the end. Measurement scheduling policies in shared environments in the presence of mid-circuit measurement is an open problem. On the other hand, if the measurements of the shorter program are delayed when programs are of unequal lengths to protect the programs in progress, the qubits may decohere by the time they are measured. To overcome this problem, we propose that two parallelizable programs be context aware.

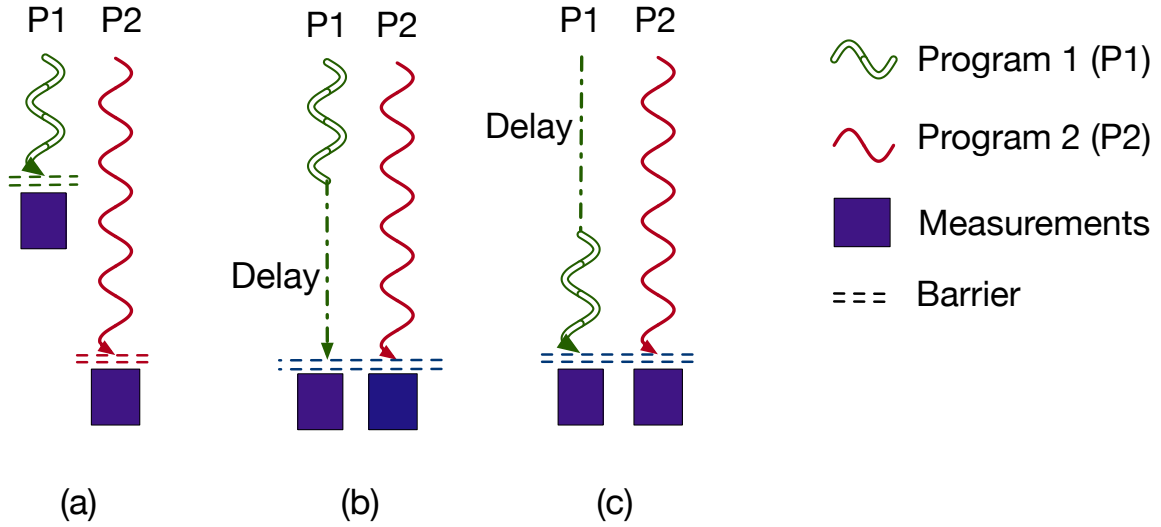


Figure 5.6: (a) Programmer expects two programs to be decoupled completely (b) Default instruction scheduling from IBM Compiler where all measurements are performed at the end (c) Our proposed delayed instruction scheduling policy where the shorter program thread starts late.

5.5.2 Scheduling Algorithm for Multi-programs

In order to minimize the impact of qubit measurement operations on on-going gate operations and to maximize the reliability of both the applications, this section presents the details of the proposed DIS policy such that both the applications complete around the same time and all qubit measurements are performed only after gate operations for both programs conclude. The same two programs shown in Figure 5.6(a) will be scheduled as shown in Figure 5.6(c) as per the delayed scheduling policy. The scheduling policy moves all the measurements to the end by inserting appropriate *barriers* as shown in Algorithm 3 and statically schedules instructions by analyzing the data flow graph of both the applications. As quantum programs are reversible, it is possible to determine the late starting point of the shorter program by scheduling the reverse of the programs and inserting barriers once each of the programs are fully scheduled. The final schedule is obtained by reversing the schedules. The overall schedule of the multi-programs is generated by Algorithm 4. The default implementation considered in this dissertation is concurrent execution of two programs on a quantum computer.

Algorithm 3 Instruction Scheduling

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

The multi-programming framework proposed in this dissertation is scalable and can be implemented by both the programmer and service provider. For allocating resources on a system with a large number of qubits, information on reliable regions can be saved once they are located and the information can be reused by other programs. As detection of reliable regions only depends on the calibration data; it can be done offline even when the quantum hardware is being used. If cloud providers opt for a utility computing model, such as IBM's Pay-As-You-Go model [194], programmers can optimize for both reliability and resource utilization by writing parallel programs themselves and requesting a specific qubit allocation. Many device providers, such as Rigetti [195], also allow users to only reserve a specific set of qubits prior to their execution and the proposed framework is compatible with such access models as well. Although the default implementation proposed in this dissertation runs only two programs concurrently, depending on the size of each program, the number of available qubits on a machine, and hardware error rates, it may be possible to run more than two programs in parallel in the shared environment. This has the potential of improving the throughput even further.

Algorithm 4 Multi-program Compilation

```
1: function (Chip coupling graph, N programs, calibration data)
2:   for Each physical qubit ( $phy_i$ ) on chip do
3:     Compute  $utility[i] = \frac{\text{Number of links of } phy_i}{\sum \text{Error rates of links of } phy_i}$ 
4:     Create 3 utility groups in order of number
5:     of links and utility
6:   end for
7:   if Delayed Instruction Scheduling then
8:     Program  $P_i \leftarrow$  Reverse  $P_i$ 
9:   end if
10:  for Each program  $P_i$  in N programs do
11:    for Each program qubit ( $PrgQ_j$ ) in  $P_i$  do
12:       $Usage[i][j] \leftarrow \frac{\text{Instructions using } PrgQ_j \text{ in } P_i}{\text{Total instructions in } P_i}$ 
13:       $Interaction[i][j] \leftarrow$  Set of program qubits
14:      that interacts with  $PrgQ_j$ 
15:      Calculate compute to measurement ratio:
16:       $CMR_i \leftarrow$  Number of Operations
17:    end for
18:    Rank Program qubits in order of their usage
19:  end for
20:  // Independent qubit allocation and scheduling
21:  for each program ( $P_i$ ) in N programs do
22:     $Graph_i \leftarrow \text{create\_sub\_graph}(P_i, utility, CMR_i)$ 
23:     $qalloc_i \leftarrow \text{fair\_and\_reliable\_partition}(Graph_i, usage[i],$ 
24:     $interaction[i])$ 
25:     $indp\_sched_i \leftarrow \text{variation\_aware\_SABRE}(Graph_i, P_i)$ 
26:  end for
27:  // Shared qubit allocation and scheduling
28:  for each program ( $P_i$ ) in N programs do
29:     $Shared_i \leftarrow \text{create\_sub\_graph}(P_i, utility, CMR_i)$ 
30:    Re rank unmapped physical qubits
31:     $qalloc_i \leftarrow \text{fair\_and\_reliable\_partition}(Shared_i, usage[i],$ 
32:     $interaction[i])$ 
33:     $sched_i \leftarrow \text{Variation\_aware\_SABRE}(Graph\_shared_i, P_i)$ 
34:  end for
35:  for each program ( $P_i$ ) in N programs do
36:     $Glob\_Sched \leftarrow \text{generate\_schedule}(\text{all } sched_i)$ 
37:    if mean error rate of all links in
38:      ( $Graph_i \times \text{tolerance} < Shared_i$ ) then
39:      Generate Warning
40:    end if
41:  end for
42:  return  $Glob\_Sched, indp\_sched$  for N programs
43:  ( $sched_i$ ), Warning
44: end function
```

5.6 Dynamically Switching between Multi-programming and Isolated Execution

Although our proposed FRP algorithm and DIS policy attempt to minimize the impact on application fidelity through optimized resource allocation and instruction scheduling, it may still be possible that the fidelity of one of the applications is low. Therefore, it is important that there is a mechanism to detect such occurrences and disable multi-programming for those instances. This dissertation presents a design that can dynamically monitor a program's fidelity at execution time and mitigate this impact caused by multi-programming by reverting back the program to isolated execution.

5.6.1 Adaptive Multi-Programming Design

This section presents a light-weight *Adaptive Multi-Programming (AMP)* design as shown in Figure 5.7, that resides at the user-quantum computer interface, manages the multi-programming environment, monitors program fidelity at the execution time using statistical tests, and reverts back a program to isolated execution when the impact on its fidelity is beyond an acceptable threshold.

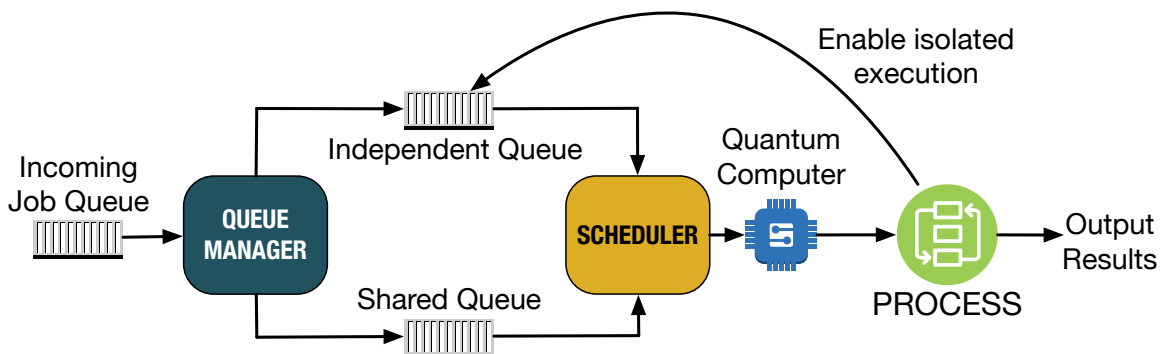


Figure 5.7: Design of Adaptive Multi-Programming

The *AMP* design uses the following mechanism to perform the aforesaid tasks:

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

The *AMP* design accepts jobs from different users in the *incoming job queue*. The Queue Manager (QM) maintains two queues: a) *shared queue* and b) *independent queue*. Besides accepting incoming jobs from the cloud and managing the additional outgoing queues to the Job Scheduler, it also hosts the compiler. The QM monitors incoming jobs and performs either of the following actions:

1. Compiles a program and puts its object code in the *independent queue* under any of the four conditions: 1) if it requires access to the entire quantum computer, 2) a user requests exclusive access to the entire machine, 3) the program cannot share resources with other programs, or 4) a user has already provided or requested a specific qubit allocation. This gives users an opportunity to control the region of the quantum hardware their program gets mapped to, which is desirable by some advanced users. Users can still choose to not share resources or request exclusive access to the machine.
2. Monitors the resource requirements of the jobs in the *incoming queue* and if it finds two independent jobs that can share resources, the QM allocates them qubits as per the algorithm described in Section Section 5.4 and compiles them. Post compilation, it puts the object code on the *shared queue*. It also compiles the individual jobs using the best available resources and puts the compiled job object into the *independent queue*. The size of reliable partitions where programs can be mapped may be predetermined using the calibration data. For larger machines, we expect the number of such available partitions to be higher, and therefore more than two jobs can be compiled and executed together.

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

requiring exclusive access to the machine as long as fairness is guaranteed. Even a simple round-robin policy and first-come-first-served approach may be sufficient.

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

5.6.2 Dynamic Reliability Monitoring Scheme

The design of the proposed reliability monitor is shown in Figure 5.8. It resides in the **Process** block shown in Figure 5.7. Programs that execute in parallel execute two types of trials: *independent trials* and *shared trials*. The reliability monitor leverages these two types of trials to dynamically detect significant impact on fidelity at runtime. It accepts the distributions from the two types of trials per application and decides to re-execute trials in an isolated environment for an affected program when it detects a deterioration in its fidelity. For detection, it uses two statistical metrics: *Entropy* and *Hellinger Distance*.

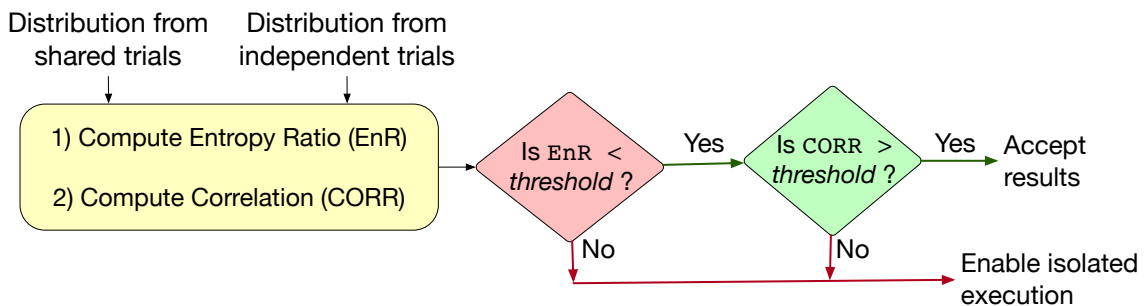


Figure 5.8: Reliability monitor and controller

Entropy [196] describes the amount of disorganization or randomness in a system quan-

tatively. Entropy in a system is higher in the presence of noise.² In quantum systems, fidelity or the probability to get the correct answer is highest when the noise is minimum. Depending upon the amount of noise, the system entropy changes. Thus, by computing the difference in the entropy between the two types of trials conducted on a program, the reliability monitor can detect any impact on application fidelity. If $X = \{x_1, x_2, x_3, \dots, x_N\}$ is a set of random phenomena, and $p(x_i)$ is the probability of occurrence of x_i , entropy can be computed using (Equation 5.1). In quantum computing, X is the state space of the N qubits. If $Entropy(I)$ is the entropy of the independent trials and $Entropy(S)$ is the entropy of the shared trials, the entropy ratio, EnR , can be computed using (Equation 5.2). A higher EnR indicates that the trials conducted while sharing are noisier.

$$Entropy(x) = - \sum_{i=1}^N p(x_i) \log_{10} p(x_i) \quad (5.1)$$

$$Entropy\ Ratio\ (EnR) = \left| \frac{Entropy(S)}{Entropy(I)} \right| \quad (5.2)$$

Entropy only estimates the amount of noise in the system and cannot capture the similarity between the distributions obtained from both types of trials. For example, two probability distributions $d_1 = \{0.2, 0.2, 0.2, 0.4\}$ and $d_2 = \{0.4, 0.2, 0.2, 0.2\}$ corresponding to the state space of 2 qubits can lead to an EnR of 1, even though the merged distribution will have low fidelity. Hence, we also compute the correlation ($CORR$) between these two output distributions, using a metric derived from Hellinger distance. Hellinger distance ($H-Dist$) [155] is used to quantify the similarity between two probability distributions. If $X = \{x_1, x_2, x_3, \dots, x_k\}$ and $Y = \{y_1, y_2, y_3, \dots, y_k\}$, are two discrete probability distribution functions, the Hellinger distance is computed by (Equation 5.3). It is bounded between 0 and 1. Two identical probability distributions have a $H-Dist$ of 0, whereas, the maximum distance of 1 is obtained when there is most dissimilarity. We define *correlation* ($CORR$)

²Entropy in a system is maximum when it has reached equilibrium. In the field of signal processing, noise is treated as a signal in thermal equilibrium.

using (Equation 5.4). This means, two identical probability distributions have a *CORR* of 1 whereas completely non-identical distributions have a *CORR* of 0. If *S* and *I* correspond to probability distribution functions obtained from shared trials and independent trials, for higher fidelity, a higher correlation (*CORR(S, I)*) is desirable.

$$H-Dist(X, Y) = \frac{1}{\sqrt{2}} \sqrt{\sum_{i=1}^k (\sqrt{p_i} - \sqrt{q_i})^2} \quad (5.3)$$

$$CORR(X, Y) = 1 - H-Dist(X, Y) \quad (5.4)$$

As shown in Figure 5.8, depending upon the computed *EnR* and *CORR(S, I)*, the monitor performs one of the following:

1. If *EnR* is lower than a preset threshold and *CORR(S, I)* is greater than a preset threshold, the trials are deemed reliable. The distributions are merged and returned to the user.
2. If *EnR* is greater than and *CORR(S, I)* is lower than corresponding preset thresholds, the trials are deemed unreliable. In that case, the monitor enables isolated execution for the affected program as shown in Figure 5.7.

The threshold for *EnR* depends upon the acceptable range of reliability and can be offered as a controllable knob to the software, whereas the threshold for *CORR(S, I)* can be predetermined conservatively for a given program size by profiling typical quantum benchmarks. The default implementation proposed here sets a threshold for *EnR* to be lower than 1.2 and *CORR(S, I)* to be greater than 0.8 (i.e. accepting 20% more noisy trials).

At its heart, the AMP design incurs the overhead of minimal changes to existing job schedulers, additional queues, and logic to merge two distributions making it scalable. The statistical tests in the run-time system involve arithmetic operations on classical machines which are readily available at data centers. The operations are not on the critical path and can be performed in conjunction with other jobs.

5.7 Evaluation Methodology

The evaluations for multi-programming are performed on the IBM Q16 machine.³ To ensure a fair comparison, all the experiments are performed in the same calibration cycle and by launching experiments one after another.

5.7.1 Figure-of-merit: Trial Reduction Factor

Multi-programming is designed to improve machine throughput. When the throughput increases, the effective number of trials performed on both programs combined is reduced. We define *Trial Reduction Factor* (TRF) as the ratio of the number of trials performed when both programs execute individually (baseline) to the total number of trials performed when the programs share resources. If $T_i^{independent}$ is the number of trials performed when the i^{th} program is executed independently and T_{ij}^{shared} is the number of trials performed when i^{th} and j^{th} programs share resources, their TRF is defined by (Equation 5.5).

$$TRF_{ij} = \frac{T_i^{independent} + T_j^{independent}}{T_{ij}^{shared}} \quad (5.5)$$

5.7.2 Quantifying Application Fidelity

To evaluate application fidelity, we use the metric: *Probability of a Successful Trial* (PST), defined as the ratio of number of successful trials to the total number of trials performed [47], as described in (Equation 5.6). For greater application fidelity, a higher PST is desirable.

$$PST = \frac{\text{Number of successful trials}}{\text{Total number of trials}} \quad (5.6)$$

The benchmarks used in the evaluations produce a single correct answer and therefore, PST is equivalent to the fidelity computed using any distance-based metrics such as Total Variational Distance (TVD) or Hellinger Distance (HD).

³IBM Q16 Melbourne was the largest freely available quantum computer in the year 2018.

5.7.3 NISQ Benchmarks

For the baseline, we use NISQ benchmarks derived from prior works on noise-aware compilation policies [48, 47] described in Table 5.1. As out of 14 qubits on IBM Q16, only a few of them offer low gate and measurement error rates, small benchmarks are used for the evaluations. For the evaluation of our proposed policies for multi-programming, we use a set of workload mixes derived from the NISQ benchmarks specified in Table 5.1.

Table 5.1: NISQ Benchmarks

Benchmark	Description	Qubits	Number of Instructions	Number of CNOTs
bv_n3	Bernstein-Vazirani [157]	3	8	2
bv_n4	Bernstein-Vazirani [157]	4	11	3
Toffoli_n3	Toffoli gate	3	15	6
Fredkin_n3	Fredkin gate	3	16	8
Peres_n3	Peres gate	3	16	7

5.7.4 Baseline Setup

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

5.7.5 Overview of the Proposed Framework

The evaluation framework is shown in Figure 5.9. It accepts two workloads W1 and W2 (equivalent to two independent jobs) and the most recent calibration data. Depending upon the number of qubits required for each workload, the partitioning algorithm decides if they can both be executed reliably on the given quantum computer. If there exist two regions where W1 and W2 can be mapped and executed reliably, they are compiled together using the qubit allocations received from the partitioning algorithm and executed. The PST of

each individual workload is calculated. The impact on reliability of each program is computed by comparing with the PST obtained by individually executing the same program using the best qubit allocation (baseline).

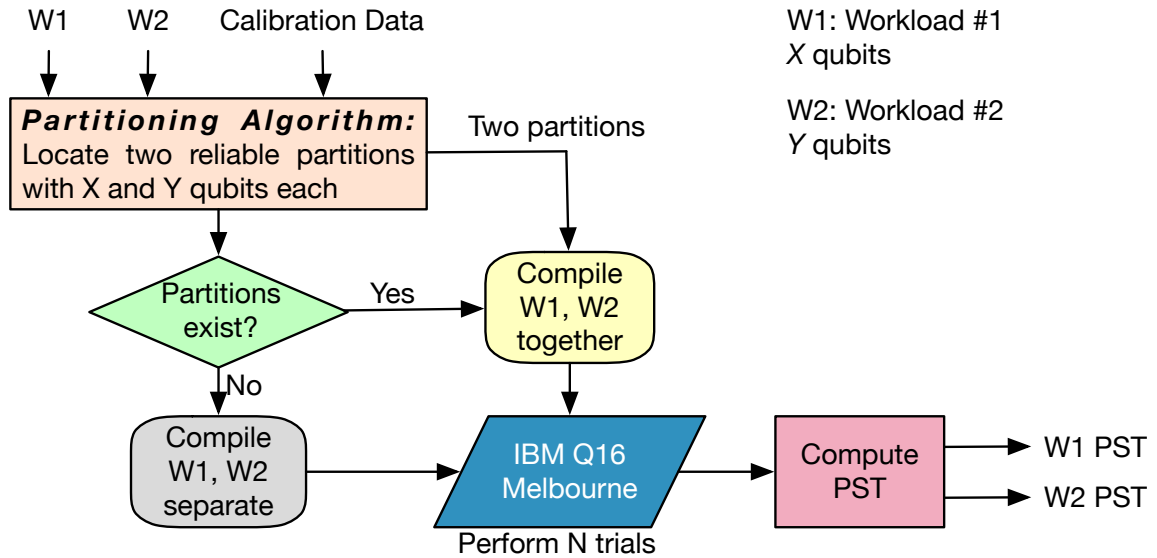


Figure 5.9: Overview of the proposed multi-programming framework. The partitioning algorithm locates two reliable regions on the NISQ computer, with X and Y qubits each. If it can find two such regions, both workloads execute together. If it is unable to locate the requested regions, it defaults to the baseline and each benchmark is run individually.

5.8 Final Evaluations

This section presents the evaluation results for our proposed policies.

5.8.1 Impact of Fair and Reliable Partitioning Algorithm

With multi-programming for certain workload sizes on IBM Q16, the resource utilization and throughput can be improved by 2x with a slight loss of program fidelity. The Trial Reduction Factor (TRF) is 0.5, indicating that the throughput of the machine is doubled and the overall number of trials is reduced by 50%. Table 5.2 compares the PST of each individual workload for the baseline and in a multi-programmed environment, highlighting the impact of application fidelity in the shared execution environment.

Table 5.2: Probability of Successful Trial (PST) under isolated execution and in a multi-programmed environment using the FRP algorithm.

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

Both Figure 5.10 and Table 5.2 show that the PST decreases by 11% on an average and by up to 22% in the worst case even when the error rates are very high and there exists a large variance in the error rates as shown in Figure 5.11. The worst case degradation in PST motivates us to design policies that can minimize the impact of multi-programming on fidelity. Our proposed DIS policy and AMP design techniques reduce this impact.

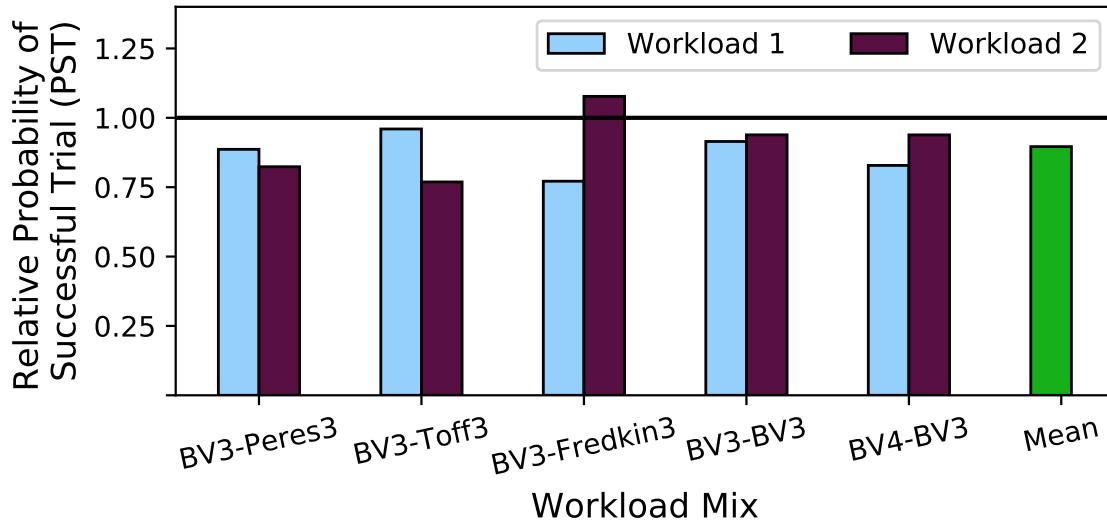


Figure 5.10: Probability of Successful Trial (PST) relative to baseline for multi-programming model

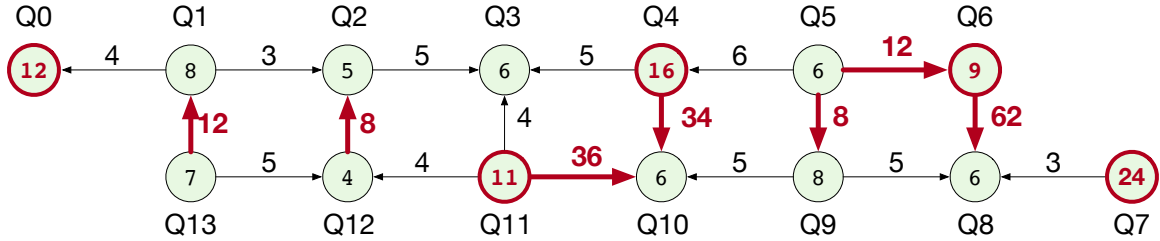


Figure 5.11: Error rates on IBM Q16 during multi-programming experiments

5.8.2 Impact of Delayed Instruction Scheduling Policy

Figure 5.12 shows the relative PST for the different measurement operation scheduling policies relative to the execution of the program in an isolated environment. When the FRP algorithm is combined with the DIS policy, multi-programming is even more effective due to reduced impact on the application fidelity. Overall, the TRF is still 0.5 and the fidelity drops by 10% on average. However, we see an improvement in the reliability of the longer program (3% average improvement in PST) and the worst-case degradation in PST is about 22%. Note that these experiments were performed on one of the earliest generations of quantum computers with very high error rates.

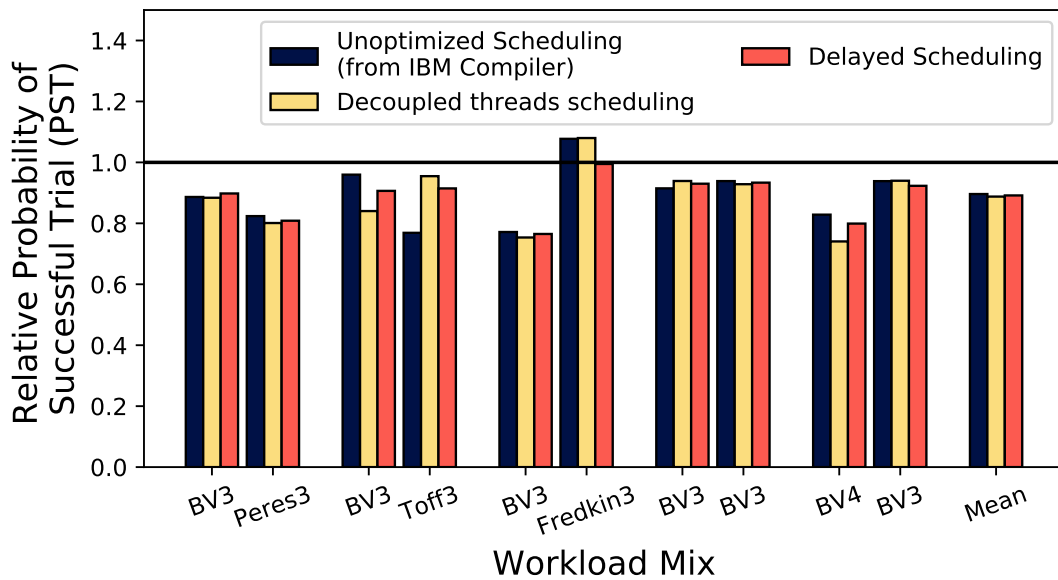


Figure 5.12: Probability of Successful Trial (PST) of proposed delayed instruction scheduling policy relative to baseline

5.8.3 Impact of Adaptive Multi-Programming

The TRF and fidelity vary depending upon the selected multi-programming mode as shown in Figure 5.13 and Table 5.3 respectively. The average impact on reliability is within 8% across all the multi-programming modes. In the evaluations presented here, the multi-programming mode S=4K, I=4K incurs minimal impact on application fidelity, and the average and maximum loss in PST are 6% and 12% respectively. Note that extreme PST degradation such as 12% is detected by our dynamic reliability monitor.

Table 5.3: Probability of Successful Trial (PST) for various modes of multi-programming

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

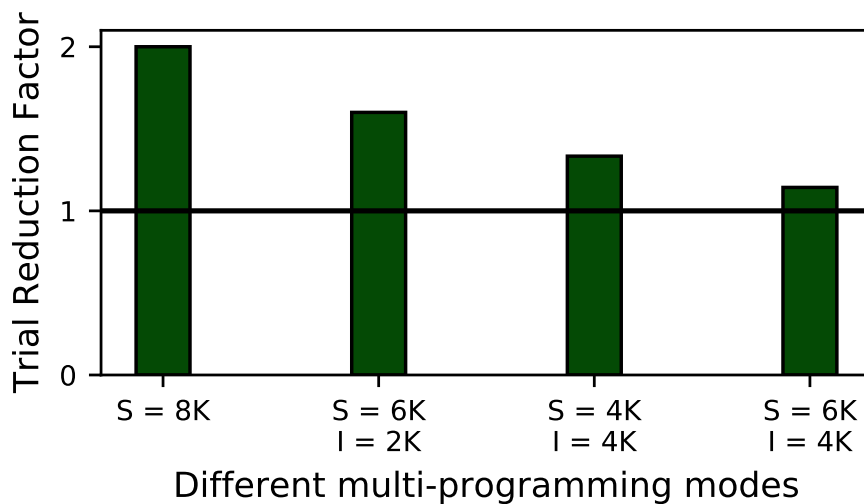


Figure 5.13: Effective reduction in the number of trials enabled by multi-programming

5.9 Discussion

The evaluations presented in this dissertation were performed on one of the earliest publicly available quantum computers with extremely large error-rates and a relatively naive job scheduling policy in the cloud. In addition to impacting application fidelity in the shared environment, the high error-rates and the limited system size restricted our evaluations to only co-running two benchmarks. This quantum computer also did not allow mid-circuit measurement operations with high accuracy. The capabilities of quantum hardware and quantum cloud services have evolved in the last couple of years and therefore, the proposed solutions may have to be adapted for future generations of quantum computers. For example, more recent quantum architectures support multiplexed readout operations and mid-circuit measurements with greater fidelity and it would be interesting to see how the DIS policy would need to be adapted for such systems. Similarly, now there exist several quantum cloud service providers, each with their own job scheduling policies and the AMP design must be adapted to be compatible with these policies.

Key Takeaways

- Quantum resources are scarce which leads to long cloud latencies.
- Quantum resources are typically under-utilized owing to high error-rates.
- Multi-programming quantum computers can enable us to improve the utilization and throughput of quantum computers, enabling us to execute more programs per unit time.
- To multi-program a quantum computer with negligible impact on application fidelity, the Fair and Reliable Partitioning algorithm ensures the allocation of high-fidelity qubits to each of the co-running applications. The Delayed Instruction Scheduling policy enables context-aware instruction scheduling to minimize interference between applications in the shared environment.
- The Adaptive Multi-Programming design detects any substantial degradation in application fidelity at runtime and reverts a program to isolated execution mode.

5.10 Evolution of the Multi-Programming Landscape and Follow-Up Works

The first proposal for multi-programming quantum computers was presented in 2019 based on the work presented in this dissertation [144]. The idea was evaluated on one of the largest publicly available quantum computers, the IBM Q16 Melbourne, available during that time. However, the application fidelity degraded slightly on this machine owing to very high error-rates. In the following year, Liu et al. proposed a tree-based resource allocation technique that reduced the impact of multi-programming on application fidelity compared to the original proposal [197]. In the same year, Murali et al. presented an instruction scheduling policy for mitigating CNOT-CNOT crosstalk between ongoing concurrent gate operations [90] by characterizing high crosstalk link-pairs and scheduling CNOT operations on those links simultaneously, even if it is possible to schedule them in parallel. By drawing insights from this work, Niu et al. proposed advanced resource allocation and instruction scheduling policies for multi-programming [198, 199, 200]. On the hardware side, the error rates of the devices improved, leading to a significantly lower impact of multi-programming on application fidelity. In subsequent years, other advanced policies for multi-programming have been proposed [201, 202]. Recent multi-programming studies on trapped-ion systems (with much higher operational fidelities compared to superconducting systems) show that multi-programming has negligible impact on application fidelity [203]. Other works have proposed how multi-programming can be specifically tailored to improve the throughput of variational quantum algorithms [204, 205, 206] and Grover’s search algorithm [207]. Multi-programming techniques are also being developed for quantum annealers [208, 209]. Most recently, job scheduling policies for multi-programming in cloud environments with access to multiple devices have been explored [210, 211]. Researchers have noted that crosstalk can enable an adversary to learn about a co-running quantum circuit in multi-programmed environments [212, 213] and developing solutions for multi-programming in a secure manner remains an active area of research [214, 215, 216, 217].

CHAPTER 6

EFFICIENT REAL-TIME DECODING SOLUTIONS FOR QUANTUM ERROR CORRECTION ON EMERGING QUANTUM COMPUTERS

This chapter of the dissertation presents *LILLIPUT* [218], a Lightweight Low Latency Look-Up Table decoder, for decoding small surface codes in real-time. It achieves the accuracy of the idealized minimum weight perfect matching decoder with a latency of up to 42 nanoseconds. *LILLIPUT* requires up-to 7% logic utilization on off-the-shelf FPGAs and thus, is lightweight and suitable for practical adoption.

6.1 Motivation

QEC is essential to run the most practical quantum algorithms. As the device quality and system size continues to improve, there is increasing interest in studying QEC codes and real-time decoding. Several demonstrations of repetition codes, Bacon-Shor code have been successful [219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 170, 229, 230]. While these experiments represent a significant milestone in QEC, unlike surface codes, these codes can only correct either phase-flip or bit-flip errors but not both. In the year 2022, Google demonstrated exponential suppression of errors using surface codes, using redundancy up to distance 5.¹ However, QEC experiments so far have mainly relied on offline software-based error decoding. [170, 44, 231, 232]. With improving device quality, quantum hardware has currently reached the level of 1% error per data qubit per syndrome cycle, where QEC can function (involving surface codes of distance 3 and beyond). Thus, QEC experiments that demonstrate small surface codes using real-time decoding are a reasonable major milestone for quantum computing in the next few years. Figure 6.1 shows a summary of the scale of experimental demonstrations of QEC codes on real systems so far.

¹*LILLIPUT* was done prior to the surface code demonstrations from Google (using offline decoding).

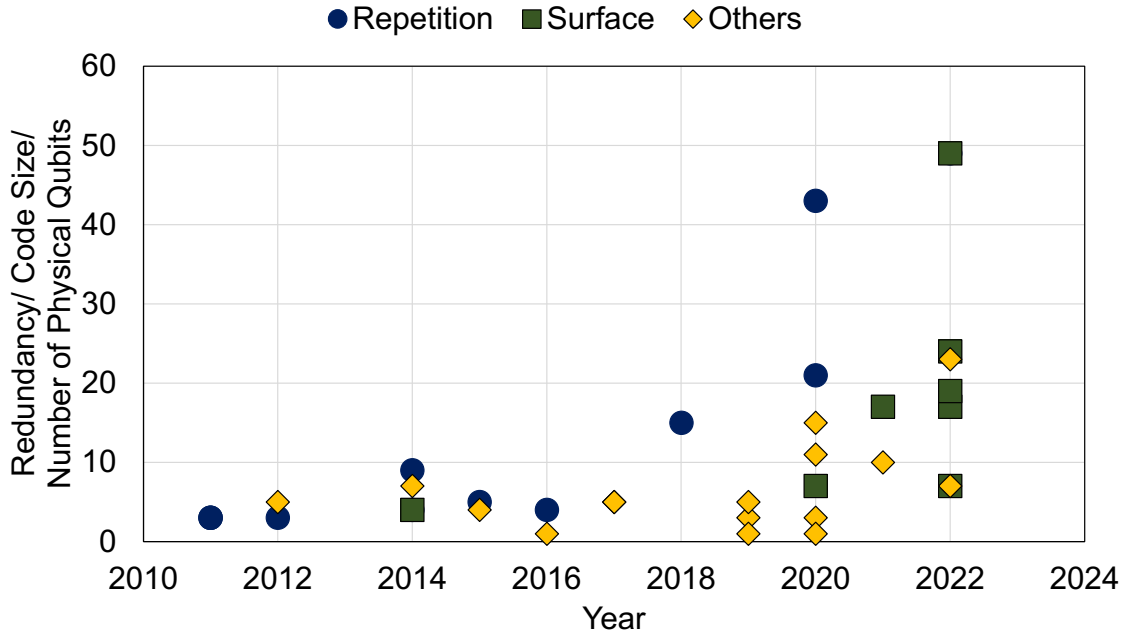


Figure 6.1: Summary of demonstrations of QEC codes on real systems.

6.2 Challenges in Real-Time Decoding

Real-time decoding is necessary to prevent the accumulation of errors on data qubits. If errors are not corrected within a QEC cycle before the arrival of the next syndrome, errors may accumulate resulting in a logical failure. The decoding complexity depends on the error events and software decoders may be too slow for real-time decoding [136]. They also incur significant communication overheads in transmitting the syndromes into software running on room-temperature CPUs. This high latency of software decoders limits the use of general-purpose computing for online decoding and therefore, almost all QEC experiments that use software decoders have resorted to offline decoding. In the first instance of real-time decoding [233], software decoders have been used for color codes [234]. However, this study uses trapped-ion systems that can tolerate up to a few milliseconds of decoding latency, about 3 orders of magnitude higher than superconducting systems. The alternative is hardware decoders that are faster and promise real-time decoding. However, they have poor accuracy due to algorithmic and implementation limitations and rely on

custom hardware [137] or specialized devices [115, 138]. Custom accelerators (ASICs) are impractical for adoption on emerging quantum systems as we are not at a stage where a company would make millions of quantum computers to make ASICs viable. On the other hand, given the current state of superconducting device technology, it is not even feasible to implement the SFQ decoders [115, 138] in the near-term. The number of devices required to fabricate these decoders far exceeds the device densities of existing superconducting device technologies, making them unsuitable for adoption in emerging quantum systems.

6.3 Overview of Proposed Solution: LILLIPUT

The classical counterpart of QEC consists of three key steps– (1) error detection from stabilizer measurements, (2) identification of errors and error assignment to data qubits every cycle, and (3) computation of logical error.² Figure 6.2 shows the micro-architecture of LILLIPUT that accomplishes these steps. It communicates with the readout interface, translates the stabilizer measurement outcomes into error detection events, identifies errors, and computes the logical error. Note that X and Z errors are decoded independently. In the next subsections, the implementation details of LILLIPUT are described.

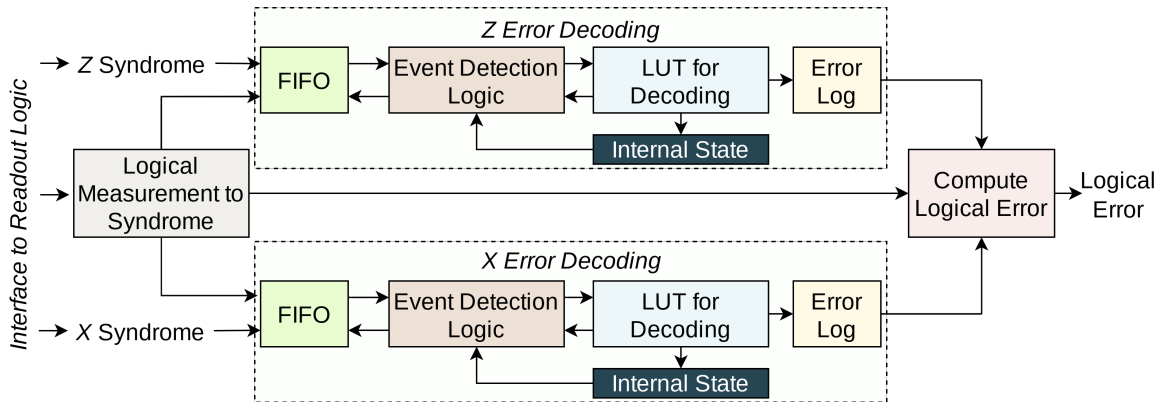


Figure 6.2: Overview of LILLIPUT.

²QEC studies on emerging systems typically focus on memory or state-preservation experiments that prepare an initial state, inject errors, and measure the data qubits to perform a logical measurement. If there is no logical failure, the output of the logical measurement must match the prepared initial state.

6.3.1 Detection of Errors from the Stabilizer Measurements

In QEC, syndromes are generated every cycle by measuring the stabilizers. LILLIPUT generates *error detection events* by comparing the stabilizer measurement outcomes from two consecutive QEC cycles. Any change in the measurement outcome of the stabilizers between two cycles indicates an error. Alternately, no event is detected if the stabilizer measurement outcomes remain the same. For example, Figure 6.3 shows Z stabilizer measurements outcomes of a distance 3 surface code, where an error event is detected in cycles 1 and 3, whereas no error event is detected in cycle 2. This step is accomplished by the Event Detection Logic block shown in Figure 6.2. Detection events enable us to track errors in a given cycle and are used to identify the optimal correction.

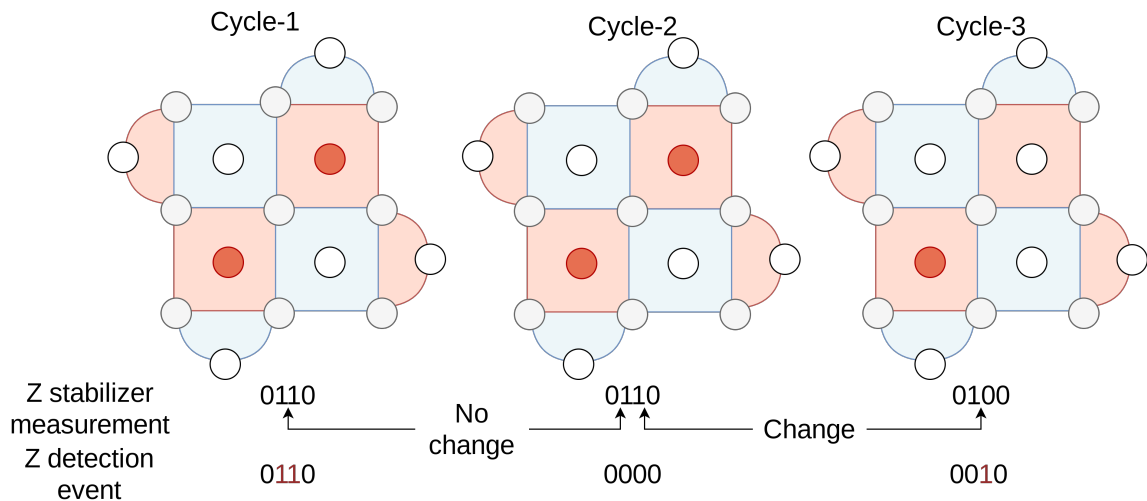


Figure 6.3: Steps in translating the stabilizer measurement outcomes (red denotes an error is identified) to error detection events in each QEC cycle. The bitstream for stabilizer measurements is specified from left to right (by convention) on the surface code lattice.

6.3.2 Error Identification as a Matching Problem

Error identification is the step in which a decoder assigns errors to each data qubit which is tracked throughout the QEC cycles. To perform this step, the detection events are represented on a graph, often termed as the *decoding graph*, where the nodes and edges represent the parity and data qubits respectively. The minimum weight perfect matching al-

gorithm [104] uses the decoding graph and matches each detection node with another or to the surface code boundary such that the total weight of the matched edges is minimal. MWPM is widely considered to be the most accurate algorithm. A more recent method, the Union-Find algorithm [135, 134], uses a different approach to matching to generate the error assignments. It is faster than MWPM but has lower accuracy [235]. In LILLIPUT, instead of dynamically assigning errors in real-time, the matching step is accomplished using Look-Up Tables (LUTs), as shown in Figure 6.2 and described in Section 6.4.

6.3.3 Handling Data and Measurement Errors

For greater accuracy, a decoder must handle both errors on the data qubits as well as errors in the syndrome extraction circuit. Overall, there are four key sources of errors: (a) errors on data qubits (b) errors in the gate operations during syndrome extraction (c) measurement errors on parity qubits during stabilizer measurements, and (d) measurement errors on data qubits during logical measurement. Below is a description of the steps related to how LILLIPUT handles each of these errors, where the “space” and “time” directions are used to describe when detection events are generated in the same round or two consecutive cycles, respectively.

(a) An error on a data qubit is detected by its neighboring parity qubits, producing a *space-like* detection event on them in the same cycle. For example, Figure 6.4(a) shows a space-like error detection event in cycle 1.

(b) Two-qubit gate errors during syndrome extraction produce *space-time like* detection events, as shown in cycles 1 and 2 in Figure 6.4(b).

(c) Measurement errors on parity qubits exhibit temporal behavior and translate into *time-like* detection events in two consecutive cycles, as shown in cycles 2 and 3 of Figure 6.4(b).

(d) Measurement errors on the data qubits are handled in the last boundary cycle by generating an extra syndrome of either X or Z type and are discussed in the next subsection.

To tackle the first three sources of errors, LILLIPUT processes more than a single cycle of the syndrome and performs matching on a 3-dimensional decoding graph spanning space and time (multiple cycles). For errors in logical measurement, LILLIPUT uses an extra syndrome as discussed in Section subsection 6.3.4.

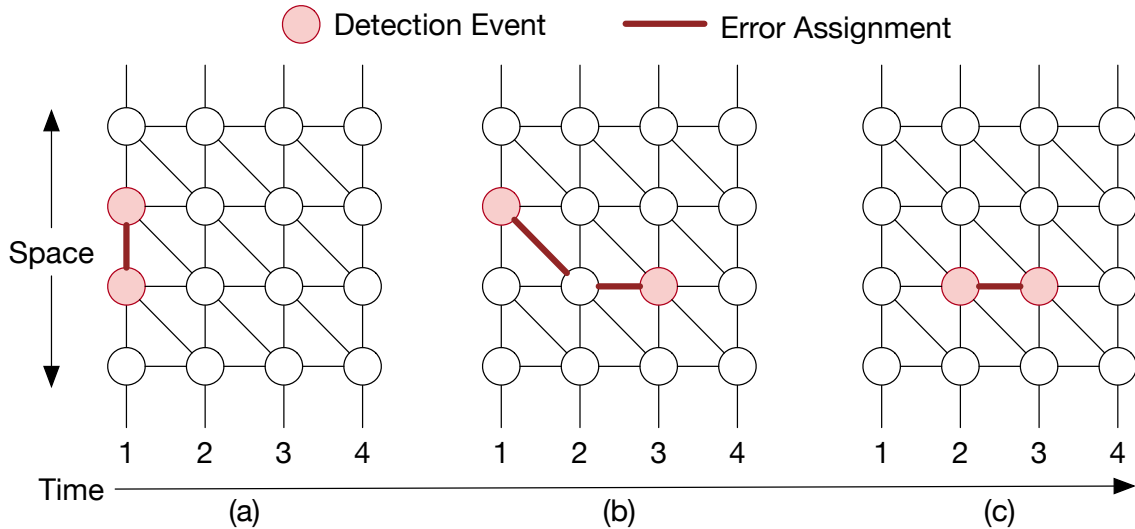


Figure 6.4: A distance 4 repetition code decoding graph, where nodes and edges denote parity and data qubits respectively. Note that this decoding graph is for the purpose of illustration only and our evaluations in this paper are based on surface codes. Examples of (a) space-like (b) space-time like and (c) time-like detection events.

LILLIPUT assumes the readout logic writes the measurement outcomes on a buffer which gets overwritten every cycle. This interface works on a much slower clock (e.g., with $1 \mu s$ latency) that depends on the duration of the syndrome extraction circuit. LILLIPUT polls for new syndromes and tracks the most recent history in a FIFO, as shown in Figure 6.2. The FIFO must store the history of the last m cycles, where m is the number of syndrome rounds that are simultaneously decoded. Since LILLIPUT decodes in real-time, storing only the last m rounds is sufficient. The impact of the number of measurement rounds on the logical error rate is discussed in Section 6.6.

6.3.4 Handling Boundary Cycles

A QEC experiment has two time-boundary cycles- one at the beginning and the other at the end. In QEC, cycles are also called *rounds*. The term cycle is used in this dissertation to avoid any confusion with the number of syndrome rounds used for decoding. For the beginning time boundary cycle, the detection event is computed by comparing the first round of stabilizer measurements and the qubit initialization data. In the last time boundary cycle, the data qubits are measured. To tolerate measurement errors in data qubits, LILLIPUT translates the measurement outcomes of the data qubits into a detection event of either X or Z type by comparing them with the stabilizer measurement data from the second-last cycle. Since a logical measurement in surface codes is implemented by performing a transversal measurement (measuring all the data qubits in either X or Z basis), the measurement basis of the data qubits determines the type of stabilizer that can be constructed from the measurement outcomes. By default, LILLIPUT translates the data qubit measurement outcomes into a Z syndrome because of the assumption (for simplicity) that the data qubits are measured on the Z basis, as shown in Figure 6.2.

6.4 Error Assignments in LILLIPUT

Instead of performing minimum weight perfect matching at runtime, LILLIPUT determines the optimal error assignment using Look-Up Tables (LUTs). The LUTs are programmed offline before the experiment by determining the error assignments corresponding to all possible error detection events. Every cycle, the history of detection events is used to index an LUT and the LUT entry assigns errors to the data qubits in the oldest cycle. The length of the history depends on the number of syndrome rounds used for decoding. LILLIPUT also maintains an *internal state* which is updated every cycle. The details of using LUTs in LILLIPUT and tracking the internal state are described next.

6.4.1 Streaming Mode of Operation

LILLIPUT operates in *streaming* mode and maintains an *error log* for each data qubit. The log is updated every cycle as errors are identified. LILLIPUT considers a fixed number of cycles at a time which is equal to the number of syndrome rounds used in decoding. We call this a *sliding window* because it slides forward as an experiment proceeds. For example, Figure 6.5 shows the sliding window for four consecutive decoding steps. LILLIPUT uses the detection events in the sliding window and the internal state to determine the LUT address every cycle. The error assignment is obtained from the corresponding LUT entry.

6.4.2 Error Assignment

LILLIPUT determines the best correction by using all the detection events in each sliding window and assigns errors only to the oldest layer. For example, Figure 6.5 shows the subset of edges that the decoder uses to make the error assignment for the current sliding window. The optimal error assignment is stored in the LUTs and targets to *neutralize* the effect of all the errors in the oldest cycle and decoding completes once the sliding window covers all the syndromes. The error assignments are made only to the oldest layer in each sliding window to prevent any *premature matching*. Premature matching may result in sub-optimal performance if syndromes in future cycles can change the assignment. Premature matching may also result if we use disjoint windows where the window proceeds forward by its full length. In that case, errors that span over two non-overlapping windows can cause inaccurate error assignments. As LILLIPUT uses a sliding window, it does not encounter this problem and therefore, does not suffer from sub-optimal performance. For the time boundary cycles, LILLIPUT constructs a full sliding window by padding zeros. Figure 6.6(c) shows a time boundary cycle where additional zeros are padded. Since the LUT is programmed such that, it assigns the most optimal error for a sliding window, it is guaranteed to never assign errors to the zero-padded regions.

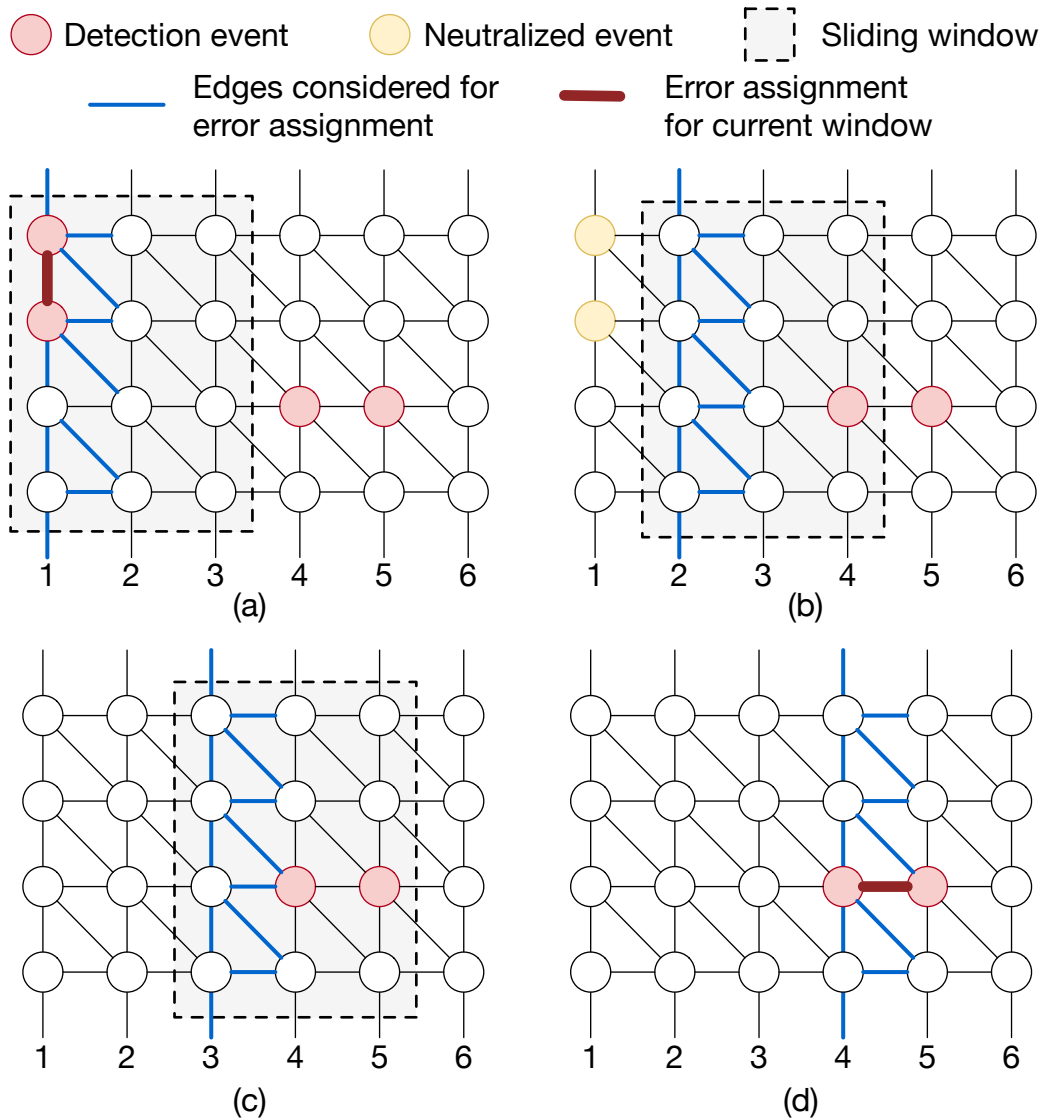


Figure 6.5: (a) The sliding window uses detection events from cycles 1, 2, and 3 and assigns error to the oldest layer i.e., cycle 1. Note that all detection events in the window are considered for matching but only the edges touching the oldest cycle are considered during error assignments, shown in blue. (b-d) The sliding window streams forward one cycle at a time.

6.4.3 Tracking the Internal State

LILLIPUT assigns errors only to the oldest layer. However, as it performs matching across all the events in a sliding window, it can neutralize some errors in the second oldest cycle if the detection event has a time-like or space-time like behavior. For example, Figure 6.5(d) shows a time-like detection event that requires neutralizing events in both cycles 4 (oldest

cycle) and 5 (second oldest cycle). Consequently, matching within this window removes the detection event from the second oldest layer. Similarly, detection events may be added as well. For example, Figure 6.6(a) shows an example of decoding steps where detection events are added. To accommodate these scenarios, LILLIPUT maintains a record of the detection events added or removed in an internal state register. To obtain the LUT address to be accessed in a cycle, LILLIPUT concatenates the detection events in the current window and modifies the oldest detection event using the most recent internal state, as shown in Figure 6.6. Each LUT entry provides the error assignment for the oldest layer as well as the detection events that are added or removed in the current cycle (1 bit per parity qubit). This value is used to update the internal state register every cycle.

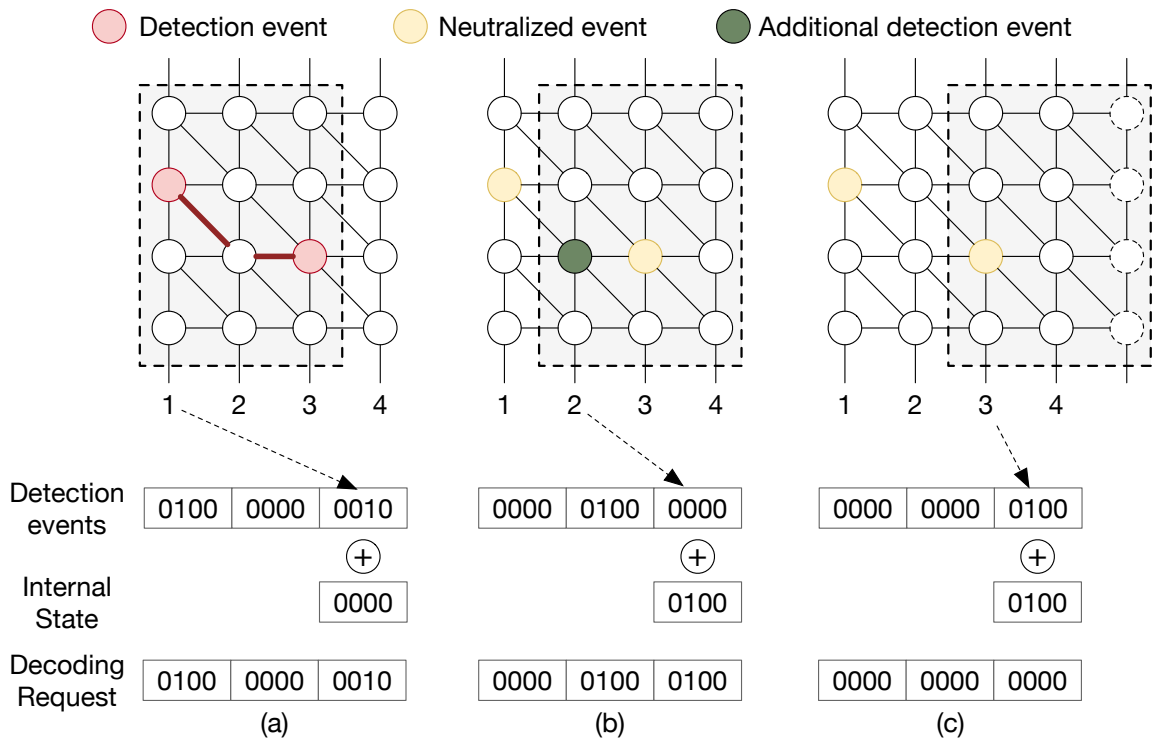


Figure 6.6: (a) LILLIPUT finds the optimal matching in the current window but adds a detection event in cycle 2 which is tracked in the internal state register. (b-c) The additional event is considered from the register. In the boundary cycle, zeros are padded. By convention, detection events of the oldest cycles occupy the least significant bits in our design. The bits for each cycle run bottom to top from the most significant position to the least.

6.4.4 Programming the LUT

LILLIPUT programs the LUT entries offline by generating all possible detection events for a given code distance and size of the sliding window. We use a software Minimum Weight Perfect Matching (MWPM) decoder [236]. For some events, the MWPM decoder may provide more than one possible error assignment. In other words, a single detection event may have multiple matching possibilities on the decoding graph that result in minimal weight. For example, Figure 6.7 shows possible error assignments on a distance 4 surface code lattice for the same detection event. Here, the Z stabilizer at the center of the lattice indicates an error. The MWPM decoder can assign X errors to either data qubits (a) E and F or (b) G and H or (c) I and J. For high accuracy, we account for the error model of the quantum hardware (we consider Google Sycamore [170]) to select the most probable error assignment. LILLIPUT can also accommodate variability in device error rates [47, 48] by re-programming the LUTs. As device characteristics remain stable over short periods of time and mainly exhibit variability over extended periods (weeks or months) [237], the LUTs need not be re-programmed frequently. The reconfigurability allows LILLIPUT to be adapted to other QEC codes, decoding algorithms, and quantum systems.

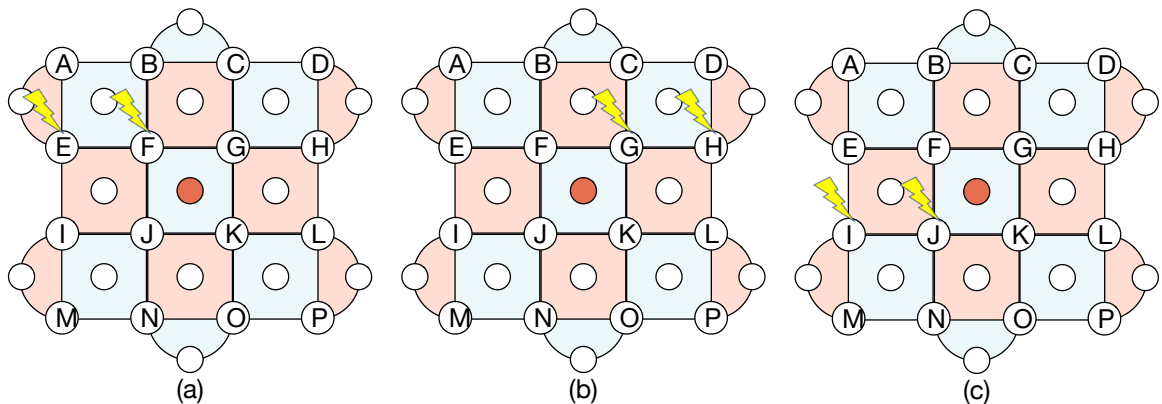


Figure 6.7: Valid possible error assignments for the detection event at the center of the distance 4 surface code lattice.

6.4.5 Decoding in Presence of Non-Clifford Gates

We design LILLIPUT to operate on surface codes that use magic states for non-Clifford gates [238, 107]. A magic state is a resource state created using an unprotected physical non-Clifford gate that is ‘injected’ at the creation step of a surface-code logical qubit. A non-Clifford gate, such as a T gate, is enacted by consuming this encoded magic state. This consumption process, which can include distillation of errors from magic states, is composed entirely of Clifford gates and measurements, as described in Section-XVI of [238].

An error in the creation of a magic state cannot be detected by any decoder, which is an unavoidable problem with state injection. To remedy this, an ensemble of noisy magic states is ‘distilled’ into one higher-fidelity magic state [239, 107]. Distillation is implemented by a circuit of Clifford operations, so this can be decoded using LILLIPUT (or another suitable decoder). Hence, LILLIPUT can support universal fault-tolerant logic in the surface code because all decoding occurs within Clifford gates.

6.4.6 Determination of Logical Error

Early demonstrations of QEC will perform a memory experiment, simply showing that error correction preserves the initial quantum state. The probability of logical error is determined by comparing the logical measurement outcome with the expected outcome for the state prepared. This logical measurement is computed using the error log. Which error log (X or Z) is used depends on the measurement basis of the logical measurement. The logical measurement bit is computed by using a reduction XOR operation on the bitwise XOR results of the error log and the logical measurement outcome. If the data qubits are finally measured on the X basis, the X error log is used. Alternately, if the logical measurement is performed on the Z basis, the Z error log is used.

6.5 Evaluation Methodology

In this section, the benchmarks, experimental setup, and the figure-of-merit used to evaluate LILLIPUT are described.

6.5.1 Surface Code Parameters

Rotated surface codes of distances 3, 4, and 5 are considered for the purpose of evaluations of LILLIPUT, presented in this dissertation. The details of the layouts are described in Table 6.1. The total number of physical qubits required ranges from 17 to 49. Emerging quantum hardware with a few hundred qubits would be able to fit these layouts.

Table 6.1: Parameters of Rotated Surface Codes

Code Distance	Data Qubits	X-ancilla Qubits	Z ancilla Qubits	Total Physical Qubits
3	9	4	4	17
4	16	8	7	31
5	25	12	12	49

6.5.2 Monte Carlo Simulation Infrastructure

Figure 6.8 shows an overview of the Monte Carlo simulator used. The simulator generates a surface code lattice for a given distance. Depending on the noise model, the simulator injects errors onto the data qubits and measurement errors onto the parity qubits, producing a syndrome every cycle. The X and Z syndromes are then decoded independently, and an error log is maintained for both error types. To model a QEC experiment, the simulator repeats the process for multiple cycles and terminates when the maximum number of cycles is reached. The simulator maintains the internal state of the qubits throughout the experiment which is then used to compute the logical measurement outcome and the logical error. Each such execution is called a *trial*. For evaluations, 1 million random trials are used.

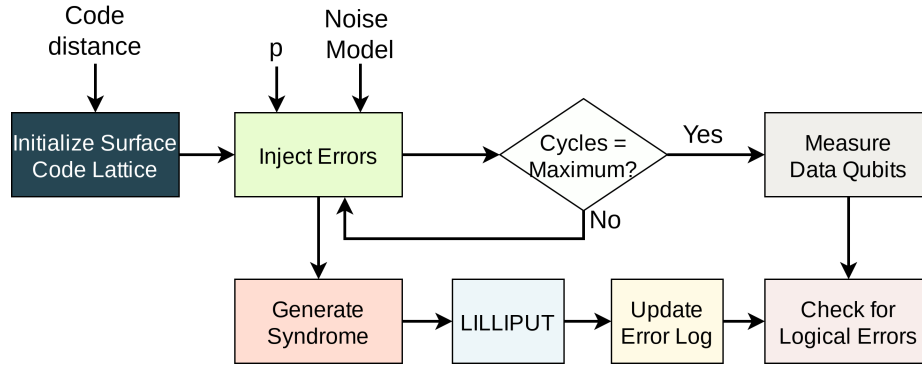


Figure 6.8: Monte Carlo simulation framework.

6.5.3 Noise Model

The *phenomenological noise model* [106], which inserts errors on both data qubits and on the measurement of parity qubits is used. This is a standard noise model in QEC and has been used in several prior works. In this noise model, a data qubit encounters an error with probability p in each cycle. The type of error is chosen uniformly from Pauli X, Y, and Z errors. Additionally, each parity qubit encounters measurement error with probability p . For simplicity, the probabilities of data qubit errors are assumed to be the same as measurement errors. This is consistent with prior works in QEC. Physical error rates ranging from $p = 10^{-3}$ to $p = 5 \times 10^{-2}$ are considered as this is a suitable range of error rates for emerging quantum systems. Nonetheless, if the device quality improves further, LILLIPUT can still support those quantum architectures.

6.5.4 Target Hardware Platforms

Our target is to implement the decoder on off-the-shelf FPGAs as existing quantum systems already use FPGAs for control and readout interface logic [60, 240]. For our studies, we use FPGA from the Intel Cyclone 10 LP [241], Arria V [242], and Stratix 10 [243] family as these are commercially available.

6.6 Final Evaluations

In this section, the accuracy, latency, and hardware complexity of LILLIPUT for the different decoder configurations are discussed.

6.6.1 Results for Accuracy

Figure 6.9 shows the logical error rate (LER) for distances (d) 3 and 4 respectively. By default, each experiment consists of 5 cycles. The LER scales $O(p^2)$ which is expected because distances 3 and 4 can correct at least one error, but sometimes fail with two errors; quadratic scaling results from errors being independent in the model used for simulation.

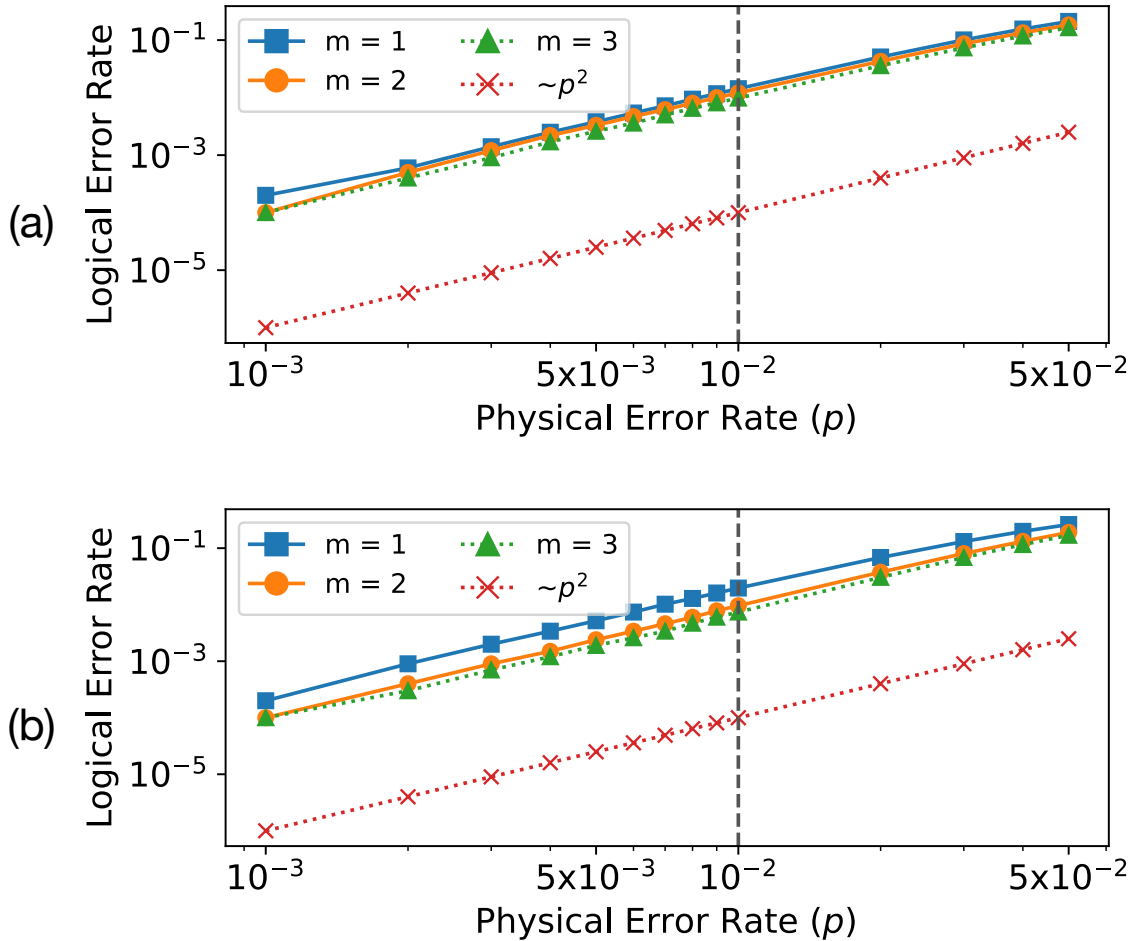


Figure 6.9: Accuracy of distance (a) 3 and (b) 4 surface codes for the different number of syndrome measurement rounds (m). Note that the p^2 line is only for visual air purposes.

Figure 6.9 also shows the impact of the number of syndrome rounds on decoding accuracy. While $m = (d - 1)$ rounds are needed to detect as many errors as the code is capable of correcting [108], the performance of LILLIPUT saturates at $m = 2$ syndrome rounds for $d = 4$. The LER reduces by 1.23x and 1.21x on average for distance 3 by going from 1 to 2 and 2 to 3 rounds respectively. Similarly, the LER reduces by 1.99x and 1.24x on average for distance 4 by going from 1 to 2 and 2 to 3 rounds respectively.

Figure 6.10 shows the logical error rate for distance 5 surface codes using 1 and 2 rounds of syndrome measurements. The number of rounds is limited to 2 to keep the LUT sizes tractable for the simulations. The LER scales $O(p^2)$, which is expected as using just $m = 2$ rounds will lead to some configurations of two errors that span beyond two rounds not being accurately corrected by the decoder.

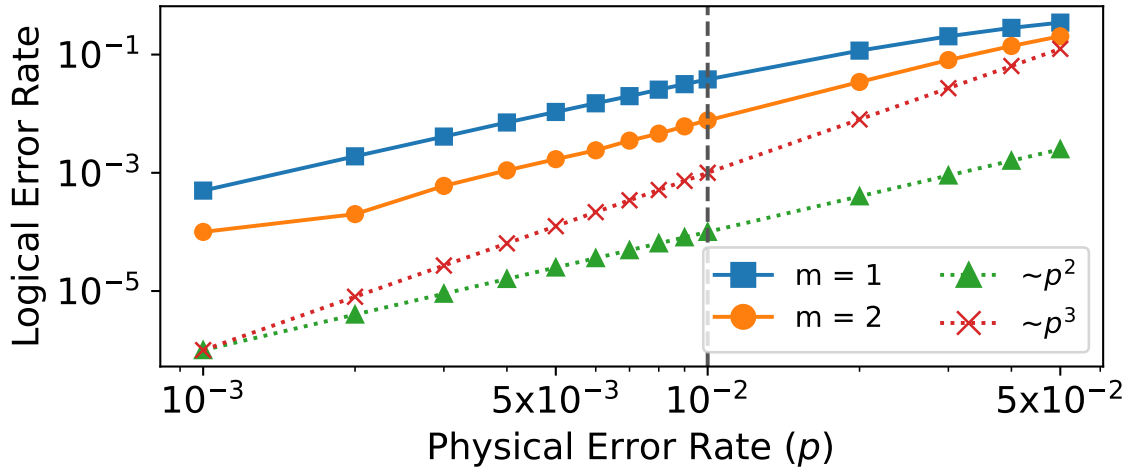


Figure 6.10: Accuracy of distance 5 surface codes for different number of syndrome measurement rounds (m).

The impact of the number of cycles on the LER of the same decoder configurations is shown in Figure 6.11. This analysis allows a user to budget cycles in a QEC experiment based on the device error rates by accounting for the decoder performance- when the error rates are too high, fewer cycles can be supported, whereas experiments can be set up with a greater number of cycles when errors are relatively infrequent.

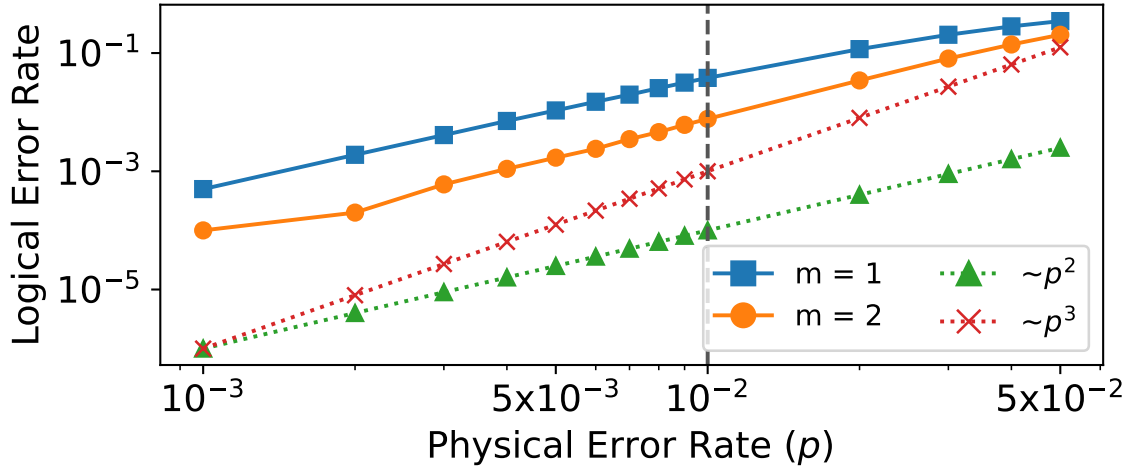


Figure 6.11: Impact of number of cycles on LER.

6.6.2 Results for Hardware Complexity

The key component of LILLIPUT is the LUTs for both types of errors (X and Z). The size of each LUT depends on the *decoder configuration*. We denote a decoder configuration as $[d, m]$, where d and m are the code distance and number of syndrome rounds considered for decoding respectively. The address size is the length of the detection event which is equal to the syndrome length multiplied by the number of syndrome rounds. The length of each LUT entry is equal to the sum of the number of data (for error assignment) and parity qubits (for internal state). Table Table 6.2 shows the size of LUTs for different decoder configurations. Asymmetry in the code leads to X and Z syndrome lengths being different for distance 4, so the LUT sizes are different for X and Z syndromes. The Cyclone 10 LP family of FPGAs is used for the decoder configurations $[d = 3, m = 2]$, $[d = 3, m = 3]$, and $[d = 4, m = 2]$ as it only supports up to 486 KB of embedded memory [241]. To support configurations with larger LUTs such as $[d = 4, m = 3]$, and $[d = 5, m = 2]$, the Arria V FPGAs are used where the LUTs are accessed from an external SRAM (using QDR II) or SDRAM (using DDR2) [242]. Alternately, the LUTs can be compressed using traditional data compression techniques (as will be discussed in Section 6.7) and do not require access to any external memory in that case [243].

Table 6.2: LUTs for different decoder configurations

Decoder Configuration	Address Size	Entry Size	LUT Size	Total Memory	Memory Type
$[d = 3, m = 2]$	8	13	416 B	832 B	Embedded
$[d = 3, m = 3]$	12	13	6.5 KB	13 KB	Embedded
$[d = 4, m = 2]$	14/ 16	23/ 24	46/ 192 KB	238 KB	Embedded
$[d = 4, m = 3]$	21/ 24	23/ 24	5.75/ 48 MB	53.75 MB	External
$[d = 5, m = 2]$	24	37	74 MB	148 MB	External

Table 6.3 shows the FPGA utilization for the different decoder configurations. The Quartus-20.1 tool-chain is used to synthesize the LILLIPUT designs on Intel devices. The logic utilization for LILLIPUT is less than 7%, making it extremely lightweight and leaving enough room for other circuits such as readout interface logic and control logic for delivering instructions to the qubits. The LUTs consume up to 40% of the memory bits for designs that use embedded memory. Different Cyclone devices are used to synthesize the LILLIPUT designs corresponding to distance 3 and $[d = 4, m = 2]$ to enable higher clock frequencies and lower decoding latency (or in other words higher performance). Configuring the distance-3 design on the latter FPGA reduces the maximum frequency to 245 MHz but still enables real-time decoding within a micro-second. From a price point perspective, Arria family of FPGAs is more expensive compared to the Cyclone family of FPGAs.

Table 6.3: FPGA utilization for different configurations

Decoder Configuration	FPGA Family	Total LEs/ ALMs	Total Registers	Utilization	
				Area	Memory
$[d = 3, m = 2]$	Cyclone 10	353	209	6%	1%
$[d = 3, m = 3]$	Cyclone 10	418	239	7%	21%
$[d = 4, m = 2]$	Cyclone 10	557	340	< 1%	40%
$[d = 4, m = 3]$	Arria-V	217	409	< 1%	–
$[d = 5, m = 2]$	Arria-V	246	486	< 1%	–

6.6.3 Results for Latency

The decoding latency typically depends on the error event (which determines the decoding graph) and implementation, so average decoding latency is usually lower than worst-case latency [137]. However, as LILLIPUT requires only a single memory access to the LUT, it incurs a fixed latency irrespective of error detection event. LILLIPUT is a fully pipelined design and requires 7 cycles to determine the correction after a cycle of stabilizer measurements is received. The maximum clock frequency depends on the decoder configuration and is listed in Table 6.4. To compute the latency of the decoder configurations that rely on external memory access, the calculations are based on the most conservative off-chip memory access time estimate (slowest clock frequency and maximum number of cycles to access a memory location) which is added to the latency of the rest of the logic. The decoding latency of LILLIPUT is up-to 24x lower than the target latency of $1\mu\text{seconds}$ in the worst-case (considering the largest decoder configuration studied). LILLIPUT also meets the 400 nanoseconds target latency considered in a few prior works [115, 138, 137] and thus, will remain useful even if the duration of syndrome extraction decreases in the future with improving device quality.

Table 6.4: Maximum Frequency (in MHz) and Latency (in ns)

Metric	Decoder Configurations				
	[d=3,m=2]	[d=3,m=3]	[d=4,m=2]	[d=4,m=3]	[d=5,m=2]
Frequency	250	240.7	209.8	244.4	232.9
Latency	28	29.1	33.4	40.8	42

The low logic utilization and re-usability of the LUTs allow LILLIPUT to support QEC experiments spanning more than one logical qubit. As there is sufficient slack between the decoding latency and the target latency for all decoder configurations studied in this paper, the real-time decoding capability will not be impacted.

6.7 A Case for Compressed LUTs

The size of a lookup table increases exponentially in the code distance because there is an entry for every possible syndrome bitstring. This limits the scalability of LILLIPUT, and some of the decoder configurations studied rely on memory external to the FPGAs. Moreover, it is infeasible to scale the LUT to higher configurations, $[d = 5, m = 4]$ for example. To implement these configurations without requiring external memory or scale LILLIPUT to higher configurations, this dissertation proposes Compressed LUTs (CLUTs).

6.7.1 Not All Error Events are Equally Likely

Each LUT entry corresponds to an error event. However, *not all error events are equally likely*, and Compressed LUTs exploit this using the following insights:

1. On surface codes, errors are detected by neighboring stabilizers. When error rates are low, fewer bit flips are observed in the detected events. Alternately, higher error-rates cause longer chains of errors. However, it results in only a few bit flips overall as the parity qubits inside the chain are likely to be zeros (can be thought of as flipped twice). Thus, the average Hamming weight of the accessed memory addresses is low. For example, the $[d = 3, m = 2]$ configuration requires LUTs with 8-bit address ranging from 0x00 to 0xFF. Figure 6.12 shows the probability distribution of the Hamming weight of addresses accessed over 1 million trials for different error rates and both X and Z LUTs of this decoder configuration. Note that not all addresses are accessed with equal probability, and some are seldom or never accessed (address 0xFF for example). The trend persists even when the number of QEC cycles increases. This is expected because extra cycles lead to more errors and parity qubits flip back and forth between consecutive cycles. *Therefore, the memory required for the LUTs can be reduced by removing entries that are unlikely to be accessed.*
2. For small distances, a decoder can assign only a limited number of errors, the LUT entries contain a large number of zeros. *Thus, the LUT entries themselves are compressible.*

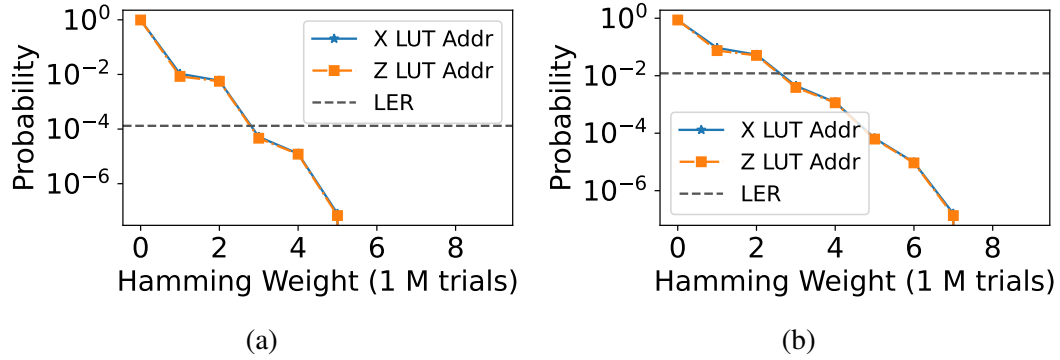


Figure 6.12: Probability distribution of the Hamming weight of memory addresses for (a) $p = 0.1\%$ and (b) $p = 1\%$ for 5 cycles

To summarize, not all LUT entries are accessed with equal probability and therefore, the size of the LUTs can be reduced by eliminating entries that are unlikely to be accessed. Furthermore, the data entries corresponding to the most probable error events that must be stored can be compressed too.

6.7.2 Compressing the LUTs in Software

The scope of compression is discussed using the $[d = 3, m = 2]$ configuration. Figure 6.12 shows that the probability of accessing LUT addresses with 3 or more ones is equal to or lower than the logical error rate. Even if these entries are not stored, the decoder accuracy is unlikely to be affected as the events would result in a logical error with low probability.

To implement the CLUT, we split the address space into two groups— Segments A and B, as shown in Figure Figure 6.13(a). These segments consist of data frames (DFs)- a contiguous block of 16 and 10 entries respectively. A block of 16 consecutive LUT entries is assigned a 16-entry or 10-entry DF depending on the Hamming weight of the addresses of the block. Addresses with all zeros or a single one in the four most significant bits are assigned a 16-entry DF, whereas addresses with two ones in the four most significant bits are assigned a 10-entry DF. For example, as shown in Figure 6.13(a), 0x00 to 0x0F correspond to a 16-entry DF in Segment A, whereas 0xA0 to 0xAA are assigned a 10-entry DF in Segment B. Addresses 0xAB to 0xAF are not stored as their Hamming weight is

more than 3. Although we want to store LUT entries for addresses of Hamming weights up-to 3 only, as memory addresses are not contiguous in terms of Hamming weight, this results in memory fragmentation. Instead, data frames are used that cause some addresses with higher Hamming weights to be stored as well (such as 0x1F), but this ensures a simpler addressing scheme. This reduces the number of entries by 1.83x from 256 to 140.

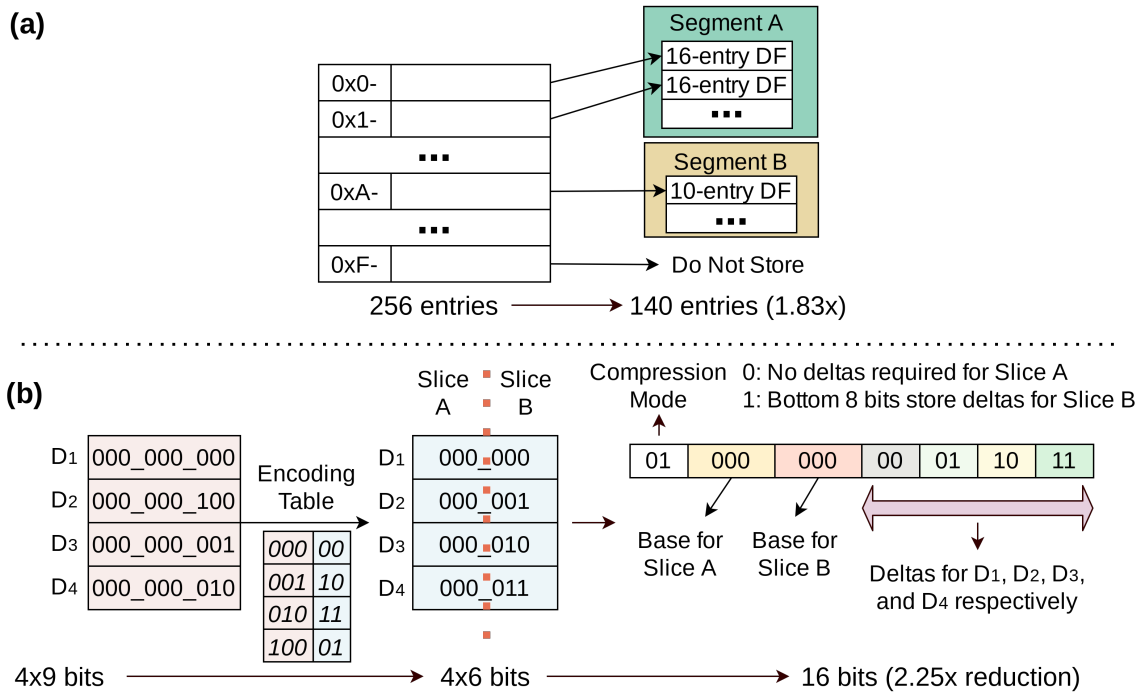


Figure 6.13: Steps (in software) involved in compressing LUTs.

Next, the DF entries are compressed by re-encoding using the least number of bits and Base-Delta-Immediate [244] compression schemes. For example, 9-bit error assignments from four consecutive entries are packed into a 16-bit word, as shown Figure Figure 6.13(b). First, each 9-bit data is encoded into 6-bit data using an encoding table. Next, four 6-bit entries are vertically sliced (Slice A and B) and packed into a 16-bit word. The 2-bit *compression mode* is needed because other memory regions require packing slices A and B differently. For example, when all slice B bits are 0s and slice A is stored using a 3-bit base and four 2-bit deltas, the mode is 10. We investigated other compression schemes and found these to be the most effective. Our studies show that for this configura-

tion, compressing the internal state part of the DF entries needs a different scheme and the overheads of decompression exceed the benefits of compression as the data is only 4 bits. Finally, each CLUT reduces to 140 bytes, 3x lower than the full LUT (416 bytes). As this is done in software offline prior to the actual QEC experiments, it does not incur additional hardware overheads in the LILLIPUT micro-architecture.

6.7.3 Accessing CLUTs and Decompression in Hardware

To service a decoding request, it is routed to the appropriate CLUT segment and DF. The CLUT entry is decompressed to obtain the error assignments and internal state, as shown in Figure 6.14. The hardware overhead to use CLUTs is the logic required to obtain the segment address and perform decompression.

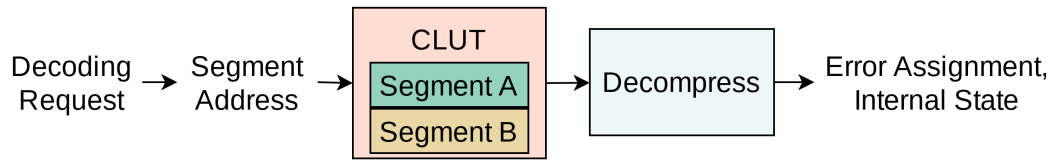


Figure 6.14: Steps (in hardware) involved in obtaining error assignment data from CLUTs.

6.7.4 Performance and Overheads of CLUTs

Figure 6.15(a) compares the logical error rate of the baseline design with respect to LILLIPUT using CLUTs. We observe that CLUTs offer similar performance as the baseline and do not degrade the LER. Figure 6.15(b) shows the decoder failure rate (due to missing LUT entries) and we observe that using CLUTs does not lead to events that increase the failure rate. A decoder failure here refers to failure to decode because the LUT address requested is not present in the CLUT. The performance remains stable even when the number of cycles is increased.

Table 6.5 shows the logic overhead (increases from 6% to 11%) and memory reduction (by 3x) in using CLUTs on Cyclone FPGAs. On Arria V, both designs require less than 1% logic utilization, and thus, the overhead is acceptable.

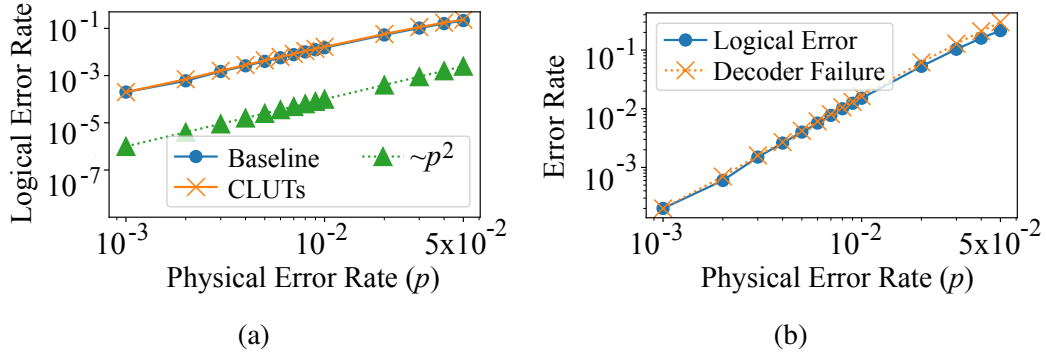


Figure 6.15: (a) Comparison of logical error rate for the baseline and LILLIPUT with CLUTs (b) Logical error rate and decoder failure rate with CLUTs.

Table 6.5: Logical Overhead for Implementing CLUTs

LILLIPUT (Baseline)			LILLIPUT with CLUT		
LEs	Registers	Memory	LEs	Registers	Memory
353 (6%)	209	832 Bytes	688 (11%)	261	280 Bytes

6.7.5 Scaling to Other Decoder Configurations

The LUTs for distance 4 and 5 surface codes can be reduced in a similar fashion. Figure 6.16 shows the probability distribution of the Hamming weight of the LUT addresses accessed for decoder configurations $[d = 4, m = 3]$ and $[d = 5, m = 2]$ respectively. We observe that storing only the LUT entries corresponding to Hamming weights 5 and below in the CLUTs is sufficient for both decoder configurations.

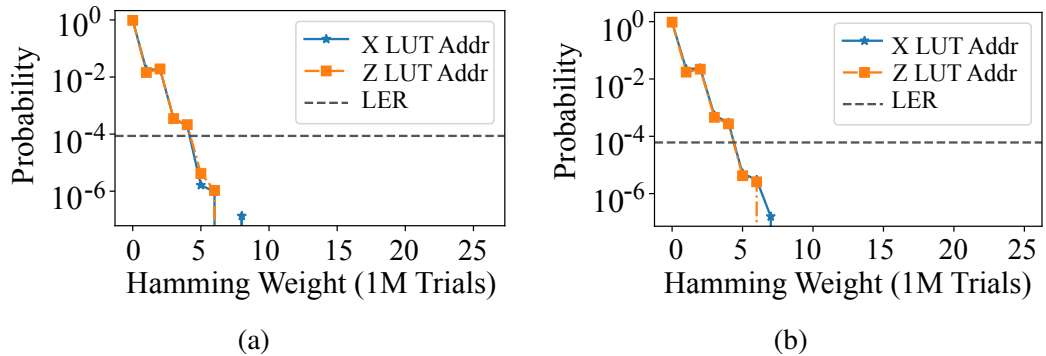


Figure 6.16: Probability distribution of Hamming weight of LUT accesses for (a) $[d=4, m=3]$ and (b) $[d=5, m=2]$.

Table 6.6 compares the size of CLUTs for distance 4 and 5 decoder configurations. Using CLUTs can reduce the memory requirement by up-to 107x. The CLUTs fit within the embedded memory available on Arria II/V [242] and Stratix 10 devices [243] and does not require any external memory access. Note that this estimate does not consider compression of the DF entries in the CLUT (second step discussed in Section subsection 6.7.2 as distances 4 and 5 require a different compression scheme and a detailed discussion is beyond the scope of this dissertation. To avoid the address translation for CLUT lookups, an alternative option is to use Cuckoo hashing [245] which incurs 2x memory overhead, but still fits the budget of these FPGA boards. Lastly, to support LILLIPUT for higher decoder configurations $[d = 5, m = 3]$ and $[d = 5, m = 4]$, the LUTs can be compressed and accessed from external memory.

Table 6.6: Memory Requirement for LILLIPUT with CLUTs

Decoder Configuration	Design	Memory (Baseline)		
		X	Z	Total
$[d = 4, m = 3]$	Baseline	5.75 MB	48 MB	53.75 MB
	W/ CLUT	245 KB	457 KB	702 KB (78.4x)
$[d = 5, m = 2]$	Baseline	74 MB	74 MB	148 MB
	W/ CLUT	704.6 KB	704.6 KB	1.38 MB (107x)

Key Takeaways

- Emerging quantum computers will be used to study QEC at a small scale.
- QEC requires real-time decoding to prevent the accumulation of errors.
- LILLIPUT is a low-cost lightweight lookup table real-time decoder that is compatible with off-the-shelf FPGAs. The LUTs are indexed by the error events and the corresponding entry provides the correction.
- Compressed LUTs reduce the memory complexity of LILLIPUT by only storing entries corresponding to correctable error events and compressing the table entries further.

6.8 Evolution of the Decoding Landscape and Follow-Up Works

Developing accurate and fast decoders for QEC has been an area of research for several years. However, the first proposal for developing custom hardware solutions for decoding, called the AFS decoder appeared in an arXiv preprint early 2020 [137]. A few months later, the NISQ+ decoder appeared at the International Symposium on Computer Architecture conference [115]. While these works use approximate decoding techniques and their performance is far from an idealized minimum weight perfect matching decoder, they still represent significant milestones in the development of fast decoders and were followed up with other works [138, 246, 141, 247, 142, 248], they remained far from practical adoption for studies on emerging quantum systems. The quest for accurate real-time decoders that are compatible with commodity hardware such as FPGAs used on near-term quantum computers continued and LILLIPUT was proposed mid-2021 [249] and presented later in the year 2022 at the Architectural Support for Programming Languages and Operating Systems (ASPLOS) conference [218]. LILLIPUT achieves the performance of an idealized MWPM decoder in real-time. Concurrently, Ryan-Anderson et al. used software LUT decoders to achieve real-time decoding for color codes on trapped-ion systems [233]. However, the scalability of LILLIPUT beyond distance 5 remained an open problem and an approximate decoder using the Union-Find decoding algorithm was later proposed to cater to larger distances [139]. This implementation trades off accuracy for latency, scalability, and practical adoption. Vittal et al. recently proposed Astrea [140] to bridge the accuracy gap and achieve accuracy comparable to MWPM for up to distance 9. Overall, there have been two key paradigm shifts in the decoding landscape in the last few years. First, the transition to hardware solutions for decoding proposed by the AFS decoder. Second, the transition to decoding solutions using commodity hardware, such as FPGAs for emerging quantum systems. In the future, solutions for decoding surface codes in real-time while achieving the accuracy of MWPM on commodity hardware beyond distance 9 need to be investigated.

CHAPTER 7

CONCLUSION AND FUTURE DIRECTIONS

Quantum computers promise computational advantages for applications that can have a profound impact on our society and humanity overall. While the initial idea of using quantum-mechanical properties for computing was cultivated in the early 1980s, technological advancements over the years have pushed the field to a point today where anyone anywhere in the world can run a quantum program on a real quantum computer using some of the world's most advanced cloud services. Dedicated research groups within industrial organizations are now focused on building quantum systems with increasing numbers of qubits. For example, IBM is on track to deliver a machine that surpasses a thousand qubits soon. Simultaneously, global investments in quantum science and technology research and development have increased substantially in recent years with several nations proposing dedicated programs and initiatives. While the prospects of compromising national security by breaking cryptographic protocols look scary, the utility of quantum computers in enabling solutions for many other crucial applications in agriculture, material sciences, healthcare, etc. provides a huge impetus to invest in the field (financially and intellectually). Building a large-scale fully-functional quantum computer is a grand engineering challenge that requires interdisciplinary research and extensive collaborations. Quantum computing research spans across multiple layers ranging from applications to algorithms, programming languages, compilers, classical control processor micro-architectures, and qubit devices. However, unlike the quantum algorithms or devices layers, the systems aspect of quantum computing (focused in this dissertation) that transform the mathematical properties of the quantum states into state changes on the actual physical devices is a relatively less well-studied and emerging research area with a lot more work remaining.

The key challenge in quantum computing is that quantum hardware is noisy with qubit devices having limited lifetimes and imperfections in quantum operations causing their error-rates to be several orders of magnitude higher than classical systems. These factors pose tremendous challenges in leveraging quantum systems for running practical applications because program execution on erroneous quantum hardware typically leads to incorrect outputs that are often indistinguishable from the correct ones. Reducing the impact of quantum hardware errors is crucial to unlock the potential of quantum computing and remains the primary focus of research in each layer of the quantum stack. For example, algorithms that are specifically tailored for noisy hardware and can tolerate certain amounts of errors are being developed. Similarly, quantum compilers optimize for fewer operations and reduced program length while translating programs to reduce the vulnerability to errors, whereas the devices community focuses on building qubits with increased lifetimes and lower operational error-rates.

The solutions required to tackle noise in quantum hardware depend on the regime or the size of the system— Noisy Intermediate Scale Quantum or NISQ systems with a few hundred to thousands of qubits heavily rely on error mitigation techniques, whereas quantum error correction is required for fully fault-tolerant quantum computing in the future. Error mitigation techniques attempt to suppress hardware errors as much as possible, typically via enhanced program scheduling and output post-processing. On the other hand, quantum error correction focuses on identifying and correcting errors in real-time as they occur, at the expense of redundant resources and operations. This dissertation presents software and architecture solutions to improve the performance (or reliability) of quantum computers by exploring techniques for error mitigation as well as error correction. The objective is to close the gap between quantum algorithms and qubit devices on emerging quantum systems by reducing the impact of hardware errors. The next sub-section provides a summary of the presented works in this dissertation.

7.1 Summary of Presented Works

Chapter 3 of this dissertation presents JigSaw, a software framework for mitigating measurement errors in emerging quantum systems. JigSaw identifies measurement crosstalk as a key source of these errors, which increases with the number of measurements. Furthermore, compilers are forced to use physical qubits with high measurement error-rates due to spatial variation. JigSaw overcomes these limitations by measuring all the qubits for half of the execution trials to obtain low-fidelity high-correlation information, measures subsets of qubits using additional program copies in the remaining trials to capture high-fidelity low-correlation partial information, and combines both using Bayesian updates.

Chapter 4 of this dissertation presents ADAPT, a software framework to reduce the impact of idling errors. Although mitigation of idling errors using dynamical decoupling (DD) sequences is well-studied in the devices community, naively applying them to all idle qubits is sub-optimal at the application-level, when the errors introduced by the additional DD operations exceed the idling errors.¹ ADAPT leverages the insight that idling errors in a program is dictated by its operational structure, builds decoy circuits with a similar structure as the input program that has a known output, uses the decoy circuits to identify the subset of qubits that may benefit from DD, and applies DD pulses only to those qubits.

Chapter 5 of this dissertation presents solutions to improve the throughput of quantum systems for enabling wider adoption of software error mitigation which typically requires additional quantum circuit executions. As quantum computers are limited resources accessible via cloud services, solutions that enable the execution of more circuits per unit time are desirable and this dissertation advocates for multi-programming as a potential solution. This dissertation presents some of the earliest solutions for fair and reliable allocation of quantum resources and context-aware instruction scheduling to minimize interference between co-running applications in multi-programmed environments.

¹Dynamical decoupling (DD) is implemented by the repeated execution of a sequence of single-qubit operations that return a qubit to its original state. Thus, DD operations do not change the overall state of the qubits as they collectively behave as an identity gate while keeping the idle qubits busy.

Although near-term quantum computers are promising for accelerating certain domain-specific applications, the ultimate goal of the quantum computing community is to build a fully fault-tolerant quantum computer as only then the true potential of quantum computing can be unlocked. Therefore, as device error-rates are reducing and system sizes are increasing, there are increasing interests in running small QEC codes on emerging quantum machines. Fault-tolerant quantum computers require real-time decoding to identify and correct errors in real-time (within a μ second) and prevent the accumulation of errors. Chapter 6 of this dissertation presents solutions that enable real-time decoding for QEC codes on emerging quantum computers. LILLIPUT is a fully reconfigurable lightweight look-up table (LUT) based solution for QEC, where the LUT is indexed by the error event and the LUT entry provides the correction. To reduce the overheads of the LUTs, LILLIPUT only stores the entries that correspond to error events that are correctable.

7.2 Key Takeaways

This dissertation focuses on using software and architecture solutions to bridge the gap between quantum algorithms and qubit devices. Although the scope of each solution presented in the thesis is relatively narrow, the insights and design principles guiding these solutions are generalizable and can be adopted broadly in other works. Some of these key takeaways are presented in the next subsections.

7.2.1 A Case for Specialized Fine-grained Software Error Mitigation

The earliest compiler optimizations for NISQ systems mainly focused on reducing the length and depth of the quantum circuits or enabling noise-aware optimizations to achieve better than worst-case error rates during program execution. In contrast, the solutions presented in the dissertation focus on developing solutions that target each individual source of error at a finer granularity to reduce their impact. For example, JigSaw specifically focuses *only* on measurement errors and prioritizes reducing the impact of measurement crosstalk.

Similarly, ADAPT focuses *only* on reducing the impact of idling errors. Such fine-grained techniques focus on targeted error mitigation and improve the application fidelity even further compared to generalized software optimizations.

7.2.2 Breaking the Abstractions

Breaking the assumptions and abstractions surrounding existing execution models can enable us to develop solutions that maximize the efficacy of software error mitigation. For example, JigSaw breaks the abstraction that all program qubits must be measured in each trial to obtain the program outcome. Such approaches also increase the flexibility of the software. For instance, JigSaw has more flexibility to map the subset measurements by recompiling the additional program copies to perform those measurements on qubit devices with the lowest error-rates. Similarly, ADAPT breaks the assumption that naively adopting device-level techniques at the application level can automatically offer optimal performance. Instead, it rethinks this approach and enables a more robust adoption of dynamical decoupling at the application level.

7.2.3 Quantum Overheads of Software Error Mitigation Must Be Minimized

Software error mitigation schemes typically incur huge resource overheads in terms of execution of additional quantum circuits (and. When multiple such optimizations are used, the total overheads can become prohibitively large, especially for variational quantum algorithms. Therefore, for practical adoption software error mitigation schemes must optimize across two vectors- their capability to improve fidelity and their scalability. Furthermore, other approaches that reduce the total execution time of quantum programs are also feasible solutions. Multi-programming quantum computers is an important step in this direction that improves the throughput of emerging quantum computers. In the future, more advanced solutions will be required given cloud providers enable access quantum systems built from different qubit device technologies.

7.2.4 Quantum Cloud Resource Management Requires A Different Approach

While modern-day cloud services are extremely advanced, quantum cloud services require a different approach that stems from the wide range of users ranging from quantum enthusiasts to experts who access the quantum systems, constraints associated with differences in qubit device technologies, and limited availability of quantum hardware. Moreover, as the error-rates of the devices change over time, long cloud latencies may affect the quality of the solutions because the efficacy of the software optimizations may diminish. This is contrary to conventional cloud services where the same compiled program code would yield identical results over different time periods.

7.2.5 Expand the Scope: A Case For Full-stack Solutions

The efficacy of error mitigation and error correction techniques improve when full-stack solutions are used. For example, exposing the hardware details to the software improves the performance of error mitigation schemes. As an example, JigSaw uses the device error-rates to recompile the program copies with subset measurements for enhanced performance. Similarly, achieving real-time decoding latencies is almost impractical using software decoders which have been widely studied in the QEC community for decades. Specialized hardware solutions that accelerate the theoretical algorithms enable us to address this bottleneck.

7.2.6 Adapting Solutions To Meet System Constraints

Engineering solutions are a function of their constraints— as the constraints change, the solutions must too. For example, although the earliest hardware decoding solutions use specialized custom ASICs or superconducting designs, LILLIPUT rethinks these solutions in light of practical adoption in emerging quantum systems. It also focuses on reconfigurability to adapt to the changing device error models as well as commodity hardware to enable ease-of-use and seamless integration.

7.3 Future Research Directions

Designing a fully-functional quantum computer that can accelerate real-world applications is a grand engineering challenge in the next few decades. This dissertation presents some preliminary solutions for improving the reliability of emerging quantum computers using software and architecture solutions, but a lot more work remains in the future. The next few subsections present opportunities for potential extensions of some of the key insights and solutions from prior works along with some open questions for future research directions.

7.3.1 Co-Designing Software Error Mitigation Using Application and Hardware Constraints

There has been a paradigm shift in software error mitigation techniques in the last half-decade with initial solutions primarily focusing on minimizing the total number of operations and circuit depth [82, 103, 65] to later solutions using targeted mitigation of specific types of errors (such as JigSaw [178], ADAPT [165]). However, recent studies show that the effectiveness of these software error mitigation techniques can be further enhanced by using application-specific optimizations. For example, the ZZ compiler [97] is an application-specific compiler that proposes instruction re-ordering policies specific to QAOA problems, whereas FrozenQubits [250] is a software tool-box that reduce the overheads of circuit-cutting for QAOA by leveraging the properties of real-world application graphs. Similarly, VarSaw extends JigSaw for VQE applications by exploiting the redundancy in subset measurements and reduces the overheads of executing the additional program copies in the subset mode [164]. These solutions have been provably more effective in improving the application fidelity compared to generalized compilation techniques. The question is- *How to develop scalable compiler optimizations that maximize the fidelity of near-term quantum applications by accounting for both application as well as device-specific properties?*

7.3.2 Striking A Balance Between Overheads and Benefits Of Software Error Mitigation

Most software error mitigation schemes incur overheads in terms of execution of additional characterization circuits that learn more fine-grained information related to specific types of errors. When multiple such techniques are combined, the overheads can become prohibitively large. One potential direction to reduce these overheads is to create characterization circuits that combine multiple optimizations or reuse them over multiple iterations for variational quantum algorithms. For example, can we design Clifford circuits that can learn both the optimal native gate selection [190] and subset of qubits to apply dynamical decoupling [165] on? Furthermore, exploiting the trade-off between the overheads for additional optimizations and improvements in fidelity remains an open problem. For example, can we determine the minimum subset of optimizations that must be used for a given application to obtain most of the benefits of software error mitigation? The question is- *How to strike a balance between the overheads and benefits of software error mitigation?*

7.3.3 Adapting Device-level Software Error Mitigation To Large-scale Applications

Similar to dynamical decoupling, many software error mitigation schemes are primarily studied at a smaller scale, such as zero-noise extrapolation [251] and randomized compilation [252]. However, using them for programs with large numbers of gates is not straightforward and may incur huge overheads. Also, the overheads become prohibitively large for variational algorithms. For example, ZNE requires the execution of many additional circuits each with increasing levels of noise so that the expectation value of the circuit at zero level of noise can be extrapolated. This method also works under the assumption that there is a clear trend in diminishing fidelity with increasing levels of noise, which may not always be true (similar to how always applying dynamical decoupling to idle qubits may not be optimal). Under such circumstances, the question is- *How to rethink and best integrate existing device-level software error mitigation techniques for larger programs?*

7.3.4 Exploring Trade-offs in Qubit Device Technologies

Quantum cloud services typically host a wide range of devices, each with its unique capabilities and constraints depending on the qubit technology. The software stack must, therefore, accommodate these differences and maximally exploit the trade-offs to attain higher application fidelity. For example, running functionally identical copies of the same program on different machines improves fidelity due to differences in the noise characteristics of the individual machines [88]. However, some machines are slower than others (trapped-ion systems for example are slower than superconducting ones) and cloud latencies may differ across devices (certain machines have more pending jobs and longer wait times than others). Similarly, running multiple compiler optimizations across several devices increases the overall number of circuit executions required on the quantum hardware which can cause a slowdown. On the other hand, some optimizations may be more effective on certain systems than others. The question is- *How should the software stack account for the unique trade-offs in each qubit device technology to improve application fidelity?*

7.3.5 Automated Software Tool-chains

Quantum programs may be hand-optimized when it includes only a few tens of operations. However, most practical quantum programs involve several hundreds to thousands of gates, making the reliance on hand-crafted code generation impractical. Even deciding amongst a wide range of available software optimizations is non-trivial. Therefore, automated toolchains that integrate multiple compiler passes, select the most suitable quantum hardware for each application, and verify the generate quantum object code as well as the pulse schedules are crucial to develop scalable software solutions. Similar solutions such as frameworks for automated device calibrations are also required for managing the hardware. The question is- *How to enhance the software stack to automatically adapt to the changing device characteristics?*

7.3.6 System-level Solutions for Quantum Cloud Services

Quantum cloud services must optimize across various dimensions- different levels of user expertise and qubit technologies, seamless user experience, and high quality-of-services (high application fidelity and low latency). Given the limited availability and challenges in deploying, operating, and maintaining quantum systems, cloud providers must develop solutions that ensure ease of use where users can take their hands off from actual resource management tasks. Simultaneously, their services must be noise-aware to account for the most crucial aspect of the hardware– device-level errors and variabilities in noise characteristics, while developing solutions for device management (such as automated drift detection and device re-calibrations) as well creating device access and job scheduling policies. The question is- *How should cloud providers best manage their services to minimize cloud latencies while maintaining high application fidelity, seamless device management, and meet the growing demands of a wide range of users for using noisy quantum hardware?*

7.3.7 Enabling QEC on Real-Systems: Rethinking Existing Approaches

The landscape of QEC is evolving rapidly with the introduction of newer QEC codes with lower redundancies, availability of scalable and low-latency approaches to decoding, and a greater understanding of different types of hardware errors. For example, QLDPC codes are becoming increasingly popular owing to their reduced overheads. Similarly, the impact of leakage errors on QEC codes is now better understood and leakage reduction techniques are now being developed. In light of this changing landscape, we must rethink existing solutions to best meet the constraints of the QEC space. For example, recent works LIL-LIPUT and Astrea have taken some important steps in this direction by not compromising accuracy while enabling real-time decoding. The question is- *How should we design the software stack, the control processors, and error correction architectures for different qubit device technologies at scale?*

7.3.8 Modeling architecture and system-level solutions for cryogenic control and QEC

Emerging quantum systems are typically controlled using FPGA-based architectures operating at room temperatures. While FPGAs present low-cost hardware development as well as reconfigurability and are being widely used in existing quantum platforms, their scalability and operational feasibility at low temperatures is an open problem. In the future, we need more energy-efficient solutions that operate closer to the quantum substrate as the number of qubits scale. The technology for implementing the control processors and decoding architectures remain an open problem and currently, both superconducting devices (greater energy-efficiency at 4 Kelvin with poor device densities and lack of dense memories) as well as cryo-CMOS (at 77 Kelvin with greater device densities but lower energy efficiency) are considered as potential option for large-scale fault-tolerant systems. Cryogenic control processors are already being designed at a smaller scale [61, 60]. However, more advanced solutions will be needed in the future. Designing an efficient system-level architecture for control and QEC depends on the number of qubits in the system (dictates the complexity of the classical hardware), qubit device technology (determines the latency and bandwidth constraints), and the choice of the classical electronics. Therefore, we need modeling tool-chains to explore this vast design space and recent works have taken some preliminary steps in this direction [253]. The question is- *How should we build tool-chains that enable us to model and explore the vast design space for cryogenic control processors and error correction architectures in large fault-tolerant systems?*

7.4 What Can We Do With Quantum Computers In The Next 5, 10, 20 and 30 years?

As with any computing system, the question always remains about the practical advantage of using the machine. With 1000-qubit, 4000-qubit, and 100,000-qubit systems to become available over the next few years and the promise of a-million-qubit systems in the next few decades, the ultimate question from the systems perspective is what real-world applications can we accelerate on each of these generations of quantum computers. More importantly, most of the research for emerging quantum systems faces fierce competition from the classical computing domain that has successfully evolved over multiple decades. This usually results in continuous technical back-and-forth, each time some key results are announced regarding quantum advantage. For example, Google's quantum supremacy experiments faced severe criticism within a few days of publication. Similarly, the recent study related to the advantage of quantum computers for tensor network simulations [39] was questioned by researchers within two weeks [254]. However, it is important to note that the research and development of complex engineering systems often face severe criticism at first and require time from inception to actual deployment. For example, the Manhattan Project and Apollo Missions are some of the most complex initiatives of mankind and a key reason behind their success is that scientific curiosity persevered to push the boundaries of human imagination. Quantum computing must go through a similar transition. Additionally, it is important to note that research developments in one area often lead to newer solutions in another, which is also an element of success. For example, the quantum supremacy experiments led to improvements in classical algorithms for random circuit sampling problems to reduce the execution time on conventional systems. This dissertation presents software and architecture solutions to bridge the gap between quantum applications and qubit devices as well as presents opportunities to pursue future research. Overall, the challenges and prospects of quantum computing make it an ideal time for a computer architect to contribute to the advancements of this emerging computing paradigm.

REFERENCES

- [1] Wikipedia, *Intel 4004*, https://en.wikipedia.org/wiki/Intel_4004.
- [2] Google, *Tech Talk: John Martinis, "Design of a Superconducting Quantum Computer"*, https://www.youtube.com/watch?v=HQmFEt6l6Tw&ab_channel=GoogleTechTalks, 2013.
- [3] G. E. Moore, *Cramming more components onto integrated circuits*, 1965.
- [4] R. H. Dennard, F. H. Gaensslen, H.-N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of solid-state circuits*, vol. 9, no. 5, pp. 256–268, 1974.
- [5] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Proceedings 35th annual symposium on foundations of computer science*, Ieee, 1994, pp. 124–134.
- [6] C. Gidney and M. Ekerå, "How to factor 2048 bit rsa integers in 8 hours using 20 million noisy qubits," *Quantum*, vol. 5, p. 433, 2021.
- [7] R. Zwanzig, A. Szabo, and B. Bagchi, "Levinthal's paradox.," *Proceedings of the National Academy of Sciences*, vol. 89, no. 1, pp. 20–22, 1992.
- [8] A. Robert, P. K. Barkoutsos, S. Woerner, and I. Tavernelli, "Resource-efficient quantum algorithm for protein folding," *npj Quantum Information*, vol. 7, no. 1, p. 38, 2021.
- [9] R. P. Feynman, "Simulating physics with computers," in *Feynman and computation*, CRC Press, 2018, pp. 133–153.
- [10] 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.
- [11] E. Bernstein and U. Vazirani, "Quantum complexity theory," in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, 1993, pp. 11–20.
- [12] D. R. Simon, "On the power of quantum computation," *SIAM journal on computing*, vol. 26, no. 5, pp. 1474–1483, 1997.
- [13] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, 1996, pp. 212–219.

- [14] S. Lloyd, “Universal quantum simulators,” *Science*, 1996.
- [15] J. Biamonte, P. Wittek, N. Pancotti, P. Rebentrost, N. Wiebe, and S. Lloyd, “Quantum machine learning,” *Nature*, vol. 549, no. 7671, pp. 195–202, 2017.
- [16] All About Circuits, *Ibm gives public cloud access to new 5-qubit quantum computer*, <https://www.allaboutcircuits.com/news/quantum-composer-ibm-5-qubit-quantum-computer-cloud/>, 2016.
- [17] E. Grumblin and M. Horowitz, “Quantum computing: Progress and prospects (2019),” *National Academies Press*. <https://doi.org/10.17226/17226>, p. 25 196, 2019.
- [18] Y. Wu, W.-S. Bao, S. Cao, F. Chen, M.-C. Chen, X. Chen, T.-H. Chung, H. Deng, Y. Du, D. Fan, *et al.*, “Strong quantum computational advantage using a superconducting quantum processor,” *Physical review letters*, vol. 127, no. 18, p. 180 501, 2021.
- [19] Q. Zhu, S. Cao, F. Chen, M.-C. Chen, X. Chen, T.-H. Chung, H. Deng, Y. Du, D. Fan, M. Gong, *et al.*, “Quantum computational advantage via 60-qubit 24-cycle random circuit sampling,” *Science bulletin*, vol. 67, no. 3, pp. 240–245, 2022.
- [20] L. S. Madsen, F. Laudenbach, M. F. Askarani, F. Rortais, T. Vincent, J. F. Bulmer, F. M. Miatto, L. Neuhaus, L. G. Helt, M. J. Collins, *et al.*, “Quantum computational advantage with a programmable photonic processor,” *Nature*, vol. 606, no. 7912, pp. 75–81, 2022.
- [21] IBM, *Ibm unveils 400 qubit-plus quantum processor and next-generation ibm quantum system two*, <https://newsroom.ibm.com/2022-11-09-IBM-Unveils-400-Qubit-Plus-Quantum-Processor-and-Next-Generation-IBM-Quantum-System-Two>, 2022.
- [22] IBM, *The ibm quantum development roadmap*, <https://www.ibm.com/quantum/roadmap>.
- [23] M. T. Review, *Ibm wants to build a 100,000-qubit quantum computer*, <https://www.technologyreview.com/2023/05/25/1073606/ibm-wants-to-build-a-100000-qubit-quantum-computer/>.
- [24] A. Cross, A. Javadi-Abhari, T. Alexander, N. de Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta, *et al.*, “Openqasm 3: A broader and deeper quantum assembly language,” *ACM Transactions on Quantum Computing*, 2021.

- [25] C. Developers, *Cirq*, <https://doi.org/10.5281/zenodo.6599601>, version v0.14.1, See full list of authors on Github: <https://github.com/quantumlib/Cirq/graphs/contributors>, Apr. 2022.
- [26] Rigetti, *The Quil Compiler*, <https://pyquil-docs.rigetti.com/en/stable/compiler.html>, 2021.
- [27] IBM, *IBM Quantum*, <https://quantum-computing.ibm.com/>, 2021.
- [28] G. Q. AI, *Quantum computer datasheet*, 2021.
- [29] P. W. Shor, “Fault-tolerant quantum computation,” in *Proceedings of 37th Conference on Foundations of Computer Science*, IEEE, 1996, pp. 56–65.
- [30] D. Aharonov and M. Ben-Or, “Fault-tolerant quantum computation with constant error rate,” *arXiv quant-ph/9906129*, 1999.
- [31] P. AALiferis, D. Gottesman, and J. Preskill, “Quantum accuracy threshold for concatenated distance-3 codes,” *arXiv quant-ph/0504218*, 2005.
- [32] D. Gottesman, “An introduction to quantum error correction and fault-tolerant quantum computation,” in *Proc. of Symposia in Applied Mathematics*, vol. 68, 2010, pp. 13–58.
- [33] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, “Surface codes: Towards practical large-scale quantum computation,” *Physical Review A*, vol. 86, no. 3, p. 032 324, 2012.
- [34] G. Quantum, “Exponential suppression of bit or phase errors with cyclic error correction,” *Nature*, vol. 595, no. 7867, pp. 383–387, 2021.
- [35] G. Quantum, “Suppressing quantum errors by scaling a surface code logical qubit,” *Nature*, vol. 614, no. 7949, pp. 676–681, 2023.
- [36] J. Preskill, “Quantum computing in the nisq era and beyond,” *Quantum*, vol. 2, p. 79, 2018.
- [37] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, “The theory of variational hybrid quantum-classical algorithms,” *New Journal of Physics*, vol. 18, no. 2, p. 023 023, 2016.
- [38] E. Farhi, J. Goldstone, and S. Gutmann, “A quantum approximate optimization algorithm,” *arXiv preprint arXiv:1411.4028*, 2014.

- [39] Y. Kim, A. Eddins, S. Anand, K. X. Wei, E. Van Den Berg, S. Rosenblatt, H. Nayfeh, Y. Wu, M. Zaletel, K. Temme, *et al.*, “Evidence for the utility of quantum computing before fault tolerance,” *Nature*, vol. 618, no. 7965, pp. 500–505, 2023.
- [40] G. Greene-Diniz, D. Z. Manrique, W. Sennane, Y. Magnin, E. Shishenina, P. Cordier, P. Llewellyn, M. Krompiec, M. J. Rančić, and D. M. Ramo, “Modelling carbon capture on metal-organic frameworks with quantum computing,” *EPJ Quantum Technology*, vol. 9, no. 1, p. 37, 2022.
- [41] I. Quantum. “Quantum algorithms for quantum chemistry and quantum materials science.” ().
- [42] I. Quantum. “Quantum finance.” ().
- [43] G. Q. AI, “Exponential suppression of bit or phase errors with cyclic error correction,” *Nature*, vol. 595, no. 7867, p. 383, 2021.
- [44] C. K. Andersen, A. Remm, S. Lazar, S. Krinner, N. Lacroix, G. J. Norris, M. Gabureac, C. Eichler, and A. Wallraff, “Repeated quantum error detection in a surface code,” *Nature Physics*, vol. 16, no. 8, pp. 875–880, 2020.
- [45] S. Krinner, N. Lacroix, A. Remm, A. Di Paolo, E. Genois, C. Leroux, C. Hellings, S. Lazar, F. Swiadek, J. Herrmann, *et al.*, “Realizing repeated quantum error correction in a distance-three surface code,” *Nature*, vol. 605, no. 7911, pp. 669–674, 2022.
- [46] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. F. Bogorin, M. Brink, L. Capelluto, O. Günlük, T. Itoko, N. Kanazawa, *et al.*, “Demonstration of quantum volume 64 on a superconducting quantum computing system,” *Quantum Science and Technology*, vol. 6, no. 2, p. 025 020, 2021.
- [47] S. S. Tannu and M. 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.
- [48] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, “Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers,” in *Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers*, ACM, 2019, pp. 1015–1029.
- [49] F. Chong, *Closing the gap between quantum algorithms and machines with hardware-software co-design*, <https://www.epiqc.cs.uchicago.edu/news/chong-asplos-2018-keynote>.

- [50] IBM, *Measurement Error Mitigation*, <https://qiskit.org/textbook/ch-quantum-hardware/measurement-error-mitigation.html>, [Accessed July-2020], 2020.
- [51] Wikipedia, *Bloch sphere*, https://en.wikipedia.org/wiki/Bloch_sphere.
- [52] Wikipedia, *Pauli matrices*, https://en.wikipedia.org/wiki/Pauli_matrices.
- [53] M. A. Nielsen and I. Chuang, *Quantum computation and quantum information*, 2002.
- [54] D. M. Greenberger, M. A. Horne, and A. Zeilinger, “Going beyond Bell’s theorem,” in *Bell’s theorem, quantum theory and conceptions of the universe*, Springer, 1989, pp. 69–72.
- [55] Y. Development, *Quantum technologies: Market & technology report*, <https://s3.amazonaws.com/uploads/2021/06/YINTR21211-Quantum-Technologies-2021-Flyer.pdf>, 2021.
- [56] IBM, *Ibm quantum breaks the 100-qubit processor barrier*, <https://research.ibm.com/blog/127-qubit-quantum-processor-eagle>, 2021.
- [57] M. Steffen, J. Chow, S. Sheldon, and D. McClure, *Ibm quantum’s highest performing system, yet*, <https://research.ibm.com/blog/eagle-quantum-error-mitigation>, Dec. 2022.
- [58] J. Gambetta, *Quantum-centric supercomputing: The next wave of computing*, <https://research.ibm.com/blog/next-wave-quantum-centric-supercomputing>, Dec. 2022.
- [59] S. Maurya, C. N. Mude, W. D. Oliver, B. Lienhard, and S. Tannu, *Hardware efficient neural network assisted qubit readout*, 2022. arXiv: 2212.03895 [quant-ph].
- [60] J. C. Bardin, E. Jeffrey, E. Lucero, T. Huang, O. Naaman, R. Barends, T. White, M. Giustina, D. Sank, *et al.*, “29.1 a 28nm bulk-cmos 4-to-8ghz 2mw cryogenic pulse modulator for scalable quantum computing,” in *ISSCC*, IEEE, 2019.
- [61] I. Newsroom, *Intel debuts 2nd-gen horse ridge cryogenic quantum control chip*, <https://www.intel.com/content/www/us/en/newsroom/news/2nd-gen-horse-ridge-cryogenic-quantum-control-chip.html>.
- [62] 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*, IEEE, 2017.

- [63] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, “Open quantum assembly language,” *arXiv :1707.03429*, 2017.
- [64] V. Tripathi, H. Chen, M. Khezri, K.-W. Yip, E. Levenson-Falk, and D. A. Lidar, “Suppression of crosstalk in superconducting qubits using dynamical decoupling,” *arXiv :2108.04530*, 2021.
- [65] A. Zulehner, A. Paler, and R. Wille, “Efficient mapping of quantum circuits to the ibm qx architectures,” in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, IEEE, 2018, pp. 1135–1138.
- [66] M. Y. Siraichi, V. F. d. Santos, S. Collange, and F. M. Q. Pereira, “Qubit allocation,” in *International Symposium on Code Generation and Optimization (CGO)*, ACM, 2018, pp. 113–125.
- [67] D. Bhattacharjee and A. Chattopadhyay, “Depth-optimal quantum circuit placement for arbitrary topologies,” *arXiv preprint arXiv:1703.08540*, 2017.
- [68] K. Booth, M. Do, J. Beck, E. Rieffel, D. Venturelli, and J. Frank, “Comparing and integrating constraint programming and temporal planning for quantum circuit compilation,” in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 28, 2018, pp. 366–374.
- [69] A. Lye, R. Wille, and R. Drechsler, “Determining the minimal number of swap gates for multi-dimensional nearest neighbor quantum circuits,” in *The 20th Asia and South Pacific Design Automation Conference*, IEEE, 2015, pp. 178–183.
- [70] A. Oddi and R. Rasconi, “Greedy randomized search for scalable compilation of quantum circuits,” in *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Springer, 2018, pp. 446–461.
- [71] 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*, 2013, pp. 1–6.
- [72] A. Shafaei, M. Saeedi, and M. Pedram, “Qubit placement to minimize communication overhead in 2d quantum architectures,” in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2014, pp. 495–500.
- [73] D. Venturelli, M. Do, E. Rieffel, and J. Frank, “Temporal planning for compilation of quantum approximate optimization circuits,” in *Scheduling and Planning Applications woRKshop (SPARK)*, 2017, p. 58.

- [74] 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.
- [75] R. Wille, A. Lye, and R. Drechsler, “Optimal swap gate insertion for nearest neighbor quantum circuits,” in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, IEEE, 2014, pp. 489–494.
- [76] G. Nannicini, L. S. Bishop, O. Günlük, and P. Jurcevic, “Optimal qubit assignment and routing via integer programming,” *ACM Transactions on Quantum Computing*, vol. 4, no. 1, pp. 1–31, 2022.
- [77] A. Chakrabarti, S. Sur-Kolay, and A. Chaudhury, “Linear nearest neighbor synthesis of reversible circuits by graph partitioning,” *arXiv preprint arXiv:1112.0564*, 2011.
- [78] I. B. M. Corporation, *Quantum Software Development Kit for writing quantum computing experiments, programs, and applications*, <https://github.com/QISKit/>, [Online; accessed 28-AUGUST-2020], 2017.
- [79] I. for Integrated Circuits, *QMAP - A JKQ tool for Quantum Circuit Mapping*, <https://github.com/iic-jku/qmap>, 2021.
- [80] R. S. Smith, E. C. Peterson, M. G. Skilbeck, and E. J. Davis, “An open-source, industrial-strength optimizing compiler for quantum programs,” *Quantum Science and Technology*, vol. 5, no. 4, 2020.
- [81] Google, *Routing with $t|ket \rangle$* , 2021.
- [82] G. Li, Y. Ding, and Y. Xie, “Tackling the qubit mapping problem for NISQ-era quantum devices,” in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1001–1014.
- [83] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, “Time-optimal qubit mapping,” in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 360–374.
- [84] 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*, 2019, pp. 527–540.

- [85] T. Patel and D. Tiwari, “Qraft: Reverse your quantum circuit and know the correct program output,” in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 443–455.
- [86] T. Peng, A. W. Harrow, M. Ozols, and X. Wu, “Simulating large quantum circuits on a small quantum computer,” *Physical review letters*, vol. 125, no. 15, p. 150 504, 2020.
- [87] W. Tang, T. Tomesh, M. Suchara, J. Larson, and M. Martonosi, “Cutqc: Using small quantum computers for large quantum circuit evaluations,” in *Proceedings of the 26th ACM International conference on architectural support for programming languages and operating systems*, 2021, pp. 473–486.
- [88] S. S. Tannu and M. Qureshi, “Ensemble of diverse mappings: Improving reliability of quantum computers by orchestrating dissimilar mistakes,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 253–265.
- [89] T. Patel and D. Tiwari, “Veritas: Accurately estimating the correct output on noisy intermediate-scale quantum computers,” in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, IEEE, 2020, pp. 1–16.
- [90] P. Murali, D. C. McKay, M. Martonosi, and A. Javadi-Abhari, “Software mitigation of crosstalk on noisy intermediate-scale quantum computers,” in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2020, pp. 1001–1016.
- [91] Y. Ding, P. Gokhale, S. F. Lin, R. Rines, T. Propson, and F. T. Chong, “Systematic crosstalk mitigation for superconducting qubits via frequency-aware compilation,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2020, pp. 201–214.
- [92] L. Funcke, T. Hartung, K. Jansen, S. Kühn, P. Stornati, and X. Wang, “Measurement error mitigation in quantum computers through classical bit-flip correction,” *Physical Review A*, vol. 105, no. 6, p. 062 404, 2022.
- [93] H. Kwon and J. Bae, “A hybrid quantum-classical approach to mitigating measurement errors in quantum algorithms,” *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1401–1411, 2020.
- [94] F. B. Maciejewski, Z. Zimborás, and M. Oszmaniec, “Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography,” *Quantum*, vol. 4, p. 257, 2020.

- [95] S. Bravyi, S. Sheldon, A. Kandala, D. C. McKay, and J. M. Gambetta, “Mitigating measurement errors in multiqubit experiments,” *Physical Review A*, vol. 103, no. 4, p. 042 605, 2021.
- [96] S. S. Tannu and M. K. Qureshi, “Mitigating measurement errors in quantum computers by exploiting state-dependent bias,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 279–290.
- [97] M. Alam, A. Ash-Saki, and S. Ghosh, “Circuit compilation methodologies for quantum approximate optimization algorithm,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2020, pp. 215–228.
- [98] 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, pp. 242–246, 2017.
- [99] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven, “Barren plateaus in quantum neural network training landscapes,” *Nature communications*, vol. 9, no. 1, p. 4812, 2018.
- [100] P. Gokhale, A. Javadi-Abhari, N. Earnest, Y. Shi, and F. T. Chong, “Optimized quantum compilation for near-term algorithms with openpulse,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, IEEE, 2020, pp. 186–200.
- [101] P. Jurcevic, A. Javadi-Abhari, L. S. Bishop, I. Lauer, D. F. Bogorin, M. Brink, L. Capelluto, O. Günlük, T. Itoko, N. Kanazawa, *et al.*, “Demonstration of quantum volume 64 on a superconducting quantum computing system,” *Quantum Science and Technology*, vol. 6, no. 2, p. 025 020, 2021.
- [102] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, “Validating quantum computers using randomized model circuits,” *Physical Review A*, vol. 100, no. 3, p. 032 328, 2019.
- [103] 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.
- [104] A. G. Fowler, A. C. Whiteside, and L. C. L. Hollenberg, “Towards practical classical processing for the surface code,” *Phys. Rev. Lett.*, vol. 108, p. 180 501, 18 May 2012.

- [105] A. Y. Kitaev, “Fault-tolerant quantum computation by anyons,” *Annals of Physics*, vol. 303, no. 1, pp. 2–30, 2003.
- [106] E. Dennis, A. Kitaev, A. Landahl, and J. Preskill, “Topological quantum memory,” *Journal of Mathematical Physics*, vol. 43, no. 9, pp. 4452–4505, 2002.
- [107] R. Raussendorf and J. Harrington, “Fault-tolerant quantum computation with high threshold in two dimensions,” *Phys. Rev. Lett.*, vol. 98, p. 190 504, 19 May 2007.
- [108] A. M. Stephens, “Fault-tolerant thresholds for quantum error correction with the surface code,” *Phys. Rev. A*, vol. 89, p. 022 321, 2 Feb. 2014.
- [109] D. Gottesman, “Theory of fault-tolerant quantum computation,” *Phys. Rev. A*, vol. 57, pp. 127–137, 1 Jan. 1998.
- [110] J. Ghosh, A. G. Fowler, and M. R. Geller, “Surface code with decoherence: An analysis of three superconducting architectures,” *Phys. Rev. A*, 2012.
- [111] C. Gidney, M. Newman, A. Fowler, and M. Broughton, “A fault-tolerant honeycomb memory,” *Quantum*, vol. 5, p. 605, 2021.
- [112] M. B. Hastings and J. Haah, “Dynamically generated logical qubits,” *Quantum*, vol. 5, p. 564, 2021.
- [113] B. M. Terhal, “Quantum error correction for quantum memories,” *RMP*, vol. 87, 2015.
- [114] Y. Tomita and K. M. Svore, “Low-distance surface codes under realistic quantum noise,” *Phys. Rev. A*, vol. 90, p. 062 320, 6 Dec. 2014.
- [115] A. Holmes, M. R. Jokat, G. Pasandi, Y. Ding, M. Pedram, and F. T. Chong, “Nisq+: Boosting quantum computing power by approximating quantum error correction,” in *ISCA*, 2020.
- [116] A. G. Fowler, “Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $\mathcal{O}(1)$ parallel time,” *Quantum Information and Computation*, vol. 15, no. 1–2, pp. 145–158, Jan. 2015.
- [117] G. Torlai and R. G. Melko, “Neural decoder for topological codes,” *PRL*, vol. 119, no. 3, p. 030 501, 2017.
- [118] S. Krastanov and L. Jiang, “Deep neural network probabilistic decoder for stabilizer codes,” *Scientific reports*, vol. 7, no. 1, p. 11 003, 2017.

- [119] S. Varsamopoulos, B. Criger, and K. Bertels, “Decoding small surface codes with feedforward neural networks,” *Quantum Science and Technology*, vol. 3, no. 1, p. 015 004, 2017.
- [120] C. Chamberland and P. Ronagh, “Deep neural decoders for near term fault-tolerant experiments,” *Quantum Science and Technology*, vol. 3, no. 4, p. 044 002, 2018.
- [121] S. Varsamopoulos, K. Bertels, and C. G. Almudever, “Designing neural network based decoders for surface codes,” *arXiv preprint arXiv:1811.12456*, 2018.
- [122] T. Wagner, H. Kampermann, and D. Bruß, “Symmetries for a high level neural decoder on the toric code,” *arXiv preprint arXiv:1910.01662*, 2019.
- [123] C. Chinni, A. Kulkarni, and D. M. Pai, “Neural decoder for topological codes using pseudo-inverse of parity check matrix,” *arXiv preprint arXiv:1901.07535*, 2019.
- [124] L. D. Colomer, M. Skotiniotis, and R. Muñoz-Tapia, “Reinforcement learning for optimal error correction of toric codes,” *arXiv preprint arXiv:1911.02308*, 2019.
- [125] N. Maskara, A. Kubica, and T. Jochym-O’Connor, “Advantages of versatile neural-network decoding for topological codes,” *Phys. Rev. A*, vol. 99, no. 5, p. 052 351, 2019.
- [126] N. P. Breuckmann and X. Ni, “Scalable neural network decoders for higher dimensional quantum codes,” *Quantum*, vol. 2, pp. 68–92, 2018.
- [127] R. Sweke, M. S. Kesselring, E. P. van Nieuwenburg, and J. Eisert, “Reinforcement learning decoders for fault-tolerant quantum computation,” *arXiv preprint arXiv:1810.07207*, 2018.
- [128] P. Baireuther, M. Caio, B. Criger, C. W. Beenakker, and T. E. O’Brien, “Neural network decoder for topological color codes with circuit level noise,” *New Journal of Physics*, vol. 21, no. 1, p. 013 003, 2019.
- [129] X. Ni, “Neural network decoders for large-distance 2d toric codes,” *arXiv preprint arXiv:1809.06640*, 2018.
- [130] P. Andreasson, J. Johansson, S. Liljestrand, and M. Granath, “Quantum error correction for the toric code using deep reinforcement learning,” *Quantum*, vol. 3, p. 183, 2019.
- [131] A. Davaasuren, Y. Suzuki, K. Fujii, and M. Koashi, “General framework for constructing fast and near-optimal machine-learning-based decoder of the topological stabilizer codes,” *arXiv preprint arXiv:1801.04377*, 2018.

- [132] Y.-H. Liu and D. Poulin, “Neural belief-propagation decoders for quantum error-correcting codes,” *PRL*, vol. 122, no. 20, p. 200 501, 2019.
- [133] P. Baireuther, T. E. O’Brien, B. Tarasinski, and C. W. Beenakker, “Machine-learning-assisted correction of correlated qubit errors in a topological code,” *Quantum*, vol. 2, p. 48, 2018.
- [134] N. Delfosse and N. H. Nickerson, “Almost-linear time decoding algorithm for topological codes,” *arXiv preprint arXiv:1709.06218*, 2017.
- [135] N. Delfosse and G. Zémor, “Linear-time maximum likelihood decoding of surface codes over the quantum erasure channel,” *arXiv preprint arXiv:1703.01517*, 2017.
- [136] A. Fowler, “Towards sufficiently fast quantum error correction,” Conference QEC 2017, 2017.
- [137] P. Das, C. A. Pattison, S. Manne, D. Carmean, K. Svore, M. Qureshi, and N. Delfosse, “A scalable decoder micro-architecture for fault-tolerant quantum computing,” *arXiv:2001.06598*, 2020.
- [138] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi, “Qecool: On-line quantum error correction with a superconducting decoder for surface code,” *arXiv preprint arXiv:2103.14209*, 2021.
- [139] N. Liyanage, Y. Wu, A. Deters, and L. Zhong, “Scalable quantum error correction for surface codes using fpga,” *arXiv preprint arXiv:2301.08419*, 2023.
- [140] S. Vittal, P. Das, and M. Qureshi, “Astrea: Accurate quantum error-decoding via practical minimum-weight perfect matching,” in *Proceedings of the 50th International Symposium on Computer Architecture (ISCA)*, 2023.
- [141] N. Delfosse, “Hierarchical decoding to reduce hardware requirements for quantum computing,” *arXiv:2001.11427*, 2020.
- [142] G. S. Ravi, J. M. Baker, A. Fayyazi, S. F. Lin, A. Javadi-Abhari, M. Pedram, and F. T. Chong, *Have your qec and bandwidth too!: A lightweight cryogenic decoder for common / trivial errors, and efficient bandwidth + execution management otherwise*, 2022.
- [143] C. Chamberland, L. Goncalves, P. Sivarajah, E. Peterson, and S. Grimberg, “Techniques for combining fast local decoders with global decoders under circuit-level noise,” *arXiv preprint arXiv:2208.01178*, 2022.

- [144] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi, “A case for multi-programming quantum computers,” in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 291–303.
- [145] M. R. Geller and M. Sun, “Toward efficient correction of multiqubit measurement errors: Pair correlation method,” *Quantum Science and Technology*, vol. 6, no. 2, p. 025 009, 2021.
- [146] A. Blais, R.-S. Huang, A. Wallraff, S. M. Girvin, and R. J. Schoelkopf, “Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation,” *Physical Review A*, vol. 69, no. 6, p. 062 320, 2004.
- [147] A. Wallraff, D. I. Schuster, A. Blais, L. Frunzio, R.-S. Huang, J. Majer, S. Kumar, S. M. Girvin, and R. J. Schoelkopf, “Strong coupling of a single photon to a superconducting qubit using circuit quantum electrodynamics,” *Nature*, vol. 431, no. 7005, pp. 162–167, 2004.
- [148] P. Krantz *et al.*, “A quantum engineer’s guide to superconducting qubits,” *Applied Physics Reviews*, vol. 6, no. 2, p. 021 318, 2019.
- [149] 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,” *Quantum Science and Technology*, vol. 5, no. 3, p. 034 003, 2020.
- [150] M. Khezri *et al.*, “Qubit measurement error from coupling with a detuned neighbor in circuit qed,” *Physical Review A*, vol. 92, no. 5, p. 052 306, 2015.
- [151] T. Patel and D. Tiwari, “Disq: A novel quantum output state classification method on ibm quantum computers using openpulse,” in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.
- [152] F. Arute, K. Arya, R. Babbush, *et al.*, “Supplementary information for ‘quantum supremacy using a programmable superconducting processor,’” *Nat. Int. Wkly. J. Sci.*, vol. 574, pp. 505–505, 2020.
- [153] 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.
- [154] J. Joyce, “Bayes’ theorem,” 2003.
- [155] E. Hellinger, “Neue begründung der theorie quadratischer formen von unendlichvielen veränderlichen.” *Journal für die reine und angewandte Mathematik*, vol. 136, pp. 210–271, 1909.

- [156] IBM, *Aqua (Algorithms for QUantum Applications)*, https://qiskit.org/documentation/apidoc/qiskit_aqua.html, [Online; accessed 15-April-2021], 2021.
- [157] E. Bernstein and U. Vazirani, “Quantum complexity theory,” *SIAM Journal on computing*, vol. 26, no. 5, pp. 1411–1473, 1997.
- [158] E. Ising, “Beitrag zur theorie des ferromagnetismus,” *Zeitschrift für Physik*, vol. 31, no. 1, pp. 253–258, 1925.
- [159] Wikipedia, *Total Variational Distance*, https://en.wikipedia.org/wiki/Total_variation_distance_of_probability_measures, [Online; accessed 7-March-2021], 2020.
- [160] Y. R. Sanders, J. J. Wallman, and B. C. Sanders, “Bounding quantum gate error rate based on reported average fidelity,” *New Journal of Physics*, vol. 18, no. 1, p. 012 002, 2015.
- [161] S. Kullback, *Information theory and statistics*. Courier Corporation, 1997.
- [162] P. D. Nation, H. Kang, N. Sundaresan, and J. M. Gambetta, “Scalable mitigation of measurement errors on quantum computers,” *PRX Quantum*, vol. 2, no. 4, p. 040 326, 2021.
- [163] I. Qiskit. “Mthree.m3mitigation.” ().
- [164] S. Dangwal, G. S. Ravi, P. Das, K. N. Smith, J. M. Baker, and F. T. Chong, “Warsaw: Application-tailored measurement error mitigation for variational quantum algorithms,” *arXiv preprint arXiv:2306.06027*, 2023.
- [165] P. Das, S. Tannu, S. Dangwal, and M. Qureshi, “Adapt: Mitigating idling errors in qubits via adaptive dynamical decoupling,” in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 950–962.
- [166] J. Bylander, S. Gustavsson, F. Yan, F. Yoshihara, K. Harrabi, G. Fitch, D. G. Cory, Y. Nakamura, J.-S. Tsai, and W. D. Oliver, “Noise spectroscopy through dynamical decoupling with a superconducting flux qubit,” *Nature Physics*, vol. 7, no. 7, pp. 565–570, 2011.
- [167] K. Khodjasteh and D. Lidar, “Fault-tolerant quantum dynamical decoupling,” *Physical review letters*, vol. 95, no. 18, p. 180 501, 2005.
- [168] K. Khodjasteh and D. A. Lidar, “Performance of deterministic dynamical decoupling schemes: Concatenated and periodic pulse sequences,” *Physical Review A*, vol. 75, no. 6, p. 062 310, 2007.

- [169] 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.
- [170] G. Q. AI, “Exponential suppression of bit or phase errors with cyclic error correction,” *Nature*, vol. 595, no. 7867, p. 383, 2021.
- [171] R. Harper and S. Flammia, “Fault tolerance in the ibm q experience,” *arXiv preprint arXiv:1806.02359*, 2018.
- [172] 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.
- [173] S. Aaronson and D. Gottesman, “Improved simulation of stabilizer circuits,” *Physical Review A*, vol. 70, no. 5, Nov. 2004.
- [174] S. Bravyi, D. Browne, P. Calpin, E. Campbell, D. Gosset, and M. Howard, “Simulation of quantum circuits by low-rank stabilizer decompositions,” *Quantum*, vol. 3, p. 181, 2019.
- [175] D. C. McKay, C. J. Wood, S. Sheldon, J. M. Chow, and J. M. Gambetta, “Efficient z gates for quantum computing,” *Physical Review A*, vol. 96, no. 2, p. 022 330, 2017.
- [176] IBM, *Quantum Software Development Kit for writing quantum computing experiments, programs, and applications*, 2017.
- [177] A. Li and S. Krishnamoorthy, “Qasmbench: A low-level qasm benchmark suite for nisq evaluation and simulation,” *preprint arXiv:2005.13018*, 2020.
- [178] P. Das, S. Tannu, and M. Qureshi, “Jigsaw: Boosting fidelity of nisq programs via measurement subsetting,” in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, 2021, pp. 937–949.
- [179] M. A. A. Ahmed, G. A. Alvarez, and D. Suter, “Robustness of dynamical decoupling sequences,” *Physical Review A*, vol. 87, no. 4, p. 042 309, 2013.
- [180] L. Viola and S. Lloyd, “Dynamical suppression of decoherence in two-state quantum systems,” *Physical Review A*, vol. 58, no. 4, p. 2733, 1998.
- [181] L. Viola, E. Knill, and S. Lloyd, “Dynamical decoupling of open quantum systems,” *Physical Review Letters*, vol. 82, no. 12, p. 2417, 1999.
- [182] L. Viola, E. Knill, and S. Lloyd, “Dynamical decoupling of open quantum systems,” *Phys. Rev. Lett.*, vol. 82, pp. 2417–2421, 12 Mar. 1999.

- [183] G. A. Paz-Silva and D. Lidar, “Optimally combining dynamical decoupling and quantum error correction,” *Scientific reports*, vol. 3, p. 1530, 2013.
- [184] A. M. Souza, G. A. Alvarez, and D. Suter, “Robust dynamical decoupling for quantum computing and quantum memory,” *Physical review letters*, vol. 106, no. 24, p. 240 501, 2011.
- [185] 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].
- [186] A. Zlokapá and A. Gheorghiu, “A deep learning model for noise prediction on near-term quantum devices,” *arXiv preprint:2005.10811*, 2020.
- [187] I. Quantum. “Qiskit dynamical decoupling.” ()
- [188] B. Pokharel and D. A. Lidar, “Demonstration of algorithmic quantum speedup,” *Phys. Rev. Lett.*, vol. 130, p. 210 602, 21 May 2023.
- [189] Wikipedia. “Decoy.” ()
- [190] P. Das, E. Kessler, and Y. Shi, “The imitation game: Leveraging copycats for robust native gate selection in nisq programs,” in *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2023, pp. 787–801.
- [191] K. N. Smith, M. A. Perlin, P. Gokhale, P. Frederick, D. Owusu-Antwi, R. Rines, V. Omole, and F. Chong, “Clifford-based circuit cutting for quantum simulation,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–13.
- [192] I. Quantum. “Qiskit runtime overview.” ()
- [193] I. Quantum. “Mid-circuit measurement.” ()
- [194] I. Quantum. “Qiskit runtime: A cloud-native, pay-as-you-go service for quantum computing.” ()
- [195] Rigetti, *Introduction to quantum cloud services*, 2018.
- [196] C. E. Shannon, “A mathematical theory of communication,” *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.
- [197] X. Dou and L. Liu, “A new qubits mapping mechanism for multi-programming quantum computing,” in *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques*, 2020, pp. 349–350.

- [198] S. Niu and A. Todri-Sanial, “Enabling multi-programming mechanism for quantum computing in the NISQ era,” *CoRR*, vol. abs/2102.05321, 2021. arXiv: 2102.05321.
- [199] S. Niu and A. Todri-Sanial, “Enabling multi-programming mechanism for quantum computing in the nisq era,” *Quantum*, vol. 7, p. 925, 2023.
- [200] S. Niu and A. Todri-Sanial, “How parallel circuit execution can be useful for nisq computing?” In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 1065–1070.
- [201] L. Liu and X. Dou, “Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment,” in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, IEEE, 2021, pp. 167–178.
- [202] Y. Ohkura, T. Satoh, and R. Van Meter, “Simultaneous execution of quantum circuits on current and near-future nisq systems,” *IEEE Transactions on Quantum Engineering*, vol. 3, pp. 1–10, 2022.
- [203] S. Niu and A. Todri-Sanial, “Multi-programming cross platform benchmarking for quantum computing hardware,” *arXiv preprint arXiv:2206.03144*, 2022.
- [204] S. Stein, N. Wiebe, Y. Ding, P. Bo, K. Kowalski, N. Baker, J. Ang, and A. Li, “Eqc: Ensembled quantum computing for variational quantum algorithms,” in *Proceedings of the 49th Annual International Symposium on Computer Architecture*, ser. ISCA ’22, New York, New York: Association for Computing Machinery, 2022, pp. 59–71, ISBN: 9781450386104.
- [205] L. Mineh and A. Montanaro, “Accelerating the variational quantum eigensolver using parallelism,” *Quantum Science and Technology*, vol. 8, no. 3, p. 035 012, 2023.
- [206] S. Resch, A. Gutierrez, J. S. Huh, S. Bharadwaj, Y. Eckert, G. Loh, M. Oskin, and S. Tannu, “Accelerating variational quantum algorithms using circuit concurrency,” *arXiv preprint arXiv:2109.01714*, 2021.
- [207] G. Park, K. Zhang, K. Yu, and V. Korepin, “Quantum multi-programming for grover’s search,” *Quantum Information Processing*, vol. 22, no. 1, p. 54, 2023.
- [208] E. Pelofske, G. Hahn, and H. N. Djidjev, “Solving larger maximum clique problems using parallel quantum annealing,” *Quantum Information Processing*, vol. 22, no. 5, p. 219, 2023.
- [209] T. Huang, Y. Zhu, R. S. M. Goh, and T. Luo, “When quantum annealing meets multitasking: Potentials, challenges and opportunities,” *Array*, p. 100 282, 2023.

- [210] G. S. Ravi, K. N. Smith, P. Murali, and F. T. Chong, “Adaptive job and resource management for the growing quantum cloud,” in *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, IEEE, 2021, pp. 301–312.
- [211] J. Yao, J. Wang, F. Yue, J. Xu, and Z. Shan, “Mtmc: A scheduling framework of multi-tasking mapping on multi-chips,” 2022.
- [212] A. Ash-Saki, M. Alam, and S. Ghosh, “Analysis of crosstalk in nisq devices and security implications in multi-programming regime,” in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 25–30.
- [213] A. A. Saki and S. Ghosh, “Qubit sensing: A new attack model for multi-programming quantum computing,” *arXiv preprint arXiv:2104.05899*, 2021.
- [214] A. Suresh, A. A. Saki, M. Alam, D. S. Ghosh, *et al.*, “A quantum circuit obfuscation methodology for security and privacy,” *arXiv preprint arXiv:2104.05943*, 2021.
- [215] S. Deshpande, C. Xu, T. Trochatos, Y. Ding, and J. Szefer, “Towards an antivirus for quantum computers,” in *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2022, pp. 37–40.
- [216] A. Mi, S. Deng, and J. Szefer, “Short paper: Device-and locality-specific fingerprinting of shared nisq quantum computers,” in *Workshop on Hardware and Architectural Support for Security and Privacy*, 2021, pp. 1–6.
- [217] S. Deshpande, C. Xu, T. Trochatos, H. Wang, F. Erata, S. Han, Y. Ding, and J. Szefer, “Design of quantum computer antivirus,” in *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, IEEE, 2023, pp. 260–270.
- [218] P. Das, A. Locharla, and C. Jones, “Lilliput: A lightweight low-latency lookup-table decoder for near-term quantum error correction,” in *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2022, pp. 541–553.
- [219] D. G. Cory, M. D. Price, W. Maas, E. Knill, R. Laflamme, W. H. Zurek, T. F. Havel, and S. S. Somaroo, “Experimental quantum error correction,” *Phys. Rev. Lett.*, vol. 81, pp. 2152–2155, 10 Sep. 1998.
- [220] P. Schindler, J. T. Barreiro, T. Monz, V. Nebendahl, D. Nigg, M. Chwalla, M. Hennrich, and R. Blatt, “Experimental repetitive quantum error correction,” *Science*, vol. 332, no. 6033, pp. 1059–1061, 2011. eprint: <https://www.science.org/doi/pdf/10.1126/science.1203329>.

- [221] O. Moussa, J. Baugh, C. A. Ryan, and R. Laflamme, “Demonstration of sufficient control for two rounds of quantum error correction in a solid state ensemble quantum information processor,” *Phys. Rev. Lett.*, vol. 107, p. 160 501, 16 Oct. 2011.
- [222] J. Zhang, D. Gangloff, O. Moussa, and R. Laflamme, “Experimental quantum error correction with high fidelity,” *Phys. Rev. A*, vol. 84, p. 034 303, 3 Sep. 2011.
- [223] M. D. Reed, L. DiCarlo, S. E. Nigg, L. Sun, L. Frunzio, S. M. Girvin, and R. J. Schoelkopf, “Realization of three-qubit quantum error correction with superconducting circuits,” *Nature*, vol. 482, no. 7385, 2012.
- [224] G. Waldherr, Y. Wang, S. Zaiser, M. Jamali, T. Schulte-Herbrüggen, H. Abe, T. Ohshima, J. Isoya, J. Du, P. Neumann, *et al.*, “Quantum error correction in a solid-state hybrid spin register,” *Nature*, vol. 506, no. 7487, pp. 204–207, 2014.
- [225] D. Ristè, S. Poletto, M.-Z. Huang, A. Bruno, V. Vesterinen, O. Saira, and L. DiCarlo, “Detecting bit-flip errors in a logical qubit using stabilizer measurements,” *Nat. Commun.*, vol. 6, Nov. 2014.
- [226] J. Cramer, N. Kalb, M. A. Rol, B. Hensen, M. S. Blok, M. Markham, D. J. Twitchen, R. Hanson, and T. H. Taminiau, “Repeated quantum error correction on a continuously encoded qubit by real-time feedback,” *Nat. Commun.*, vol. 7, no. 1, pp. 1–7, 2016.
- [227] J. R. Wootton and D. Loss, “Repetition code of 15 qubits,” *Phys. Rev. A*, vol. 97, p. 052 313, 5 May 2018.
- [228] J. R. Wootton, “Benchmarking near-term devices with quantum error correction,” *Quantum Science and Technology*, vol. 5, no. 4, p. 044 004, Jul. 2020.
- [229] L. Egan, D. M. Debroy, C. Noel, A. Risinger, D. Zhu, D. Biswas, M. Newman, M. Li, K. R. Brown, M. Cetina, *et al.*, “Fault-tolerant operation of a quantum error-correction code,” *Nature*, vol. 598, pp. 281–286, 7880 2021.
- [230] Y.-H. Luo, M.-C. Chen, M. Erhard, H.-S. Zhong, D. Wu, H.-Y. Tang, Q. Zhao, X.-L. Wang, K. Fujii, L. Li, *et al.*, “Quantum teleportation of physical qubits into logical code-spaces,” *Proc. of the National Academy of Sciences*, vol. 118, 36 2021.
- [231] B. Bell, D. Herrera-Martí, M. Tame, D. Markham, W. Wadsworth, and J. Rarity, “Experimental demonstration of a graph state quantum error-correction code,” *Nat. Commun.*, vol. 5, no. 1, pp. 1–10, 2014.
- [232] J. Marques, B. Varbanov, M. Moreira, H. Ali, N. Muthusubramanian, C. Zachariadis, F. Battistel, M. Beekman, N. Haider, W. Vlothuizen, *et al.*, “Logical-qubit

- operations in an error-detecting surface code,” *Nature Physics*, vol. 18, pp. 80–86, 1 2021.
- [233] C. Ryan-Anderson, J. G. Bohnet, K. Lee, D. Gresh, A. Hankin, J. P. Gaebler, D. Francois, A. Chernoguzov, D. Lucchetti, N. C. Brown, T. M. Gatterman, S. K. Halit, K. Gilmore, J. A. Gerber, B. Neyenhuis, D. Hayes, and R. P. Stutz, “Realization of real-time fault-tolerant quantum error correction,” *Phys. Rev. X*, vol. 11, p. 041 058, 4 Dec. 2021.
- [234] A. M. Steane, “Error correcting codes in quantum theory,” *Phys. Rev. Lett.*, vol. 77, pp. 793–797, 5 Jul. 1996.
- [235] S. Huang, M. Newman, and K. R. Brown, “Fault-tolerant weighted union-find decoding on the toric code,” *Phys. Rev. A*, vol. 102, p. 012 419, 1 Jul. 2020.
- [236] A. G. Fowler, “Minimum weight perfect matching of fault-tolerant topological quantum error correction in average $o(1)$ parallel time,” *Quantum Info. Comput.*, vol. 15, no. 1–2, pp. 145–158, Jan. 2015.
- [237] S. Dasgupta and T. S. Humble, “Stability of noisy quantum computing devices,” *arXiv preprint arXiv:2105.09472*, 2021.
- [238] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland, “Surface codes: Towards practical large-scale quantum computation,” *Phys. Rev. A*, vol. 86, p. 032 324, 3 Sep. 2012.
- [239] S. Bravyi and A. Kitaev, “Universal quantum computation with ideal clifford gates and noisy ancillas,” *Phys. Rev. A*, vol. 71, p. 022 316, 2 Feb. 2005.
- [240] IBM. “Quantum computing and ibm q: An introduction.” ().
- [241] Intel. “Intel cyclone 10 lp device overview.” ().
- [242] Intel. “External memory interface handbook volume 1: Intel fpga memory solution overview, design flow, and general information.” ().
- [243] Intel. “Intel stratix 10 embedded memory user guide overview.” ().
- [244] G. Pekhimenko, V. Seshadri, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, “Base-delta-immediate compression: Practical data compression for on-chip caches,” in *PACT-21*, ACM, 2012, pp. 377–388, ISBN: 9781450311823.
- [245] R. Pagh and F. F. Rodler, “Cuckoo hashing,” *Journal of Algorithms*, vol. 51, no. 2, pp. 122–144, 2004.

- [246] Y. Ueno, M. Kondo, M. Tanaka, Y. Suzuki, and Y. Tabuchi, “Qulatis: A quantum error correction methodology toward lattice surgery,” in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 274–287.
- [247] C. Chamberland, L. Goncalves, P. Sivarajah, E. Peterson, and S. Grimberg, *Techniques for combining fast local decoders with global decoders under circuit-level noise*, 2022.
- [248] S. C. Smith, B. J. Brown, and S. D. Bartlett, *A local pre-decoder to reduce the bandwidth and latency of quantum error correction*, 2022.
- [249] P. Das, A. Locharla, and C. Jones, “Lilliput: A lightweight low-latency lookup-table based decoder for near-term quantum error correction,” *arXiv preprint arXiv:2108.06569*, 2021.
- [250] R. Ayanzadeh, N. Alavisamani, P. Das, and M. Qureshi, “Frozenqubits: Boosting fidelity of qaoa by skipping hotspot nodes,” in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2023, pp. 311–324.
- [251] K. Temme, S. Bravyi, and J. M. Gambetta, “Error mitigation for short-depth quantum circuits,” *Physical review letters*, vol. 119, no. 18, p. 180509, 2017.
- [252] J. J. Wallman and J. Emerson, “Noise tailoring for scalable quantum computation via randomized compiling,” *Physical Review A*, vol. 94, no. 5, p. 052325, 2016.
- [253] D. Min, J. Kim, J. Choi, I. Byun, M. Tanaka, K. Inoue, and J. Kim, “Qisim: Architecting 10+ k qubit qc interfaces toward quantum supremacy,” in *Proceedings of the 50th Annual International Symposium on Computer Architecture*, 2023, pp. 1–16.
- [254] J. Tindall, M. Fishman, M. Stoudenmire, and D. Sels, “Efficient tensor network simulation of ibm’s kicked ising experiment,” *arXiv preprint arXiv:2306.14887*, 2023.

VITA

Poulami Das grew up in Kolkata, India. She completed a Bachelor of Technology in Electronics and Communication Engineering from the National Institute of Technology - Durgapur in 2012 and a Master of Science in Electrical and Computer Engineering from the University of Texas at Austin in 2016. Poulami completed her PhD in Electrical and Computer Engineering from the Georgia Institute of Technology, Atlanta, USA in 2023, where she was advised by Prof. Moinuddin Qureshi.

Poulami's research interests are broadly in the areas of computer architecture with a focus on improving the reliability of quantum computers. Her research has been published at several top computer architecture conferences. Poulami's PhD research has been supported in part by the Microsoft PhD Research Fellowship. Her works have also been awarded a Best Paper at Computing Frontiers, Best PhD Research Award at the DAC PhD Forum, and the Best PhD Proposal Award in ECE, Georgia Tech.