

Architecture Framework for Trapped-Ion Quantum Computer based on Performance Simulation Tool

by

Muhammad Ahsan

Department of Computer Science
Duke University

Date: _____

Approved:

Jungsang Kim, Supervisor

John Reif

Robert Calderbank

John Kubiawicz

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2015

ABSTRACT

Architecture Framework for Trapped-Ion Quantum Computer
based on Performance Simulation Tool

by

Muhammad Ahsan

Department of Computer Science
Duke University

Date: _____

Approved:

Jungsang Kim, Supervisor

John Reif

Robert Calderbank

John Kubiawicz

An abstract of a dissertation submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2015

Copyright © 2015 by Muhammad Ahsan
All rights reserved except the rights granted by the
Creative Commons Attribution-Noncommercial Licence

Abstract

The challenge of building scalable quantum computer lies in striking appropriate balance between designing a reliable system architecture from large number of faulty computational resources and improving the physical quality of system components. The detailed investigation of performance variation with physics of the components and the system architecture requires adequate performance simulation tool. In this thesis we demonstrate a software tool capable of (1) mapping and scheduling the quantum circuit on a realistic quantum hardware architecture with physical resource constraints, (2) evaluating the performance metrics such as the execution time and the success probability of the algorithm execution, and (3) analyzing the constituents of these metrics and visualizing utilization of resources to identify system components which crucially define the overall performance.

Using this versatile tool, we explore vast design space for modular quantum computer architecture based on trapped ions. We find that while success probability is uniformly determined by the fidelity of physical quantum operation, the execution time is a function of system resources invested at various layers of design hierarchy. At physical level, the number of lasers performing quantum gates impact the latency of the fault-tolerant circuit blocks execution. When these blocks are used to construct meaningful arithmetic circuit such as quantum adders, the number of ancilla qubits for complicated non-clifford gates and entanglement resources to establish long-distance communication channels, become major performance limiting factors.

Next, in order to factorize large integers, these adders are assembled into modular exponentiation circuit comprising bulk of Shor's algorithm. At this stage, the overall scaling of resource-constraint performance with the size of problem, describes the effectiveness of chosen design. By matching the resource investment with the pace of advancement in hardware technology, we find optimal designs for different types of quantum adders. Conclusively, we show that using three million qubits, the 2,048-bit Shor's algorithm can be reliably executed in five months.

To my parents, my wife and all those who encouraged me to complete my dissertation

Contents

Abstract	iv
List of Tables	xiii
List of Figures	xv
Acknowledgements	xviii
1 Introduction	1
1.1 Why Quantum Computation?	1
1.2 Quantum Noise: The Main Enemy of Quantum Computer	3
1.2.1 How Noise Affects Correctness of Computation	3
1.2.2 Achieving Tolerance Against Noise	4
1.2.3 Resource Overhead in Fault Tolerant Quantum Computing . .	6
1.3 Quantum Computer Architecture	8
1.3.1 Achieving Fault Tolerance in Quantum Architecture	10
1.3.2 The Trade-offs between Resource and Performance	11
1.3.3 Simultaneous Reduction of Execution Time and Failure Prob- ability	13
1.4 Performance Simulation and Modeling Quantum Computers	16
1.4.1 Why Performance Simulation?	16
1.4.2 Main Components of the Performance Simulation	17
1.5 The Contribution of the Thesis	18
1.5.1 A New Performance Simulation Tool	18

1.5.2	The Summary of New Research Problems	20
1.6	Summary	23
2	Basics of Quantum Computing	24
2.1	Qubits	24
2.2	Quantum Gates and Measurement	27
2.2.1	Single-qubit Gates	27
2.2.2	Multi-qubit Gates	30
2.2.3	Quantum Measurement	30
2.2.4	Classically Controlled Quantum Gates	32
2.2.5	Sample Quantum Circuit: Quantum Teleportation	33
2.3	Universal Quantum Computation	34
2.4	Quantum Noise	35
2.5	Fault-tolerant Quantum Circuits	37
2.5.1	The Choice of Quantum Error Correcting Code	37
2.5.2	Hamming Code	39
2.5.3	Quantum CSS Codes	41
2.5.4	Stabilizer description of Steane code	43
2.5.5	Fault-tolerant gates in Steane code	46
2.5.6	Concatenated Quantum Error Correction	50
2.6	Summary	59
3	Trapped-Ion Quantum Hardware	60
3.1	Introduction to the Trapped-Ion Quantum Computer	61
3.2	Architectures for Trapped-Ion Quantum Computers	63
3.2.1	Why quantum architecture?	63
3.2.2	Brief Survey of Trapped-Ion Quantum Computers	65

3.2.3	Feasibility of Quantum Architecture	71
3.3	MUSIQC Hardware	71
3.4	Architecture Support in MUSIQC Hardware	76
3.5	Summary	77
4	Our Design and Performance Simulation Tool Box	78
4.1	Introduction to the Design and Performance Simulation	78
4.1.1	Motivation Behind the Simulation	78
4.1.2	Deficiencies in the prior tools	79
4.1.3	Unique features of our tool set	80
4.2	Brief Survey of Prior Tools or Infrastructures	81
4.3	Our Design and Performance Simulation Tool Box	84
4.4	Fault-Tolerant Circuit Generator	86
4.5	Low Level Mapper	88
4.5.1	Qubit Partitioning Problem Definition	89
4.5.2	Qubit Placement Problem Definition	90
4.6	Low Level Scheduler	94
4.7	Low Level Error Analyzer	97
4.8	Tile Database	101
4.9	Quantum Application Circuit Generator	103
4.10	Brief description of basic Architecture model in ADPA	104
4.11	High Level Mapper	105
4.12	High Level Scheduler	106
4.12.1	Dynamic Resource Allocation in Cross-Layer Scheduling	111
4.12.2	The Correctness and the Optimality of the Scheduler	111
4.13	High Level Error Analyzer	113

4.14	Performance Metrics Decomposer	116
4.15	Visualizer	118
4.16	Summary	120
5	Performance Simulation based on Hardware Resources Constraints	129
5.1	Motivation of Study	130
5.2	Hardware, Architecture Model and Definitions	132
5.3	Tool Components and Overall Design Flow	134
5.3.1	Mapping	136
5.3.2	Scheduling	137
5.3.3	Quantifying Architecture Support for Fault-tolerance	140
5.4	Simulation of Bernstein-Vazirani Algorithm	140
5.4.1	Simulation Results	143
5.4.2	Analysis of Resource Reduction	146
5.5	Discussions	147
5.5.1	Tool Testing, Verification and Validation	147
5.5.2	Running Time and Scalability of the Tool	148
5.6	Summary	151
6	Optimization of a Quantum Computer Architecture	153
6.1	Motivation of Study	154
6.2	Quantum Hardware and Architecture Models	156
6.2.1	Quantum Hardware Model	156
6.2.2	Quantum Architecture Model	157
6.2.3	Error Model and Baseline Device Parameters	159
6.2.4	Benchmark Application Algorithm	160
6.3	Tool Description	160

6.4	Simulation Results	162
6.4.1	Searching in the Architecture Space	164
6.4.2	Selecting an Architecture	166
6.4.3	Improving Performance through Device Parameters	170
6.5	Summary	172
7	Designing a Million-Qubit Quantum Computer	173
7.1	Motivation of Study	174
7.2	Quantum Circuits Revisited	176
7.2.1	Universal Quantum Gates for the Steane code	176
7.2.2	Benchmark Circuits	178
7.3	Quantum Hardware and Quantum Architecture Models	183
7.3.1	Quantum Hardware Model	183
7.3.2	Quantum Architecture Model	185
7.4	Tool Description	189
7.4.1	Tool Validation and Performance	194
7.5	Simulation Results	195
7.5.1	Resource-Performance Scalability	196
7.5.2	Resource-Performance Trade-offs	199
7.5.3	Performance Scaling under Limited Resources	202
7.5.4	Design Optimization by Tuning Device Parameters	204
7.6	Conclusion	208
8	Conclusion and Future Work	210
8.1	Summary of Important Results	210
8.2	Directions for Future Research	213
8.2.1	Introducing Heuristics in Searching Design Space	213

8.2.2	Mapping and Scheduling Algorithms	214
8.2.3	Support for Other Error Models	215
8.2.4	Improving Models of Physical Device Components	216
8.2.5	Support for Multiple Quantum Technologies	217
8.2.6	Support for Multiple Error-correcting Codes	217
8.3	Summary and Final Word	218
A	Bimodal Error Correction	219
A.1	Condition for Quantum Error Correction	220
A.2	Double Error Correction	220
A.2.1	The $[[3,1,3]]$ code	221
A.2.2	Steane $[[7,1,3]]$ code	222
A.2.3	Bacon-Shor $[[9,1,3]]$ code	223
	Bibliography	226
	Biography	236

List of Tables

1.1	Gate infidelity in various quantum technologies	4
2.1	Hamming Codewords	42
2.2	Steane code Stabilizers	44
2.3	Steane code error correction	44
2.4	X and Z Error Propagation	45
2.5	Transversal logic gates in Steane code	47
3.1	State-of-art trapped-ion performance numbers	63
4.1	Qubit Placement Algorithm Steps	91
4.2	Updating Block Physical Failure Probability	124
5.1	Tabular data for Figures 5.4 and 5.5 for $AvEPR=1$	145
5.2	Tabular data for Figures 5.4 and 5.5 for $AvEPR=7$	146
5.3	Tabular data for Figure 5.6	147
6.1	L2 Tile performance numbers for chapter 6	162
6.2	The composition of total failure probability, chapter 6	167
6.3	Reducing failure probability in SA-Regime	169
6.4	Reducing failure probability in Tel-Regime	170
6.5	Effective Execution Time for modular exponentiation	170
7.1	Device Parameters (DPs)	185
7.2	Composition of L2 Tile	189
7.3	L2 Tile performance numbers chapter 7	192

7.4	The scaling of failure probability	198
7.5	Breakdown of execution time and failure probability, chapter 7	198
7.6	Optimal architecture configuration, chapter 7	199
A.1	Double error syndromes for $[[3,1,3]]$ bit flip code	221
A.2	Double X error syndromes for $[[7,1,3]]$ code	222
A.3	Double Z error syndromes for $[[7,1,3]]$ code	222
A.4	Gauge operators for $[[9,1,3]]$ code	224
A.5	Double X error syndromes for $[[9,1,3]]$ code	225
A.6	Double Z error syndromes for $[[9,1,3]]$ code	225

List of Figures

1.1	Architecture dependent Shor's algorithm performance	7
1.2	The qubit fidelity with layers of concatenation	12
1.3	Simultaneous reduction of execution time and failure probability . . .	13
1.4	The philosophy of the toolbox	20
2.1	The Bloch Sphere	26
2.2	Commonly used single-qubit quantum gates	28
2.3	Commonly used multi-qubit quantum gates	29
2.4	Controlled- U gates	29
2.5	Commonly used Measurement operators	32
2.6	Operator Measurement Circuit	33
2.7	The Teleportation Circuit	34
2.8	X and Z error detecting procedure	45
2.9	Non fault-tolerant vs fault-tolerant measurement	51
2.10	Encoding and Decoding of a 3-cat state	51
2.11	Encoding and Decoding of a 4-cat state	52
2.12	Encoding and Decoding of a 7-cat state	53
2.13	Steane Error Detection Circuit	54
2.14	Fault-tolerant Steane $ 0\rangle_L$ preparation	54
2.15	Fault-tolerant circuit for the magic state $T +\rangle$	55
2.16	Fault-tolerant circuit to perform Steane T gate using magic state $T +\rangle$	56

2.17	Fault-tolerant circuit to perform Steane Toffoli gate	57
3.1	Monolithic trapped-ion quantum hardware	65
3.2	Quantum Logic Array Architecture	66
3.3	Compressed Quantum Logic Array Architecture	68
3.4	Qalypso	70
3.5	MUSIQC hardware	72
3.6	Components of ELU	72
3.7	Heralded Entanglement	73
4.1	Overview of our design and performance simulation tool	85
4.2	Dependency Graph in Circuit Generators	87
4.3	Single Chain to Multi-chain Mapping	92
4.4	Qubit placement problem sample solution	93
4.5	Physical circuit graph	94
4.6	Fault Counter Method	99
4.7	High Level Mapper role demonstration	106
4.8	Optimality of the Scheduler	107
4.9	Cross-Layer optimization	124
4.10	Breakdown of execution time	125
4.11	Sample visualization of scheduled AQFT circuit	126
4.12	Sample visualization of scheduled QRCA circuit	127
4.13	Sample visualization of scheduled QCLA circuit	128
5.1	MUSIQC architecture for Bernstein-Vazirani algorithm	132
5.2	Tool flow used in chapter 5	135
5.3	Bernstein-Vazirani Algorithm and hardware parameters values	141
5.4	Execution time as function of available ancilla qubits and lasers	141

5.5	Success probability as function of available ancilla qubits and lasers	142
5.6	Performance variation with entanglement resources	142
5.7	Tool runtime as function of ancilla qubits and lasers	149
5.8	Tool runtime as function of entangling ions	149
6.1	Quantum architecture for QCLA circuit	155
6.2	Tool flow used in second study	163
6.3	Execution time as function of ancilla and communication Tiles	164
6.4	Failure probability variation with ancilla Tiles	167
6.5	Optimal architecture selection from SA- and Tel- regimes	168
7.1	Fault tolerant Toffoli gate	179
7.2	Shor’s algorithm block diagram	180
7.3	Fault tolerant controlled-rotation approximation	182
7.4	Architecture model used in chapter 7	186
7.5	Enhanced version of toolbox used in chapter 7	190
7.6	Tool Performance for various benchmark circuits	193
7.7	Resource-performance analysis of QCLA	195
7.8	Resource-performance analysis of QRCA	196
7.9	Resource-performance analysis of AQFT	197
7.10	Visualization of QCLA resource utilization	200
7.11	Performance scaling with the problem size	202
7.12	Reducing the failure probability of 2,048-bit QCLA circuit	208

Acknowledgements

Special thanks to the IARPA for funding the entire research, the committee members for the general support, the advisor for the necessary guidance required throughout this work.

Introduction

1.1 Why Quantum Computation?

Quantum computers can in principle efficiently solve certain practically important problems (e.g., integer factorization) which are deemed intractable using conventional computers. The quantum computation works in fashion similar to that of currently prevalent method of computation (classical computation) which processes the information stored in bits using sequence of instructions. The quantum bits (qubits) store quantum data manipulated by sequence of quantum gates. However, the computational advantage of quantum computation comes from the quantum mechanical properties of matter which include ‘superposition’ and ‘entanglement’. The superposition allows quantum computer to process exponentially larger data space than same size classical computer. In one time step, n -qubit quantum computer can process data space whose storage and processing would require 2^n classical bits and hardware units. During the execution of quantum algorithm, this exponential data space is tested for all possible solutions to the problem. However, in order to guide computation to obtain correct solution, the data space is so modified as to efficiently

eliminate incorrect solutions. This can be achieved by the quantum mechanical properties of qubits. In contrast to their classical counterparts, qubits possess not only the classical bit amplitude 0 or 1, but also the phase information $\exp(i\theta)$. These phases can be adjusted so that false candidate solutions are canceled out by the interaction among qubits through the application of quantum gates. The phase adjustment is accomplished by quantum entanglement which allows qubit to communicate both amplitude and phase information across each other. Once desired phase adjustment is obtained, data space is collapsed to correct solution which is readout as a last step of quantum algorithm. In general perfect cancellation of incorrect solutions is not possible due to either imperfect phase adjustment or the nature of the problem. In those cases quantum algorithm can be repeated multiple times to increase the probability of obtaining correct result. The computational speed of the quantum computer lies in its ability to efficiently span exponentially large data space using superposition and apply entanglement to help shrink this space to the correct solution. Both these tasks manipulate qubits amplitude and phases through appropriate sequence of quantum gates at every time step. The Shor's factorization quantum algorithm Shor (1997) is a prime example wherein this entire manipulation can be carried out in polynomial number of time steps which makes this algorithm run exponentially faster than its classical counterparts. Since the running time of an algorithm is described as a function of problem size, the difference between polynomial and exponential running time is far greater for larger size problems. In this sense we expect quantum computers to be useful in solving very large size problems which are prohibitively time expensive when attempted to be solved on classical computer systems. Hence, building scalable quantum is the main challenge in this field through which we can demonstrate practical supremacy of quantum computation. The goal of this thesis is to quantitatively evaluate the magnitude of this challenge and to provide concrete guidelines to solve this fundamental open problem.

1.2 Quantum Noise: The Main Enemy of Quantum Computer

1.2.1 *How Noise Affects Correctness of Computation*

For quantum algorithm to be practically useful, quantum computation needs to be implemented on a piece of hardware which has quantum mechanical properties. Several quantum technologies have been proposed to date Ladd et al. (2010b), however the physical implementations of quantum memory and operations in any technology are imperfect due to noisy quantum hardware Unruh (1995); Bacon (2003); Zurek (2003); Schlosshauer (2007); Mazzola et al. (2010). The qubits are fragile and quantum gates are unreliable. The qubits quickly begin to lose stored information with the passage of time due to insufficient isolation from the noisy environment, while the execution of quantum gate is inaccurate due to the incomplete control over qubit manipulation methods even in the state-of-art technology. The rate at which the qubit loses its information is called ‘decoherence rate’, while the reliability of quantum operation is described by the ‘failure probability’ or ‘infidelity’ quantifying the imperfection involved in carrying out intended operation. Several experimental methods have been proposed to quantify the decoherence rate and infidelity Blume-Kohout (2010); Schwemmer et al. (2014); Shabani et al. (2011). The Table 1.1 shows the state-of-art fidelities of quantum gates reported for various quantum technologies. Clearly these values are too high for lengthy quantum computation containing more than thousands of quantum gates. Furthermore, as qubits interact with each other during computation, the noise can easily spread from infected qubits to the non-infected qubits. The magnitude of the accumulated noise becomes significantly higher and begins to interfere with the superposition and entanglement operations of quantum algorithm. Consequently, the probability of obtaining correct solution becomes significantly lower which compromises the correctness of computation. Moreover, greater the size of computation, larger the amount of accumulated noise and

Table 1.1: State-of-art range of gate infidelity reported for various quantum technologies

Quantum Technology	Infidelity	Reference
Trapped-ion	10^{-6} - 10^{-3}	Kim (2014)
Superconductor	10^{-4} - 10^{-3}	Barends et al. (2014)
Neutral Atoms	10^{-3} - 10^{-2}	Han et al. (2014)
Photons	10^{-2} - 10^{-1}	Crespi et al. (2011)

higher the probability of quantum algorithm failing. Therefore the noise on one hand neutralizes the proficient features of quantum computation, on the other hand, it impedes the scalability of computation. Therefore, during the execution of quantum algorithm, adequate preventive measures are required to facilitate scalable quantum computation.

1.2.2 *Achieving Tolerance Against Noise*

In the battle against noise, sophisticated mechanism are devised which guard both quantum information and computation. The quantum information stored on single qubit is protected by encoded into multiple qubits (called physical qubits) using suitable error correcting code. The resulting encoded block of qubits is called ‘logical qubit’. The logical qubit provides significantly higher robustness to noise as long as the number of noise corrupted qubits in the block are fewer than certain threshold (also known as the code distance for a range of known error correcting codes). These erroneous qubits can be detected and corrected for errors using well-known ‘quantum error correction’ procedures Shor (1995); Laflamme et al. (1996); Calderbank and Shor (1996); Steane (1996); Kitaev (2003) which act as shield against noise.

The basic principle of a large set of quantum error correction is derived from classical error-correcting codes, wherein, logical qubit corresponds contains classical codewords in superposition while the erroneous bit(or qubit) location is determined by the sequence of parity checks or syndrome measurement. There are two impor-

tant difference which make the design of quantum error correcting codes (QECC) challenging Nielsen and Chuang (2000). First, quantum information needs to be protected not only from classical bit (amplitude) flips but also from phase flip errors since qubit has both amplitude and phase as discussed earlier. This demands careful construction of logical qubit state from classical codewords. Second, In the classical error detection procedure syndrome extraction is usually trivial since, the bits of the codewords can be directly compared without affecting the codeword itself. However in case of quantum computation, this is far more complicated since qubits states cannot be directly compared without modifying logical qubit! unless qubit state is first copied or saved before the comparison is performed. Unfortunately, quantum mechanics forbids qubit state to be copied to another due to the ‘no-cloning theorem’ Wootters and Zurek (1982). Can we devise an error detection procedure which can extract syndrome while stabilizing the logical qubit state? Fortunately, this can be achieved by allocating a set of helping qubits (called ancilla qubits) and assign them an special state. Ancilla qubits interact with logical qubit using sequence appropriate entangling quantum gates. Their initial state is so designed that ancilla qubits can extract error information from the logical qubit without destabilizing it. Designing these special ancilla qubit state and finding their concise description is a major milestone in the area of quantum error correction and generally described by ‘stabilizer formalism’ Gottesman (1997). The investigation of various QECC and their properties is a vibrant topic in the field and invites several expertise from the field of information theory.

To protect computation, quantum gates must be also be implemented in some ‘encoded’ procedure. This is achieved by ensuring that gates are applied without decoding the encoded logical qubit(s). Such a gate is called ‘logical gate’. The logical gate should preserve the logical sense of desired operation on the logical qubit(s). It involves several constituent (physical) gates operating on the physical qubits of the

operand logical qubit(s). These constituent gates can potentially spread errors due to the interactions among qubits. Given that the spread of noise is carefully controlled, the logical gate provides sufficiently higher protection against noise when followed by error correction to correct erroneous qubits resulting from imperfect physical gates. Hence there are two important consideration while designing a logical gate (1) it should realize to the intended operation on the logical qubit operands (2) it should restrict the spread of errors within the encoded block of logical qubit such that these errors can be corrected by the error correction procedure. These requirements are met by applying a special set of physical gates depending upon the chosen error correcting code and the intended logical gate. A logical gate fulfilling these requirements is called fault ‘tolerant quantum gate’. The fault-tolerant execution for some quantum gates only requires bit-wise application of physical gates on the constituent qubits of the logical qubit block. However, for certain gates which typically arise in interesting application such as Shor’s algorithm, the fault tolerant construct is non-trivial. In those case fault tolerance requires preparation of special ancilla qubit state which becomes resource and time consuming. In a large scale quantum computation tolerance against noise is achieved when error correction is applied between sequence of logical gates. Logical gates suppress the spread of errors which are eliminated by the subsequent error correction.

1.2.3 Resource Overhead in Fault Tolerant Quantum Computing

Fault tolerance is an essential ingredient of achieving scalable quantum computation with adequate reliability. However, the price paid in executing quantum algorithm fault tolerantly includes large number of (1) physical qubits to encode quantum information in logical qubits (2) physical gates to perform error correction and logical gates. The performance of quantum algorithm depends on the overhead of hardware resources required to realize these constituent qubits and physical gates. By invest-

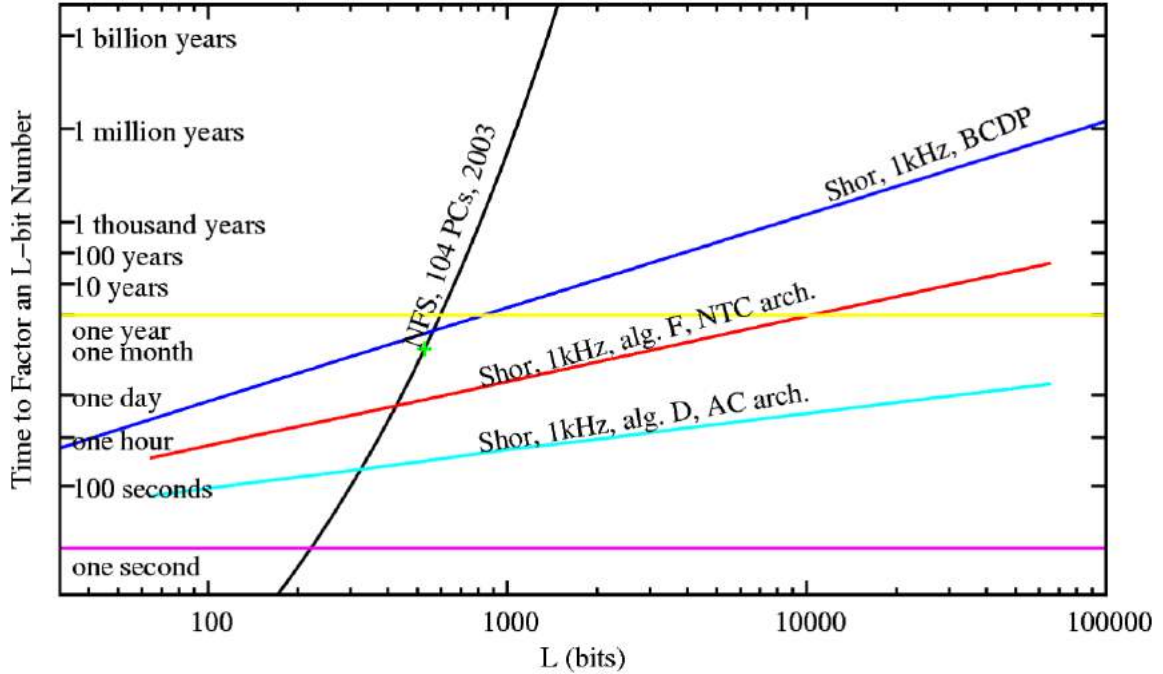


FIGURE 1.1: (Included with the permission of Ref. Van Meter et al. (2006)) For various quantum architectures, the execution time of Shor's integer factorization algorithm is plotted against problem size and compared with that of best known classical Number Field Sieve (NFS) algorithm for integer factorization. The acronym AC stands for arbitrary connectivity (no architecture constraints), NTC for the nearest-neighbor and BCDP for the more realistic architecture Beckman et al. (1996a) for the modular exponentiation part of the Shor's algorithm.

ing more qubits we can not only expedite the process of extracting the erroneous qubits but also construct larger encoded blocks which can recover from more erroneous qubits. When sufficient resources (qubits) are available, higher performance can be achieved since the total execution time (synonymously used with 'latency' in this thesis) required to obtain correct result from quantum algorithm will be short. The quantification of trade-offs between performance gains resource investment for different types of quantum algorithm is an interesting open problem which has resulted in appreciable contribution to the field. A set of some interesting results have been reported in Ref Suchara et al. (2013a).

1.3 Quantum Computer Architecture

In order to be practicable, fault tolerance should ensure that quantum algorithm can be executed on realistic noisy hardware with various types of constraints. When applications are mapped onto the physical device, the constraints imposed by the characteristics of the hardware can adversely affect the performance of algorithm. One of the main performance-deciding factor is the efficiency of qubit-qubit interaction. Qubits need to communicate with each other to allow for quantum mechanical interaction accomplished by multi-qubit gates in the algorithm and fault tolerant protocols. Many quantum device technologies only allow hardware architecture to enable interaction among qubits located in the vicinity. These are called nearest neighbor systems. However, as the number of qubits increase in a scalable quantum system, the physical locality among qubits is inevitably compromised and nearest neighbor architecture will become infeasible unless additional mechanism of communication or resources are provided. The interaction among qubits located far apart in the hardware imposes a generic challenging problem due to the no-cloning theorem. The qubit information cannot be copied and therefore cannot be transmitted by the classical wire. Fortunately, there are several alternative mechanisms for the transfer of quantum information. In some technologies it is possible to physically transported qubit from one physical location to another to ‘move’ quantum information across the system. However, there are general quantum mechanical methods in which qubit information can be ‘swapped’ or ‘teleported’ Bennett et al. (1993) to another with the help of extra qubits dedicated for communicated purposes (called communication qubits). These methods which facilitate non-local qubit communication are independent of chosen device technology. This means that for any quantum hardware we can architect ‘quantum channels’ using quantum swapping and teleportation at the cost of communication qubits overhead. The allocation and placement of these ad-

ditional qubit resources in the hardware are part of the general problem of designing a quantum computer architecture. The hardware constraints which define quantum architecture play vital role in estimating the overall performance of the quantum algorithm and its speedup over its classical counterpart. The dependence of the Shor's algorithm execution time on the choice of architecture is shown in Figure 1.1.

An important consideration in quantum architecture is the support of fault tolerance. The allocation of ancilla blocks for error correction and fault tolerant quantum gates crucially determine the ability of system to combat noise buildup which in turn affects the overall performance. During the execution of quantum algorithm, these ancilla blocks are continuously consumed by logical qubits as part of fault tolerant protocols. The distribution of these blocks among logical qubits for consumption is another example of architecture choice. We can architect a system in which each logical qubit is assigned dedicated fixed sized ancilla block *Metodi et al. (2005)*. In this architecture the speed of executing quantum algorithm fault tolerantly is achieved at the cost of large number of physical qubit resources and size of the hardware. Alternatively, one can design a system which allows the sharing of variable size ancilla blocks *Thaker et al. (2006)* among logical qubits by compromising the parallelism in the quantum algorithm. In this case the algorithm can be executed with fewer physical qubits. However, the resource efficiency comes with the price of increased execution time *Thaker et al. (2006)*. Once basic support for the fault tolerance is ensured, the architecture for connecting different blocks of ancilla and logical qubits needs to be specified. The bandwidth of communication across different types of blocks depends upon (1) architecture of ancilla blocks (2) the placement and (3) connectivity patterns application level logical qubits specified by quantum algorithm. Bandwidth may also depend on area dedicated for the physical transportation of qubits across the hardware if permitted by the device technology. The performance of the scalable quantum system critically depends upon the bandwidth

of chosen communication channels. Higher bandwidth can deliver better performance since the latency of communication among qubits can be reduced. However, increased bandwidth entails investment of communication qubits or area in the hardware which leads to the similar resource-performance trade-offs observed in the design of ancilla blocks. Conclusively, the study of fault tolerance and connectivity aspects of the architecture necessarily involve trade-offs between auxiliary resources (ancilla or communication qubits) and the performance. The detailed investigation of these trade-offs, the quantification of performance and resource consumption for range of architecture types comprise interesting open problems addressed in this thesis.

1.3.1 Achieving Fault Tolerance in Quantum Architecture

One of the main significance of analyzing resource-performance trade-offs is the quantification of architecture resources requirement for a practical scalable system executing quantum algorithm fault tolerantly. If the architecture is not taken into consideration, the rough estimate of physical resources can be derived by the quantum threshold theorem Aharonov and Ben-Or (1997) when the noise level in the device components, the target reliability of quantum algorithm (described by the probability of its (un)successful execution) and the size of the its circuit is known. The quantum algorithm circuit containing $p(n)$ gates will fail with probability at most ϵ , requires the resource overhead which scales poly-logarithmically in $\frac{p(n)}{\epsilon}$ given that each physical gate execution fails with probability less than p_{th} . This remarkable result is called quantum threshold theorem which promises that resource overhead is smaller compared to the gain in the accuracy of algorithm if noise per quantum gate is below certain threshold. When threshold condition ($p < p_{th}$) is met, multiple layers of encoding per logical qubit are used to make inaccuracy arbitrarily small by as $\epsilon = p(n)(cp)^{2k}/c$ where $c > 1/p_{th}$ and k is the number of encoding layers (called concatenation levels). A large set of QECC which are termed as concatenated

quantum codes feature multiple layers of encoding. For distance-3 Steane $[[7,1,3]]$ code Steane (1996) which can correct arbitrary single qubit error, the impact of multiple layers of encoding on the qubit fidelity is shown in Figure 1.2. With the addition of the concatenation layer, the number of correctable error events increases due to the higher code distance. However, it is important to note that the effect of increasing code distance can be simulated by ways other than concatenation, for example, by directly using higher distance or more famous topological quantum codes Kitaev (2003). The threshold values for the variety of error-correcting codes fall into the range 10^{-5} to 10^{-2} depending upon the structure of code and/or resource investment Aliferis et al. (2005). Compared to the concatenated codes, the topological codes provide superior shield against noise at the cost of significantly large number of simultaneous operations per parity check in one round of error correction. A realistic hardware architecture which can support such computational load seems to be difficult to realize in practice. Therefore in this thesis we focus on concatenated codes only. Nevertheless, for both concatenated and topological codes, several quantum computer architectures Metodi et al. (2005); Van Meter et al. (2008); Whitney et al. (2009); Kim and Kim (2009); Monroe et al. (2014); Galiutdinov et al. (2012); Fowler et al. (2012a) have been proposed which support fault tolerance execution of quantum application.

1.3.2 The Trade-offs between Resource and Performance

Even through the quantum threshold theorem provides hope for the scalable fault tolerant quantum computation, it makes very simplified assumptions about the quantum hardware and architecture. These are derived from the basic principle that by applying error correction at regular intervals, time scale over which qubit acquires errors is much longer than the time taken to correct for those errors. Hence the theorem requires that (i) hardware supports large number of parallel quantum gates

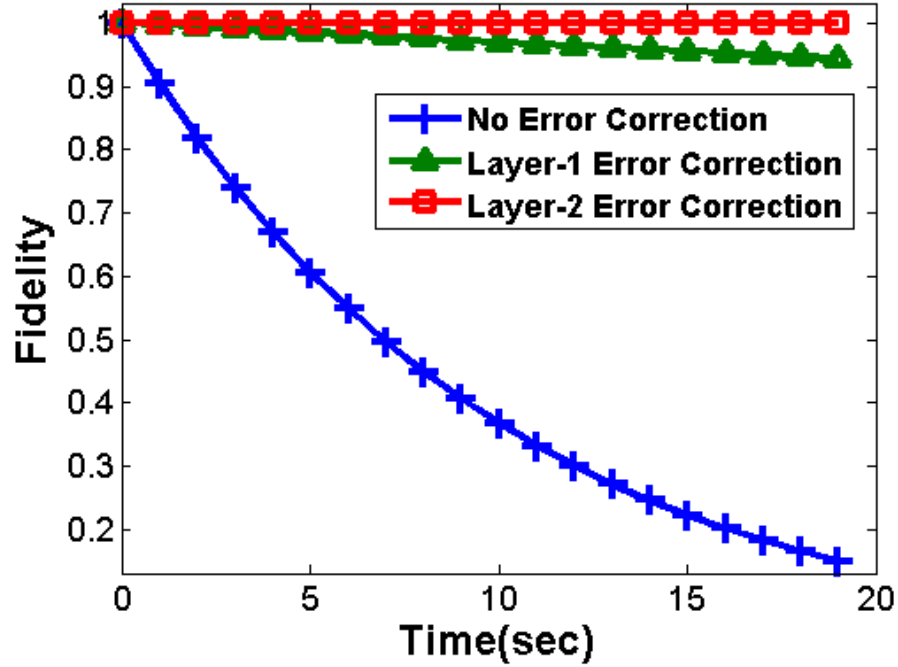


FIGURE 1.2: The impact of multiple layers of encoding on the qubit fidelity. The coherence time is assumed to be 10s, gate failure probability is taken as 10^{-5} which is slightly less than the threshold of Steane code used for error-correction.

(ii) large supply of ancilla blocks is available in the architecture (iii) the qubit communication cost is negligible. These assumption ensure that errors accumulated in the logical qubit during logical gates or due to decoherence are very few, and therefore, correctable in the upcoming round of error correction. Unfortunately, these assumptions may not hold in a realistic quantum system architecture which can only supply limited hardware resources (parallel operations support, ancilla and communication qubit). Such constraints adversely affect the capacity of system to suppress noise. When concurrently executable components of error correction circuit are serialized due to the constraints or insufficient resources, the decoherence errors add up faster than are corrected. The performance of an architecture built on limited hardware resources will be significantly worse than the one assuming unlimited resources.

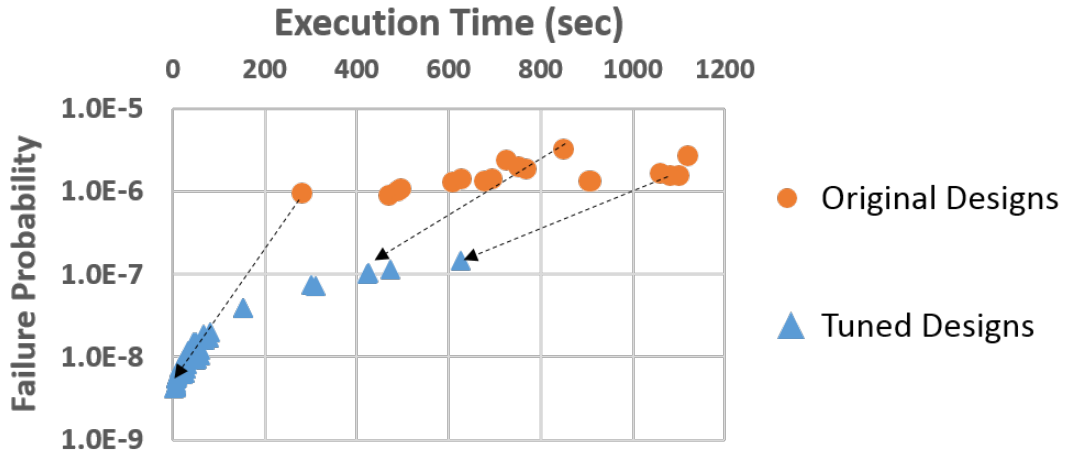


FIGURE 1.3: Design optimization by reducing both execution time and the failure probability of 2,048-bit Quantum Carry Look-Ahead Adder Draper et al. (2006). A set of original designs contains 1.5 million qubits with gate latency set to baseline values. On the other hand, a set of tuned designs contains 2.85 million qubits along with 10x reduction in the infidelity of communication channels and baseline latency of certain gates. Dotted arrows show how different original designs were tuned to allow simultaneous reduction in both execution time and failure probability.

Therefore, in the presence of architecture description quantum threshold theorem cannot adequately quantify the relationship between performance and resource overhead. For desired performance level, the theorem provides very optimistic resource estimate which can be used as initial guidance for the system design. The actual resource requirement will be significantly higher to satisfy more stringent fault tolerance requirement imposed by the practical quantum hardware. For accurate and quantification of resource-performance trade-offs, the fault tolerant quantum algorithm should be simulated on the hardware with realistic constraints.

1.3.3 Simultaneous Reduction of Execution Time and Failure Probability

The quantum threshold theorem guarantees the gain in reliability at the cost both additional qubits and gates executed in the fault tolerant procedures. Executing more gates increases the overall execution time which is a general price paid for the

decreased failure probability of the quantum algorithm. Ideally we want to not only reduce the failure probability to increase robustness against noise but also minimize the execution time to retain the speedup offer by the quantum computer as shown in Figure 1.1. When quantum algorithm is mapped onto the physical device, the hardware constraints begin to define the overall performance of application. In this subsection we reason that the goal of simultaneous reduction in the latency and unreliability becomes a task of finding suitable computer design by taking into the consideration both the nature of quantum hardware and the description of architecture.

In the quantum architecture study the efficacy of design is tested by the yardstick of evaluation metrics which should be adequately defined. One such metric is defined as Area Delay-to-Correct Result (ADCR) Whitney et al. (2009) which is the product of physical area occupied by the quantum computer and the ‘effective execution time’ of the quantum algorithm. The physical area captures the total qubit resources allocated for computation and communication while, effective execution time which is obtained by dividing the actual execution time by the success probability ($= 1 - \text{failure probability}$), describes the number of times quantum algorithm may be repeated in order to obtain the correct result. In other words, the effective execution time is a time to the successful execution of quantum algorithm. Based on the definition of ADCR metric, an ‘optimal’ quantum computer design is one which minimizes the product of effective execution time and the area of the quantum hardware.

The utility of ADCR lies in generating holistic characterization for the design. It lumps fundamental evaluation metrics by assigning same weight cost to the performance (execution time and failure probability) and resource variables (area or total qubits). However, as quantum hardware technology continues to evolve, these costs may change over time leading to different specification of constraints across various

metrics. Moreover, in order to achieve desired level of performance within the constraint of resource budget we require deeper insights into the resource utilization patterns and the performance limiting factors. Due to these reasons we believe that it is more beneficial to decompose ADCR back into the fundamental metrics and study their complicated interrelationship in a more general and flexible framework of resource-performance trade-offs. The main advantage of this analysis strategy lies in evaluating the sensitivity of performance metric to the specific component of design which can be tuned to control the amount of performance gain.

One of the most crucial merit of our approach is the ease of simultaneously reducing the execution time and failure probability. These metrics are shown as scatter plots in Figure 1.3 where each point is associated with specific design (parametrized by the variables of architecture and device hardware) of a quantum computer to execute 2,048-bit Quantum Carry Look-Ahead Adder (QCLA) circuit Draper et al. (2006). The original designs are restricted to contain no more than 1.5 million qubit resource and the performance (e.g., execution time and failure probability of physical qubits and gates) of physical hardware assigned baseline values. These designs are tuned by doubling the qubits and applying 10x reduction in the infidelity of communication channel and the baseline latency of selected physical operations. The dotted arrows in the plot show that tuned designs provide reduction in both execution time and the failure probability. In chapter 7 we use this analysis show that by lowering failure probability close to 10^{-9} and execution time to less than 0.8 seconds, 2,048-bit Shor's algorithm can be successfully executed in less than five months. This remarkable result was obtained by identifying crucial parameters of quantum computer whose tuning is responsible for the simultaneous improvement in both execution time and the failure probability. These valuable insights into the design space were obtained from our versatile performance simulation software tool which extracts performance limiting factors and patterns of resource consumptions.

The construction and the components of our tool to study quantum computer design space are detailed in chapter 4.

1.4 Performance Simulation and Modeling Quantum Computers

1.4.1 *Why Performance Simulation?*

Complete simulation a quantum circuit containing n – *qubits*, entails tracking 2^n complex numbers describing the hilbert space spanned by the superposition (and entangled) states of the qubits. It is clear that as quantum system scales, complete simulation becomes quickly infeasible due to the exponential scaling of states in the number of qubits. For certain quantum circuits in which the qubit state after each gate is describable by stabilizer formalism, we can evade this exponential tracking Gottesman (1997). Even though the set of such circuit is very small, some of them are widely used in quantum error correction as described in chapter 2. In general for practically interesting quantum circuits such as Shor’s algorithm, the full scale simulation becomes impractical due to the exponential scaling of the run time when quantum computation is simulated on the conventional computers. The fact that we cannot efficiently simulate a quantum algorithm on classical computer makes quantum computer exclusive choice for certain vital application. Nevertheless, quantum circuit simulation is an inefficient choice for investigating resource-performance trade-offs.

Alternatively, if we are only interested in the performance of the quantum circuit, we can utilize the concept of efficient ‘performance simulation’ to obtain the intended performance estimates. There are two main tasks of a performance simulation software tool. First, it generates model of quantum circuit components and hardware technology and the architecture. These models are typically described by set of parameters. Second, it simulate these models to extract resource and performance estimates. The advantage of these tools lies in efficient simulation of the

models which are described by a small set of parameters. At the same time the estimates provided by the tool are fairly accurate since the model parameters are derived from the well known physical properties of the quantum hardware, and experimental efforts to build quantum computers. One of the main use of such tools is to guide these efforts in the right direction.

1.4.2 Main Components of the Performance Simulation

Before estimates can be computed performance simulation tool also perform additional preliminary tasks of graphing fault tolerant quantum algorithm on hardware. This includes ‘mapping’ and ‘scheduling’. The mapping involves assigning physical resource to the qubits in the circuit. The scheduling generates hardware protocol and correct time instances at which of quantum gates are applied. The generated sequence of physical gates should abide by the gate-level dependencies described by the actual circuit. Both tasks should be performed respecting the hardware constraints and limited resources which make mapping and scheduling non-trivial problems. Once circuit execution is pictured on the hardware, performance simulation can be carried out to compute the resource investment, total execution time and the overall failure probability of algorithm. The resource estimate can be obtained at the time of mapping qubits to the hardware. The execution time can be obtained from the description of completely scheduled circuit. The failure probability is evaluated by using detailed fault tolerance analysis of the scheduled circuit. A set of noteworthy efforts to build such tools (see chapter 4 for details) can be found in Refs. Svore et al. (2006a); Balensiefer et al. (2005a); Whitney et al. (2007); Balensiefer et al. (2005b); Whitney et al. (2009); Fowler et al. (2012a).

1.5 The Contribution of the Thesis

1.5.1 *A New Performance Simulation Tool*

In this thesis, we present a performance simulation tool which fulfills the deficiencies in the similar set of tools reported in the prior studies. In addition to mapping, scheduling, our tool features unique capabilities to

- simulate performance over varying hardware technology parameters
- allow dynamic resource allocation by changing architecture parameters
- facilitate variable resource allocation (ancilla or communication qubits) across different layers of encodings (cross-layer optimization)
- build sophisticated models to simulate fault tolerance and analyze failure probability of the quantum algorithm (advanced error analysis methods)
- provide detailed breakdown of resource and performance metrics
- enable visualization of resource utilization over a range of benchmark applications.

Using these additional flexibilities, the deeper insights into resource-performance trade-offs can be obtained as shown in Figure 1.4. By varying technology, architecture and resource parameters, we can search very large design space for the quantum system. It can simulate variety of architectures to search for the optimal design for given quantum algorithm. The breakdown of metrics reveals the performance limiting factors in the design while the visualization enhances our understanding of resource utilization during the execution of quantum algorithm. Similarly, by matching resource allocation to the fault tolerance needs of each encoding layer, the optimization is benefited from additional degree of freedom in the design space. Finally,

the tool offers advanced error analysis method to compute the failure probability of different components of quantum circuit. By virtue of these features, our tool makes substantial contribution towards the area of quantum computer performance simulation.

With this tool we simulate different benchmarks quantum circuits on the trapped-ion based computer architecture. The trapped-ion Cirac and Zoller (1995) is one of the most promising candidate technologies has several desirable properties needed for scalable quantum computer, which are described in chapter 3. Several types of baseline architectures were analyzed through which the impact of varied resources allocation on the performance were studied. The benchmark circuits range from simple four qubit Bernstein-Vazirani Bernstein and Vazirani (1993a) circuit to sizable thousands of qubits quantum adders Draper et al. (2006); Cuccaro et al. (2004) and quantum fourier transform Kitaev (1995). The tool went through several stages of development. The initial version which could simulate small circuits (around 1,00 qubits and 1,000 gates) on a small-scale hardware was gradually evolved to handle fairly large sized circuits (between $10^5 - 10^6$ logical gates) on scalable hardware. Through different phases of tool enhancement, the mapping, scheduling and performance analysis techniques and were gradually improved according to the increasing complexity of advanced architecture. During each phase, the specific problem within the context of resource-performance trade-offs analysis was systematically defined. The insights gained from the results of one set of analysis were used to increase the scope of next set of problems. By progressively constructing towards more meaningful questions, the thesis culminates by estimating the performance of the largest size Shor's algorithm which can be executed with limited number of physical qubits.

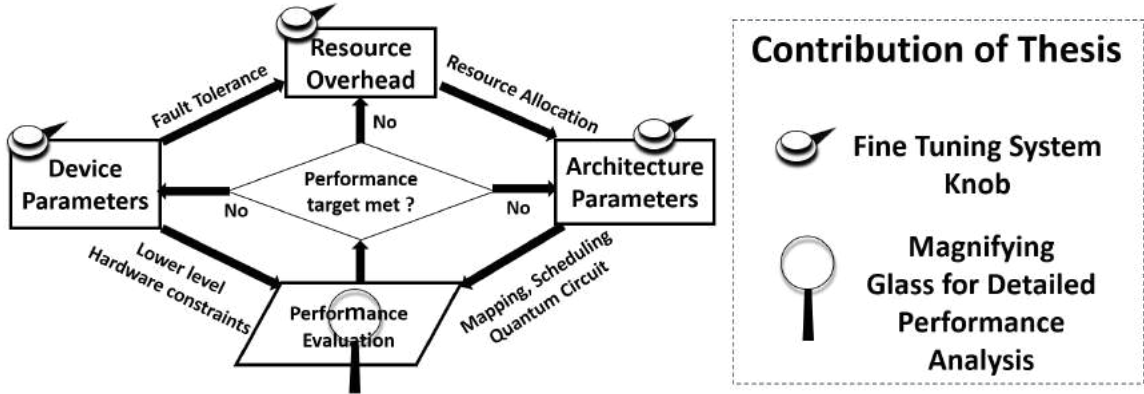


FIGURE 1.4: The philosophy of our flexible toolbox. It offers ‘knob’ for the fine tuning of various system parameters and ‘magnifying glass’ to obtain deeper insights into the system performance

1.5.2 The Summary of New Research Problems

We conclude this chapter by providing brief description of the specific problem analyzed in the thesis. The rest of the thesis is an exposition of the main ideas developed in this chapter. These summary of three important research problems considered in our work is follows:

The Hardware Constraints Impact on the Fault Tolerance

By simulating four-qubit Bernstein-Vazirani Bernstein and Vazirani (1993a) circuit on a small trapped-ion architecture, the impact of limited hardware resources on the performance was analyzed. The foundation of resource-constraint based mapping and scheduling was laid and used for the study, while basic error analysis technique was employed to measure the overall level of the fault-tolerance. The results showed how limited resources adversely affected the performance and that the gradual performance-degradation trend suddenly became sharper as resource reduction reached maximum limit. However, there was a plateau spanned by resource space, where acceptable performance levels were maintained. In that region, we could

choose our performance level by trading one type of hardware resource with another. It was also shown that chosen baseline architecture allowed higher gain in performance than the corresponding increase in resource investment. This result encouraged us to investigate the performance of larger size benchmark circuits when executed on this scalable architecture. The chapter 5 provides details of this study.

Optimizing the Performance of Benchmark by Tuning Architecture and Device parameters

A careful observation reveals that constraint variables in previous problem can be divided into two main categories: (1) Device parameters describe the physics quality of hardware technology (2) Architecture parameters signify the amount of physical resources available supplied in the hardware. This division was found particularly useful while optimizing the scalable architecture for a meaningful size benchmark circuit. We chose to study the quantum adder circuit which is the building block of the modular exponentiation Van Meter and Horsman (2013); Vedral et al. (1996a); Beckman et al. (1996b) comprising bulk of Shor’s algorithm. A method was proposed to systematically improve the performance of 1,024-bit Quantum Carry Look-Ahead Adder (QCLA) Draper et al. (2006) by stepwise tuning the performance constricting parameters. First, architecture space was thoroughly analyzed and partitioned into the subspaces. Each subspace was labeled by the dominant performance limiting architecture parameter (number of ancilla or communication blocks). By varying this parameter, the subspace produced the ‘best’ design for which further performance improvement was possible only by adjusting device parameters. The qubit decoherence time emerged as the most critical device parameter common to all selected designs. A slight improvement in this parameter was sufficient to reliably execute 1,024-bit modular exponentiation circuit. The details of this analysis are provided in chapter 6. A natural extension to this study was to investigate the applicability

of given method of optimization to the variety of benchmarks quantum circuits, in particular those used in the Shor's algorithm.

Designing a Million-qubits Quantum Computer

The performance simulation tool was equipped to handle multiple benchmarks to be mapped and scheduled over highly reconfigurable architecture. The proposed architecture contained varying number of computation and storage units to match the resource requirement of different application. The scalability of such flexible system was validated when optimal designs found for increasing size of a benchmark, produced desired level of performance. In this study, the scalability of an architecture for various applications was thoroughly quantified using three different types of benchmarks circuits sufficient to implement Shor's algorithm. These include two vastly different types of adders which included the logarithmic depth QCLA and the linear depth Quantum Ripple Carry Adder (QRCA) Cuccaro et al. (2004). In addition we also analyzed Approximate Quantum Fourier Transform (AQFT) Fowler and Hollenberg (2004) which is applied as the last step in the Shor's algorithm. It was found that proposed architecture showed desired performance scalability for all three benchmarks given no limitation of resources. At the same time, QCLA which demonstrated the shortest execution time, was far more resource hungry than QRCA and AQFT. Being the building blocks of modular exponentiation, quantum adders play greater than AQFT in the context of resource-performance trade-offs analysis. The overwhelming speed advantage of QCLA over QRCA was tested in the presence of fewer resources. We observed that within the resource budget of 1.5 million-qubits, the architecture could support the execution of 1,024-bit QCLA, 2,048-bit QRCA and at least 4,096-bit of AQFT. The most remarkable result was the sudden sharp rise of logarithmic QCLA execution time curve which eventually went past QRCA linear curve as the problem size approached 1,024-qubit addition. From this study

we concluded that the optimal choice of quantum adder critically depended on both the problem size as well as the size of a quantum system factorizing large integers. See chapter 7 for comprehensive analysis and results of this study.

1.6 Summary

In this chapter we established research space for our thesis. We argued that the unreliability of quantum bits and noisy quantum gates ranked on top of the list of challenges involved in constructing scalable quantum computer. Next, we briefly described how quantum computation was protected from noise by investing in various types of system resources and designing appropriate architecture for resource utilization. Once the problem of finding reliable architecture translated into the exploration of large parametric design space, we proposed to develop an efficient performance estimation software tool to investigate the crucial trade-offs between resource investment and performance gains. We argued that in contrast to the prior studies, our tool analyzed the performance of application quantum circuits on vast combinatorial design space spanned by the architecture attributes and parameters of the quantum hardware. Our tool also detailed breakdown of performance metrics and supports visualization of resource utilization. Using this tool we studied three important problems which could provide valuable guidelines to experimentalists and system architects to build large scale quantum computer.

Basics of Quantum Computing

2.1 Qubits

Quantum computing employs quantum mechanical properties of the matter in order to perform mathematical calculations. The basic theme of quantum mechanical computing is derived from our standard computers which process information stored in the memory as bits. However, instead of bits, we work with quantum bits (qubits) in quantum computation. A qubit is realized by the two-state quantum mechanical system where each state correspond to either binary 0 or 1 analogous to the classical bit. For qubit, these states are represented by notation $|0\rangle$ and $|1\rangle$. The $|0\rangle$ and $|1\rangle$ states form one possible set computational ‘basis’ states. The symbol $|\rangle$ is called ‘ket’ notation widely used in quantum mechanics along with ‘bra’ $\langle|$. One of the remarkable characteristic of the qubit is that unlike bit which stores either 0 or 1, it can occupy both $|0\rangle$ and $|1\rangle$ simultaneously. This unique state is called ‘superposition state’ of the qubit. The superposition allows qubit to hold $|0\rangle$ and $|1\rangle$ states in different proportion which can be described by their respective amplitudes. An interesting question arises at this point: what will happen if we try to readout the

precise state of the qubit? Before answering this question, we define the reading out of the qubit state in quantum mechanical term as ‘measurement’. Now, when measurement is performed on a qubit, it collapses superposition into one of the basis states; which in our case will be either $|0\rangle$ or $|1\rangle$ with the probability distribution described by the amplitude of states. For example a qubit in the state $|\psi\rangle = a|0\rangle + b|1\rangle$ will give measurement result in the basis states with the probability a^2 of reading out $|0\rangle$ and b^2 for reading out $|1\rangle$. In order for probability interpretation of these amplitudes to make sense, we require $a^2 + b^2 = 1$ when a and b are assigned complex values. The reason for using complex numbers in the description of the qubit will be given shortly.

According to the theory of quantum mechanics, qubit states can interfere in a complex manner resulting in the range of patterns: from reinforcement to the cancellation of amplitudes. To model this quantum interference, the co-efficients a and b of the basis states are assigned complex. By representing these complex co-efficients in magnitude and phase notation, we find that common phase factor: γ called global phase, can be factored out from the description of qubit $|\psi\rangle = e^{i\gamma}(x|0\rangle + ye^{i\phi}|1\rangle)$. where $x = \cos(\frac{\theta}{2})$ and $y = \sin(\frac{\theta}{2})$. The global phase is ignored because it does not possess any physical significance. In contrast, ϕ has crucial meaning in the description; it denotes relative phase between the basis states which plays vital role in quantum interference. Therefore, the complete specification of a qubit state is composed of the spherical coordinates description of the unit vector ($\rho = 1, \theta, \phi$) as $|\psi\rangle = \cos(\frac{\theta}{2})|0\rangle + e^{i\phi} \sin(\frac{\theta}{2})|1\rangle$. During quantum computation, this vector rotates on the surface of sphere called Bloch Sphere of Figure 2.1.

The single qubit state $|\phi\rangle$ can also be presented in a linear-algebra friendly notation defined by vector in \mathbf{C}^2 as $|\phi\rangle^1 = \sum_{z=0}^{z=1} k_z |z\rangle$ such that $\sum_{z=0}^{z=1} |k_z|^2 = 1$. The ket $| \rangle$ can be deemed as a column vector while bra $\langle |$ as a row vector. For an n -qubit quantum system, this representation can be generalized to a vector spanning

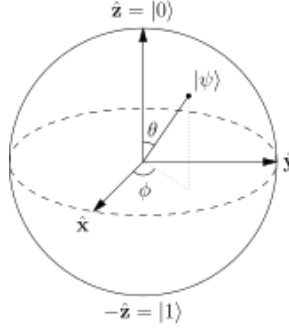


FIGURE 2.1: The Bloch Sphere representation of the qubit. [Diagram sketched by Glosser.ca - Own work. Licensed under CC BY-SA 3.0 via Wikimedia Commons]

complex Hilbert space \mathbf{C}^{2^n} in eq.2.1

$$|\phi\rangle^n = \sum_{z=0}^{2^n-1} k_z |z\rangle \quad (2.1)$$

The vector containing 2^n complex amplitudes, one for each basis state $|0_1 0_2 0_3 \dots 0_n\rangle$, $|0_1 0_2 0_3 \dots 1_n\rangle$ through $|1_1 1_2 1_3 \dots 1_n\rangle$. The linear-algebraic expression gives concise description of the states of quantum system. At the same time it aids in our understanding of important quantum mechanical properties of the system. As an example, a two-qubit state $|\phi\rangle^2 = \frac{1}{2}|0_1 0_2\rangle + \frac{1}{2}|0_1 1_2\rangle + \frac{1}{2}|1_1 0_2\rangle + \frac{1}{2}|1_1 1_2\rangle$ can be written as tensor product of two-single qubit states where each qubit is in equal superposition state. $|\phi\rangle^2 = \frac{1}{\sqrt{2}}(|0_1\rangle + |1_1\rangle) \otimes \frac{1}{\sqrt{2}}(|0_2\rangle + |1_2\rangle)$. When allowed to be written in a separable single qubit states, the quantum system is said to be disentangled. When quantum measurement is performed on this system, the outcome of reading out one qubit has no affect on the others. However, in general when the co-efficients are unequal the system may not be disentangled alluding to some degree of entanglement. The phenomena of quantum entanglement describes strange correlation between the two or more qubit states. For instance, in case of an entangled $|\phi\rangle_2 = \frac{1}{\sqrt{2}}|0_1 0_2\rangle + \frac{1}{\sqrt{2}}|1_1 1_2\rangle$, the result of measuring out one qubit will collapse other qubit state in known state.

In this case if measuring first qubit gives $|0\rangle$, the state of the second qubit will also be $|0\rangle$, likewise for $|1\rangle$. Thus we either get $|0_10_2\rangle$ or $|1_11_2\rangle$ when measured in basis states. The entangled state $\frac{1}{\sqrt{2}}(|0_10_2\rangle + |1_11_2\rangle)$ is also called Einstein-Rosen-Podolsky (EPR) pair Einstein et al. (1935) and it is used in several amazing applications of quantum computing. In general, the phenomenon of entanglement widely features in the algorithm algorithms and the construction of fault tolerant protocols as described through rest of the chapter. For the upcoming discussion we shall drop the subscript notation in the quantum state representation and assume the little-endian as the default ordering of the qubits.

2.2 Quantum Gates and Measurement

Quantum gates modify the amplitudes of eq.2.1 by evolving quantum state according to discretized simulation of the Schrodinger's equation. The solution to the equation requires the state to be evolved according to the unitary transformation. A quantum gate on n -qubit operand is described by $2^n \times 2^n$ unitary matrix which transforms the state vector of eq.2.1. During the execution of quantum algorithm, a set of chosen gates is applied in the known sequence which gradually transforms input state into the final state for the readout. The qubit states are outputted to the user by the procedure called quantum measurement.

2.2.1 Single-qubit Gates

A set of commonly used single-qubit quantum gates is shown in Figure 2.2. The Hadamard gate is often used in creating equal superposition by changing the transforming as $H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $H|1\rangle = |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. It can also be used to collapse the superposition since $H|+\rangle = |0\rangle$, $H|-\rangle = |1\rangle$. The X gate is analogous to the classical NOT. It flips the state of qubit from $|0\rangle$ to $|1\rangle$ and vice versa. The Z gate flips the relative phase ϕ between the basis states by


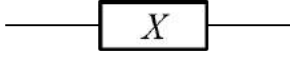
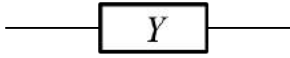
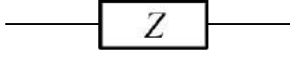


Gate Name	Gate Symbol	Matrix Description
Hadamard Gate		$\frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
NOT Gate (Pauli-X)		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Y Gate (Pauli-Y)		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Z Gate (Pauli-Z)		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Phase Gate (S)		$\begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$
$\pi/8$ or T Gate		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$

FIGURE 2.2: Commonly used single-qubit quantum gates, their symbols and matrix representation.

π radians ($k_0|0\rangle + k_1|1\rangle$ to $k_0|0\rangle - k_1|1\rangle$ and vice versa). The Y gate acts as both bit and phase flip operator. The Phase (S) gate also changes the phase between the basis states, albeit by an angle of $\frac{\pi}{2}$ while the T gate changes it by $\frac{\pi}{4}$ radians. The action of these single-qubit quantum gates on qubit state can be viewed as rotating the state vector on the surface of Bloch Sphere. Therefore X , Y and Z can be generalized to arbitrary single qubit rotation (angle = θ) gates: $R_X(\theta) = \begin{pmatrix} \cos \frac{\theta}{2} & -i \sin \frac{\theta}{2} \\ -i \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}$, $R_Y(\theta) = \begin{pmatrix} \cos \frac{\theta}{2} & -\sin \frac{\theta}{2} \\ \sin \frac{\theta}{2} & \cos \frac{\theta}{2} \end{pmatrix}$ and $R_Z(\theta) = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$ along x, y and z axis respectively. However, to allow quantum mechanical interaction among multiple qubits, the concept of single-qubit gates needs to be generalized to the need multi-qubit quantum gates.

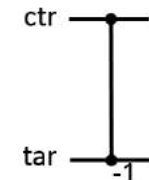
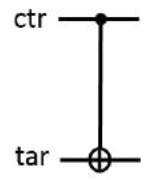
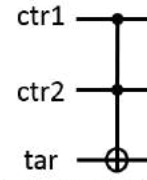
Gate Name	Controlled-Phase (CZ) Gate	Controlled-NOT (CNOT) Gate	Controlled-Controlled NOT (CCNOT or Toffoli) Gate
Gate Symbol			
Matrix Representation	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$

FIGURE 2.3: Commonly used multi-qubit quantum gates, their symbols and matrix representation.

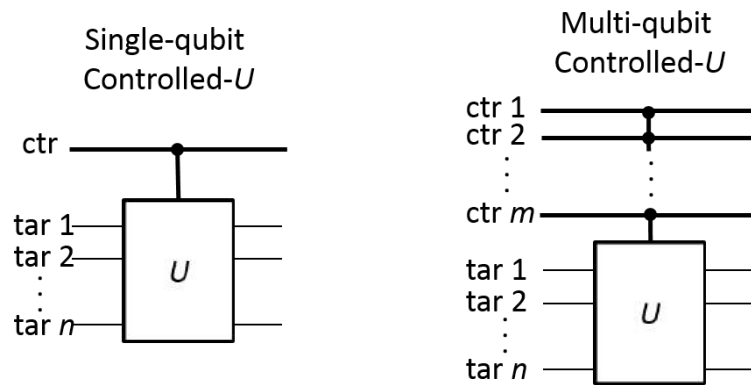


FIGURE 2.4: Generic symbols for controlled-gates

2.2.2 Multi-qubit Gates

Figure 2.3 shows a set of quantum gates which act on multiple qubit operands. The two-qubit gates has two operands: the control (ctr) and the target (tar) qubits. The Controlled-Phase (CZ). It flips phase of the tar when the ctr is $|1\rangle$. Like Controlled-NOT (CNOT) performs X gate on tar when ctr is $|1\rangle$. The Controlled-Controlled (CCNOT or Toffoli) is three-qubit gate performs X gate on tar when the two control qubits: ctr1 and ctr2 are $|1\rangle$. We can also generalize these gates by replacing the gate on *tar* by a general n -qubit quantum gate U . Thus, a symbolic representation of a generic single-qubit and multi-qubit controlled quantum gate U is illustrated in Figure 2.4. Their general matrix description will be $\begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix}$ The multi-qubit gates can entangle the states of the operand qubits and play vital role in generating complex unitary operation with the aid of single-qubit gates. It is also known that a CNOT gate and single qubit-arbitrary rotation gates can implement any multi-qubit gate Nielsen and Chuang (2000).

2.2.3 Quantum Measurement

In quantum mechanics reading out the qubit can alter its state. The quantum measurement is described by an operator which (1) produces the result of observing the qubit (2) changes the state of the qubit. The result is one of the eigenvalues while post-readout qubit state is the eigenvector of the operator. The eigenvector describes basis state in which qubit is measured. Generally, the operator is chosen to be a hermitian so that measurement basis states can be set to be orthonormal. To project $|\psi\rangle$ into the eigenstate of measurement operator M , we rely upon the projection operators P which constitute M as $M = \sum_m mP_m$. Here P_m is a hermitian projection operator for the eigenstate corresponding to eigenvalue m . Furthermore we require, $P_iP_j = \delta_{ij}P_i$ and $\sum_m P_mP_m^\dagger = I$. The last condition ensures that the probabilities of all possible measurement outcomes add to 1. The quantum measurement meeting all

these conditions is called ‘projective measurement’. It should be noted that projective measurement falls into general formalism of quantum measurement Nielsen and Chuang (2000) which relaxes some of the above mentioned conditions. However, for most quantum computing application, projective measurement suffice. Hence for the rest of the thesis we shall restrict to this formalism and use measurement implying projective measurement. The probability of collapsing an unknown quantum state $|\psi\rangle$ into the eigenstate m of operator P_m is given by $p(m) = \langle\psi|P_m|\psi\rangle$. The exact state of the qubit after the measurement is given by $\frac{P_m|\psi\rangle}{\sqrt{p(m)}}$. Typical projective measurement operators used in quantum circuits are given Figure 2.5. The M_Z operator measures qubit in the basis states $|0\rangle, |1\rangle$ while M_X in $|+\rangle, |-\rangle$. As an example, when $|\psi\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ is readout in M_Z , it will collapse to $|0\rangle$ state with probability $\frac{1}{2}$ yielding to +1 eigenstate of the M_Z operator. Similarly when $|\psi\rangle$ is read in M_X , it will produce -1 eigenstate of M_X with probability 1.

It is crucial to note that when measured in M_Z , the states $a|0\rangle + b|1\rangle$ and $a|0\rangle - b|1\rangle$ will output same probability distribution (+1 with prob. a^2 and -1 with prob. b^2). This shows that M_Z remains unaffected by phase flipped version of the qubit state. Similarly the states $a|0\rangle + b|1\rangle$ and $b|0\rangle + a|1\rangle$ will produce same probability distribution (+1 with prob. $(\frac{a+b}{2})^2$ and -1 with prob. $(\frac{a-b}{2})^2$) when measured in M_X . Hence M_X outcome remains unchanged by the bit flipped version of the qubit state. On contrary, the bit-flipped qubit state will flip the M_Z measurement outcome from +1 to -1 and vice versa. Similarly the phase-flipped qubit state will flip the M_X measurement outcome from +1 to -1 and vice versa.

The matrix representation of the M_X and M_Z is same as that of X and Z respectively and these correspond to performing measurement in two different basis states. We can generalize this concept to the measurement M_U performed in the eigenstate of the hermitian operator U having eigenvalues +1 and -1. The operator



Symbol	Measurement Operator	Eigenstates	Projection operators and Eigenvalues
	$M_z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$	$ e_{+1}\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ or $ 0\rangle$ $ e_{-1}\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ or $ 1\rangle$	$P_{+1} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ $m = +1$ $P_{-1} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$ $m = -1$
	$M_x = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$	$ e_{+1}\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ or $ +\rangle$ $ e_{-1}\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ -1 \end{bmatrix}$ or $ -\rangle$	$P_{+1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$ $m = +1$ $P_{-1} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$ $m = -1$

FIGURE 2.5: Commonly used measurement operators, their symbols, matrix representation M , eigenstates $|e\rangle$, eigenvalues m and projection operators P

measurement circuit is shown in Figure 2.6. Note that in order to collapse the qubit state ψ into eigenstates of U , the circuit needs this qubit to interact with another qubit through controlled- U gate. The life span of the additional qubit is short; it is initialized in a known state ($|0\rangle$) and measured after a short sequence of gates. Such a qubit is called ‘ancilla’ qubit. Once measured, such ancilla can be reused to execute similar operator measurement circuits.

2.2.4 Classically Controlled Quantum Gates

The outcome of the qubit measurement is an eigenvalue which is a scalar quantity and can be stored in classical computer memory. In many quantum circuits, based on the measurement outcome (+1 or -1) of one qubit, we can choose to apply (or not apply) a quantum gate on another qubit (or set of qubits). Such quantum gate is called classically controlled quantum gate. These gates have already been described earlier. To give different symbol to these gates, we append two parallel lines to the

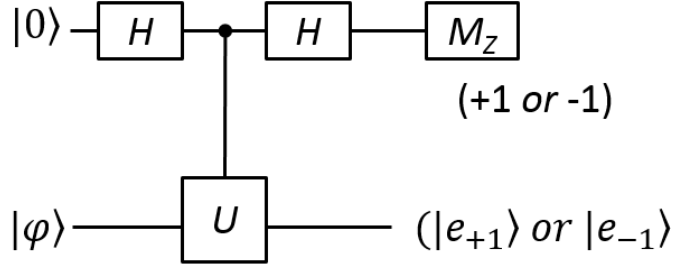


FIGURE 2.6: The quantum circuit for performing measurement in the eigenstate basis $|e_{+1}\rangle, |e_{-1}\rangle$ of operator U

corresponding quantum gates as shown in Figure 2.7. When classically controlled gate execution is dependent on the measurement outcome, one end of the parallel lines is connected to the measurement symbol and the other end to the quantum gate symbol.

2.2.5 Sample Quantum Circuit: Quantum Teleportation

Let us demonstrate a simple application of the quantum gates to construct a meaningful circuit. In this example, we implement quantum teleportation which transfers quantum state $|\psi\rangle = \alpha|0\rangle + \alpha|1\rangle$ from one qubit (call it first qubit) to another (third qubit) using an EPR pair state contained in qubit 2 and 3. The initial state of the qubit can be written as a tensor product of an $|\psi\rangle$ and EPR pair state as: $\frac{1}{\sqrt{2}}(\alpha|0\rangle(|00\rangle + |11\rangle) + \beta|1\rangle(|00\rangle + |11\rangle))$. After the CNOT gate, we obtain: $\frac{1}{\sqrt{2}}(\alpha|0\rangle(|00\rangle + |11\rangle) + \beta|1\rangle(|10\rangle + |01\rangle))$. When Hadamard gate is applied to the first qubit, the resulting state becomes: $\frac{1}{\sqrt{2}}[(\alpha(|0\rangle + |1\rangle)(|00\rangle + |11\rangle) + \beta(|0\rangle - |1\rangle)(|10\rangle + |01\rangle))]$. At this point when first and second qubits are measured there are four possibilities: For both outcomes $(+1,+1)$ we obtain desired state $\alpha|0\rangle + \beta|1\rangle$

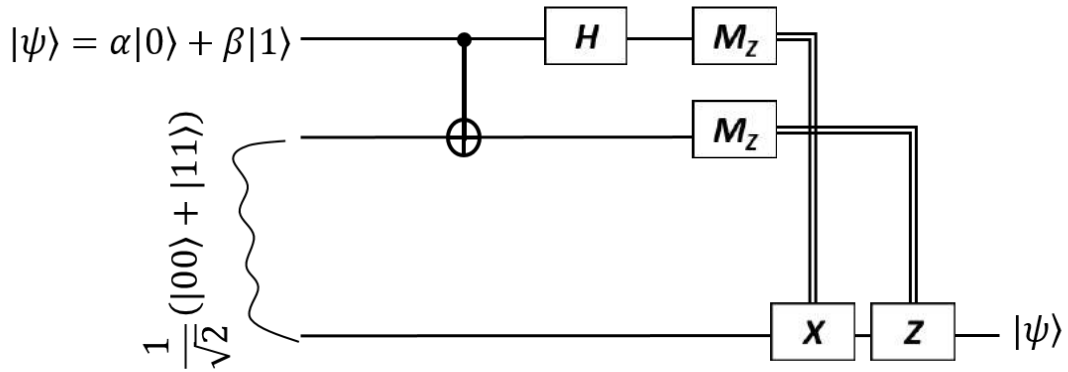


FIGURE 2.7: The quantum circuit for transferring state $|\psi\rangle$ from one qubit to another using teleportation protocol

which is $|\psi\rangle$ and no subsequent gates are applied. For $(+1,-1)$, we obtain $\alpha|1\rangle + \beta|0\rangle$ which can be converted to $|\psi\rangle$ by applying X gate. For $(-1,+1)$, $\alpha|0\rangle - \beta|1\rangle$ which can be converted to $|\psi\rangle$ by applying the Z gate. Finally in case of $(-1,-1)$ $\alpha|1\rangle - \beta|0\rangle$ which can be transformed back into $|\psi\rangle$ by applying both X and Z gate. Thus we have perfectly transferred $|\psi\rangle$ from one qubit to another.

2.3 Universal Quantum Computation

Similar to classical digital circuit which can be constructed entirely from NAND or NOR gates, it is known that an arbitrary quantum circuit can be constructed using a finite set of gates called universal quantum gates, a set which is not unique. There are several possible sets of universal quantum gates. For example CNOT and arbitrary single qubit rotation comprise one universal set Nielsen and Chuang (2000). However, these arbitrary rotations are not conducive to the precise physical implementation and fault-tolerant computation. Alternatively, we can use Universal set $\{\text{CNOT}, H, S, T\}$ or $\{\text{CNOT}, H, S, \text{Toffoli}\}$ to approximate arbitrary unitary

operation with desired level of accuracy Nielsen and Chuang (2000). The general composition of a universal set is described as a union of a set of gates which can generate all ‘Clifford group’ gates Gottesman (1998b) and one gate which belongs to ‘non-Clifford group’. The Clifford group gates include X, Y, Z, S, H and CNOT gates while non-Clifford include Toffoli and T gates.

To shield against noise, quantum information is encoded and processed using specific quantum error correcting code. To ensure continuous protection against the accumulation of errors, the gates in the universal set should be performed on the encoded qubits. For many quantum error correcting codes, the execution of the non-Clifford gates is substantially complicated and far resource consuming than the Clifford gates. Due to this reason we prefer to minimize the number of non-Clifford gates in the universal set. Therefore we construct this set by adding either Toffoli or T gate to the set of Clifford gates. The decision to include either Toffoli or T gate can be based on the nature of benchmark application circuit. In chapter 7, we show that for quantum arithmetic circuit such as quantum adders, it is advantageous to use Toffoli gates, while T gates are preferred for quantum Fourier Transform.

2.4 Quantum Noise

So far we have described qubit by its state which is coherent in a sense that qubit is kept perfectly isolated from interacting with the undesired quantum system. However, as mentioned in chapter 1, the quantum components are very fragile as they tend to lose information due to unwanted coupling with the surroundings (called environment) or faulty operations. This loss is quantified by the uncertainty induced in the coherent state of the qubit using a concept called ‘decoherence’. The incoherent qubit state is modeled as a statistical mixture of coherent states. The ‘coherent state’ is called the pure state and statistical mixture of pure states is called ‘mixed state’. The mixed state is represented as positive semi-definitive, unit trace hermi-

tian matrix called ‘density matrix’ ρ . For example a qubit in the mixture of pure states: $|\psi\rangle_1, |\psi\rangle_2, \dots, |\psi\rangle_n$ w.p. p_1, p_2, \dots, p_n respectively, is expressed by a density matrix: $\rho = \sum_{i=1}^{i=n} |\psi\rangle_i \langle \psi|_i$. Note that density matrix a richer description of a qubit in a sense that pure state can be represented a density matrix made of single pure state. A unitary operator U acting on ρ transforms to $U\rho U^\dagger$ while measurement operator M_m changes ρ as $\frac{M_m \rho M_m^\dagger}{M_m^\dagger M_m \rho}$. The detailed description of density matrix is given in chapter 2 Nielsen and Chuang (2000).

Quantum noise process is formulated using operator-sum representation in which qubit is entangled with the specific type of environment which is then measured to render qubit in a mixed state. The collective effect of the noise process can be represented by changing the density matrix ρ to $E(\rho) = \sum_k A_k \rho A_k^\dagger$. The operational elements A_k are conditioned to preserve the trace of the ρ . The choice of A_k operators define the nature of quantum noise which arise in different physical systems. In this thesis, we focus on ‘depolarizing channel’ which affects qubit information loss equally among all three axes of the Bloch Sphere. The depolarizing noise changes the state of the qubit as $E(\rho) = (1 - p)\rho + \frac{p}{3}(X\rho X + Y\rho Y) + Z\rho Z$ which means that a qubit undergoes bit-flip (X), phase-flip (Z) and both bit and phase flip (Y) gates with equal probability $p/3$ and remains unchanged with probability p . In the context of quantum noise, we call these flips as ‘errors’ since they can cause corruption in the qubit state resulting in the incorrect computation. The ‘similarity’ between noise affected state $\sigma = E(\rho)$ and the original state ρ is quantified by the ‘fidelity’ $F(\sigma, \rho)$ defined as $\text{tr}\sqrt{\rho^{1/2}\sigma\rho^{1/2}}$ where tr designates the trace of the matrix. Naturally, fidelity is a function of p , the probability of error (or failure probability) so that they can both be used to quantify the amount quantum noise. From this definition we can also relate failure probability to the ‘infidelity’ which can be defined as 1-fidelity. For more details on quantum noise and other noise models, reader can refer to chapter

8 of Nielsen and Chuang (2000). In the next section we discuss how quantum error correction provides protection against these errors and paves way for the reliable computation.

2.5 Fault-tolerant Quantum Circuits

In this section we discuss a general mechanisms for guarding components of the quantum circuits against noise. We first describe the basic criterion for choosing quantum error-correcting code and justify the usage of Steane code Steane (1996) in our study. To understand Steane code we first describe classical error correcting code called Hamming [7,4,3] code. Using Hamming code, we shall describe general construction of quantum CSS code Calderbank and Shor (1996) and converge our discussion to specific CSS code called Steane $[[7,1,3]]$ (Steane) code. Next, the error propagation mechanism will be illustrated along with the description of Stabilizer formalism. Finally, we shall detail the fault tolerant construction of the universal set of quantum gates used in Shor's algorithm.

2.5.1 *The Choice of Quantum Error Correcting Code*

Several factors affect the choice of quantum error correcting code. We identify following attributes provide decent guidelines to simplify the selection procedure.

- Well known fault tolerant encoding and error correction procedures
- Bitwise (Transversal) implementation of Clifford gates
- Distillation-free realization of fault tolerant non-Clifford gates
- Scheduling on hardwares of different types of constraints
- Support for the variety of noise models

We find that Steane $[[7,1,3]]$ code is one of the few error correcting codes which features all these attributes. First, the procedure for encoding and performing error correction on the encoded qubit block are well defined and can be easily verified (chapter 10 Nielsen and Chuang (2000)). Second, the execution of Clifford gates can be done in a bitwise or transversal fashion which ensures that error in one qubit doesn't spread to multiple qubits in the encoded qubit block. Third, in case of the non-Clifford gate such as Toffoli and T gate, the execution protocol is slightly complex but can be broken down into two steps (1) magic-state preparation (2) data injection into the magic state. Both these steps only need transversal application of gates which prevents the spread of error, thus enable 'fault tolerant' implementation as explained in section 2.5.5.

It is worth pointing out that for majority of error-correcting codes, magic-state preparation step cannot be realized fault tolerantly. In that case we resort to the general 'distillation' procedure in which several instances of non-fault tolerantly prepared magic-states are processed to extract single high fidelity state Bravyi and Kitaev (2005). Since distillation procedures are generally enormously time and resource consuming Meier et al. (2012); Knill (2004); Bravyi and Kitaev (2005), therefore distillation-free magic-state preparation is highly attractive feature of any quantum error correcting code. The non-Clifford gates construct basic building blocks of the Shor's algorithm and their fault tolerant implementation is key to practical computational speed of the quantum computer. The efficient fault tolerant implementation of the non-Clifford gates is one of the main reason of choosing the Steane code.

Fourth, the fault tolerant procedures of Steane code can be efficiently scheduled on hardwares with different qubit connectivity constraints specified by the quantum device technology. In contrast to the widely famous topological code Kitaev (2003) which are tailored to the hardware which supports 2-D nearest-neighbor interaction, Steane code can be mapped onto range of hardware architectures such as linear

chain of qubits (chapter 5), 2-D nearest neighbor hardware Svore et al. (2006b) and the one which supports flexible physical transportation mechanism for the qubits (e.g., trapped ion computers) for the implementation non-distance fault tolerant quantum computation Monroe and Kim (2013); Monroe et al. (2014). Historically, Steane code been considered as premier choice for protecting trapped ion qubits which are considered as one of the strongest candidate for the realistically achievable large scale quantum system Monroe and Kim (2013). Even though the topological codes generally achieve higher noise threshold to provides greater noise protection against noise as compared to the Steane code, when it comes to the overall resource overhead, reliability and the time to execute large size quantum algorithms on the trapped ion computer, the Steane code edges ahead of the topological code Suchara et al. (2013b). Since in this thesis we choose to model trapped ion technology for underlying quantum hardware, Steane code is the natural choice for error correcting code.

Lastly, Steane code is one of the few codes which have been studied in the context of noise models other than depolarization channel (e.g., Amplitude damping channel Gutiérrez and Brown (2014)). It is also known that Steane code can be used to correct errors which are correlated in time Lu and Marinescu (2007) as well as in space (see Appendix A). In a quantum hardware where qubits suffer from errors specified by multiple types of noise processes, Steane code acts as a powerful tool of protection against noise. Although, in this thesis we restrict our analysis based on depolarizing noise, the performance simulation tool (described in chapter 4) based on Steane code can be easily extended to other noise models for the future investigation.

2.5.2 Hamming Code

The Hamming code $[7,4,3]$, encodes four data bits into a seven bit codeword whose ‘code distance’ is three. There are sixteen codewords in the Hamming $[7,4,3]$ code,

each assigned to one of the sixteen possible 4-bit binary strings. These codewords provide protection against noise (e.g. during transmission over some physical channel) which can be determined by the code distance. The code distance is the minimum hamming distance between any two codewords. It means when if a codeword is corrupted by single bit error (which corresponds to change the codeword by hamming distance of 1), it is still closest to the original codeword. By carefully choosing a set of codewords and designing the method to determine the unique codeword closest in the hamming distance to the corrupted codeword, we can detect and correct single bit-flip error. The codewords are obtained by appending three parity bits to the original 4-bit string using the Generator matrix G in 2.2.

$$G := \begin{pmatrix} 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.2)$$

The rows of G , compute the codeword bits. The rows 3,5,6 and 7 will simply copy the data bits (call them components v_1, v_2, v_3, v_4 of bit strings v) to the bits 3,5,6 and 7 of the codeword. Whereas, the bits 1,2 and 4 of the codeword store three parity bits $v_1 \oplus v_2 \oplus v_4$, $v_1 \oplus v_3 \oplus v_4$ and $v_2 \oplus v_3 \oplus v_4$ respectively. Therefore, in order to encode v , the codeword c is obtained by multiplying $c = Gv$ with matrix arithmetic performed in the Galois binary field $GF(2)$. For example the bit string $v = (1, 0, 1, 0)$, will be encoded into the following codeword $c = (1, 0, 1, 1, 0, 1, 0)'$. The codeword can undergo bit flips when it is transmitted over the noisy channel. The $[7,4,3]$ code can correct up to one bit flip error whose location can be determined by performing parity check operations on the corrupted codeword. A procedure called 'syndrome extraction' is employed to determine if the parity relationships among the codeword

bits has been preserved during the transmission. In case of Hamming code the syndrome is obtained by computing three parity bits p_1, p_2 and p_3 in 2.3. The c_i are components of bit vector of the codeword c . These parity checks can be written in the matrix form H given in 2.4

$$p_1 = c_4 \oplus c_5 \oplus c_6 \oplus c_7, p_2 = c_2 \oplus c_3 \oplus c_6 \oplus c_7, p_3 = c_1 \oplus c_3 \oplus c_5 \oplus c_7, \quad (2.3)$$

$$H := \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} \quad (2.4)$$

The location d of the bit flipped in the corrupted codeword c' is determined by $d = Hc'$. For example, if the fourth bit of the codeword c is flipped, then $c' = (1, 0, 1, 0, 0, 1, 0)$ would give syndrome $d = (1, 0, 0)^T$ which is the binary representation of 4. It is worth noting that in an event of multiple errors in the codeword, the error correction procedure will result in the false codewords. Moreover, when there is no bit flip error in c' (or highly unlikely event of c' corrupted to become another codeword due to burst of errors), it will give all zero (no error). For all uncorrupted 16 codewords listed in Table 2.1, the parity check will give 0. Since G describes the basis set for the codewords, it is easy to see that $HG^T = 0$. Hamming code falls into the category of classical linear codes in which the sum of two codewords in $\text{GF}(2)$ is another codeword in the set.

2.5.3 Quantum CSS Codes

Quantum error correcting codes encodes several physical qubits into one ‘logical qubit’ in order to correct bit (X) phase (Z) and bit-and-phase flip (Y) errors. Since Y can be decomposed into X and Z error, it suffices to construct a code with

Table 2.1: Hamming Codewords

Set 1	Set 2
0000000	1111111
1010101	0101010
0110011	1001100
1100110	0011001
0001111	1110000
1011010	0100101
0111100	1000011
1101001	0010110

corrects X and Z error. The protection against these basic errors is sufficient to protect quantum information from any type of noise Nielsen and Chuang (2000). One simple yet powerful recipe to design quantum relies on classical linear code which can correct for the bit flip error(s). Once X errors are corrected, the phase errors can be converted into bit flips using bitwise Hadamard on the codeword qubits: $HXH = Z$. If this transformation keeps us within the desired codeword space, we can also correct for Z errors and we have a satisfactory quantum error correcting code. Such quantum code is called Calderbank Shor Steane (CSS) code Calderbank and Shor (1996). More formally, we need two classical codes C_1 which is $[n, k_1]$ and C_2 which $[n, k_2]$ code, such that $C_2 \subset C_1$ and both C_1 and C_2^\perp can correct up to t errors. Using such C_1 and C_2 we can construct $[[n, k_1 - k_2, t]]$ quantum code which can correct up to t errors.

We construct a specific CSS code called Steane $[[7, 1, 3]]$ using classical seven-bit Hamming code. This code encodes seven ‘physical qubits’ into one Steane ‘logical qubit’. We choose Hamming $[7, 4, 3]$ for C_1 which has all 16 codewords while $C_2 = C_1^\perp$ is a $[7, 3]$ code containing Set 1 of Table 2.1. Note that $C_2 \subset C_1$ and similar to C_1 , C_2^\perp can correct for single error since $C_2^\perp = (C_1^\perp)^\perp = C_1$. The remarkable feature of the CSS codes lies in bi-directional transformation between C_1 and C_2^\perp using bitwise Hadamard on the codewords if they are contained as superposition states of the

logical qubit. Based on this observation, we can derive logical computational basis states $|0\rangle_L$ and $|1\rangle_L$ for the Steane logical qubit. Each state is carefully assigned set of codewords as follows. Since $H|0\rangle = |+\rangle$ and $H|+\rangle = |0\rangle$, the codewords comprising Steane $|+\rangle_L$ should be orthogonal to codewords contained in Steane $|0\rangle$. The only way this can be achieved is by assigning all codewords to $|+\rangle_L$ (code space: C_1) while $|0\rangle_L$ the codewords in C_2 . This means that $|1\rangle_L$ will be the codewords contained in the set $C_1 - C_2$.

$$\begin{aligned}
|0\rangle_L &= \frac{1}{\sqrt{8}}[|000000\rangle + |1010101\rangle + |0110011\rangle + |1100110\rangle + |0001111\rangle + |1011010\rangle + \\
&\quad |0111100\rangle + |1101001\rangle] \\
|1\rangle_L &= \frac{1}{\sqrt{8}}[|1111111\rangle + |0101010\rangle + |1001100\rangle + |0011001\rangle + |1110000\rangle + |0100101\rangle + \\
&\quad |1000011\rangle + |0010110\rangle]
\end{aligned}$$

2.5.4 Stabilizer description of Steane code

The parity check matrix H in 2.4 determines whether a bit string c lies inside or outside the code space. $\forall c, Hc' = 0$, it perseveres or ‘stabilizes’ the code space. In classical syndrome calculation, the parity checks computed by Hc' employs summation over GF(2). By replacing summation with multiplication we can formalize the syndrome vector calculation for quantum CSS code. The multiplication is carried over the group $\{+1, -1\}$ which correspond to the outcome of the operator measurement procedure (section 2.2.3). These operators are design to calculate syndrome locating bit (X) or phase flip (Z) errors in the encoded qubit and can be derived from the parity check matrix 2.4. Using identities of Table 2.4, X and Z errors can be caught using operator measurement procedure shown in Figure 2.8. By measuring X operator (substitute X for U in Figure 2.6), we can propagate Z error in $|\psi\rangle$ to flip the measurement outcome from +1 to -1 or vice versa. Similarly by measuring Z operator (substitute Z for U in 2.6), we can propagate X error in $|\psi\rangle$ to flip the measurement outcome from +1 to -1 or vice versa.

Table 2.2: Steane Code Stabilizers

S_X	S_Z
$Z_4Z_5Z_6Z_7$	$X_4X_5X_6X_7$
$Z_2Z_3Z_6Z_7$	$X_2X_3X_6X_7$
$Z_1Z_3Z_5Z_7$	$X_1X_3X_5X_7$

Table 2.3: General error detection procedure for Steane code. The qubit infected with X and/or error is identified by measuring the syndromes

S^1	S^2	S^3	Error Qubit
+1	+1	+1	None
+1	+1	-1	1
+1	-1	+1	2
+1	-1	-1	3
-1	+1	+1	4
-1	+1	-1	5
-1	-1	+1	6
-1	-1	-1	7

For Steane code, we can translate parity checks of H into corresponding syndrome operators S_X and S_Z detecting X and Z error respectively. For three parity checks in 2.3 we measure bit-flip syndromes operators $S_X^1 = Z_4Z_5Z_6Z_7$, $S_X^2 = Z_2Z_3Z_6Z_7$, $S_X^3 = Z_1Z_3Z_5Z_7$. For the phase-flip syndromes operators can be obtained by replacing X with Z by virtue of CSS code construction. Hence $S_Z^1 = X_4X_5X_6X_7$, $S_Z^2 = X_2X_3X_6X_7$, $S_Z^3 = X_1X_3X_5X_7$. The set S of all six syndrome operator defines the stabilizers for the Steane code since it can validate correct (error free) state of the Steane logical qubit. These are listed in Table 2.2. When all six stabilizer measurements output +1, no single qubit error is inferred (as in $Hc' = 0$ for Hamming code). Otherwise, the error is located by adopting similar procedure for the Hamming code, described in Table 2.3. Once bit- and phase-flip qubits are identified, the recovery procedure is applied to undo these errors: by applying X gate on bit-flipped qubit and Z gate on the phase flipped qubit in the logical block.

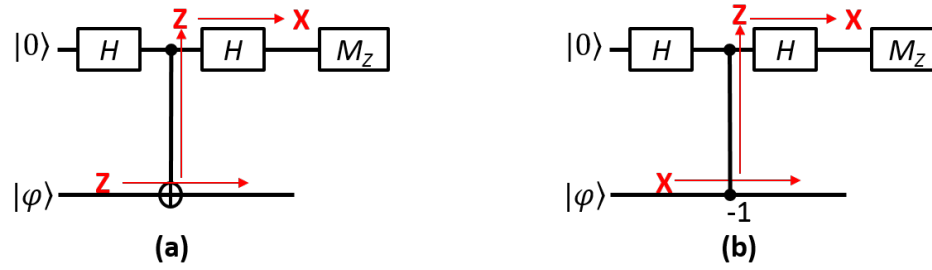


FIGURE 2.8: X and Z error detecting procedure using operator measurement. (a) shows that X error is propagated to change measurement outcome by measuring Z while (b) shows that Z error is propagated to change the measurement outcome by measuring X

Table 2.4: Propagation of X and Z through different gates

Gate	Input	Output
<i>Controlled-NOT</i> (CX_{12})	X_1	X_1X_2
	X_2	X_2
	Z_1	Z_1
	Z_2	Z_1Z_2
<i>Controlled-Phase</i> (CZ_{12})	X_1	X_1Z_2
	X_2	Z_1X_2
	Z_1	Z_1
	Z_2	Z_2
H	X	Z
	Z	X
S	X	Y
	Z	Z
X	X	X
	Z	$-Z$
Y	X	$-X$
	Z	$-Z$
Z	X	$-X$
	Z	Z

2.5.5 Fault-tolerant gates in Steane code

We have seen Steane logical basis states $|0\rangle_L$ and $|1\rangle_L$ which encode computational basis states one qubit. The next step involves performing computation directly on the logical qubit without decoding. Otherwise we cannot reap the benefits of the quantum error correction which ensures protection against noise. We define ‘logical gate’ which performs which is an encoded version of the physical or uncoded quantum gate (see Figure 2.2 and 2.3) discussed earlier in the chapter. A logical gate is designed to transform the state of the logical qubit in the same way as physical quantum gate transforms the state of the physical qubit. It is constructed from physical gates which act on the constituent qubits of the logical qubit block. Since imperfect physical gates are tend to be noisy, the application of logical which can cause errors in the logical qubit. For Steane protected logical qubit, we can detect and correct for single error. Therefore, as long as physical gate execution is corrupted to cause up to one error in each operand logical qubit, the logical gate is fault-tolerant. This is because, we can error correct each operand to restore its pristine logical state. The condition that at most one physical qubit is corrupted by noise can be met easily if physical gates are applied bitwise on the logical block. A logical gates constructed in this manner is naturally fault-tolerant and called is ‘transversal gate’. For Steane code, it can be verified that logical gates X, Y, Z, H, S and $C_{A,B}$ ($CNOT$ between logical qubit A and B) are all transversal. The bitwise implementation of these gates is listed in Table 2.5. For the remainder of discussion we use subscript L to differentiate logical gate or qubit state from that of the physical.

To show that Steane code can support universal set of gates, the fault-tolerant construction of T (or Toffoli) gate is required. In addition, we need to provide the fault-tolerant circuits in order to obtain Steane code syndromes and perform logical M_X and M_Z . For all these logical operations, a basic circuit construct of a logical

Table 2.5: The list of transversal logical gates in Steane code. The subscript in the right column indicate physical qubit position in the logical block.

Logical gate	Physical implementation
X	$X_1X_2X_3$
Z	$Z_1Z_2Z_3$
Y	$Y_1Y_2Y_3$
H	$H_1H_2H_3H_4H_5H_6H_7$
S	$Z_1S_1Z_2S_2Z_3S_3Z_4S_4Z_5S_5Z_6S_6Z_7S_7$
$C_{A,B}$	$C_{A_1,B_1}C_{A_2,B_2}C_{A_3,B_3}C_{A_4,B_4}C_{A_5,B_5}C_{A_6,B_6}C_{A_7,B_7}$

operator U measurement is required. Figure 2.9(a) expands the circuit of Figure 2.6 by adding single qubit controlled multiple U gates on the Steane logical block. However, this circuit is non fault-tolerant since a an error (e.g. X) in the ancilla can propagate to multiple physical qubits in the block (e.g. when $U = X$). Since Steane code incorrectly recovers the logical state of the qubit when the block has more than one bit-flips (or phase flips) errors, this error event will result in false logical state ensuing logical error.

A generic fault-tolerant operator measurement circuit is given in Figure 2.9(b). Unlike its non fault-tolerant counterpart, this circuits employs several ancilla qubits, out of which one acts as verifier which prevents the spread of uncorrectable errors into the logical block as follows. Assuming $U = X$, a single X -error anywhere in the first three ancilla qubits will either remains single- X error or spreads into two or more X -errors and inevitably propagates to the fourth ancilla qubit. This qubit acts a verifier whose Measurement result indicates multiple X -errors in the ancilla preparation part of the circuit. Thus when M_Z gives -1, it we know that other three ancilla qubits carry multiple errors which can propagate into data when controlled- U gates are applied. In this case we start over and the ancilla preparation part of the circuit is repeated until M_Z gives +1. Once desired measurement outcome is obtained, controlled- U are applied with ancilla contain single X -error or two (or more) X -errors which cancel each other on the fourth ancilla qubit before the measurement.

Single error propagating from ancilla into the logical block is correctable. However, the possibility of uncorrectable two-error event needs to be minimized. If p is the probability of one X -error in the circuit, then the probability of two X -errors is $O(p^2)$. The defining characteristic of the fault-tolerant circuit is that it ensures that number of ways K_2 in which two-error event can occur with sufficiently small probability so that $K_2 p^2 \ll p$ when p is low.

Next, we note that although single Z -error from ancilla does not propagate into the logical block, however it can easily propagate to second ancilla qubit through CNOT gates. The H gate will turn the Z -error into X -error which flips the final measurement outcome, thus the incorrect output eigenvalue of logical operator. The same logical error can occur if the final measurement fails or ancilla qubit undergoes bit-flip just before the measurement. When each one of these events occur with probability p we obtain logical error with the same probability unless special measures are taken to make it $O(p^2)$. The solution turns out to be very simple: repeat the entire circuit of 2.9(b) three times and take the majority vote of the second ancilla qubit measurement. This means that the outcome needs to be incorrect at least two out of three trials in order to cause logical error. Thus logical failure probability is $O(p^2)$. The circuit is fault-tolerant because (1) it does not allow single error anywhere in the circuit to cause logical error and (2) the probability of multiple errors propagating to cause logical error is $O(p^2)$.

The fault-tolerant circuit in Figure 2.9 contains two main components: ancilla state preparation and ancilla state decoding. The special ancilla state used here is called ‘cat state’. For the remainder of discussion we need three- four- and seven-cat state preparation and decoding. These are shown in Figures 2.10 2.11 2.12 respectively. These simple fault-tolerant constructs along with transversal controlled- U gates can be used to develop complicated fault-tolerant blocks to allow for universal quantum computation in Steane code.

The error correction circuit involves six stabilizer measurements, each involving preparation of 4-cat state preparation, transversal CNOT/CZ followed by 4-cat decoding. Each stabilizer measurement is repeated three times, the outcome is computed from the majority vote of the three measurement trials. Once error location is identified, the recovery procedure is used to undo the error by applying X or Z gate on the affected qubit.

In addition to correcting for errors, the Steane error correction circuit can be used to initialize the logical block in the desired basis state. The fault-tolerant circuit to prepare $|0\rangle_L$ is shown in Figure 2.14. A block of seven physical qubits in any arbitrary state is projected on to the Steane state: $|s\rangle = \alpha|0\rangle_L + \beta|1\rangle_L$ by error correction circuit. The state $|s\rangle$ produces $|0\rangle_L$ or $|1\rangle_L$ when logical M_Z is performed. If needed, the resulting state can be flipped from $|1\rangle_L$ to $|0\rangle_L$ by logical X .

The execution of the Steane T_L on $|\phi\rangle_L$ is a complicated two-step process. First, the magic state: $T|+\rangle$ is prepared by measuring the logical operator $e^{\frac{i\pi}{4}}SX = TXT^\dagger$ on $|0\rangle_L$ state logical ancilla, shown in the Figure 2.15. The controlled- TXT can be broken down into transversal T , $CNOT$ and T^\dagger . The $+1$ eigenstate of this operator is logical $T|+\rangle := \frac{1}{\sqrt{2}}(|0\rangle + e^{\frac{i\pi}{4}}|1\rangle)$. The operator is measured three times and the preparation concludes if majority vote is $+1$. Otherwise, Z_L is applied to the state in order to obtain logical $T|+\rangle$. Once magic state is prepared in the logical ancilla, the operand $|\phi\rangle_L$ is teleported into the ancilla using logical CNOT, M_Z and the SX gates contingent upon the M_Z outcome (-1).

Finally, the Steane Toffoli gate also requires magic state preparation in which operand qubits are teleported. In this case The magic state will be logical $\frac{1}{2}(|000\rangle + |010\rangle) + |100\rangle + |111\rangle$ and it is prepared in three logical ancilla state $|\phi_+\rangle_L$ shown in Figure 2.17(a). The logical operand qubits $|X\rangle_L$, $|Y\rangle_L$ and $|Z\rangle_L$ are teleported into the magic state using a sequence of Clifford group gates, some of which are controlled

by Measurement output as shown in Figure 2.17(b).

We have provided brief discussion of the Steane code logical operation circuits. The detailed description can be found in provided in chapter 10 Nielsen and Chuang (2000). The circuits described in this section are all fault-tolerant since their component are made of fault-tolerant circuit constructs and they prevent single error occurring at any location in the circuit to induce multiple errors in any logical qubit block. Although the number of ways in which these errors can occur linearly grows with the size of the circuit, the fault-tolerance construction ensures that none of these event produces uncorrectable error pattern. However, the impact of two-error events K which generally lead to the logical error still needs to be quantified to estimate logical failure probability $p_L = Kp^2$. Although, K scales (quadratically) in the size of the circuit, the actual value may be higher when these circuits are mapped and scheduled onto the hardware. In chapter 5, we shall estimate p_L as well as the execution time of Steane fault tolerant logical operations graphed on the realistic quantum computer hardware.

2.5.6 Concatenated Quantum Error Correction

We have seen that failure probability of the logical quantum operation reduces from p to $p_L = Cp^2$ with one layer of encoding where C represents the number of faulty locations in the equivalent physical circuit. With two layers of encoding, the failure probability can be further suppressed to $p_L = C(Cp^2)^2$. Generalizing to n layers of concatenation, we have $p_L = C^{2n+1}p^{2^n}$. The double exponential decrease in the p_L with the addition of encoding layer is a great news for the scalable quantum computation if the cost of additional layer is adequately lower than the decrease in p_L . Fortunately, the double exponential reduction in failure probability can be met with only exponential increase in qubits and operations with each added encoding layer. To show this, we present the general construction of fault-tolerant operation

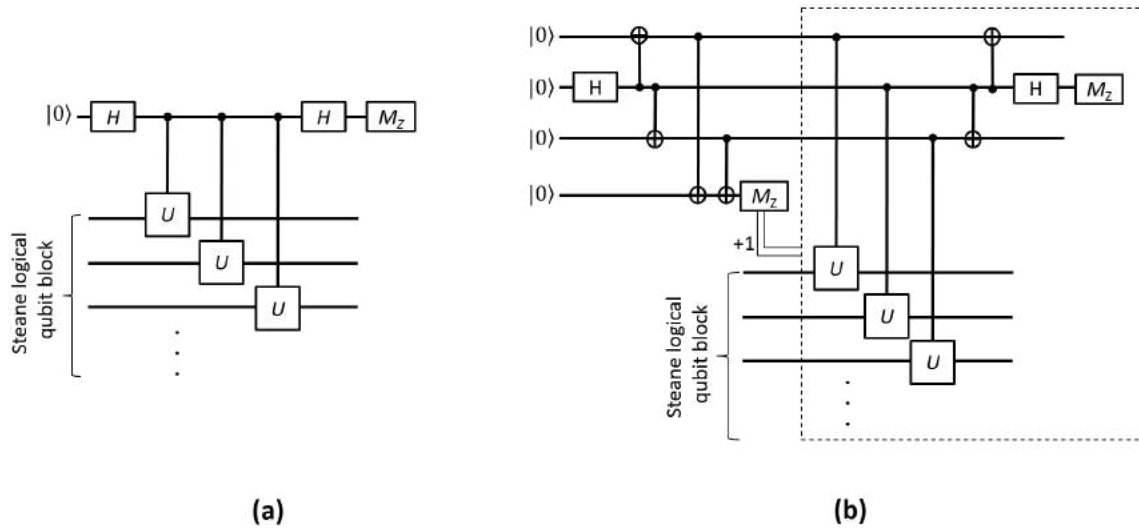


FIGURE 2.9: Comparison of non fault-tolerant (a) with fault tolerant (b) Measurement of logical operator U .

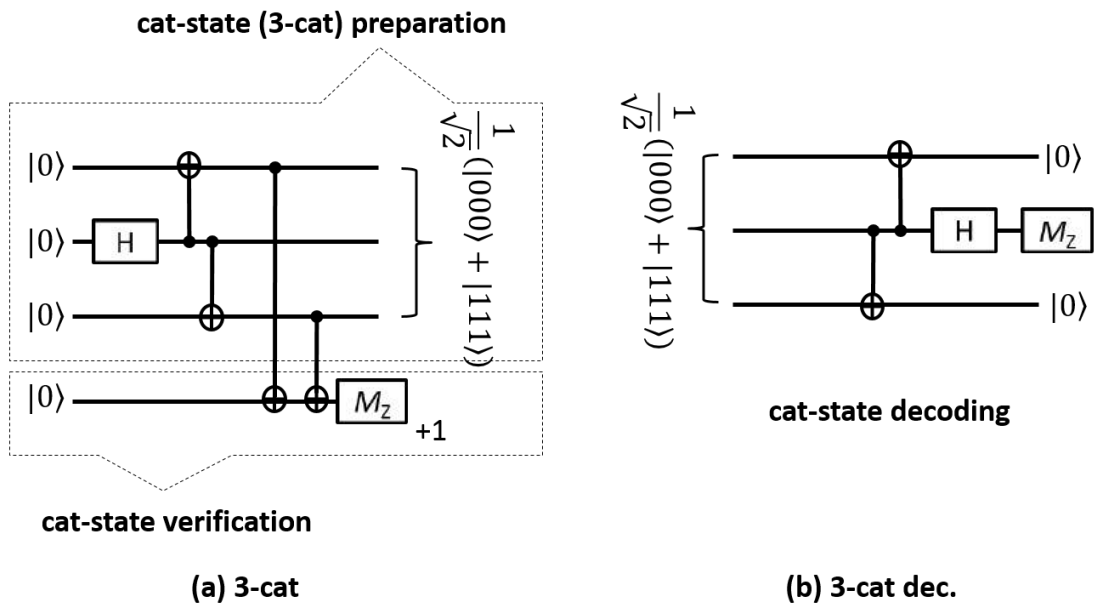


FIGURE 2.10: Encoding and Decoding circuits of a 3-cat state

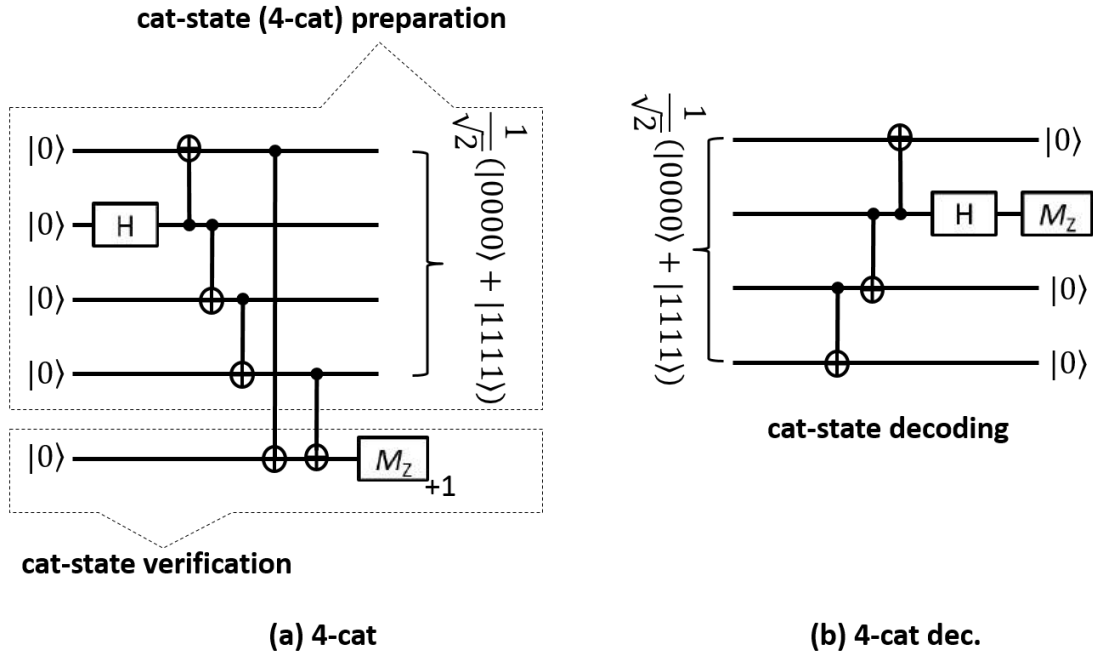


FIGURE 2.11: Encoding and Decoding circuits of a 4-cat state

protected by multiple layer of concatenation using circuits from the last subsection.

Construction of the concatenated code

We know that using encoding circuit of Figure 2.14, 7 physical qubits encode one Steane logical qubit. Since only one layer of encoding is used, we call it layer-1 or L1 qubit. The logical gate acting on L1 qubit operand is termed as L1 gate. When second layer of encoding is employed, we can use the same encoding circuit by replacing each (1) physical qubit by corresponding L1 qubit and each (2) physical gate by equivalent L1 gate. This method can be generalized to obtain logical operation at second layer of encoding (L2-operation) wherein, we replace each constituent physical qubit or operation with corresponding L1-qubit and operation respectively. Using this recipe we can easily translate fault-tolerant operations at first layer of encoding shown in the previous section, into fault-tolerant circuit at second layer of encoding. Conclusively, an L2 operation can be constructed from an constituent L1 operation

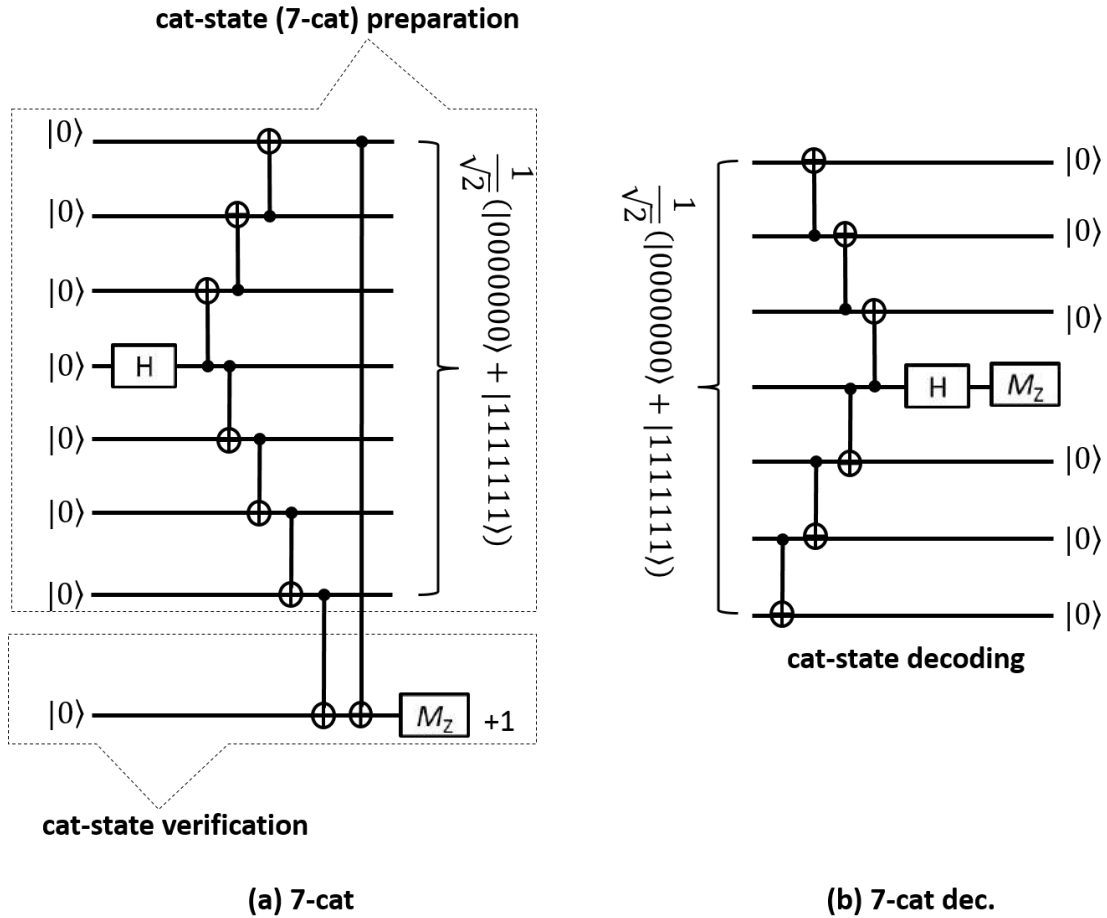


FIGURE 2.12: Encoding and Decoding circuits of a 7-cat state

in the same way as L1 operation derived from physical (L0) operations. Hence if we know how to construct fault-tolerant operation circuit at first layer of encoding, we can recursively construct fault tolerant operation at any layer i using $i - 1$ fault-tolerant operations. In general different codes can also be used to provide varying degrees of protection across layers of concatenations. This summarizes the key idea behind concatenated quantum error correction. We use Steane code at all layers of concatenation in this thesis.

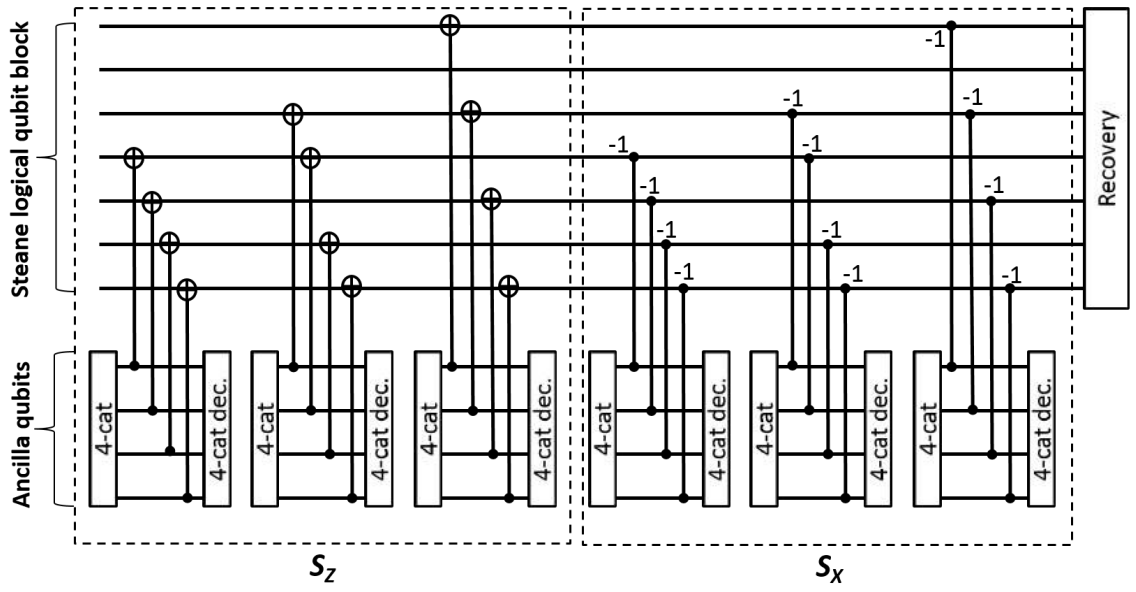


FIGURE 2.13: Fault tolerant circuit for detecting errors in the Steane logical block. The sub-circuit S_Z extracts syndrome for phase flips while the sub-circuit S_X extracts syndrome for the bit flips

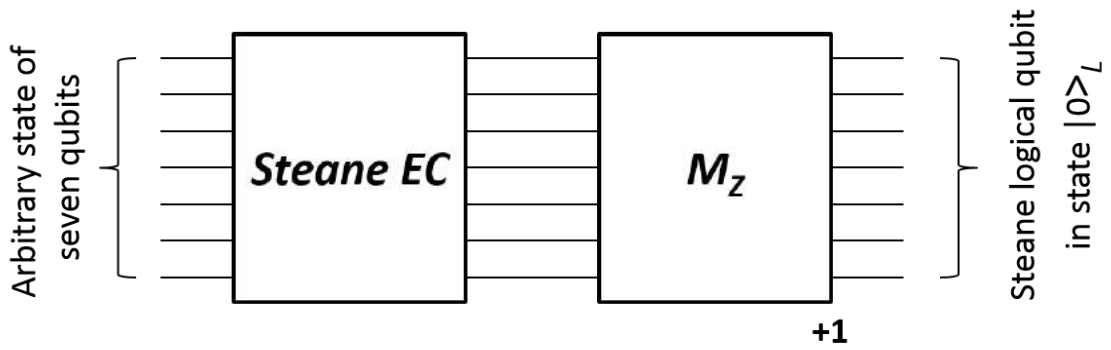


FIGURE 2.14: Circuit for the preparation of Steane $|0\rangle_L$ state. The logical measurement M_Z circuit is obtained by substituting Z for U in Figure 2.9(b)

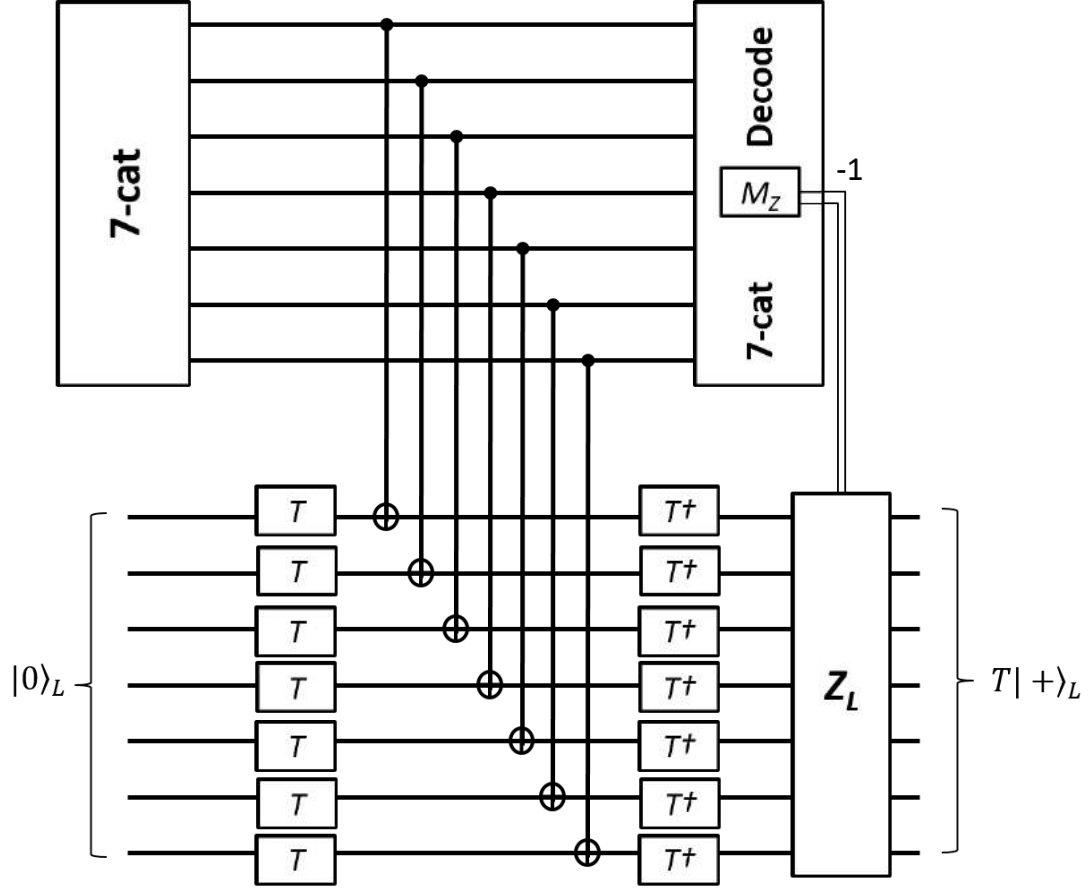


FIGURE 2.15: Fault tolerant circuit to prepare Steane $T|+\rangle$ state. The circuit requires the preparation of Steane $|0\rangle_L$ state. The fault-tolerant measurement of an operator $e^{-\frac{i\pi}{4}}SX = TXT^\dagger$ requires transversal application T , $CNOT$ and T^\dagger gates

Overhead scaling in the concatenated code

Let us now analyze the overhead incurred in using additional encoding layer. To quantify the worst case overhead we choose a logical operation which comprises most number of physical operations (e.g., T or Toffoli gate), call this op^{worst} . It is easy to verify that if L1 op^{worst} requires k physical operations, the corresponding L2 operation will consume no more than $k op^{worst}$ L1 operations or k^2 physical operations. Similarly an L3 operation will consume $k op^{worst}$ L2 operations or k^3 physical operations. In general operation at layer n will consume k^n physical operations.

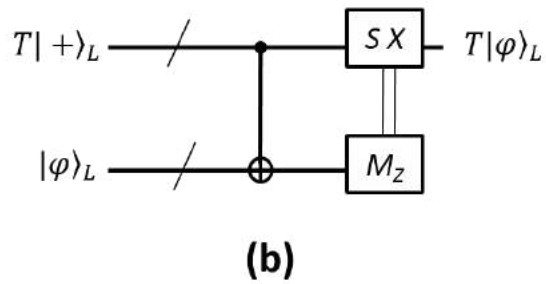
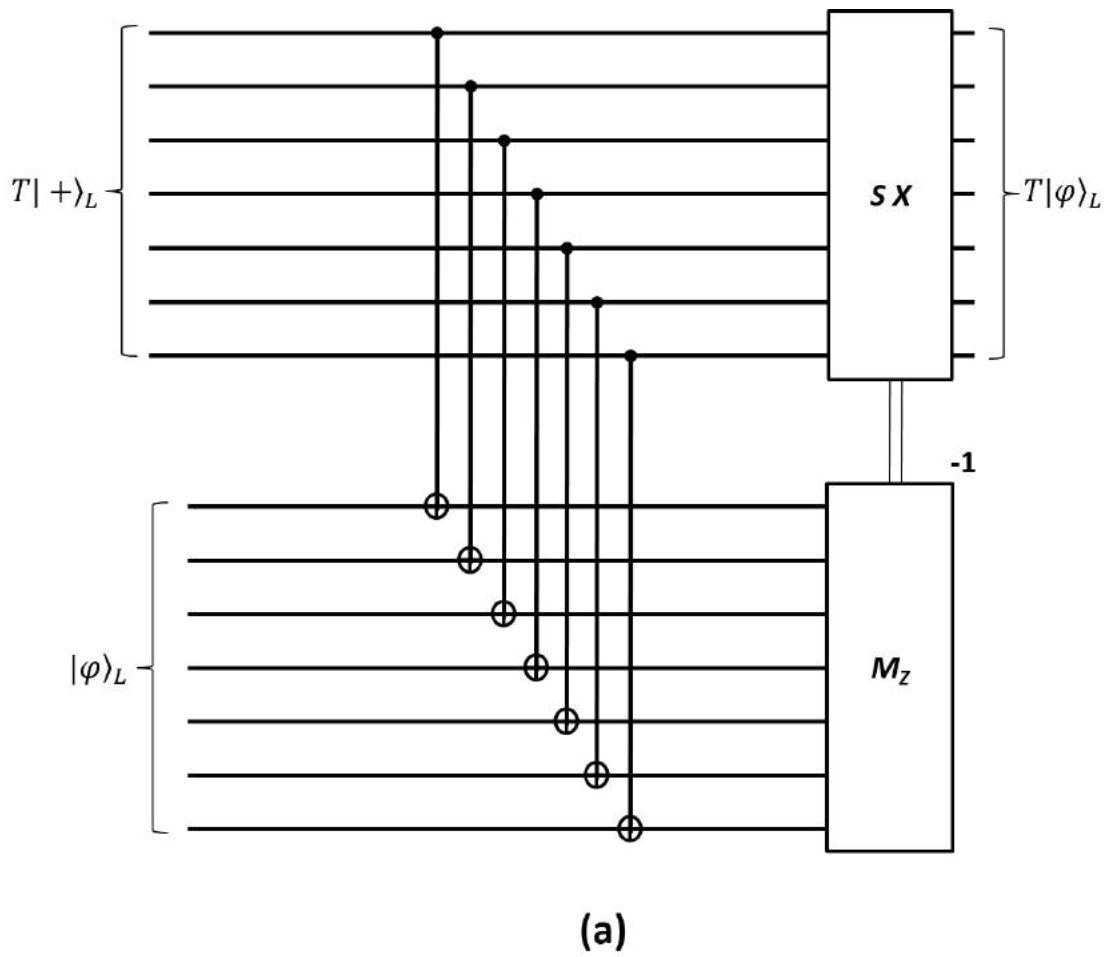


FIGURE 2.16: (a) Fault-tolerant circuit to perform Steane T gate using magic state $T|+\rangle$. The sub-figure(b) is a concise presentation of fully expanded circuit in (a)

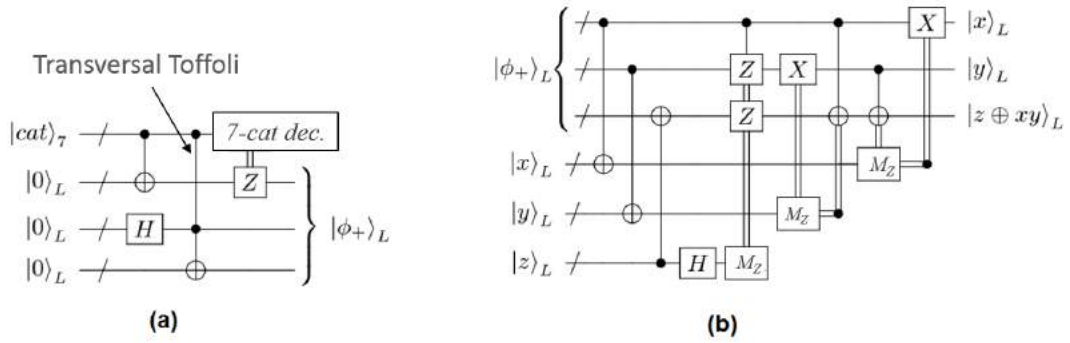


FIGURE 2.17: (a) The Toffoli magic state $|\phi\rangle_L$ (which requires the application of transversal Toffoli gate) (b) Data teleportation into the magic state $|\phi\rangle_L$.

Hence, computational overhead scales exponentially with concatenation layers. To evaluate the resource overhead scaling, we perform similar worst case analysis by replacing operation count by qubit count. Since we know that qubit count is always bounded by the operation count, therefore we conclude that in the worst case, resource overhead scales exponentially with the layers of concatenation. Now if we compare the overhead of the concatenation with the reduction in failure probability, we find that double exponential decrease in failure probability comes at the cost of only single exponential increases in operation and resource overhead. In other words the gain in performance exponentially outweighs the cost of concatenation. This means that by adding levels of encoding, the failure probability per logical operation sufficiently can be exponentially reduced to arbitrarily low value in order to execute sizable quantum circuits. This concept is formally summarized by quantum threshold theorem Aharonov and Ben-Or (1997).

Quantum Threshold Theorem

The formal statement of quantum threshold theorem is given in Theorem 1.

Theorem 1. *The quantum circuit containing $g(n)$ gates will fail with probability at*

most ϵ , by incurring operation and resource overhead scaling only poly-logarithmically in $\frac{g(n)}{\epsilon}$, given that each physical gate fails with probability p strictly less than some threshold value: p_{th} .

The condition $p < p_{th}$ is called threshold condition which ensures that for $C > 1/p_{th}$, the logical failure probability $p_L = Cp^2$ of a encoded qubit/operation is strictly smaller than p when error correction is performed. This ensures that there is a net gain in reliability due to the computation performed on encoded qubit. Since p_L strongly depends on the number of fault points C in the fault-tolerant circuit block, the threshold value p_{th} is often determined by the most complicated fault-tolerant operation for given error correcting code. Once threshold condition is met, multiple layers of encoding per logical qubit are used to recursively minimize the failure probability as $\epsilon = p(n)(Cp)^{2d}/C$ where $C > 1/p_{th}$ and d is the number of concatenation layers.

Limitations of the quantum threshold theorem and new research direction

Theorem 1 is an encouraging result, it only gives rough estimate of overhead of qubit resources and execution time of the quantum application circuit. We find that there is substantial space for the resource-performance co-optimization. For example, the ancilla qubits used in the complicated logical operation can be shared and reused among multiple logical data qubits during the circuit scheduling. The sharing may increase execution time but reduces overall resources, leading to an interesting study of trading performance with resource investment. Similarly, the frequency of error correction opens another avenue of optimization. Instead of inserting error correction after each logical gate, we can selectively error correct logical qubits depending upon the estimate of accumulated physical noise. Finally, as quantum system scale, the role of communication channels and the support of concurrently executable gates becomes heavily dependent on the hardware constraints and resource distribution

architecture. Therefore, complete characterization of resource-performance needs in addition to quantum threshold theorem, the design of quantum hardware system which will be described in the next chapter.

2.6 Summary

In this chapter we described the basic concepts of quantum computation including qubits and various quantum gates used in practical applications. We also illustrated how to protect quantum information from noise by error correction using Steane code as an example. The fundamentals of the constructing encoded qubit basis states were provided as well as procedure of applying gates on encoded qubits was thoroughly discussed. Using these concepts, the basic principles of fault-tolerant computation were developed. In the end we showed that while quantum threshold theorem guaranteed desired level of fault tolerance at the cost of modest resource overhead, it did not provide adequate insights into the resource-performance trade-offs in the presence of limited hardware resources.

Trapped-Ion Quantum Hardware

Quantum computation needs to be physically realized in order to gather practical utility. The basic criterion of achieving computation in realistic hardware stems from the operational requirements of the components of general quantum circuits. While it is possible to experimentally validate quantum mechanical behavior in several hardware technologies, the ability to harness these properties for meaningful computation needs to be specifically characterized. In this regard the famous Loss DiVincenzo criterion Loss and DiVincenzo (1998) details fundamental requirements for candidate quantum device as follows:

- It provides ‘well-defined’ representation of qubits
- It can ‘initialize’ qubits in the basis state
- It ‘retains’ the qubits state with sufficiently low decoherence
- It allows reliable mechanism of ‘measuring’ qubit states
- It performs ‘universal set’ of quantum gates to enable arbitrary quantum computation

The technological advancements to explore quantum mechanical properties of matter has led to several types of physical implementations of quantum systems Ladd et al. (2010a). However, only few of these have shown potential to become serious contenders for the large scale quantum computer in the face of several technological challenges. In this chapter we describe trapped-ion based quantum computers which is one of the strongest candidate for the large scale quantum computation Cirac and Zoller (1995).

3.1 Introduction to the Trapped-Ion Quantum Computer

The trapped-ions offers very reliable hardware platform for the quantum computation which has sparked great interest and significant progress in this technology Nielsen and Chuang (2000); Monroe et al. (2014); Monroe and Kim (2013). It not only adequately fulfills DiVincenzo criterion but also possesses several prominent properties demonstrated experimentally which are conducive to architectures for the fault-tolerant quantum computation. The qubit is realized by the atomic ion (e.g., $^{171}\text{Yb}^+$ ion Olmschenk et al. (2007)) and its basis states(e.g., $|0\rangle$ or $|1\rangle$) can be mapped to highly stable internal spin states of the ion. These may be the two ground state of ‘hyperfine qubit’ or a ground and an excited states of optical qubit. The ions can be held in the free space by adequate application of electromagnetic field and DC voltage on the physical trap. A set universal quantum gate can be executed by firing appropriate frequency lasers beams to the operand ions which results in the transitions between the spin states. In case of single-qubit gates (arbitrary rotation around Bloch-Sphere), we can direct the laser beam to the target ion and change its internal state through Raman Transitions. For two-qubit gates(e.g., CZ), the laser induces coulombic interaction between the operand ions such that spin of one ion can change the other’s. Multiple ions can be trapped to form a chain to share common vibrational mode which can be used to enact multi-qubit gates. The mode

of vibration acts as quantum bus which can be used to transfer qubit information from one ion to another or to affect their states. The internal spin of an ion can be coupled with the bus. The spin-coupled vibrational mode is shared by all the ions in the chain. For ion whose spin needs to be affected, the vibrational mode is swapped with its spin, thus changing the state of target qubit. Both coupling and swapping are again accomplished by lasers of appropriate frequencies. It should be noted that when ions tend to heat up when operated using vibrational modes, this heat can adversely affect the state of the stored qubit. To remove heat from the ion, either low frequency laser (doppler's cooling) or interaction with the additional low energy ion (sympathetic cooling) is employed.

Both initialization and measurement in the desired basis state can be easily implemented using laser tuned (or detuned) to the state. For initialization, the laser continues to excite the ion to collapse into a state (chosen to be the basis state) which is impervious to the laser excitation. In case of measurement, depending upon the state of ion, the laser application leads to two distinct types of state-transition; one in which ion aggressively emits photons and the other in which it does not. Thus qubit states can be adequately readout.

The trapped-ion technology is making rapid progress Benhelm et al. (2008) to attain high quality quantum memory and basic operations (gates, initialization and measurement). There are two important factors of these advancements. First, the methods of trapping ions or group of ions have achieve sufficient level of sophistication Noek et al. (2013b). Once initialized, the ions can retain their state for period much longer than the execution time of the gates and measurement Mount et al. (2013). Thus qubit memory is very robust and reliable component of large scale computation. Second, the trapped ions are accessible for operations; they can be addressed individually for quantum gate Crain et al. (2014a) as well as measured with high fidelity Noek et al. (2013a). Third, the amount of control with which

Table 3.1: State-of-art infidelities of various quantum operations in the trapped-ion hardware. The variable T in the last row, represents time in seconds

Quantum operation	Infidelity	Reference
Single-qubit gate	10^{-6}	Brown et al. (2011)
Two-qubit gate	10^{-3}	Ballance et al. (2014)
Initialization	10^{-4}	Harty et al. (2014)
Measurement	10^{-4}	Harty et al. (2014)
Memory	$1 - e^{-T/10}$	Langer et al. (2005)

lasers can manipulate the ion state has matured enough to allow for reliable quantum operations. The quantum control techniques such as dynamic decoupling and compensation pulses empower quantum gates with higher accuracy and greater precision. Table 3.1 shows state-of-art quality of quantum operations measured by their infidelity. With the current pace of advancement, these numbers are expected to nosedive providing physical quantum component with even greater reliability Benhelm et al. (2008)

3.2 Architectures for Trapped-Ion Quantum Computers

3.2.1 *Why quantum architecture?*

One of the major themes in classical computer design revolves around compensating the inefficiencies of a physical hardware with additional resources efficiently utilized by an adequate architecture. For example, an appropriate memory hierarchy can substantially hide the latency of communication between slow storage units and fast microprocessor. The power dissipation in the hardware can be controlled by distributing computation among multiple data processor cores. The clock skew arising in the FPGA and ASIC systems is minimized by sophisticated clock-tree architectures. In this sense, the current status of quantum hardware technologies justifies the quest for suitable architecture space. To design a quantum computer containing millions of qubits, the system architecture needs to be defined. Even though the lab

demonstration has been restricted to dozen qubits, several designs have been proposed to scale trapped-ion technology to larger systems. The goal of scaling small sized lab computer fiddling with toy problems in the lab to the sizable computers solving meaningful practical problems, invites expertise from the field of physics as well as computer science. On one hand, device technology continues to evolve to provide us with quality components. On the other hand, we need to architect meaningful circuit blocks from these components which can be used for computation. The task of designing quantum system for given application amounts to mapping quantum circuit on the hardware within the framework of its physical constraints. The study of quantum architecture sheds insights into various design strategies in order to systematically construct larger quantum systems. The salient features of quantum architecture study includes:

- Assignment of qubits and gates to the resources provided by the hardware e.g., ions and lasers
- Strategies to group physical qubits according to their collective task (e.g., basic fault-tolerant operation)
- Hierarchical design of complicated blocks (e.g., logical Toffoli or T gate) by assembling simpler blocks
- Choice of communication channels to connect different qubit blocks
- The structuring of associated classical control system to drive quantum hardware resources to perform specific set of tasks (e.g., physical quantum operation)

While all these aspects of architecture study are vital, the design of control system can be aggressively leveraged from the maturity in classical computer architecture or

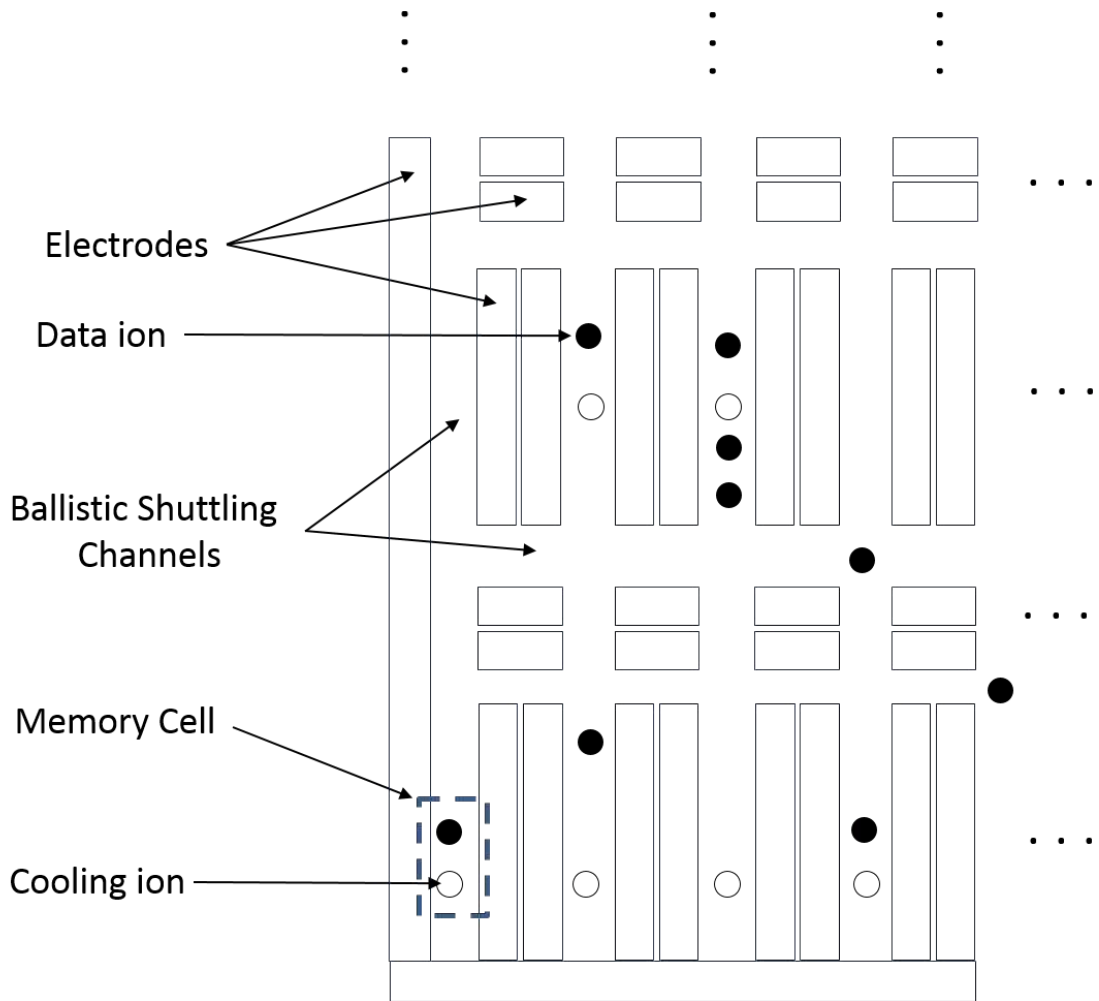
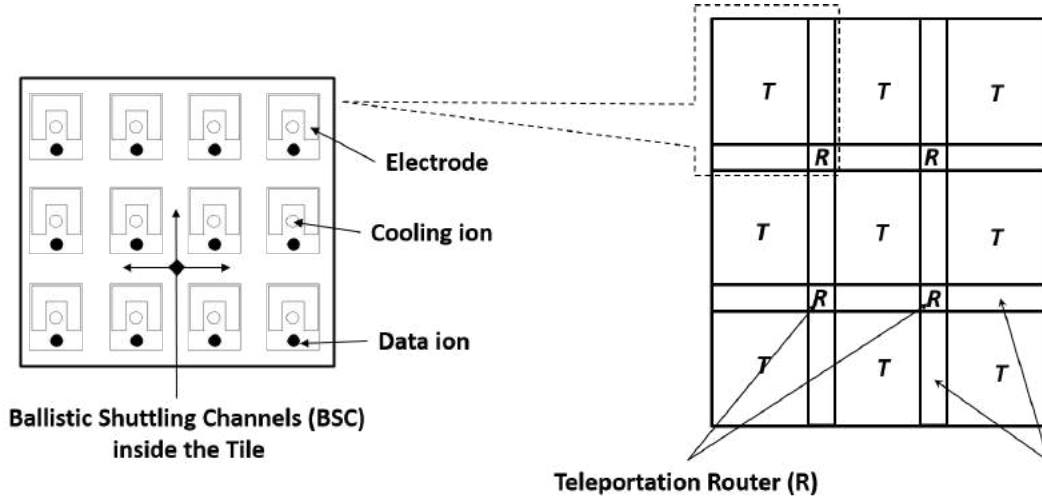


FIGURE 3.1: Description of a monolithic trapped-ion quantum hardware

perhaps borrowed from the field of ASIC design. The frontier of quantum architecture needs far greater attention since the technology is still nebulous, thus, invites detailed investigation.

3.2.2 *Brief Survey of Trapped-Ion Quantum Computers*

Several architectures for ion traps based quantum hardware have been propounded. A large set of architecture was constructed by connecting surface traps holding short chain of ions arrays to form a monolithic 2-D planar surface in which all necessary



(a) Zoomed in QLA Tile

(b) The QLA Architecture

FIGURE 3.2: Description of Quantum Logic Array Architecture

operations for quantum processor were executed. The Figure 3.1 shows ions were trapped in the space between electrodes and could move from one point to another within and across different electrodes. This is called ‘ballistic shuttling’ of ion and it is accomplished by continuously changing the voltages electrodes through which ions were incidentally passing. The heat acquired by ion during shuttling was removed by sympathetic cooling. The architecture was modeled as 2-D grid of identical ‘cells’ (See Figure 3.2(a)). A cell represented location for the storing qubit or performing single- or two-qubit gates or shuttling passage for the ion.

Based on the abstraction framework, a generation of trapped-ion architectures were developed to study the performance of Shor’s algorithm and/or its component circuit blocks. The circuits were encoded using Steane $[[7,1,3]]$ code. These are described as follows:

Quantum Logical Array (QLA) Architecture

The Quantum Logic Array architecture Metodi et al. (2005) in Figure 3.2, divides trapped-ion hardware into uniform sized blocks comprising basic cells. These blocks are arranged on a regular 2-D grid, were named ‘Tiles’. The cells within the Tile can be classified as either memory cells for storage and manipulation of qubits for quantum gates, and transportation cells which act as building blocks of ballistic shuttling channels (BSC). The Tile clusters sufficient physical qubits to allow the (1) execution of basic set of fault-tolerant operations within the Tile and (2) communication with qubits located in the different Tile. The physical qubits in the Tile can be classified as

- Data qubits: These qubits constitute Steane logical qubit block.
- Ancilla qubits: These qubits are used in error-correction and fault-tolerant logical operation.
- Communication qubits: These qubits are dedicated to form and purify communication channels between Tiles.

These qubits are stored and operated in the memory cells while rapid communication to enable two-qubit gates for the fault-tolerant operations is facilitated by the ballistic shuttling channels. These channels enable communication within the Tile and also, in part, across the Tile. Using BSC laid down between the boundaries of the Tiles, the qubits dedicated for communication from two different Tiles come together to form several EPR pairs. These EPR pairs form entanglement channels between Tiles which arbitrate inter-Tile logical gates using quantum teleportation (please refer to section 2.2.5 to understand the teleportation protocol which physically transfers quantum information from one qubit to another). Before applying

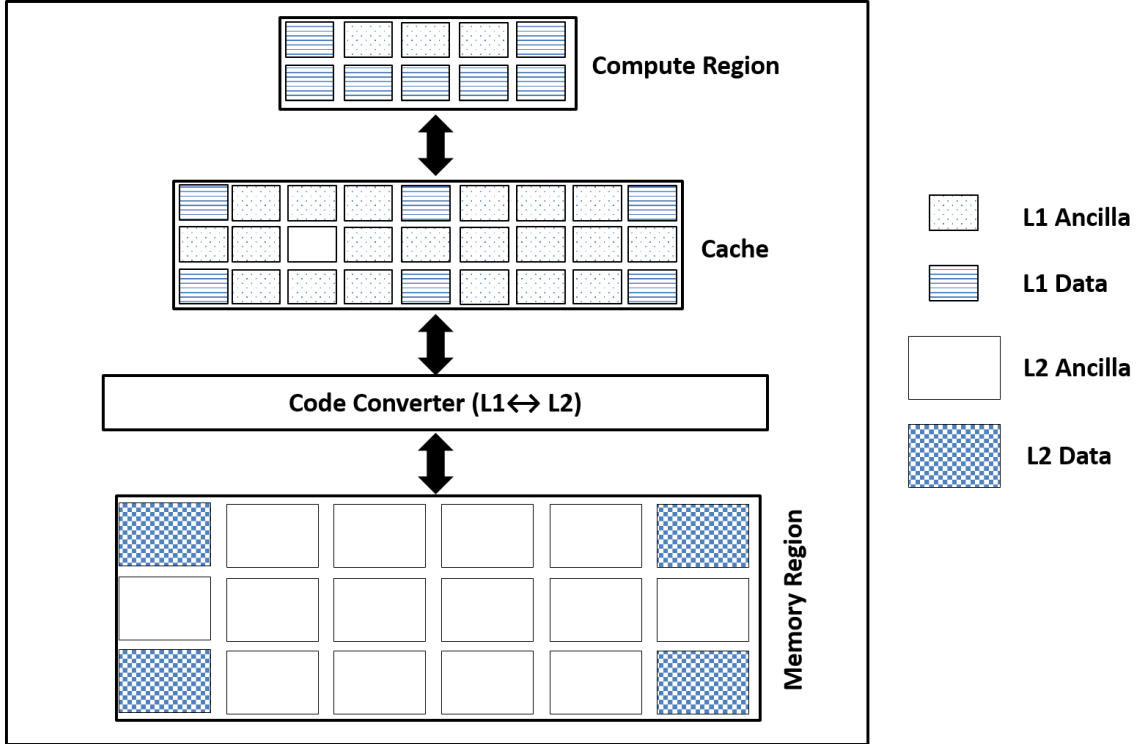


FIGURE 3.3: Description of Compressed Quantum Logic Array Architecture

teleportation, EPR pairs were sanitized using purification protocols implemented as quantum repeaters Briegel et al. (1998) which components the Tile.

The QLA architecture replicates same Tile design over the entire architecture. This way it emulates multi-processor distributed system to enable maximally parallel execution of quantum circuits. The speed comes at the cost of large number of resources leading to substantial amount of hardware area $1m^2$ to factor 1024-bit Shor’s algorithm.

Compressed Quantum Logical Array (CQLA) architecture

Compressed Quantum Logical Array (CQLA) architecture Thaker et al. (2006) in Figure 3.3, reduces the QLA area by allocating just enough resources to match parallelism quantum available in the quantum application circuit. It divides the architecture into two types of regions:

- Memory Regions: These act as storage units for the logical qubit blocks and support error-correction to retain their fidelity
- Compute Regions: These regions temporarily hold operand logical qubits for computation. Once logical operation execution, the space is vacated for next operation and operands are sent back to memory.

Using this organization, CQLA implements classical von-Neumann micro-architecture for quantum computers. The communication component of the architecture was leveraged from the QLA work. However, unlike QLA which tiles similar qubit blocks, the CQLA regions comprise different types of blocks. Both memory and compute regions contain blocks of ancilla qubits and the number of these blocks differs significantly. The compute region clusters large number of smaller ancilla blocks (Data-to-Ancilla ratio 1:2) while memory regions contain fewer large size blocks (Data-to-Ancilla ratio 8:1). Furthermore, different layers of encoding are used to protect quantum data in each region. Two layers of encoding are deployed to enhance reliability of storage. Whereas, single encoding layer was used for faster processing of gates in the compute regions. A fault-tolerant encoding code converter is used to change the encoding layer of data as it moves across two regions. Using these optimizations, CQLA provides 13x improvements in the area and 8x improvement in the speed compared to the QLA architecture.

QALYPSO Architecture

QALYPSO architecture Isailovic et al. (2008) in Figure 3.4, recognizes that the critical path of quantum circuit execution is dominated by ancilla preparation used in error-correction and fault-tolerant operation. By separating the preparation of ancilla state from the main stream computation (involving only data operands), overall performance can be optimized. A lumped parameter called Area-Delay-to-Correct

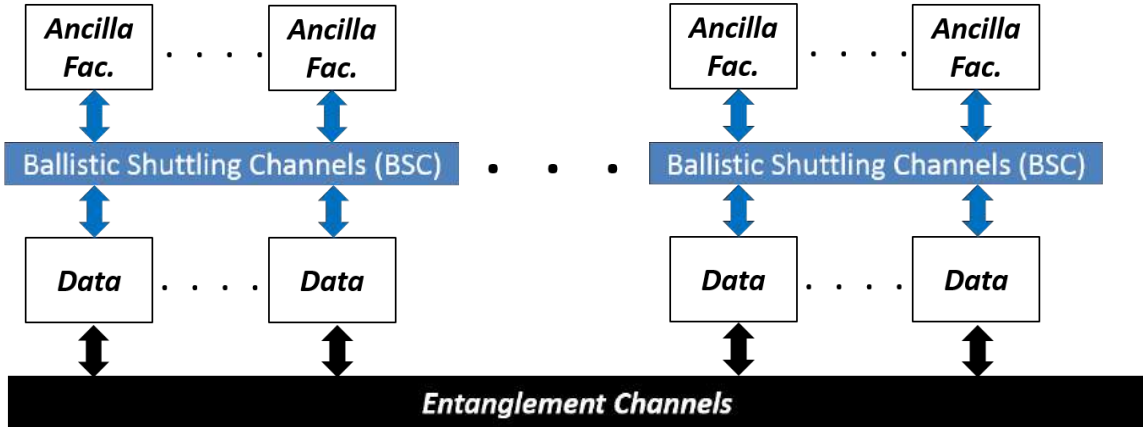


FIGURE 3.4: Description of Qalypso architecture

Result (ADCR) is defined to capture the resource investment in the architecture and performance of quantum application. QALYPSO design specializes in dedicated ancilla preparation units (called ancilla factories) which are separated from data processing units (containing logical qubits blocks only). Depending upon the application, the amount of ancilla preparation overhead is minimized by (1) optimizing the layout and the placement of ancilla factories (2) matching the production bandwidth of ancilla factory to the demand of data units. Multiple types of factories are introduced to encompass both error-correction and magic state preparation. BSC connect factories with data units while flexible entanglement channels fulfill long-distance communication for non-local gates between data operands. Moreover, unlike QLA and CQLA, the architecture is equipped with different bandwidth channels in order to match the application level communication needs. With flexible ancilla and communication resource distribution, QALYPSO architecture advances the state-of-art design by improving ADCR of QLA, CQLA by 6 to 7 orders of magnitude. At the same time it significantly reduces the area needed for Shor's algorithm from $1m^2$ in QLA to $76.6cm^2$ in QALYPSO.

3.2.3 Feasibility of Quantum Architecture

The generation of the above mentioned architectures utilizes scalable network of planar ion traps. At the time of these proposals, it was anticipated that technology advancements would soon deliver hardware which could trap and manipulate large number of ions in the planar traps. A decade has passed since this pioneering work in ion-trap architecture was first published, however, the scalability of planar ion-traps still faces wide range of technological challenges Guise et al. (2014). For example it has been noticed that the individual addressing of ions becomes prohibitively difficult due to the geometry constraints of the planar traps chips containing in excess of hundred electrodes. Hence processing very large number of ions in these chips seems practically difficult. The problem of scalability can be tackled if reliable means of connecting these restricted size chips can be found. On recent proposal envisions large number of smaller trapped-ion units connected by fast optical switch network Kim and Kim (2009). It combines two complementary quantum hardware technologies to constructed an integrated hybrid-quantum computer which uses ions for computation and photons to assist communication. The physics of trapped-ion hardware features reliable quantum memory and local gates but sluggish for non-local interaction. In contrast, photons which make relatively unreliable basic quantum components (poor memory and noisier gates), are perhaps the only viable choice for long-distance multi-qubit gates. Based on these complementary features, we now delve into the proposed ModUlar Scalable Ion-trap Quantum Computer (MUSQIC) hardware.

3.3 MUSIQC Hardware

MUSIQC contains three main components Elementary Logic Units (ELU), reconfigurable Optical Switch (OS) and photon processing circuitry (Beam splitters and photon detectors) as shown in Figure 3.5. An ELU contains chain(s) of ions ei-

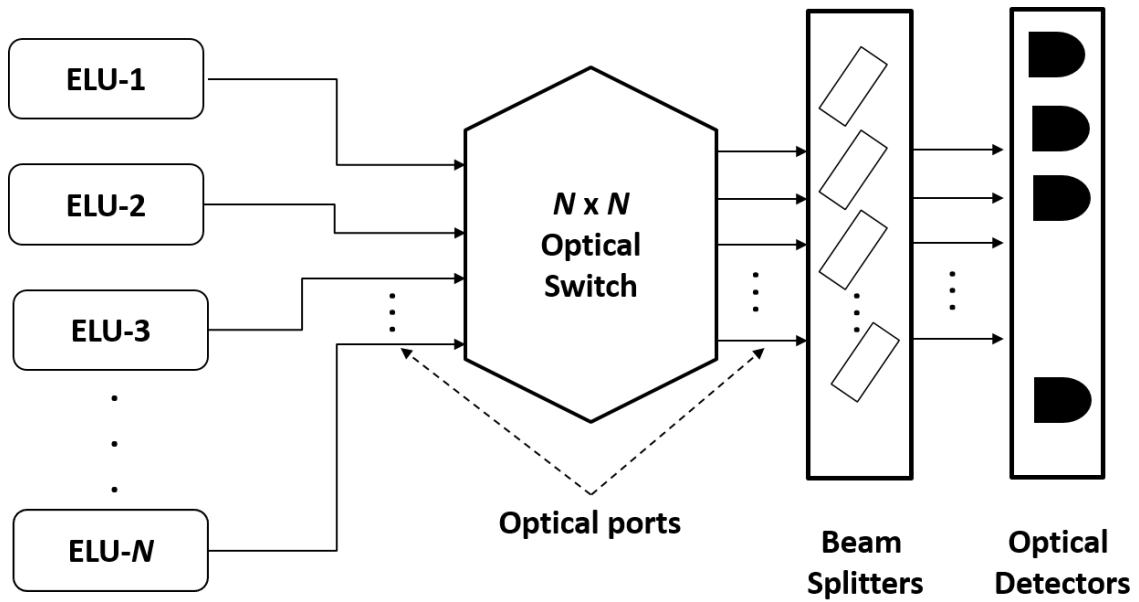


FIGURE 3.5: Description of Modular Scalable Ion-trap Quantum Computer (MUSIQC) and its important components

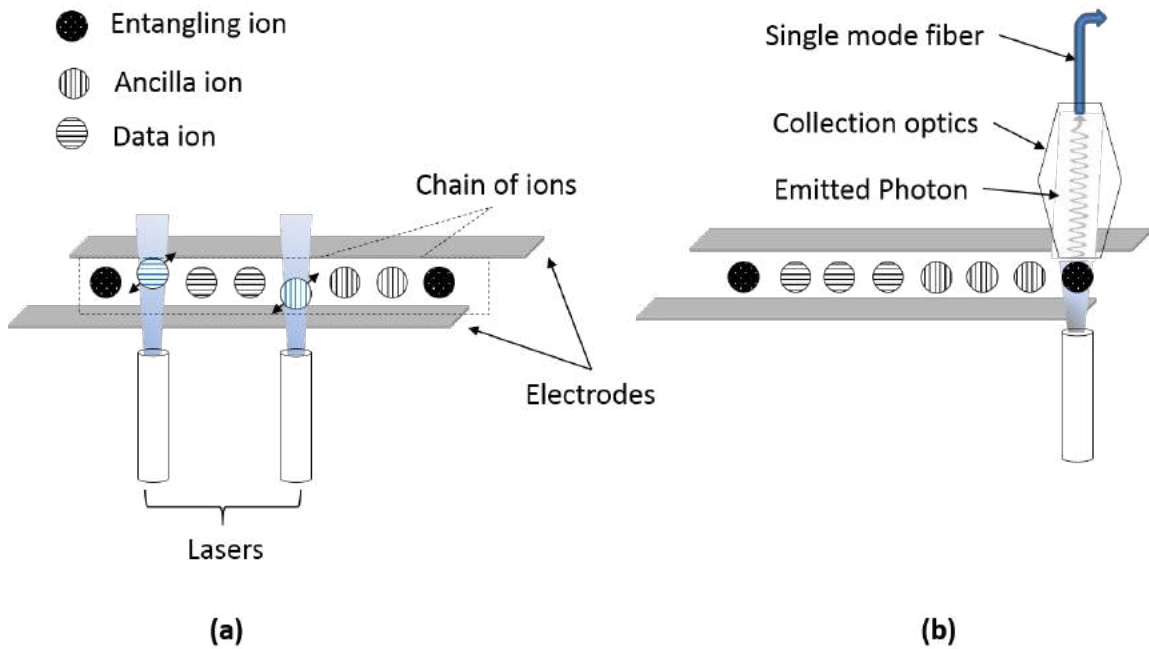


FIGURE 3.6: Important components of an Elementary Logic Unit (ELU)

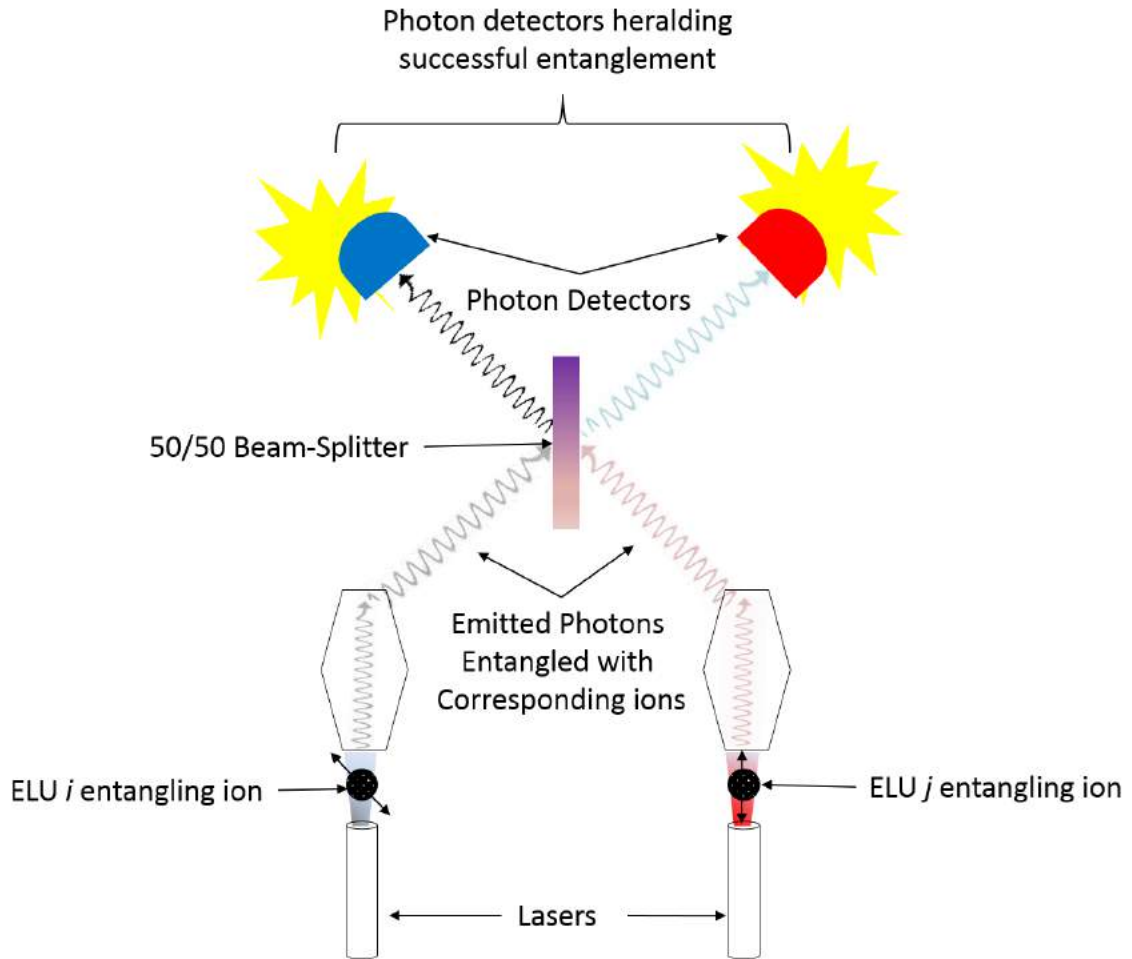


FIGURE 3.7: Description of heralded entanglement procedure to connect different ELUs.

ther held as the Quantum Charged Coupled Device (QCCD) or in the small planar traps described earlier. It acts as basic resource unit of computation to support fault-tolerant operations on very small of logical qubit(s) contained within. The ions inside an ELU are assigned specific computational roles. The data-ions hold physical qubits comprising logical block, the ancilla-ions map ancilla qubits needed in the error correction and fault-tolerant operation, the entangling-ions (e-ion) which form EPR pairs with the entangling-ions of the different ELU. See Figure 3.6 for details. These ions are manipulated by lasers using Micro-Electro-Mechanical Sys-

tems (MEMS). This system provides finer control while steering the laser beam from ion location to another within an ELU. It facilitates reliable addressing of individual ions while preventing interference with neighboring ions Crain et al. (2014a).

Multiple ELUs are connected by establishing EPR pairs based entanglement channel between their respective e-ions. Using these channels, the logical qubits across ELUs can undergo non-local two-qubit gates using local operations and classical communication (LOCC). An $N \times N$ optical switch containing N input ports and N output ports, can be configured to connect any pair of ELUs in the system. A 1000×1000 switches has already been demonstrated Kim et al. (2003). To generate entanglement link between ELUs, the e-ions first need to communicate with their remote partners using photonic interaction guided by the optical switch. A protocol named ‘heralded entanglement’ Duan et al. (2004) entangles two e-ions as follows. Each ion initialized to desired state, emits photon whose quantum state (polarization or frequency) is entangled with the spin of ion. The two emitted photons are guided to interfere at 50/50 beam-splitter. After interference, the photons states are measured in appropriate basis by optical detectors. The occurrence of successful detection is a probabilistic event but it is heralded by the detector output. In brief, the heralded entanglement first entangles ions with photons, then entangles the photons and measures them, hence indirectly creating an entanglement resource between remote e-ions. The pictorial summary of this procedure is shown in Figure 3.7. The resulting EPR pair allows non-local gates across the two ELUs via quantum teleportation Gottesman and Chuang (1999) (or similar) protocol which simply needs classical communication preceded by the less costly quantum operations localized to the ELU.

It should be noted that the heralded entanglement procedure has small success probability due to technology limitations such as low photon collection efficiency and imperfections of optical path length. Fortunately, it is possible to repeat the en-

tire procedure until entanglement is accomplished. Hence the effective rate of EPR pair generation can be increased by investing more e-ions per ELU and photonic hardware components which in turn requires multiple OS port per ELU. This leads to significant resource overhead. Alternatively, we can use Pipelined Entanglement Generation (PEG) Monroe et al. (2014) which relies on the observation that the latency bottleneck in one round of entanglement is the ion initialization process taking ($1 \mu s$), about 100 times longer than photon emission, transmission and detection (collectively estimated to be $10 ns$). This lopsided latency contributions can be used to speed up the EPR pair generation process using PEG. This scheme also employs multiple ions per port, however these are time-multiplexed in a way that another ion which has already been initialized and have sent in its photon, occupies the port for an entanglement attempt while initialization for previously unsuccessful ion is underway. By hiding ion time consuming ion-initialization process through efficient pipelining, PEG achieves high EPR pair throughput with relatively fewer resources.

The MUSIQC architecture modularizes the complex design of large scale quantum system by recognizing the limitations of state-of-art technology. The hybrid assembly of MUSQIC extracts good of both ionic and photonic worlds, while at the same time countering the frailties of both technologies. Because of its enhanced versatility, practicality and scalability, we choose MUSIQC as the underlying quantum hardware for our performance simulation analysis. Following attributes summarize unique features of MUSIQC hardware.

- It provides feasible means to scale trapped-ion based quantum without using impractically large monolithic ion traps. MUSIQC can connect large number of small sized trapped-ion chips using reconfigurable optical switch to allow global connectivity
- It combines the distinguishing features of two candidate technologies: trapped-

ions for reliable storage and computation while photons for communication channels. Such hybrid quantum system can benefit from rapid advances in the techniques which integrate optical components on the trapped-ion chips

- The latency cost of slow EPR pair generation can be completely masked by the execution time of essential fault-tolerant procedures which can be scheduled in parallel with heralded entanglement. Based on the pace of technology advancement, the expected latency of EPR pair generation takes around $5,000\mu s$ which can be smaller than the time spent in one round of layer-2 error correction (ranging from $5,000 - 48,000\mu s$ depending upon frequency of layer-1 error correction). Comprehensive details can be found in chapters 6 and 7.
- Since the speed of photons is negligible compared to the shuttling speed of ions, the latency of communication is entirely dominated by EPR pair generation and therefore independent of spatial distance between the distant qubits. The distance-oblivious global connectivity allows speedy execution of the quantum arithmetic circuits comprising several long distance gates (e.g., QCLA). These circuits are extensively used in Shor's algorithm.

3.4 Architecture Support in MUSIQC Hardware

Several remarkable features of MUSQIC hardware allows us to define range of architectures by varying resource allocation and interconnection hierarchy. An architecture built on MUSIQC hardware can easily emulate previously proposed designs. For example, Tiles in QLA, CQLA and QALYPSO can be mapped to the ELUs while long distance ballistic shuttling channels can be simulated by the optical channels. In addition, a vast architecture space spanned by the size, and composition of qubit resources in the ELU as well as the hierarchy and topology of optical switch network can be analyzed for interesting resource-performance trade-offs. In chapter 5, 6

and 7, different types of architecture realized from MUSIQC hardware will be studied which will guide us to choose appropriate design for given quantum application circuit.

3.5 Summary

In this chapter we presented trapped-ion as candidate technology to construct quantum computer. The qubit information was stored in the internal energy states of the ions. The quantum gate was realized by changing these energy levels by firing appropriate frequency lasers on the operand ion qubits. We also described that role oriented clustering of ion qubits, hardware constraints and benchmark performance optimization necessitated the study quantum computer architecture. We surveyed a generation of trapped-ion based architectures proposed in the last decade and detailed their merits and demerits. Because the scalability of these design critically hinged on building prohibitively large planar ion traps, the next generation quantum architecture MUSIQC was proposed which connected small ion trap chips using photon assisted fast communication channels. Several distinguishing features of this architecture were presented of which practicality and scalability were the most prominent. We showed how MUSIQC modularized the construction of large scale quantum computer using the advances in optics technology and efficiently executed the arithmetic circuits acting as building block of Shor's algorithm. The qualitative prowess of the MUSIQC presented at the end of the chapter will be quantified in the second half of the thesis which studies the performance of running practically interesting quantum circuits on this hardware.

Our Design and Performance Simulation Tool Box

4.1 Introduction to the Design and Performance Simulation

4.1.1 Motivation Behind the Simulation

The development of a large scale quantum computer falls in the initial design stages as experimental efforts have produced small sized computers. There are two important pillars of these effort which can be conceived as basic framework of quantum system design. First, the quality of quantum device technology needs speedy improvement to provide reliable components for basic quantum computation. Second, a mechanism must be carefully devised to integrate these components at a very large scale to construct fully functional quantum system capable of solving real-world problems. The quality of hardware depends on the physical attributes of the quantum device technology while the integration depends on the architecture of the system resources which are needed to execute quantum application. As noted in previous chapter that the inefficiencies of physical hardware can be compensated by designing adequate resource architecture. Naturally, when both hardware and architecture parameters are taken into consideration, the combinatorial design space substantially increases.

The spectrum of feasible designs is often spanned by the optimistic values of the parameters which are essential to achieve desired level of performance. Based on the targeted parameter values, the technologists and architects can be guided to focus on specific aspects of the system design which needs crucial improvements. However, as quantum device technology continues to evolve and new ways integrating device components are invented the emerging candidate design inevitably changes the standards for system parameters. In order to evaluate dynamically changing vast design space we need an efficient software tool which can not only model system components using appropriate parameters, but also estimates the performance of quantum algorithm execution on chosen design. The important features of such tool can be listed as:

- Modeling system hardware and architecture using appropriate set of parameters
- Synthesizing circuit from the quantum algorithm
- Mapping the qubits of quantum circuit on the hardware
- Scheduling the gates of quantum circuit on the hardware
- Evaluating the resource requirement and performance of quantum algorithm from scheduled circuit.

4.1.2 Deficiencies in the prior tools

A number of software tools had been developed which contained these features in whole or in part. These contribution explored various mapping and scheduling techniques in order to optimize performance and resource requirements for some chosen hardware and architecture. Using these tools significant advances were made towards refining a design of scalable quantum system some of which were highlighted in the last chapter. However, there are three important deficiencies in the work related

to the prior tools. First, the resource-performance analysis assumed fixed hardware parameters while led to less flexible design choices. The evolution of device technology in the past decade has raised questions about the assumptions under which those tools hard coded the hardware parameters. Fixing parameter values limits our horizon which can result in ignoring superior candidate designs. Second, these tools do not provide breakdown of overall performance metrics into meaningful components indicating system parameters crucial for the chosen design. In other words, it is difficult to estimate the performance improvement (or degradation) by tuning specific hardware or architecture parameters. Third, the previous work does not provide suitable yardstick to test the resource-performance scalability of proposed design. For example, as application size increase, how does the performance behavior change under different scale of qubits resources? An important problem in the ongoing experimental efforts is to calculate the maximum size of application which can be executed within fixed number of qubits in the system. Although existing tools may adequately optimize quantum system for fixed size application (e.g., Shor’s algorithm), they do not justify why does the chosen design provide desired performance level as application size continues to increase. These interesting insights for missing for two reasons (i) they are not equipped with the flexible framework of analyzing resource-performance trade-offs and (ii) they do not provide any suitable means to visualize the utilization of various system resources.

4.1.3 Unique features of our tool set

We present a performance simulation tool which addresses the these aforementioned deficiencies. Our tool adds following new features to those listed in 4.1.1.

- It explores vast combinatorial design space by simulates performance over varying hardware technology and architecture parameters.

- It provides detailed breakdown of resource and performance variables which can estimate the desired level of improvement in the system parameters for chosen design.
- It enables visualization of resource utilization, changing application size and analyzing the performance behavior under different budget constraints.

By using these new capabilities of our tool, we were able to conduct comprehensive analysis of resource-performance trade-offs as function of system parameters for different benchmark applications. The insights obtained from the analysis can be used to direct the endeavor to build scalable quantum computer systems. Before providing the details of our tool, we briefly discuss the specific contributions of prior related work. We then detail different components of our tool box, their integration and unique features that lie at the heart of this thesis.

4.2 Brief Survey of Prior Tools or Infrastructures

One of the very first proposal of tool chain was introduced by Svore et al. (2004). The basic framework of compiling quantum algorithm into technology specific micro instructions were throughly laid out. However, it was Balensiefer et al. (2005a) who featured first complete basic tool which contained a module to compile high level quantum application into equivalent fault-tolerant quantum circuit. The circuit was translated into the sequence of low level quantum micro-architecture instructions for trapped-ion hardware. The performance of circuit execution was evaluated a simulator called QUALE Balensiefer et al. (2005b). One of the major contribution of this work was the efficient calculation of the failure probability of different circuit components without performing full blown quantum state space analysis. The Metodi et al. (2006) investigated the scheduling of quantum gates in the planar ion traps. The tool converted quantum gate into sequence of low-level physical qubit move-

ment operations which were scheduled on the hardware through careful resolution of routing congestion.

The Whitney et al. (2007) specialized in comprehensive the Computer Aided Design (CAD) flow to support automated design and layout of fault-tolerant quantum circuit on trapped-ion hardware. The work pioneered sophisticated modeling of planar ion trap architecture. It recognized that the finding the optimal layout of quantum circuit on the hardware was at least as hard as counterpart problem in classical VLSI design. Therefore it introduced several heuristic based interesting mapping and scheduling techniques which were used for placement and routing of large sized quantum circuits. The layout was optimized for the design Shor's algorithm architecture Whitney et al. (2009). In addition general trade-offs between area and latency (execution time) were studied and guidelines for optimal design were also presented. The Patel (2010) complemented this work by studying greedy heuristics and linear-programming based approximation to optimize the network design component of the quantum computer systems. Both Whitney et al. (2007) and Patel (2010) were utilized to analyze Qalypso architecture Isailovic et al. (2008).

Dousti and Pedram (2012) introduced a tool called Quantum mapper based on Scheduling Placement and Routing (QSPR) for planar ion traps. Using this tool, the overall execution time of quantum circuit was significant minimized (reportedly 41%) compared to prior work Balensiefer et al. (2005b) using (1) improved placement of qubits and (2) advanced shuttling techniques to minimize qubit traffic congestion. The Dousti and Pedram (2013) provided an efficient method of quantifying circuit execution time without explicitly scheduling it on the hardware. The efficient tool called quantum algorithm latency estimator (LEQA) could directly compute the latency from the closed form expression describing critical path of quantum circuit. The tool runtime was one to two orders of magnitude faster than Dousti and Pedram (2012) for range of benchmarks circuits. The SQUASH tool Dousti et al. (2014) ex-

amined mapping problem wherein ancilla qubits could be dynamically shared among fault-tolerant operations. This reduced overall resource overhead at the expense of increased latency. These trade-offs were analyzed using multi-core reconfigurable quantum processor named Requp. This architecture which improved CQLA Thaker et al. (2006) using flexible ancilla allocation in the compute regions. The tool set Dousti and Pedram (2012, 2013); Dousti et al. (2014) was used to optimize the performance of a trapped-ion based universal logical block, capable of performing any logical fault tolerant operation Goudarzi et al. (2014).

The tools described above assumed trapped-ion technology as underlying quantum hardware. The benchmark circuits were encoded using concatenated codes. It should be noted that the tools for technologies other than trapped-ion and have also been developed. For example, a tool called Autotune Fowler et al. (2012b) could accurately compute and optimize failure rate of quantum circuits encoded using topological code, mapped on neutral atoms quantum hardware. In Maslove et al. (2008); Maslov et al. (2007) quantum circuit placement on the Nuclear Magnetic Resonance (NMR) hardware was formalized and a heuristic based algorithm was proposed. However, the analysis was restricted to small quantum circuits with no details of automated tool included in the papers.

We would conclude this section by differentiating performance simulation with actual simulation of quantum circuit which is beyond the scope of this thesis. The actual simulation deals with the efficient management of state space spanned by the quantum system. In principle, an n -qubit quantum circuit requires 2^n complex quantities to be stored and manipulated. This exponential resource requirement hardens the simulation of large quantum circuit on classical computer. A set of interesting methods of simulating general quantum circuit and related tools can be found in Viamontes et al. (2009); Garcia and Markov (2013); Viamontes et al. (2003); Gottesman (1998a); Aaronson and Gottesman (2004). In contrast the performance

simulation described above, evade this problem by only keeping track of variables of interest. These include infidelity of quantum state or the failure probability of physical or logical qubits, and total execution time as quantum circuit is scheduled on hardware.

4.3 Our Design and Performance Simulation Tool Box

The intention of designing a new toolbox is to obtain detailed insights into the key factors involved in resource overhead and performance gains (or degradation) when quantum algorithm is execution on a realistic quantum device. We chose to model MUSIQC as primary hardware for our analysis Steane using $[[7,1,3]]$ error correcting code to generate fault-tolerant circuits. Technologies other than MUSIQC and concatenated codes other than Steane can be supported with slight modification in the basic infrastructure of our tool. The overall flow in the tool chain is given in Figure 4.1

There are two main components in the tool: Tile Designer and Performance Analyzer (TDPA) and Architecture Designer and Performance Analyzer (ADPA). TDPA works in the back end of the tool and simulates the fault tolerant construction of the logical qubit operations using specified Device parameter values. The **Fault Tolerant Circuit Generator** synthesizes the logical operation by specifying physical qubits and sequence of physical gates. The **Tile Builder** constructs an Tile emulating the capabilities of an ELU which acts as basic computational unit to execute fault-tolerant operation. The Tile is designed according to the user specified ratio between different types of qubits such as data, ancilla and communication. The **Low Level Mapper** maps qubits of the circuit to the physical qubits in the Tile and **Low Level Scheduler** generates an order in which gates needed to be scheduled in the circuit. **Low Level Error Analyzer** computes the failure probability of the specified fault tolerant quantum operation based on the output of the. The

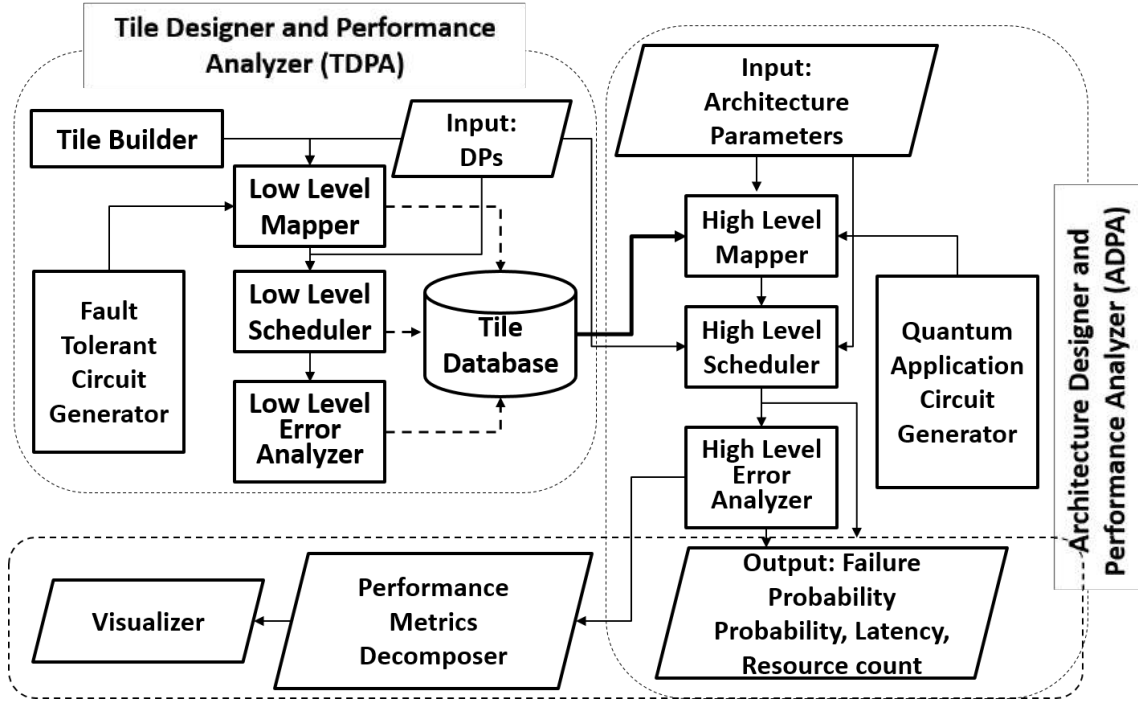


FIGURE 4.1: Different components of our tool and their interconnectivity

Tile parametrized by DP and the performance metrics is stored in **Tile Database**.

ADPA is the front end of the tool that interfaces with the user. It takes user-specified architecture design parameters as inputs and defines architecture by placing and connecting different types of Tiles obtained from Tile Database. These Tiles contain logical operands of the application circuits generated by **Quantum Circuit Generator**. The **High Level Mapper** maps logical qubits to Tiles grouped in large clusters (called Segments) described in chapter 6. **High Level Scheduler** maps application level gates into the sequence of logical gate operation and controls the data movement across the hardware, outputting total execution time (latency) of the circuit and resource count. **High Level Error Analyzer** calculates success probability of the overall scheduled logical circuit. The **Performance Metrics Decomposer** provides the detailed breakdown of performance metrics (such as execution time and failure probability) from scheduled output. Finally, the **Visualizer**

provides pictorial representation of circuit execution indicating utilization of different types of resources. The flexibility of changing device and architecture parameters, the splitting of performance metrics and visual representation of resource usage define prominent characteristics of our tool chain. The rest of the chapter is dedicated to the detailed description of various tool components described above.

4.4 Fault-Tolerant Circuit Generator

It synthesizes an acyclic dependency graph (DG) from the given fault-tolerant quantum circuit implementing a logical operation. The circuit gates are mapped to the nodes while the dependencies among the operations are translated into the directed edges of the graph. The node is specified by a set of following variables.

- Operation name: The name of quantum operation
- Operand qubits: The identifiers of qubit operands
- Start time: The time (in μs) before which the operation can be scheduled
- Execution time: The execution time (in μs) of an operation
- Parent nodes: The list of immediate predecessors of the node
- Children nodes: The list of immediate descents of the node
- Dependency height: The number of nodes in the circuit which cannot be scheduled before this operation

A sample construction of a dependency graph from quantum circuit is shown in Figure 4.2. The graph makes enables for the Low Level Scheduler to simulate the execution of gates in correct order and timing information. When DG is used to model fault-tolerant operation, the nodes act as constituent physical operations which needs

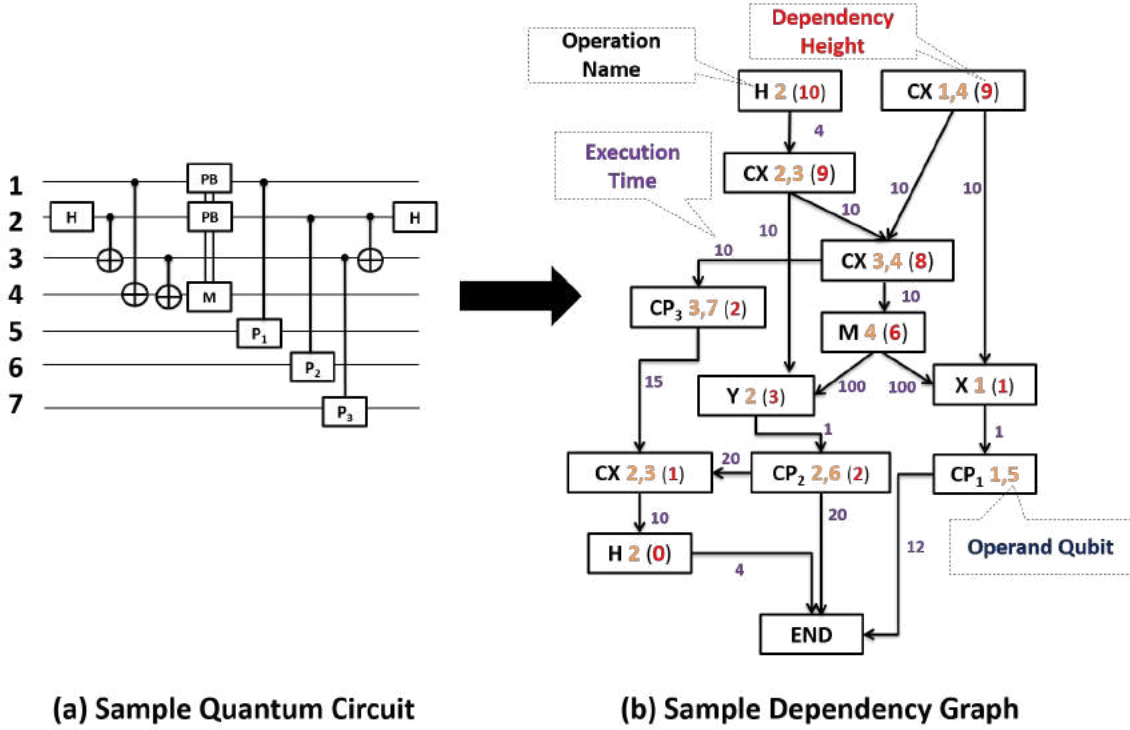


FIGURE 4.2: Converting quantum circuit into equivalent dependency graph. Note that P_A, P_B and P_C represent Pauli gates.

to be properly annotated with additional variables. These new pieces of information defines the role of physical operation in the context of macro operation (such as transversal logical gate, state preparation, syndrome measurement etc.) and they are crucial in avoiding scheduling deadlocks and resource reuse. Thus based on the fault-tolerance needs of DG, we add following new variables to the in the description of node:

- Operand type: Labels qubit operand as data or ancilla
- Operation type: Categorizes physical operation in the context of macro operation.
- Logical qubit: Specifies for which logical qubit is this physical operation constituting the macro operation.

- Syndrome measurement ID: Identifies the measurement whose result registers the syndrome value (+1 or -1)
- Syndrome type: The type of syndrome measurement (X or Z)
- Failure probability: The probability that operation was unsuccessful

We know ancilla qubits play vital role in the fault-tolerant operations. During the execution of sequence of fault-tolerant operation, the ancilla qubits are repeatedly prepared in specific state, interact with data and finally measured. After the measurement ancilla can be re-initialized for similar tasks, thus allowing dynamic allocation. To capture this effect, we make a prototype of ancilla qubits containing two variables to describe its status (i) ancilla context: indicates macro operation which is currently using the ancilla (ii) associated logical qubit: designates which logical qubit macro operation is employing the ancilla. During the scheduling, the variables ancilla context and associated logical qubit of ancilla qubit are compared with variables operation type and logical qubit of the node in order to synchronize the ‘software’ operands with physical qubits. Ancilla status dynamically changes between ‘inuse’ (when occupied by macro operation) or ‘available’ after it is measured. This way Low Level Scheduler can schedule fault-tolerant operation with different number of physically available ancilla qubits in the system.

4.5 Low Level Mapper

The Low Level Mapper assigns qubits in the circuit to the ions in the ELU (or Tile). During this assignment, we want mapper to place frequently interacting qubits ‘closer’ to each other so that communication cost can be minimized. The ‘frequency’ of interaction among qubits depends upon the connectivity of multi-qubit gates in the circuit. We can simplify mapping problem by converting quantum circuit into a

familiar data structure such as that chosen for the Fault-Tolerant Circuit Generator. A quantum circuit can be transformed into an undirected graph called connectivity graph whose vertices correspond to qubits and edges signify two-qubit gates between qubits. The weight of an edge is a positive integer indicating the recurrence of CNOT or CZ between the qubits. The three- or more operands gate, a clique is introduced which increments the weight of edges between operand nodes by 1. The formulation of connectivity graph allows us to solve mapping problem by using wealth of computer science algorithmic techniques. Specifically, in case of MUSIQC architecture, the mapping requires qubits to be first distributed among ions in the ELUs such that the number of CNOT/CZ whose operand are mapped to different ELUs is minimized. The main motivation to minimize inter-ELU gates is that they tend to be resource consuming and time expensive due to non-local nature of communication. Reducing the number of such gates is the first task of Low Level Mapper which can be reduced to the following Qubits Partitioning Problem:

4.5.1 Qubit Partitioning Problem Definition

Qubit partitioning problem accepts connectivity graph $G(V, E)$ and ELUs and their capacities $(\{ELU_i, S_i\}_{i=1}^m)$ pairs as input and asks for distributing vertices V among fixed capacity ELUs (partitions) such that $\forall_{i,j}, V_i \cap V_j = \phi, V_1 \cup V_2 \cup V_3 \cdots \cup V_m = V$ and $\forall_{k=\{1,2,\dots,m\}}, |V_k| \leq S_k$. Note that S_i is the capacity of ELU_i . The objective is minimizing total number of edges $e_{ab} \in E$ across partitions $v_a \in ELU_i, v_b \in ELU_j$ where $ELU_i \neq ELU_j$. These edge crossing correspond to inter-ELU gates. The constraints are sizes of partitions; that each partition (ELU) cannot hold more vertices (ions) than its capacity. It is known that this kind of partitioning problem is NP-hard.

Assuming that once qubits are optimally partitioned, they need to be assigned to specific ions trapped as linear chain(s) inside ELUs. During placement, we want to

minimize the communication cost inside an ELU by maximizing the locality among the qubits. The cost is a function of spacial distance between ions which map the qubits. Regardless of number of ion chains, we can easily leverage QLA Tile abstraction (Chapter 3, Figure 3.2(a)) to model the arrangement of ions inside ELU. We show that solution of placement problem for single chain can easily be used for the multiple chain case as well as for the partitioning problem described earlier. In case of single chain comprising n -equidistant ions (vertices), we can formalize the qubit placement problem as follows:

4.5.2 Qubit Placement Problem Definition

Qubit placement problem accepts connectivity graph $G(V, E)$ to map each vertex v_k to point x_k on a straight line (specified by equidistant neighboring points) such that the quantity $\sum_{i,j} w_{ij} |x_i - x_j|$ is minimized. The points are labeled $1, 2, 3, \dots, n$. The w_{ij} is the weight of an edge between vertex v_i and v_j in G . This problem is also called as optimal linear arrangement problem.

Solution to the Qubit Placement Problem

The qubit placement problem is also NP-hard which means that there is no known polynomial time algorithm which finds optimal solution to the general form of this problem. Therefore, we rely on an approximate algorithm Andreev and Racke (2006) providing ‘approximately optimal’ solution efficiently. The Table 4.1 describes graph-theory algorithm for the qubit placement problem. It converts adjacency matrix representation of the connectivity graph $G(V, E)$ into corresponding Laplacian matrix $S(G)$ which is positive semi-definitive. When diagonalized, the spectrum of $S(G)$ produces non-negative eigenvalues $\lambda_i \geq 0$. We obtain eigenvector V_2 corresponding to the second smallest eigenvalue λ_2 . Index the components of V_2 using vertex identifiers x_i for $i = 1, 2, 3, \dots, n$. Thus, we can associate $V_2[1]$ with vertex 1,

Table 4.1: Qubit Placement Algorithm Steps

1. Obtain Laplacian matrix $S(G)$ of the adjacency matrix representation of Connectivity Graph $G(V, E)$.
2. Diagonalize $S(G)$. Let $\lambda_1(G) \geq \lambda_2(G) \geq \lambda_3(G) \dots \geq \lambda_n(G)$ are sorted n eigenvalues.
3. Find eigenvector V_2 corresponding to second smallest eigenvalue $\lambda_2(G)$.
4. Index V_2 by vertex number x_i for $i = 1, 2, 3, \dots, n$.
5. Sort V_2 to obtain V'_2 . For each x_i , find new index j by comparing $V_2[i] = V'_2[j]$.
6. Place vertex x_i at location j on the line.

$V_2[2]$ with vertex 2 and so on. Finally, the placement position of vertex x_i is the index of $V_2[i]$ in sorted V_2 . The time complexity of the algorithm is dominated by the computation of eigenvector V_2 which can be done in polynomial time (in the number of vertices n). Even though Andreev and Racke (2006) did not provide the theoretical approximation of solution optimality, based on our experience, the algorithm generally outputs sufficiently adequate qubit placement when tested on range of quantum circuits. The algorithm also runs fast in practice. A sample application of this algorithm is shown in Figure 4.4.

In case of multiple chains, we can first map qubits to ions in the single chain and then fold the chain into a 2-D serpentine shape, see Figure 4.3. The philosophy of this technique lies in the observation that folding the chain in this manner preserves the locality among the qubits. To see this, define spatial distance between mapped qubits as the number of intermediately placed qubits in a 2-D grid. We find that spatial distance between qubits can only reduce when we when the chain is folded onto the grid. Hence the algorithm in Table 4.1 can easily be extended to multiple chains. To solve qubit partitioning problem, we note that the one long chain mapping

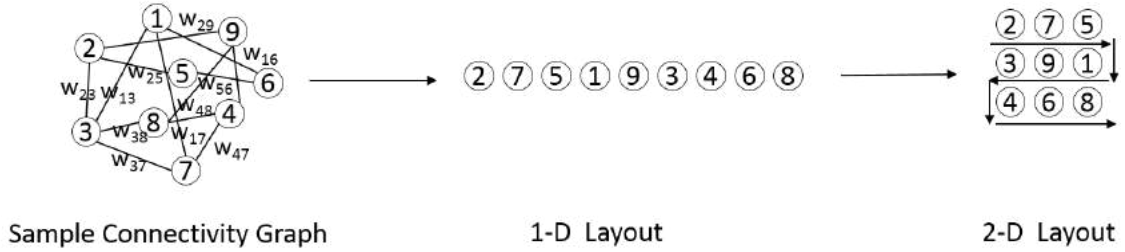


FIGURE 4.3: Demonstration of folding 1-D Chain onto 2-D grid to map multiple chains in ELU

all the qubits can be split into smaller chains, one for each ELU. Thus if the capacity of each ELU is k qubits, we can partition the chain into adjacent $\lceil \frac{n}{k} \rceil$ sub-chain partitions similar to that in Figure 4.3. However, the number of resulting inter-ELU gates can increase if the weight of an edge between qubits located at partition border is large. Fortunately, a careful investigation reveals that pair of qubits never interact ‘too frequently’ in most quantum circuits of our interest. Therefore, the edge weights are generally bounded by some small integers which allows us to employ the efficient partitioning solution with negligible loss of optimality. Hence Table 4.1 algorithm solves both partitioning and placement problem.

One of the distinguishing feature of our tool is the ability to simulate quantum circuit containing more qubits than the physical ions present in the system. So far we have assumed one-to-one mapping between qubits and ions. This is assumption can be satisfied for data qubits which generally stay ‘alive’ till the end of computation. On contrary, the reusable qubits (ancilla or communication) are ‘short lived’ because they are consumed rapidly. For example, the ion allocated for an ancilla qubit can easily map another ancilla once the first ancilla is measured. Although dynamic allocation of reusable qubits can be handled during the scheduling, however, it invalidates one-one-mapping assumption when available of ions are fewer in number than the reusable qubits in the quantum circuit.

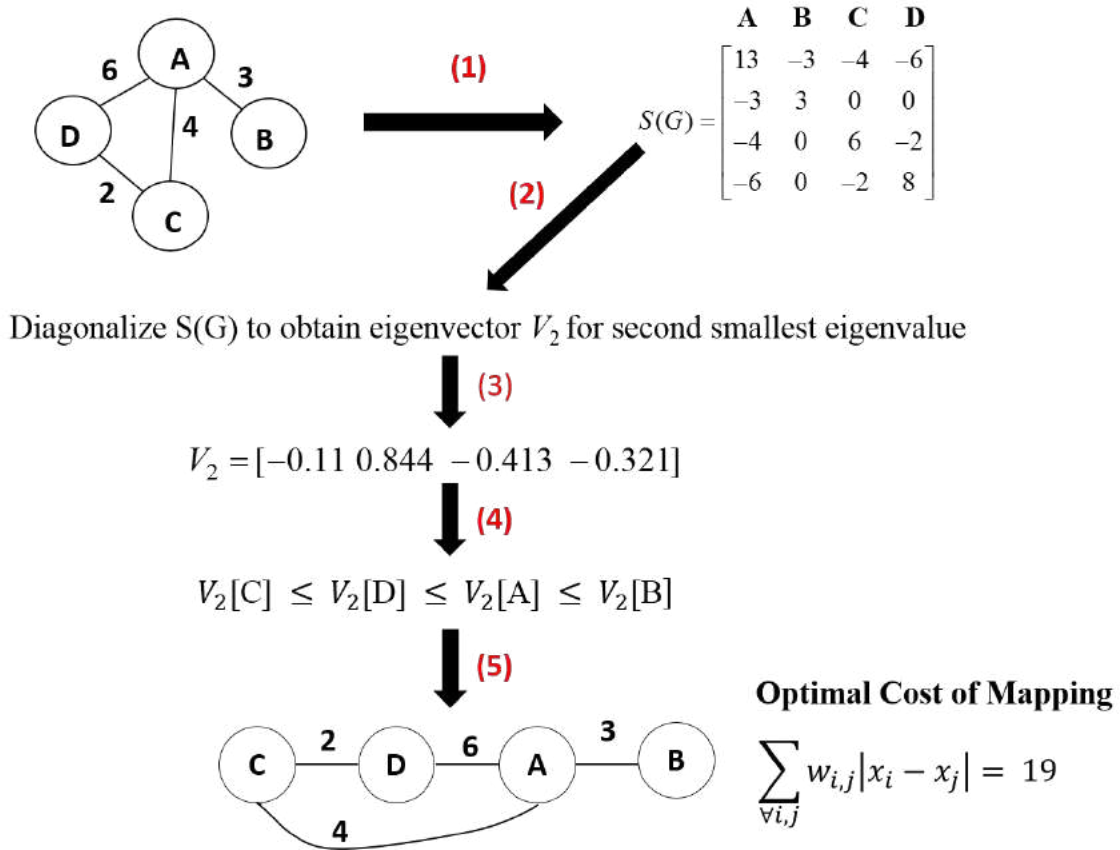


FIGURE 4.4: Demonstration of solution to qubit placement problem

For a hardware containing m ions dedicated to map k reusable qubits, then for $k > m$, there are at least $\frac{k}{m} P_m^k$ possible mappings $f(k) \rightarrow m$. Instead of searching through all possible mapping functions, Low Level Mapper can learn the pattern in which reusable qubits are utilized and interact with data qubits during fault-tolerant operation. If this pattern can be learned by running the Low Level Scheduler without considering physical qubit map. During this run, the scheduler will dynamically allocate k reusable qubits to m hardware ions, thus providing us with unique $f(k) \rightarrow m$. Once this auxiliary run of scheduler is complete, the mapping function is obtained to enable Low Level Mapper construct the correct connectivity graph whose nodes represent physical ions instead of circuit qubits. This new graph is called ‘physical

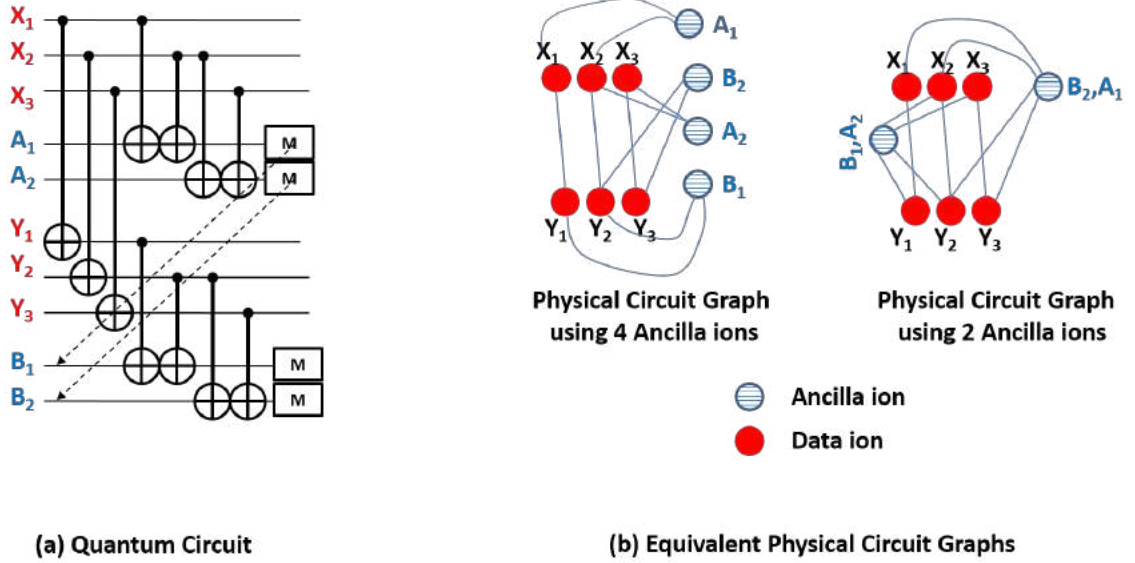


FIGURE 4.5: An example circuit shows how one quantum circuit generates different physical circuit graphs for different physical ions dedicated to ancilla qubits

circuit graph’. Figure 4.5 shows that same quantum circuit generates two different physical circuit graphs depending upon different number of ancilla ions to map ancilla qubits. The physical circuit graph is then used by qubit partitioning and placement algorithms to complete the role of Low Level Mapper.

4.6 Low Level Scheduler

Scheduler takes mapped hardware and generates the execution order of gates in the given quantum circuit by outputting the time T_{start} at which the gates is begins execution. The time T_{finish} at which gate execution ends is the sum of its execution time T_{exec} and T_{start} . The main challenge of scheduling lies in dynamic allocation of reusable qubits and hardware-imposed constraints on the execution of gates. A typical set resource constraints considered in the Low Level Scheduler includes limited number of:

1. Lasers

2. Ancilla ions

3. entangling ions

The number of available lasers bounds the concurrently executable gates, fewer ancilla ions increases the latency of fault-tolerant operation while small number of entangling ions will reduce the bandwidth of quantum communication channel.

For scheduling, a dependency graph is generated from the fault-tolerant quantum circuit. The structure of the graph ensures the correctness of circuit scheduling. A directed edge from node i to node j in the graph means that gate j of circuit cannot be scheduled before gate i is executed. Our Scheduler in Algorithm 1 works in an iterative way: In each iteration, a set of mutually independent nodes(gates) is selected for execution and removed from the graph, along with their outgoing edges. These nodes had no parent nodes and their removal would pave the way for their children nodes to be scheduled in subsequent iteration. After satisfying gate level dependencies, the nodes are further short-listed due to insufficient hardware resources to schedule all selected gates. The gates are screened for their usage of entangling ions, followed by their consumption of ancilla ions. When resources are fewer than candidate gates, the ones with higher dependency height are selected. The chosen gates are then dispatched and their respective order of execution is determined by constraints on lasers. This is similar to classical instruction processing wherein processor maintains reorder buffer and incoming instructions are queued until their operands are available. Here, the operands correspond to qubits and number of slots in reorder buffer which models the sets of lasers in our case.

The dynamic resource allocator within the scheduler is responsible for deadlock-free scheduling through careful management of resource assignment and release. The most frequent tasks encountered in scheduling are fault tolerant protocols largely containing quantum error correction. It often requires a minimum number n_{min} of

ancilla qubits to schedule main subtask of syndrome calculation, which is composed of three steps. In the first step, it needs fresh ancilla prepared in an appropriate state (*e.g.*, a maximally entangled state of several qubits), which amounts to resource assignment. In the second step, ancilla interact with data to obtain the signature of the error. In the last step, ancilla are decoded and measured to extract final error information. Once measurements are performed, ancilla qubits are released and can be reused for another syndrome calculations or other tasks. The syndrome calculations subtasks can be scheduled in parallel if sufficient ancilla qubits are available, or (partially) in series otherwise. When fully parallel scheduling is infeasible due to limited availability of ancilla qubits and the resource allocator distributes ancilla such that each one is assigned less than n_{min} ancilla, the schedule can potentially encounter deadlock because none of the subtasks can be completed and no allocated ancilla can be freed for other (sub)tasks. Therefore, a smaller quantum computer can suffer from scheduling deadlock due to inefficient resource allocation.

To address this issue, our resource allocator follows ‘all or nothing’ strategy, where each subtask is scheduled only when enough ancilla are available for its completion. In addition, scheduling of operations within each subtask is prioritized to facilitate quick ancilla release. This is achieved by assigning highest priority to the measurement since they free qubits to be used by other tasks. The same strategy is applied to entangling ions resource management, except that in case of constraint, the associated tasks are operations local to ELU for each non-local CNOT. J. Eisert and Plenio (2000) and can be scheduled with only one entangling ion per ELU. On the contrary, the laser constraint does not create any deadlock since unlike quantum resources which are freed only after qubit measurement of qubits, the classical resource laser can be released and reused once the current operation (whether it is a gate, preparation or measurement) is completed.

or memory) is the possibility that it introduces a bit-flip error (X) error, phase-flip (Z) error or both (Y error) in the operand qubit(s). For our analysis, we assume depolarizing channel as our underlying error model in which all X , Y and Z error occur independently with same probability p . The logical failure probability of a noisy logical quantum component (physical operation or memory) is the possibility that it introduces a logical bit-flip error, phase-flip error or both (logical Y error) in the operand qubit(s). The logical error is established if the error correction step fails to correct the errors in the operand logical qubit.

The fault counter method computes the logical failure probability of the operation from the physical failure probability by counting the number of ways in which physical errors propagation resulting in the logical failure. A fault tolerant circuit block constructed for an error correcting code of distance d , can perfectly recover from k errors occurring anywhere in the block if $k < d$. However for $k > d$, errors propagation can result in the incorrect logical state of the qubit with probability $\mathcal{O}(p^k)$. Since our analysis is based on distance-3 Steane $[[7,1,3]]$ code, events containing two or more errors cause the logical failure should be counted. However, when p is small (we assume $p = 10^{-7}$ in our chapter 6,7 simulations), the higher order terms in the logical failure probability $\mathcal{O}(p^3)$, $\mathcal{O}(p^4)$, ... can be neglected at the cost of nominal inaccuracy. Hence fault counter simply counts number of failures C , resulting from exactly two-errors events.

Based on our error model, we define two-error event by introducing a combination of Pauli X or Z gate at two different locations in the circuit block. Since each error models physical fault, the probability of a two-error event is p^2 . We need to test whether such event corrupts the logical state of the qubit as these errors propagate through the circuit. The error propagation is equivalent to commuting the corresponding error Pauli gates through the constituent circuit gates according to the Table 2.4. The logical failure is detected when the logical qubit acquires

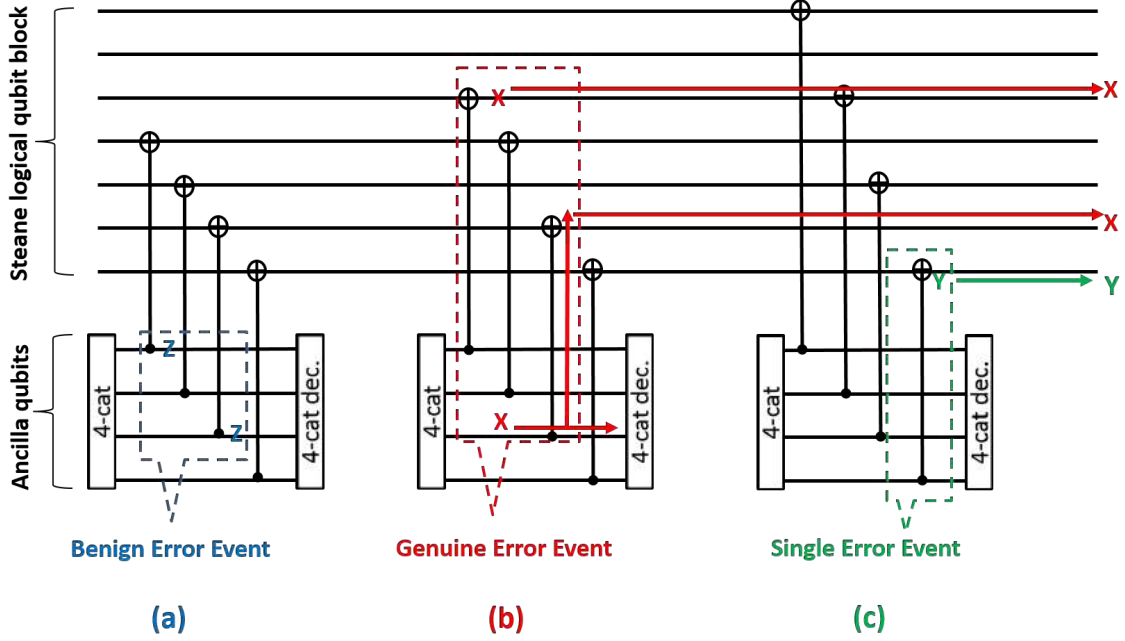


FIGURE 4.6: An example demonstration of fault counter method to distinguish between benign error event, genuine error event and single error event.

uncorrectable error combination at the end of propagation. If a two-error event results in the logical failure it is termed ‘genuine error event’, otherwise, it is labeled ‘benign error event’. An example distinction between genuine and benign errors is shown in Figure 4.6(a,b). This procedure is repeated for all combinations of two-error events each occurring at $\binom{n}{2}$ locations for a circuit block containing n possible fault sites. This way, the fault counter computes total number of genuine error events C causing logical failures and outputs logical failure probability as Cp^2 .

During the execution of the fault tolerant quantum algorithm, single-qubit errors occurring in different a fault tolerant operation can also accumulate to produce logical failure in the logical qubit Q . For example, if Q undergoes sequence of gates: $H_L X_L S_L Z_L$, a bit-flip error occurring in H_L and Y error arising in X_L can lead to the logical error in Q when these error fall on different constituent physical qubits. Such error-event can be accounted for by tracking the probability that fault-tolerant

block produces single qubit error in the logical qubit. We call this Block Physical Failure Probability (BPFP). In order to accurately model the error propagation, for each logical qubit we keep track of its logical failure probability as well as Block Physical Failure Probability.

The basic philosophy behind recording BPFP is that a logical qubit containing single qubit error can be considered as ‘logically correct’ (since single error can be corrected by error correction) but ‘physically incorrect’. To illustrate this concept, consider an imperfect transversal gate which can introduce a single qubit error into perfect logical qubit. As a result the BPFP level is raised for the operand logical qubit. The acquired single qubit error can be corrected in subsequent error correction steps which amounts to the reduction in BPFP. During the execution of quantum algorithm, regular error correction on logical qubit ensures that the accumulated physical error probability remains within acceptable limits. Therefore by controlling BPFP, error correction preserves the correct state of the logical qubit. On contrary when error correction is scarcely executed, BPFP may grow to dangerously large value, resulting in the logical failure. Thus BPFP is translated into logical failure probability when error correction projects logical qubit into the code space. After error correction is performed, BPFP is reset to initial value precomputed by the fault counter. The BPFP can easily be computed by considering all single-error event in the circuit block which result in single qubit error in the logical qubit. An example of single-error contributing to BPFP is shown in Figure 4.6(c). It should be noted that only error correction and operator measurement operations reset BPFP value by explicitly translating BPFP into the logical failure probability. All other fault-tolerant operations monotonically increase its value since they are not entitled to interpret the logical state of the qubit.

The main advantage of BPFP will be highlighted when we explain how High Level Error Analyzer calculates the overall failure probability of the entire quantum

algorithm for different insertion configurations of error correction. For now, we conclude this section by explaining the implementation details of Low Level Analyzer. The main task of this tool component is to identify error events which propagate to logical qubit block. Since fault-tolerant operation for Steane code are made of Clifford group gates, we can easily track error propagation by commuting corresponding Pauli operators with the circuit gates. A frame of Pauli errors is constructed after propagation through each gate, at the same time we also register the set of stabilizers whose joint +1 eigenvalue eigenvector describe the quantum state of the circuit after each gate. After each gate, we test if the updated pauli error frame falls into the stabilizer space, in which case the event is termed benign, otherwise the propagation continues. The propagation is also terminated if an error is caught during cat-state verification used in operator fault-tolerant operator measurement. The complete description of fault counter testing two-error event is provided in Algorithm 2.

4.8 Tile Database

Tile data base stores the resource and performance parameters of an ELU defined in the general framework of Tile (see section 3.2.2 for details). These parameters describe the qubit resource investment and the quality of the fault-tolerant operation localized to the Tile. These are listed as follows:

1. Set of supported logical operations
2. Layers of encoding
3. Area (total QLA cells) and dimensions
4. Total physical qubits
5. Operation execution time

6. Operation logical failure probability

7. Operation block physical failure probability (BFPF)

The parameters (1)-(4) come from user specified design while (5)-(7) are calculated from computed from Low Level Mapper, Scheduler and Error Analyzer. When multiple layers of encodings are used, the resource and performance numbers can be calculated by either direct flattening of the fault-tolerant circuit or hierarchical evaluation of higher layer Tile parameters from precomputed lower layer Tiles parameters. The hierarchical method is useful in two ways: first it modularizes the construction of higher layer Tiles by assembling different types of lower layer Tiles. Second, it provides a convenient framework to optimize resource overhead by (i) allocating reusable qubit at each layer according to its computational load (ii) assembling lower layer Tiles to perform multiple tasks for Tiles at higher layer of concatenation. The framework of cross-layer optimization is an integral part of our tool, which will be detailed in chapter 7. Once adequate Tiles are designed and parametrized, they are stored in the Tile Database. It should be noted that Tile object stored is associated with assumed values of device parameters which determine its performance. Multiple Tile objects with varying performance levels can be created and stored for different sets of user specified device parameters. These Tiles can be assembled and connected in several different ways to define variety of architectures to graph algorithm level circuit in ADPA.

A parametrized Tile provides a macro-model to define quantum architecture which can schedule application level quantum circuit. The logical qubit of the encoded circuit are mapped to the block of physical qubits inside the Tile and logical gates are translated into the functional roles of the Tiles. In chapter 6,7 we use Tile macro to abstract trapped-ion hardware unit containing one application level qubit which is an operand of a gate, application level ancilla for error-correction or

magic-state preparation or an e-bit of logical EPR pair to non-local logical gates. These Tiles generally contain sufficient physical qubits which can be used to execute desired set of fault-tolerant quantum operations. In general one can define multiple types of Tiles, each performing different functional role, to allow wider range of building blocks in order to optimize resource consumption and the performance of quantum application. An example of such optimization is described in section 4.12.1 and utilized in the study of chapter 7.

Even though, we use Tile to abstract trapped-ion hardware in this thesis, we wish to point out that Tile can also encapsulate other technologies such as superconductors, quantum dots and neutral atoms to define corresponding application level quantum architecture. Once technology specific Tiles are performance parametrized based on device level implementation of the logical qubit, we need to specify mechanism of communication among Tiles to allow application level logical gates whose operands are contained in different Tiles. Therefore Tiles are needed to be connected by the communication channel applicable to given technology (e.g., ballistic shuttling for trapped-ion, successive swapping for superconductors, optical interconnects for photons). The communication component of the architecture can be modeled using dedicated ‘Communication Tiles’ (see chapter 6,7) or adding ‘Port’ to the Tile attributes which interface one Tile with another. Once adequate interface to connect different Tiles is appropriately modeled, application level quantum architecture can be completely defined for the chosen device technology to execute quantum algorithm.

4.9 Quantum Application Circuit Generator

The Quantum Application Circuit Generator leverages the dependency graph formulation used in Fault Tolerant Circuit Generator (see Figure 4.2). The quantum algorithms are constructed from variety of complicated gates which needs to be de-

composed into universal set of gates whose fault-tolerant implementation is known. The decomposition allows us to architect hardware using Tiles implementing these familiar constituent fault-tolerant operations. For example quantum arithmetic circuit widely use Toffoli gates which need to be converted into the fault-tolerant circuit of Figure 2.17. Similarly, another famous benchmark application Quantum Fourier Transform uses small rotations about Z -axis of Bloch sphere $R_Z(\theta)$ which is generally translated into sequence of T and single qubit Clifford gates Kliuchnikov et al. (2013); Fowler and Hollenberg (2004).

In the dependency graph the nodes which model complicated gates are replaced by the sub-dependency graph to include the equivalent fault-tolerant circuits required for complete scheduling. This replacement is performed dynamically during the high level scheduling as explained in Section 4.12. Using the dependency graph abstraction, Quantum Application Circuit Generator models necessary structural component of application level circuit to ensure correctness of scheduling.

4.10 Brief description of basic Architecture model in ADPA

Before describing various components of ADPA, we introduce base architecture analyzed in the simulations. We assume large size ELUs called Segments connected by optical switch. Each Segment consists of several QLA Tiles which map logical qubits (data, ancilla or communication). The qubits across multiple Tiles in the same Segment communicate through Ballistic Shuttling Channels while those across Segments using Entanglement Channel. For multi-qubit transversal logical gate, the operand logical qubits are brought into the same Tile wherein the gate can be performed locally. Thus considering Toffoli as the bottleneck, we provide space to accommodate at least three logical qubits. Each Segment contains dedicated ancilla to perform error correction locally while cross-Segment qubit movement is arbitrated by the dedicated Communication Tiles. Certain Segments contain additional magic

state Ancilla Tiles to perform non-Clifford Group gates (T or Toffoli). In this case operands need to be first teleported into the designated Segment before the gate is scheduled. For cross-Segment CNOT, the operands stay in the same segment as the non-local gate can be implemented without operand teleportation J. Eisert and Plenio (2000). However, in case of Toffoli gate, we require all operand qubits in the same designated Segment which can lead to multiple cross-Segment teleportations. Each Communication Tile has dedicated ancilla which perform error correction to enhance the fidelity of non-local operation. Finally, unless there is extra space available, the Data logical qubit teleported to new Segment is swapped with the resident Data logical qubit. Thus cross-Segment transfer includes two teleportations steps to implement non-local swaps.

4.11 High Level Mapper

The High Level Mapper uses the Qubit Placement Algorithm described in section 4.5.1. It maps application circuit qubits (data logical qubits) to the Tiles in the Segment so that the cost of long distance cross-Segment communication is minimized. The High Level Mapper generates initial map of the logical data qubits which may be later teleported to different ELUs for (i) non-local multi-qubit operation (Toffoli gates) (ii) special Segments containing dedicated ancilla for magic-state preparation (such as for T gate). Even though these logical qubits keep moving across the Segments, the initial map plays critical role in reducing the impact of the long-distance communication latency as well as number of teleportation across ELUs. For example in Figure 4.7, the initial distribution of data logical qubit in the Segments obtained from qubit placement algorithm, will bear significantly lower cross-Segment movement than that of random distribution. Once initial map obtained, it is fed to the High Level Scheduler.

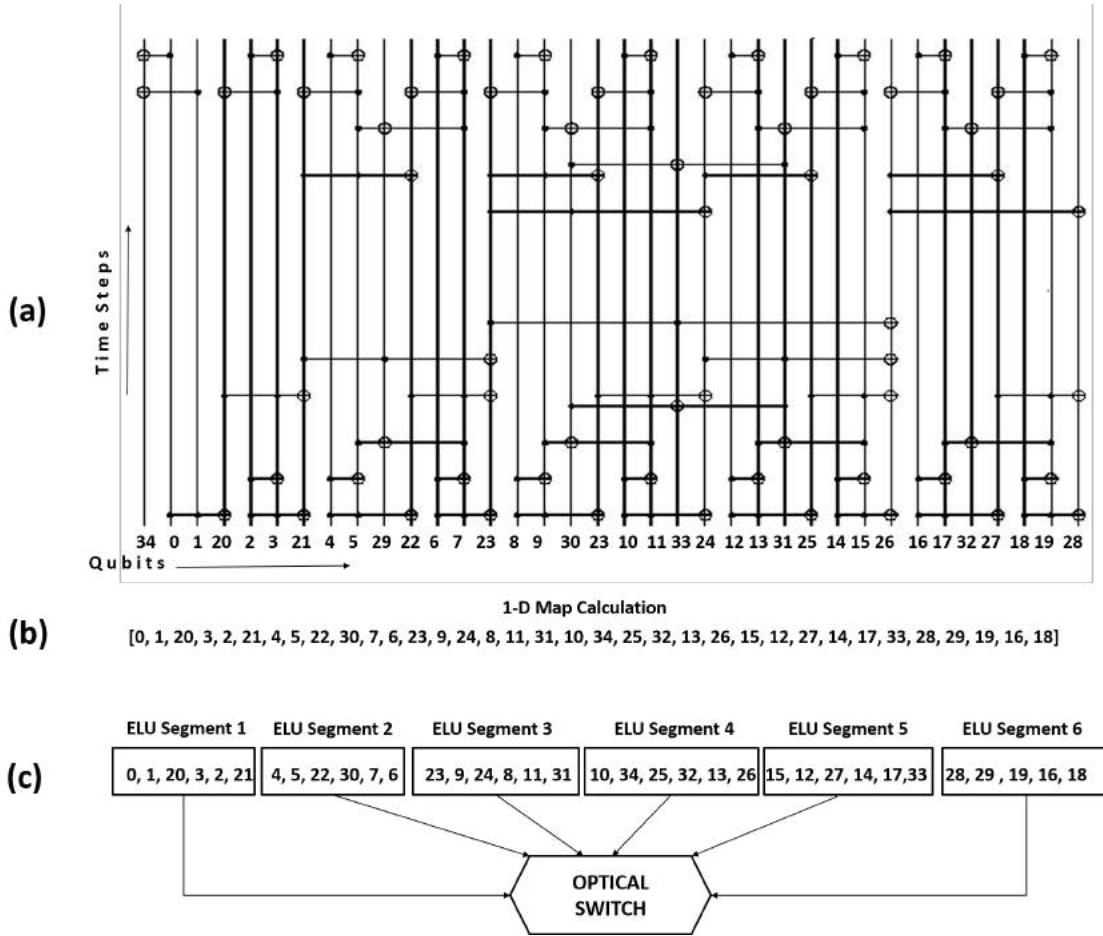


FIGURE 4.7: The demonstration of the role of High Level Mapper. The qubits of sample quantum algorithm circuit shown in (a) mapped onto a line (b) using Algorithm in Table 4.1. These qubits are mapped to Tiles in Segments (c).

4.12 High Level Scheduler

The High Level Scheduler generates the correct sequence of logical gates in the application circuit, uses ASAP heuristic to minimize the total execution time. The Quantum Application Circuit Generator provides circuit description as dependency graph, while the initial map of operand qubit is obtained from the High Level Mapper. Using these inputs, High Level Scheduler implements Algorithm 3 for scheduling. For this tool component, there are two important tasks. First the dynamic allocation

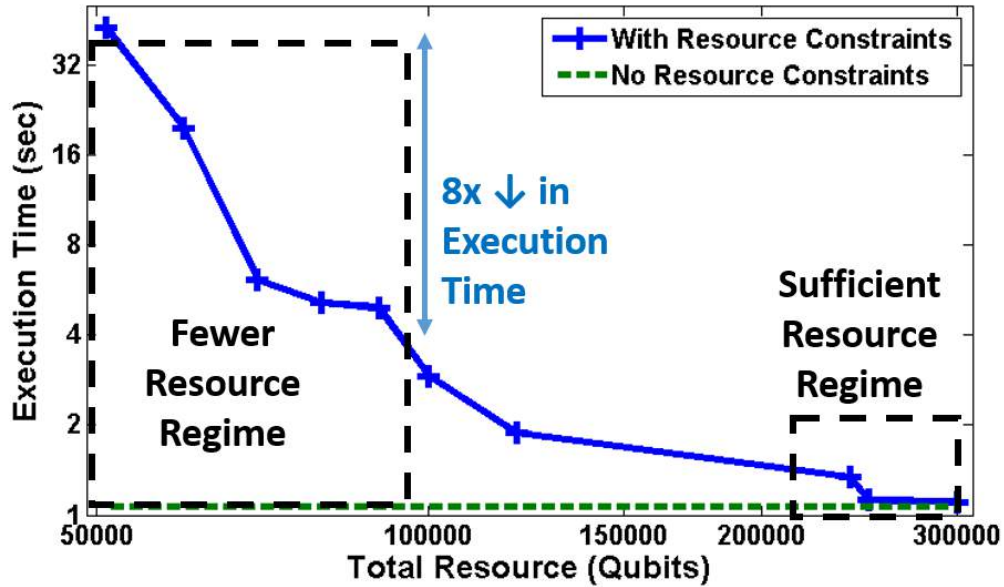


FIGURE 4.8: Optimality test of the Scheduler by plotting the execution time of 64-bit Quantum Carry Look-Ahead Adder (QCLA) circuit against the available hardware resources (qubits). The solid line shows resource constrained circuit execution time while dotted line shows circuit execution time with no constraints on resources. The resource axis is divided into fewer resource regime and sufficient resource regime. The optimality of the Scheduler can be confirmed by noting the desired behavior shown by the solid execution time curve in the two regimes.

of Segment resources to perform different types of logical operations. Second, track availability of operands before which gate cannot be scheduled. Several resource constraints can delay the scheduling which include:

1. Ancilla Tiles for error correction
2. Ancilla Tiles for magic state preparation
3. Communication Tiles for cross-Segment teleportation
4. Communication Tiles for cross-Segment CNOT
5. The congestion in the Ballistic Shuttling Channel for intra-Segment communication

Denote T_{start} the time (in μs) at which gate can be originally scheduled without considering above constraints, then depending upon the above mentioned constraint T_{start} is increased to include waiting time in which all operands are ready to initiate gate execution. The reusable resources such as Ancilla and Communication Tiles are annotated with T_{avail} : the time at which these resources can be initialized for their usage in next logical operation. In case of error correction and magic state preparation, the Scheduler allocates subset of Ancilla Tiles which has available at the earliest. In other words, if time available for N Ancilla Tiles can be ordered as: $T_{avail}^{A_1} \leq T_{avail}^{A_2}, T_{avail}^{A_3}, \dots, T_{avail}^{A_k}, \dots, T_{avail}^{A_N}$, it will choose first k Ancilla Tiles $A_1, A_2, A_3, \dots, A_k$. In the same way, the Communication Tiles for cross-Segment teleportation or CNOT are chosen. In this case it chooses two Communication Tiles, one from each Segment with available times $T_{avail}^{C_1}$ and $T_{avail}^{C_2}$ respectively. Then, time before which these Tiles cannot generate mutual EPR pair will be $T_{avail}^{C_1, C_2} = \max(T_{avail}^{C_1}, T_{avail}^{C_2})$. We describe mathematical expressions for the operand available times T_{avail} for different types of operation. By convention D is used for Data, A for Ancilla and C for Communication Tile. The expressions will use following latency variables whose symbol and description is listed as follows:

- $T_{BShutt}^{X,Y}$: Time taken to ballistically shuttle logical qubit contained in Tile X to the space available in Tile Y
- $T_{EPR}^{V,W}$: Time taken to prepare EPR pairs Communication Tile V and W located in different Segments
- $T_{LOCC}^{D,C}$: Time taken to perform local operation and classical communication to implement non-local CNOT or teleportation. Since we ignore the cost of classical communication, this latency mainly includes time to perform transversal CNOT between logical Data in D and logical e-bit contained in C followed by the logical measurement on the e-bit

- $T_{EC_{EPR}}$: Time taken to execute error correction in the Communication Tile
- T_{EC}^D : Time taken to execute error correction on the logical data qubit in D .

The same Segment CNOT between Data D_1 and D_2 :

$$T_{avail}^{D_1} = T_{avail}^{D_2} = T_{BShutt}^{D_1, D_2} \quad (4.1)$$

The same Segment CNOT between Data D and Ancilla A :

$$T_{avail}^D = T_{avail}^A = T_{BShutt}^{D, A} \quad (4.2)$$

The same Segment CNOT between Data logical qubits A_1 and A_2 :

$$T_{avail}^D = T_{avail}^A = T_{BShutt}^{A_1, A_2} \quad (4.3)$$

The Cross-Segment CNOT between D_1 of S_1 using C_1 with D_2 of S_2 using C_2 :

$$T_{avail}^{D_1} = \max(T_{avail}^{C_1, C_2}, T_{BShutt}^{D_1, C_1}) + T_{EPR}^{C_1, C_2} + T_{EC_{EPR}} + T_{LOCC} \quad (4.4)$$

$$T_{avail}^{D_2} = \max(T_{avail}^{C_1, C_2}, T_{BShutt}^{D_2, C_2}) + T_{EPR}^{C_1, C_2} + T_{EC_{EPR}} + T_{LOCC} \quad (4.5)$$

$$T_{avail}^{C_1} = T_{avail}^{C_1, C_2} + T_{EPR}^{C_1, C_2} + T_{EC_{EPR}} + T_{BShutt}^{D_1, C_1} + T_{LOCC} \quad (4.6)$$

$$T_{avail}^{C_2} = T_{avail}^{C_1} \quad (4.7)$$

Cross-Segment teleportation swap between D_1 from Segment S_1 and D_2 of Segment S_2 . Assuming S_1 selected C_1, C_3 and S_2 selected C_2 and C_4 Communication Tiles:

$$T_{avail}^{D_1} = \max(T_{avail}^{C_1, C_2}, T_{BShutt}^{D_1, C_1}) + T_{EPR}^{C_1, C_2} + T_{EC_{EPR}} + T_{LOCC} + T_{BShutt}^{C_2, D_2} \quad (4.8)$$

$$T_{avail}^{C_1} = T_{avail}^{C_1, C_2} + T_{EPR}^{C_1, C_2} + T_{ECEPR} + T_{BShutt}^{D_1, C_1} + T_{LOCC} \quad (4.9)$$

$$T_{avail}^{C_2} = T_{avail}^{C_1} \quad (4.10)$$

$$T_{avail}^{D_2} = \max(T_{avail}^{C_3, C_4}, T_{BShutt}^{D_2, C_4}) + T_{EPR}^{C_3, C_4} + T_{ECEPR} + T_{LOCC} + T_{BShutt}^{C_3, D_1} \quad (4.11)$$

$$T_{avail}^{C_4} = T_{avail}^{C_3, C_4} + T_{EPR}^{C_3, C_4} + T_{ECEPR} + T_{BShutt}^{D_2, C_3} + T_{LOCC} \quad (4.12)$$

$$T_{avail}^{C_3} = T_{avail}^{C_4} \quad (4.13)$$

Using these equations High Level Scheduler runs Algorithm 3 to order the execution of gates by setting their start time: T_{start} . The Scheduler also registers important component of delays that increases T_{start} for each gate which are listed as follows:

- Ancilla Delay (D_{ANC}): Delay due to the magic state preparation
- Shuttling Delay (D_{SHT}): Delay due to the transportation of operand qubits of the gate, through Ballistic Shuttling Channel inside the Segment
- Tel Delay (D_{TEL}): Delay due to the EPR pair generation for communication
- Cross-Seg-Swap Delay (D_{SWP}): Delay due to the cross-segment swapping

These delays are fed to Visualizer and Performance Metrics Decomposer which provide insight into the breakdown of execution time. The output of Scheduler is received by the High Level Error Analyzer to computer logical failure probability of entire quantum algorithm. We conclude this section by discussing how High Level Scheduler deals with multiple layers of concatenation.

4.12.1 *Dynamic Resource Allocation in Cross-Layer Scheduling*

The Figure 4.9 shows High Level Scheduler dynamically allocating resources for scheduling across multiple layers of concatenation. For quantum operation, denote L1 as first layer and L2 as second layer of concatenation. The L2 Toffoli magic state preparation requires two resource consuming steps (1) L1 Toffoli gates and (2) L2 $|0\rangle_L$ error correction *EC*. Both these functions will be performed by group of L1 Ancilla Tiles. There are two-types of L1 Tiles. Seven Type-1 Tiles are assembled to form one L2 Tiles which hold logical qubit (or L1 7-cat) of the Toffoli preparation circuit. The Type-2 Tiles are dynamically allocated to perform multiple tasks. First, 28 Type-2 Tiles are used to generate seven L1 Toffoli magic states. Once consumed by the L1 Toffolis, they can be reinitialized to perform L2 Error Correction for the L2 Tiles.

It is also interesting that for an architecture of Figure 4.9, when macro model is employed to abstract an L2 Tiles, the schedule for L1 operations spanning across multiple L2 Tiles, can easily be translated into schedule at L2. In this example, to generate schedule of an L1 gate (e.g., L1 CNOT or transversal Toffoli) whose operands are stored in different L2 Tiles, we can use Shuttling Delay between the corresponding L2 Tiles to adjust the T_{start} of the L1 gate. By designing appropriate resource architecture, we can avoid flattening an higher layer Tile to zoom into lower layer Tiles operations. This way the High Level Scheduler runs efficiently by performance-simulating the quantum circuit only at the top layer of concatenation.

4.12.2 *The Correctness and the Optimality of the Scheduler*

We would conclude this section by analyzing different ways in which the output of (both High and Low Level) Scheduler can be tested for correctness and optimality.

Correctness

To verify that Scheduler generates correct sequence of gates, we can examine its output which contains time instances at which each gates is scheduled for execution. This output can be compared with the ordering and dependencies among gates specified in the original quantum circuit to validate the correctness. This fine-grained verification approach is feasible for smaller circuits containing fewer gates so that entire schedule can be fully read out compared with the circuit level gate connectivity. As circuit size grows larger, we rely on more holistic approach which utilizes the output of Visualizer or analyzes the performance trend of the circuit as a function of application size. The former method is explained in section 4.15 containing several examples in which patterns of resource utilization is compared with the gate sequence in the circuit to validate the Scheduler. In the latter method, we schedule application circuit on a hardware containing sufficient resources and analyze the scaling of its execution time with the application size. The matching between hardware mapped execution time scaling with that predicted in the theory, allows us to certify the correctness of the Scheduler. Using this method we verified our Scheduler for the range of large size applications circuit by showing resource-performance scalability of the quantum architecture in section 7.5.1.

Optimality

When computational resources are limited, the job of our Scheduler is reduced to the general classical resource constrained scheduling problem known to be NP-hard. This means that in principle we cannot guarantee if the Scheduler outputs optimal sequence of gates within the resource constraints such that total execution time of quantum application circuit is minimized. However, despite the fact that rigorously proving optimality is tough task, we can test our Scheduler for ‘approximate’ optimality by comparing its performance in the selected resource-performance space

of our interest. In order to expose any sub-optimality, the Scheduler is tested for highly resource intensive quantum application. Once appropriate benchmark is selected, we analyze its execution by scheduling it on the hardware for varying resource investment categorized into following regimes:

1. Fewer resource regime
2. Sufficient resource regime

In the fewer resource regime, we want Scheduler to maximally utilize resources to lower the execution time. Ideally Scheduler should feature an x -fold (or more) reduction in execution time for x -fold increase in resources in the fewer resource regime. In the sufficient resource regime, we want scheduler to minimize execution time to the lowest achievable value (which is obtained considering no hardware constraints). To test the optimality of our Scheduler we select 64-bit Quantum Carry Look-Ahead Adder Draper et al. (2006) due to its resource consuming nature (see section 7.5.2 for details). Using this benchmark we obtained Figure 4.8 which shows benchmark execution time as a function of available resources. It can be seen that in a fewer resource regime, Scheduler achieves sufficiently higher (eight-fold) decrease in the execution time compared to the increase (two-fold) in resources. On the other hand, in case of higher resource regime, Scheduler reduces the execution time to its baseline value (two execution time curves meet in this regime). Based on these observations, we can conclude that Scheduler performs optimally over wide range of resource distribution patterns.

4.13 High Level Error Analyzer

The High Level Analyzer computes overall failure probability P_{fail} of the quantum algorithm circuit. It manipulates two types of quantities precomputed by the Low

Level Error Analyzer: (1) The failure probability of each logical operation (op) denoted P_{Log}^{op} and (2) The block physical failure probability BFPF denoted P_{phy}^Q of each logical qubit Q . As logical operation are processed for their scheduling, these probabilities to systematically utilized to compute P_{fail} according to the specific rules which are different from those defined for Low Level Error Analyzer. First, the quantum error correction performed at layer i can either correct for error at layer $i - 1$ or its converts layer $i - 1$ error into layer i logical error (if large number of errors are accumulated at $i - 1$). If layer i is the highest layer of encoding, then a logical error occurring at this layer is uncorrectable. Therefore we assume that a single logical failure at the highest layer is enough to declare faulty execution of entire quantum application circuit. In other words, the quantum algorithm will yield correct result when none of the operations fails, thus simplifying the expression of failure probability:

$$P_{fail} = 1 - \prod_{\forall op} \prod_{j=1}^{j=n^{op}} (1 - P_{Log}^{op}) \quad (4.14)$$

where n^{op} represents total repetition of an operation op . Both these variables had been precomputed by the Low Level Error Analyzer. We define following categories of operational failures:

- Shuttling Failure P_{Log}^{SHT} : Failure due to the noisy Ballistic Shuttling Channel
- Teleportation Failure P_{Log}^{TEL} : Failure due to the infidelity of an EPR pair for communication
- Idle Failure P_{Log}^I : Failure due to the fidelity degradation of qubit during no-operation(no-op)
- Gate Failure P_{Log}^{GATE} : Failure due to the noisy quantum gates

- Measurement Failure P_{Log}^M : Failure due to the noisy measurement
- EC Failure P_{Log}^{EC} : Failure due to the infrequent error correction

For Shuttling and Memory Failure, we assume exponential degradation of physical fidelity $F \sim \exp(-at)$, where a ($=1/T_{coh}$) is determined by the coherence time T_{coh} of the qubit, and t is the time between quantum gates over which qubit sits idle (no-op) or travels through Ballistic Shuttling Channel. The failure probability of physical qubit due to infidelity is: $p_f = 1 - F^{NStep}$, where $NStep$ represents the time steps spent in Shuttling or sitting idle. Therefore we compute Shuttling and Memory logical failure as follows:

$$P_{Log}^{SHT}, P_{fail}^I = 42p_f^2 \quad (4.15)$$

Similarly, the P_{Log}^{GATE} is computed from physical failure probability p due to the noise in physical operation. This is given as:

$$P_{Log}^{GATE} = Cp^2, \text{ where } C \in \{42, 84, 126\} \quad (4.16)$$

where C is 42 for single-, 84 for two- and 126 for three-operand transversal gates. Next, in case of cross-Segment communication using EPR pairs, the fidelity is improved by performing error correction on logical e-bits in corresponding Segments. In this sense P_{Log}^{Tel} and P_{Log}^{EC} are computed in the same manner. For error correction we need to know the amount of physical error P_{phy}^Q accumulated in the logical qubit (or logical e-bit) Q . Thus P_{Log}^{EC} will be calculated as:

$$P_{Log}^{EC} = 42[P_{phy}^Q]^2 + 491[P_{phy}^Q]p + 7512p^2 \quad (4.17)$$

The quantum error correction is a special case of measuring logical operator after which logical qubit is projected into correct or incorrect logical state depending upon

the value of P_{phy}^Q . Similarly, the logical measurement of Q will fail to produce correct outcome depending on P_{phy}^Q . Thus, for P_{Log}^M we define:

$$P_{Log}^M = 16[P_{phy}^Q]^2 + 83[P_{phy}^Q]p + 796p^2 \quad (4.18)$$

The operator measurement for magic state preparation (for T gate) using 7-cat state, we define P_{Log}^{Magic} as:

$$P_{Log}^{Magic} = 42[P_{phy}^Q]^2 + 392[P_{phy}^Q]p + 6083p^2 \quad (4.19)$$

The pre-factor C used in the eq. 4.15,4.16,4.17,4.18,4.19 was obtained from fault counter of Low Level Error Analyzer. We can generalize these equation to compute failure probability at layer i from layer $i - 1$ failure probability. Therefore, by recursively computing performance, we can efficiently track P_{fail} for any layer of concatenation. Finally, we describe how P_{phy}^Q is updated by the High Level Error Analyzer. P_{phy}^Q increases monotonically as gates operate on Q as noise accumulate due to faulty gates. However, once error correction is performed, we calculate P_{Log}^{EC} using eq.4.17 and reset P_{phy}^Q to default value 11. The Table 4.2 shows the effect of different operations on P_{phy}^Q . These values are also computed by the Fault Counter. The output of High Level Error Analyzer is also fed to the Performance Metrics Decomposer to obtain constituents of P_{fail} .

4.14 Performance Metrics Decomposer

The Performance Metrics Decomposer provides detailed breakdown of algorithm execution time and failure probability. First, the components of T_{exec} by keeping track of the different types of latency overhead comprising the critical path of the quantum circuit execution. Our iterative scheduler selects a gate for execution during each

iteration and updates the critical path. When a gate is selected for execution, the operand qubits should be available for computation otherwise it will be delayed. If we let T_{start} be the time at which the gate was executable but operands were available at T'_{start} where $T'_{start} \geq T_{start}$, then the time at which the gate execution is complete is given by $T_{finish} = T'_{start} + T_{exec}$. The gate execution selected in scheduling iteration i will update the critical path if $T_{finish} > T_{TotalExec}^{i-1}$ where $T_{TotalExec}^{i-1}$ is the total execution time (the length of critical path) computed in iteration $i - 1$.

When the gate lies on the critical path, the Scheduler computes $\Delta T = T_{finish} - T_{TotalExec}^{i-1}$ and $\Delta D = T'_{start} - T_{start}$ which can be broken down as $\Delta D = D_{ANC} + D_{SHT} + D_{TEL} + D_{SWP}$. To define the components of critical path in iteration $i - 1$ we split $T_{TotalExec}^{i-1}$ into its components as: $T_{TotalExec}^{i-1} = T_{ANC}^{i-1} + T_{SHT}^{i-1} + T_{TEL}^{i-1} + T_{SWP}^{i-1} + T_{GATE}^{i-1}$. For iteration i , these components are updated as follows:

- Ancilla Preparation Overhead:

$$T_{ANC}^i = T_{ANC}^{i-1} + \frac{D_{ANC}}{\Delta D + T_{exec}} \times \Delta T$$

- Shuttling Overhead:

$$T_{SHT}^i = T_{SHT}^{i-1} + \frac{D_{SHT}}{\Delta D + T_{exec}} \times \Delta T$$

- Teleportation Overhead:

$$T_{TEL}^i = T_{TEL}^{i-1} + \frac{D_{TEL}}{\Delta D + T_{exec}} \times \Delta T$$

- Segment Swap Overhead:

$$T_{SWP}^i = T_{SWP}^{i-1} + \frac{D_{SWP}}{\Delta D + T_{exec}} \times \Delta T$$

- Gate Overhead:

$$T_{GATE}^i = T_{GATE}^{i-1} + \frac{T_{exec}}{\Delta D + T_{exec}} \times \Delta T$$

In case of the critical path, we also update the total execution time $T_{TotalExec}^i = T_{finish}$. An example breakdown of execution time as shown in Figure 4.10, where magic state preparation for Toffoli gates and cross-segment swapping delay the execution of gates in turn and comprise the bulk of the critical path.

The decomposition of the failure probability P_{fail} is relatively easier. The eq.4.14 can be broken down into sub-equations each correspond to the noise source tracked by High Level Error Analyzer during the scheduling of logical operations. For any operation type op , we can obtain $P_{fail}^{op} = 1 - \prod_{i=1}^{i=n^{op}} (1 - P_{Log}^{op})$, where op can be Shuttling, Idle, Teleportation or Gate, Measurement and Error Correction and n^{op} and P_{Log}^{op} are the total operation count and failure probability for op , respectively.

4.15 Visualizer

Our Visualizer provides pictorial representation of the scheduled quantum circuit. In contrast to the critical path analysis of the quantum algorithm execution, it displays footprint of entire schedule. Therefore, Visualizer augments the utility of Performance Metrics Decomposer in three ways. First, it can help us in verifying the correctness of scheduling. Second, it can track the distribution of computational load across different parts of the quantum system. Third, it can evaluate utilization of resources for different types of operations.

Our Visualizer portrays latency incurred in scheduling quantum circuits by plotting two types of lines in Resource-Time plane. The slanted line represent delay in scheduling cross-Segment CNOT. The strictly horizontal lines designate delays in scheduling gates inside the Segment, such as gates comprising magic-state preparation and fault-tolerant implementation of non-Clifford gates. The length of line represents latency associated with the type of gate. To demonstrate the efficacy of this tool component, we present three 64-bit circuit examples for visualization: the Approximate Quantum Fourier Transform (AQFT) Fowler and Hollenberg (2004),

the CDKM Quantum Ripple Carry Adder (QRCA) Cuccaro et al. (2004) and the Quantum Carry Look-Ahead Adder Draper et al. (2006), their visual execution is shown in Figure 4.11,4.12,4.13 respectively, encoded with two layers of concatenation. In these figures, the horizontal axis represent time while vertical axis shows the numeric Segment ID for visualization plots. Each Segment contains four Data Tiles in case of AQFT and three in case of adders. Using these simulation parameters, interesting insights into the execution of these circuits can be obtained.

We find that the computational load in AQFT successively shifts from one Segment to the neighboring Segment, following the staircase connectivity pattern found in the circuit (comparing (a) and (b) in Figure 4.11). However, the lack of slanted lines show the absence of cross-Segment delays, indicating that AQFT execution time is dominated by the magic-state preparation. In case of QRCA, the load of computation trends into V-shape, imitating the connectivity structure of CNOT and Toffoli gates in the circuit (compare (a) and (b) in Figure 4.12). However, unlike AQFT, the presence of slanted lines in the plot indicates cross-Segment CNOT gates contributing to the overall execution time. Note that AQFT and QRCA plots look sparse which hints at the underutilization of Segment resources due to the serial nature of these circuits. In contrast, QCLA is constructed from several concurrently executable gates which keep several qubits busy at any given time. Thus, Figure 4.13 shows several Segments concurrently executing Toffoli gates represented by horizontal lines in the visualization. It is interesting to note that variation in the magic state preparation workload distribution across Segments, strikes sufficient matching with the concurrency of Toffoli gates in the circuit. The long vertical lines at the far end of visualization plot shows cross-Segment CNOT scheduled at the end of circuit. Based on these plots we can conclude that QCLA is more resource intensive circuit that AQFT and QRCA.

This example shows that our Visualizer provides added insights into the events

occurring while circuit is being scheduled. Most importantly, it sheds light on the workload distribution and resource utilization patterns. In chapter 7, we show that Visualizer can help us in optimizing the design of large scale quantum computer architecture for Shor's algorithm. Both Visualizer and the Performance Metric Decomposer are unique feature of our tool and vital contribution of thesis.

4.16 Summary

In this chapter we presented a case for the performance simulation software tool to explore design space for quantum systems. We highlighted basic components of such tool and surveyed prior work in this area and analyzed their strengths and limitations. A new design and performance estimation tool was presented which fulfilled the deficiencies in previous tools. These included the flexibility to change the parameters defining quantum hardware and architecture, the detailed breakdown of performance metrics and the ability to visualize resource utilization. The entire design flow was provided along with comprehensive details of each component. The functionality of various part of the tool was illustrated using underlying algorithm pseudocode or example demonstration of the technique or both. The tool was developed to pack very large number of qubits on the scalable quantum hardware (e.g., MUSIQC) in hierarchical manner. Using modularity, the tool also supported execution of circuit encoded with multiple layers of Steane code and optimized resource allocation across encoding layers. The efficiency of the tool came from explicit scheduling at the highest layer using per-computed schedules of immediate lower layer gates. With all these unique attributes, the tool lies at the heart of the thesis which describes flexible framework to study quantum computer architectures.

It should be noted that the tool presented in the chapter is the end product of an effort which went through different phases of development. At the end of each phase, we conducted a study using the hitherto version of the tool. These studies comprise

next three chapters of the thesis, each revisits parts of the tool relevant to the topic at hand.

Input: A fault-tolerant operation acting on logical qubit Q and containing sequence of physical gates $\mathcal{G} = g_1, g_2, g_3, \dots, g_m$ and n possible fault locations.

Output: Logical Failure Probability P_L^{fail} of logical qubit Q

initialization Precompute Stabilizers set $\{S_{g_i}\}_{i=1}^{i=m} \forall g_i \in \mathcal{G}$

$U \leftarrow \{\text{two-qubit uncorrectible errors defined for } Q\}$

LogicalErrorsCount = 0

for $i = 1 : n$ **do**

for $j = 1 : n$ **do**

if $i \neq j$ **then**

$E \leftarrow \{(P_i, P_j)\} \quad P \in \{X, Y, Z\}$

$\triangleright E$ is the set of all two-error events defined by combinations of Pauli errors pair (P_i, P_j) tested for faulty location for i and j

for $e \in E$ **do**

$Error = True$

while e continues to propagate **do**

 Find gate g_a for location i and g_b for location j

if $e(1) \in \{S_{g_a}\} \wedge e(2) \in \{S_{g_b}\} \wedge e$ caught in cat-state

verification then

$Error = False$

 Discontinue e propagation

end

else

 Propagate $e(1)$ through g_a

 Propagate $e(2)$ through g_b

 Update new faulty locations for i and j

 Update g_a and g_b to next gates in the sequence

end

end

if $e \in U \wedge Error$ **then**

 LogicalErrorsCount += 1

end

end

end

end

end

Algorithm 2: Pseudocode of the fault counter

Input: Dependency graph $G(V, E)$ with vertices V for gates in the circuit and directed edges E for dependencies among gates

Output: List containing pairs $(v_i, T_{start}^{v_i})$, $v_i \in V$ for start time of each gate

initialization $\forall i \in V \text{ Set}(v_i, 0)$

Define for a vertex x , $T_{finish}^x = T_{start}^x + T_{exec}^x$

while V is not empty **do**

$S \leftarrow$ Independent vertices from V

for $gate \in S$ **do**

if $gate \in \{\text{non- Clifford Group}\}$ **then**

 Obtain equivalent sub-dependency graph $W(x, d)$ for $gate$

 Substitute W for $gate$ in G

$V \leftarrow V - gate$

$S' \leftarrow$ Independent vertices from W

$S = S' \cup S$

$op \leftarrow gate$ operands

end

if Number of(op) = 3 **then**

\triangleright It is a transversal Toffoli gate

if op in different Segments **then**

 Teleport to bring all op in same Segments

 Update gate $T_{start}^{gate} = T_{avail}^{op}$ according to eq. 4.8 4.9,
4.10,4.11,4.12,4.13

end

 Shuttle logical qubits of op in the same Tile to execute Toffoli

 Update $T_{start}^{gate} = T_{avail}^{op}$ according to eq.4.3, 4.1 and 4.2

end

if Number of(op) = 2 **then**

\triangleright Its a CNOT gate

if all op in different Segments **then**

 Create EPR pairs to implement non-local CNOT

$T_{start}^{gate} = T_{avail}^{op}$ according to eq. 4.4, 4.5, 4.6, 4.7

end

else

 Shuttle logical qubit of op in the same Tile to execute CNOT

 Update $T_{start}^{gate} = T_{avail}^{op}$ according to eq.4.3, 4.1 and 4.2

end

end

if Error Correction needed for on or more op **then**

 Shuttle error-correction Ancilla Tiles into the Tile containing op

 Update $T_{start}^{gate} = T_{avail}^{op}$ according to eq.4.2

$T_{start}^{gate} + = T_{EC}^D$

end

 Update $T_{finish}^{gate} = T_{start}^{gate} + T_{exec}^{gate}$

end

for $w \in \{\text{children of gate}\}$ **do**

$T_{start}^w = \max(T_{start}^w, T_{finish}^{gate})$,

end

end

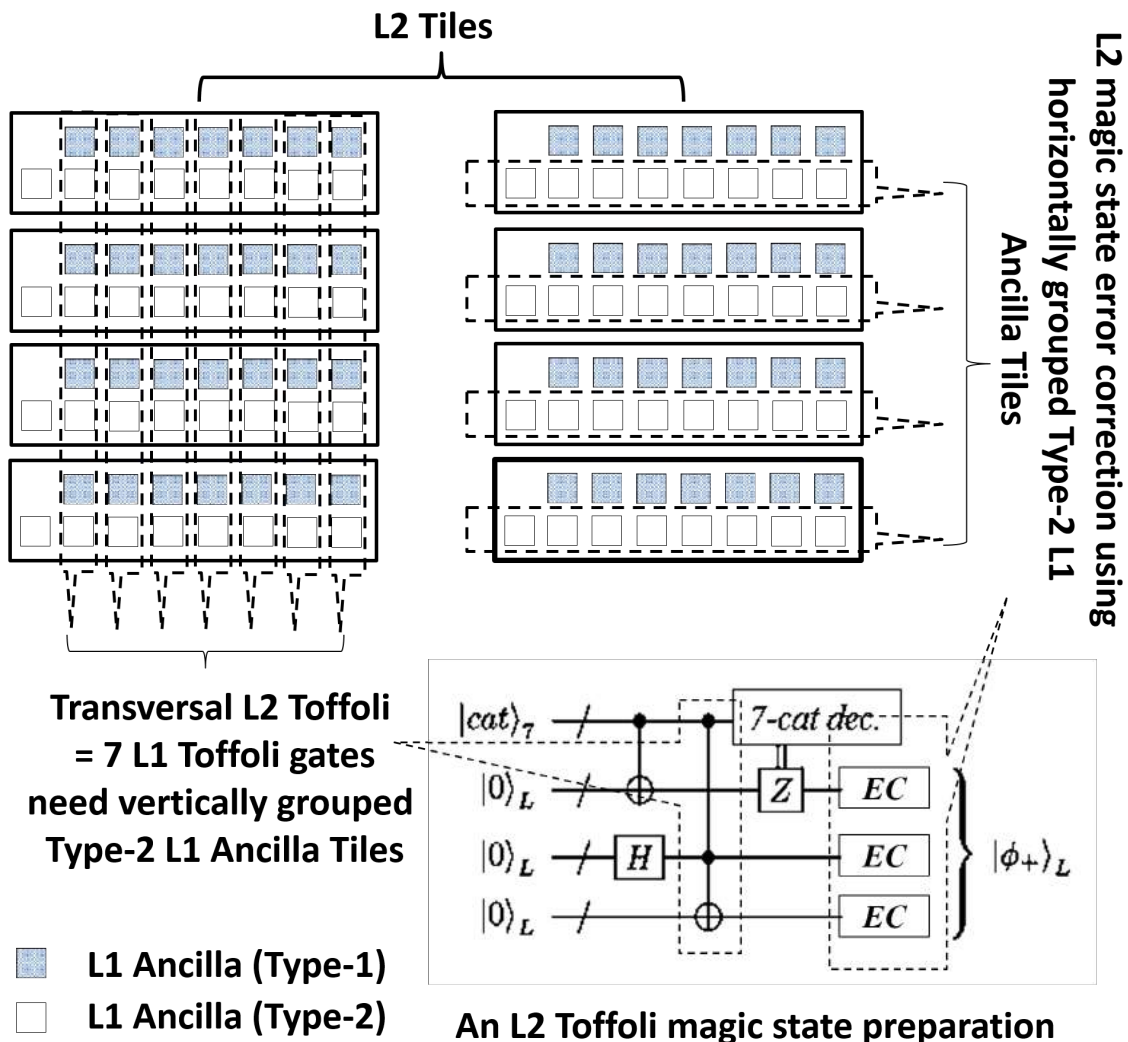


FIGURE 4.9: An example demonstration of cross-layer resource optimization during High Level scheduling using L2 Toffoli magic state preparation circuit. A block of Type-2 L1 Tiles used to perform seven L1 Toffoli gates can be reallocated to perform L2 error correction

Table 4.2: Updating Block Physical Failure Probability

Logical Operation	Addition to P_{phy}^Q	Update P_{phy}^Q
Single-qubit Clifford	$2p$	NA
Multi-qubit transversal	$2p$	NA
T gate	$20p$	NA
Error Correction	NA	$11p$

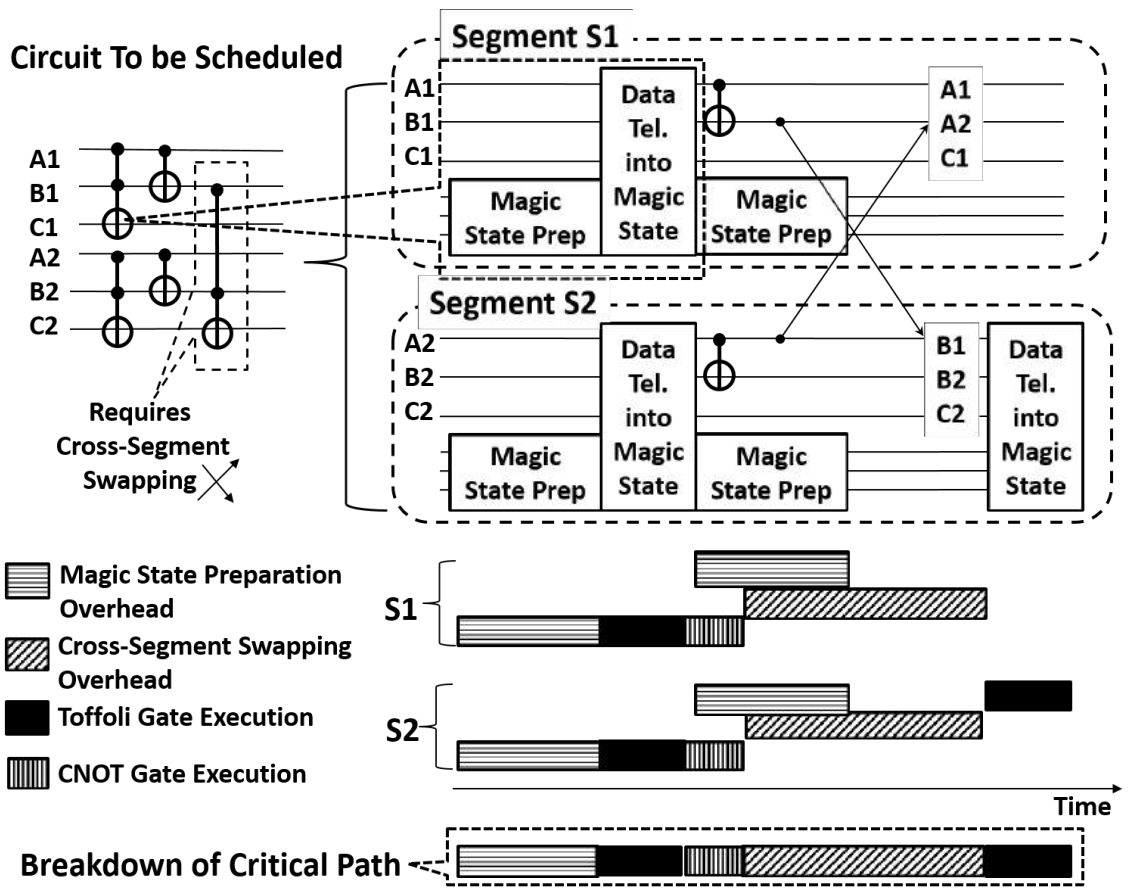


FIGURE 4.10: An example shows tracking of latency overheads comprising critical path of the circuit.

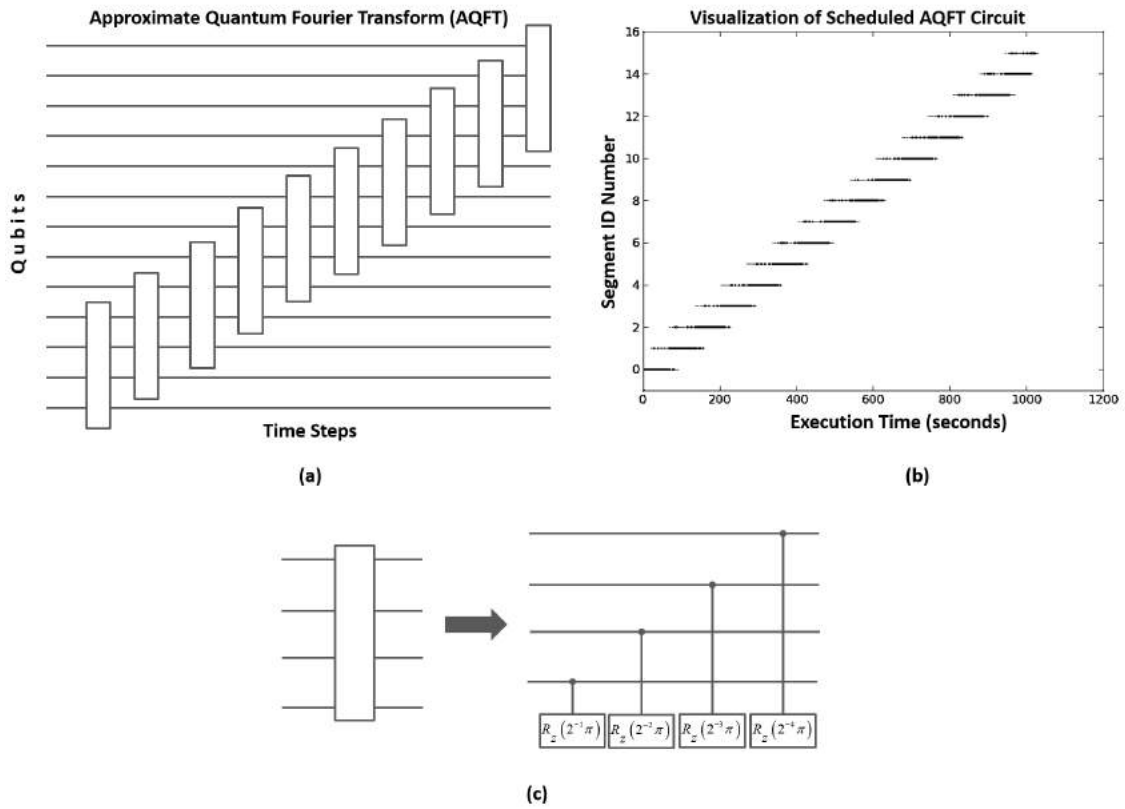


FIGURE 4.11: An example visualizer demo of 64-bit Approximate Quantum Fourier Transform circuit scheduled with 4 Data, 4 Ancilla and 1 Communication Tiles per Segment. A sample circuit is shown in (a) while visualization is presented in (b). The staircase structural connectivity of the macro blocks in the circuit, perfectly matches with the stairs in visualization. The internal circuitry of the block is shown in (c)

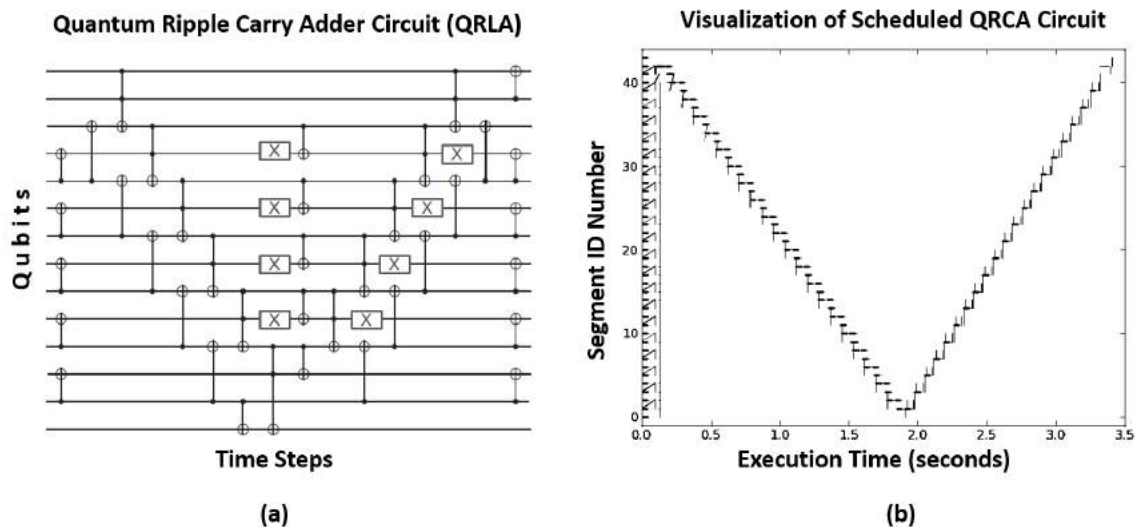


FIGURE 4.12: An example visualizer demo of 64-bit Quantum Ripple Carry Adder circuit scheduled with 3 Data, 4 Ancilla and 4 Communication Tiles per Segment. A sample circuit is shown in (a) while visualization is presented in (b). The “V” shape connectivity of the CNOT and Toffoli gates in the circuit maps similar pattern in the visualization shown in (b)

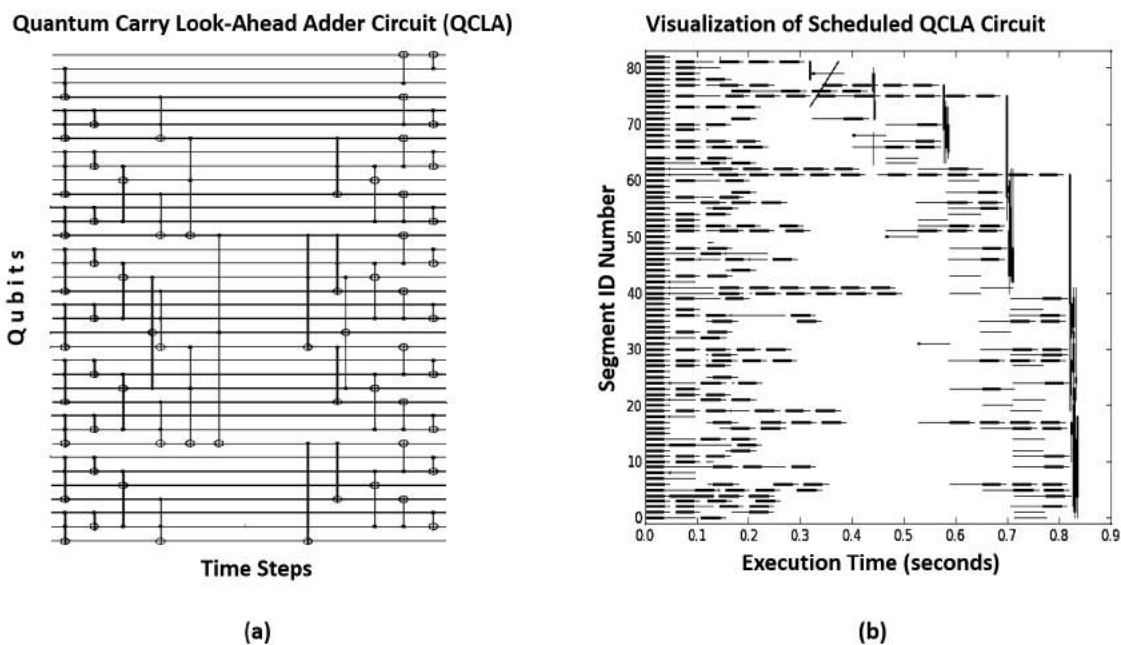


FIGURE 4.13: An example visualizer demo of 64-bit Quantum Carry Look-Ahead Adder circuit scheduled with 3 Data, 4 Ancilla and 4 Communication Tiles per Segment. A sample circuit is shown in (a) while visualization is presented in (b). Large number of parallel operations in the circuit translate into denser map in the visualization which shows busy Segment resources

Performance Simulation based on Hardware Resources Constraints

In chapter 3 it was shown that efforts to build quantum computers using ion-traps have demonstrated all elementary qubit operations necessary for scalable implementation. Modular architectures have been proposed to construct modest size quantum computers with up to $10^4 - 10^6$ qubits using technologies that are available today. Concrete scheduling procedure to execute a given quantum algorithm on such a hardware is a significant task, but existing quantum CAD tools generally do not account for the underlying connectivity of the qubits or the limitation on the hardware resources available for the scheduling. We present a scheduler and performance simulator that fully accounts for these resource constraints, capable of estimating the execution time and error performances of executing a quantum circuit on the hardware. We outline the construction of tool components, and describe the process of mapping the qubits to ions and scheduling the physical gates in the MUSIQC based architecture. Using this tool, we quantify the trade-off between hardware resource constraints and performance of the computer and show that at an expense of x fold increase in latency, a minimum of $1.6x$ resource reduction is possible for executing a

three-qubit Bernstein-Vazirani algorithm encoded using Steane code.

5.1 Motivation of Study

Realization of scalable quantum computing device has foundational importance of demonstrating quantum speedup in actual problem solving, as well as several known practical applications. In contrast to the rapid advances in theory, the frontier of experimental effort of building a quantum computer of significant size still faces a wide range of technological challenges. The experimental demonstrations so far have been limited to only small sized computers Marx et al. (2000); Vandersypen et al. (2001); Negrevergne et al. (2006); Hanneke et al. (2009); Monz et al. (2011); Mariani et al. (2011). Architectures for constructing large-scale quantum processors have been considered Monroe et al. (2014); Galiutdinov et al. (2012), and one can readily estimate the performance of these systems by simulating quantum circuit on the hardware. Existing tools that were developed for this purpose Svore et al. (2004, 2006a); Balensiefer et al. (2005a); Whitney et al. (2007); Balensiefer et al. (2005b) make simplifying assumptions about the physical hardware that is not well justified in practical settings: examples of such assumptions include (1) arbitrarily large number of qubits available in the quantum hardware, (2) unlimited classical control resources that support many qubit manipulation operations in parallel, and (3) arbitrary connectedness of the qubits allowing two-qubit gate between any qubits in the quantum hardware regardless of their relative location. One of the important considerations in designing performance simulation tool is to handle resource intensive fault-tolerance requirements of quantum circuits Nielsen and Chuang (2000). Since quantum systems are much more sensitive to errors compared to their classical counterparts, fault-tolerant protocols must be introduced to increase the probability of correct circuit execution by performing computation on qubits encoded in quantum error correcting codes. A quantum circuit is fault-tolerant if the failure rate of

the encoded circuit execution reduces to $O(p^2)$, where p is the failure rate of each physical gate. Although a fault-tolerant quantum circuit is successfully constructed, successful fault-tolerant execution requires low enough error probabilities in the physical operations and memory, along with the availability of fresh supply of qubits and the hardware resources needed to execute the frequent error correction procedures in parallel to efficiently overcome the memory errors Preskill (1998b).

In this chapter we quantitatively analyze the requirements on the fault-tolerant operation of a quantum circuit and the necessary hardware resources given acceptable memory error rates. We shall base our analysis on performance simulation of a MUSIQC as the underlying quantum hardware. Using our tool, we show that there is a manifold in which reliable quantum computation is possible respecting the constraints of the hardware, such as the number of qubits, their interconnectivity and number of concurrent gate operations. Furthermore we show that we can trade one type of hardware constraint variable for another in achieving a similar performance. This trade-off opens up useful design space for the quantum computer architects to focus their effort in pursuit of meaningful design parameters. Finally, based on simulation results, we will show that there is a significant reduction in resources at the cost of modest degradation in execution time as long as one can find a schedule under hardware constraints in which qubit idle time is held within reasonable limits.

In section 5.2, we briefly describe the MUSIQC architecture, important terminology and definitions used throughout the chapter. Section 5.3 describes the overview of the tool design flow, the resource bound mapping and scheduling strategies, and methods to quantify fault tolerance for a given architecture. Simulation results and their analysis comprise section 5.4. Further discussions on the validity, scalability and limitations of our tool is detailed in section 5.5, and section 5.6 contains the summary.

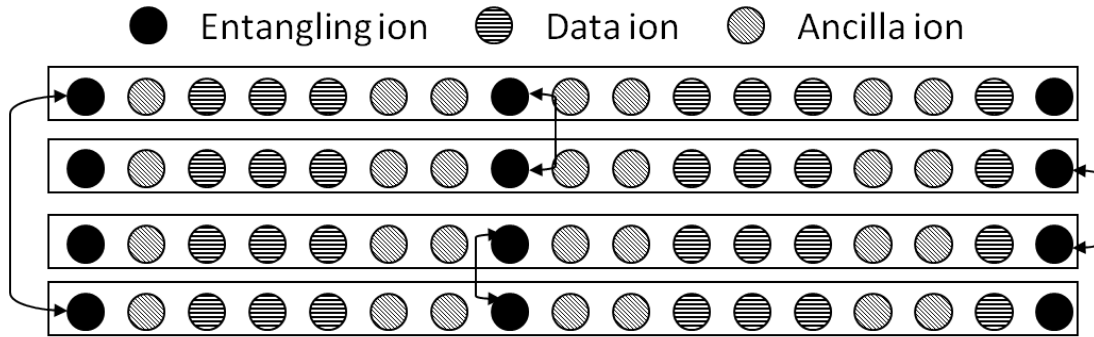


FIGURE 5.1: Schematic of a modular ion trap quantum computer architecture used for performance analysis in our simulation

5.2 Hardware, Architecture Model and Definitions

While all physical models of quantum computer hardware will be subject to a range of constraints, a specific physical hardware platform has to be defined for concrete analysis. In our study, we analyze a modular universal scalable ion-trap MUSIQC as the underlying quantum hardware. Compared to other idealized architectures [Metodi et al. \(2005\)](#); [Thaker et al. \(2006\)](#); [Isailovic et al. \(2008\)](#), MUSIQC utilizes a simple 1-D chain of ions as a unit of quantum register within which multi-qubit gates are accomplished using transverse vibrational mode of the chain as the inter-qubit interaction [Zhu et al. \(2006\)](#). Such chains form ELUs in the scalable architecture. Multiple ELUs implemented on the same ion trap chip can be connected through physical shuttling of ions between them [Kielpinski et al. \(2002\)](#), while a two-qubit gate between ELUs that are physically separated can be realized by first establishing an entangled qubit pair between designated “entangling ions” on the ELUs using a photonic channel [Duan et al. \(2004\)](#), and then using the entangled pair as a resource to perform the desired gate via quantum teleportation [Gottesman and Chuang \(1999\)](#). The ELUs are connected to each other through a reconfigurable optical crossconnect switch, where the entangling ions from any pair of ELUs can

be connected within the system. This makes the cost of communication between ELUs for non-local (inter-ELU) gate independent of distance, a unique feature in MUSIQC architecture that provides significant advantages in implementing useful quantum circuits Monroe et al. (2014).

A simplified schematic of architecture is shown in Figure 5.1. The ions within each ELU are used as three different types of qubits: **data**, **ancilla** and **entangling** ions. Data ions correspond to data qubits which store main quantum information during algorithm execution. The ancilla ions are for ancilla qubits that carry temporary quantum information during various quantum error correction procedures, such as (error) syndrome detection, error correction, and preparation and measurements of special quantum states used in fault-tolerant quantum gate procedures Nielsen and Chuang (2000). Entangling ions are used for generating entangled qubit pairs (e-qbts) between ELUs used to communicate qubits or gates Maunz et al. (2009). The established entanglement is shown as black arrowed arcs. The process of generating entanglement between entangling ions is probabilistic, and therefore in our simulation we use average time for entanglement generation. The gate operation between ions on different ELUs (inter-ELU gates) is carried out using either quantum teleportation of data qubits from one ELU to the other, or using distant gate method based on local operations and classical communications (LOCC) J. Eisert and Plenio (2000).

The quantum error correction is performed by first measuring all the stabilizers of the code (corresponding to the parity check operations in classical linear codes) Nielsen and Chuang (2000). The measurement result of the stabilizers provide the error syndrome, which dictates the error correction procedure that should be applied to the encoded qubit. A fault-tolerant procedure for measuring the stabilizers is well established using ancilla qubits, and can be performed in parallel if sufficient resources are available. Various strategies for performing all stabilizer measurements under resource constraints will be the main analysis of this work.

The following terminology and acronyms will be used in the rest of the chapter:

- $AvEPR$: Number of available e-qbts on each ELU
- T_{entgmt} : Average time to establish an entangled pair between two ELUs.
- $NAnc$: Number of available ancilla ions on each ELU
- NP : Maximum number of permitted concurrent gate operations in the system, which depends on the amount of classical control hardware available.
- $P_{success}$: Probability that computation yields correct outcome after the execution of a quantum circuit. $1-P_{success}$ is also referred to as the error or failure probability.
- T_{exec} : Total time spent in executing the circuit

Variables such as $AvEPR$, T_{entgmt} , $NAnc$, NP describe hardware resource constraints whereas $P_{success}$ and T_{exec} are performance metrics. We do not consider a constraint on data ions: we have to assume the quantum hardware has sufficient data qubits to handle the problem size under consideration, and there is not much opportunity for reuse of the data qubits during computation.

5.3 Tool Components and Overall Design Flow

The version of tool used in this study is shown in Figure 5.2(a). It has three main parts: Physical Circuit Graph (PCG) builder, Mapper and Scheduler, with Error Analyzer as an optional component. These components belong to TDPA in the context of complete toolbox, shown in Figure 5.2(b). The quantum circuit and architecture resource specifications are provided by the user to the PCG builder which generates the PCG. This graph is constructed by translating qubit connectivity in

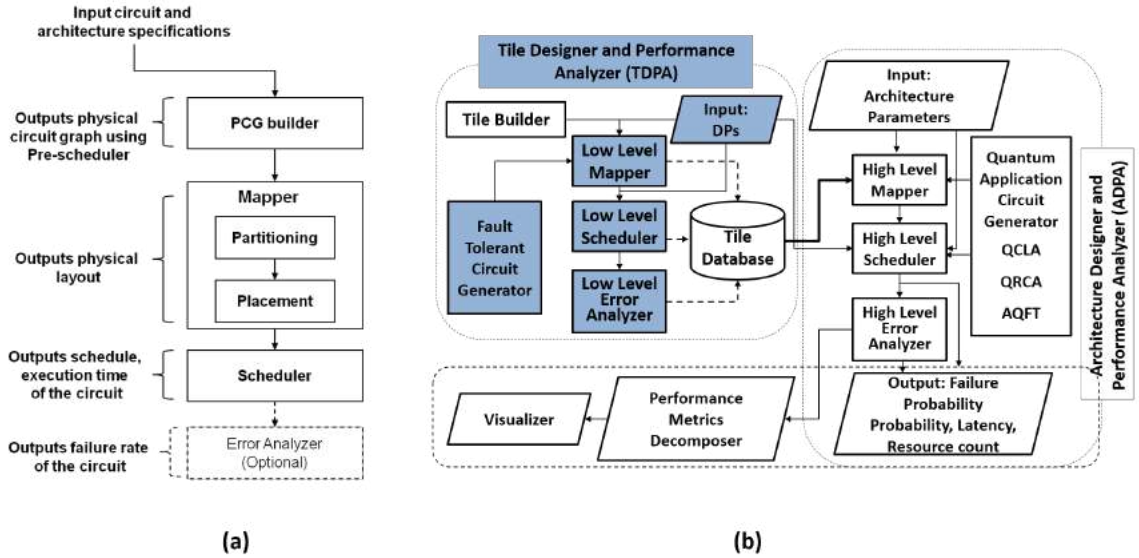


FIGURE 5.2: (a) The components and their interconnection in tool design flow. PCG: Physical Circuit Graph (b) These components map to the shaded constituent blocks of TDPA in the context of complete toolbox

the given quantum circuit to ion connectivity in the hardware with the help of a pre-scheduler. PCG is fed to the Mapper which performs Partitioning and Placement to map circuit qubits to ions in ELU. Once Mapper has produced the layout, Scheduler runs to obtain total execution time and to optionally record qubit idle times that may be used by error analyzer. Both Mapper and Scheduler are designed to operate under hardware resource constraints. Error Analyzer, in this study, takes the scheduled circuit from the Scheduler as input, uses Monte-Carlo simulation to simulate gate error in the circuit, and calculates overall failure probability of the circuit. In this chapter we assume a fixed value for the gate errors, and study the impact of memory errors on fault-tolerant operation of a quantum circuit.

Limited resources play key role in automated mapping and scheduling performance of quantum circuit on the hardware. Our algorithms keep track of each resource usage by labeling them ‘Occupied’ or ‘Available’. Resources are dynamically allocated and their allotment is prioritized such that overall latency of the quantum

circuit execution is minimized.

5.3.1 Mapping

Mapper allocates the qubits in the quantum circuit on to physical ions in the hardware. For data qubits, mapping is static, meaning that ions representing data qubit will not hold ancilla or e-qubits at any time. For ancilla and e-qubits (both of these are ‘reusable qubits’), each ELU keeps track of occupied and available qubits. In this sense allocation of ions for these qubits is dynamic and handled in the scheduler. Hence finding static map for all data qubits is the main job of Mapper, which has two parts: Partitioner and Placer. Partitioner places each data qubit into a specific ELU, and Placer identifies a physical ion for each data qubit within the designed ELU. Mapper has to first construct qubit connectivity graph from quantum circuit to facilitate application of our algorithms. It contains vertices corresponding to qubits in the circuit and edges model CNOT gate between the qubits. The weight of an edge between two vertices is determined by the number of CNOTs between the associated qubits. The selection of our algorithms is based on simplicity and efficiency to match the requirements of scheduling quantum circuits. Next we briefly describe our partitioning and placement algorithms.

The Partitioning algorithm distributes qubits among ELUs such that the number of inter-ELU gates resulting from the partitioning is minimized. To solve this partitioning problem, we implemented a simple greedy algorithm to exploit common structures in quantum circuits such as quantum error correction. We note that such repetitive structures produce qubit connectivity graphs (QCGs) with two important properties: that the edge weights (total number of two-qubit gates involving a qubit as the vertex) are often bounded by a small integer, and the variance of edge weights of the vertices are also generally small. Based on these observations, our partitioning algorithm prioritizes the assignment of qubits to the ELUs in the ascending order of

their total edge weight. In each iteration, a qubit is selected from the priority list along with the set of its interacting qubits. An ELU which can accommodate most of these qubits is selected and assigned to the set. The priority list is then updated with leftover qubits to be processed in the subsequent iteration, until every qubit is assigned to an ELU.

Placement algorithm arranges the qubits within an ELU by maximizing locality among frequently interacting qubits. It finds an approximate solution to the Optimal Linear arrangement Problem, by using a graph-theory-based algorithm discussed in Ref. Juvan and Mohar (1992). First, we translate the QCG as an Adjacency matrix, where each row and column represents the qubits in the quantum circuit, and each element corresponds to the number of two-qubit operations between the row and the column qubits. Then, the Laplacian of this matrix is obtained and its eigenvalues and eigenvectors are computed. The resulting eigenvalues are non-negative. The eigenvector corresponding to the second smallest eigenvalue is chosen to determine the location of the qubits within each ELU. For more details on the mapping problem please refer to section 4.5.

5.3.2 Scheduling

Scheduler takes mapped hardware and generates the execution order of gates in the given quantum circuit. The main challenge of scheduling lies in dynamic allocation of reusable qubits and hardware-imposed constraints on concurrent execution of gates. Our Scheduler in Algorithm 4 works in an iterative way: first, a Dependency Graph is generated at the quantum circuit level, and a set of gates is selected for execution in each iteration based on the dependencies. A subset of these gates may be further selected due to limited e-qbts and ancilla used in inter-ELU gates and quantum error correction, respectively. The selected gates are then dispatched and their respective order of execution is determined by constraints on NP . This is similar to classi-

Input: Dependency graph $G(V, E)$ with vertices V for gates in the circuit and directed edges E for dependency between gates

Output: List containing pairs $(v_i, T_{start}^{v_i})$, $v_i \in V$ for start time of each gate

initialization $\forall i \in V \text{ Set}(v_i, 0)$.

Define for a vertex x , $T_{finish}^x = T_{start}^x + T_{exec}^x$

while V is not empty **do**

$S \leftarrow$ Independent vertices from V

if S doesnot satisfy constraints on $AvEPR$ **then**

$S_1 \leftarrow$ EPR independent nodes in S

$\forall i \in S - S_1, T_{start}^i = \min_{j \in S_1} T_{finish}^j$

end

if S_1 doesnot satisfy constraints on $NAnc$ **then**

$S_2 \leftarrow$ NAnc independent nodes in S_1

$\forall i \in S_1 - S_2, T_{start}^i = \min_{j \in S_2} T_{finish}^j$

end

 Dispatch left over nodes in S_2 for execution and update their T_{start} as:

$\forall i \in S_2, T_{start}^i = \min_{1 \leq j \leq NP} L_{avail}^j$

$\triangleright L_{avail}^j$ is the time when j^{th} slot of Parallel Operation Buffer becomes available

 Update $V \leftarrow V - S_2$

for $gate \in S_2$ **do**

$T_{finish}^{gate} = T_{start}^{gate} + T_{exec}^{gate}$

for $w \in \{\text{children of gate}\}$ **do**

$T_{start}^w = \max(T_{finish}^{gate}, T_{start}^w)$

end

end

end

Algorithm 4: Scheduling Algorithm

cal instruction processing wherein processor maintains reorder buffer and incoming instructions are queued until their operands are available. Here, the operands correspond to qubits and number of slots in reorder buffer is NP . For more details on the scheduling problem, please refer to the section4.6.

The dynamic resource allocator within the scheduler is responsible for deadlock-free scheduling through careful management of resource assignment and release. The most frequent tasks encountered in scheduling are fault tolerant protocols largely containing quantum error correction. It often requires a minimum number n_{min} of ancilla qubits to schedule main subtask of syndrome calculation, which is composed

of three steps. In the first step, it needs fresh ancilla prepared in an appropriate state (*e.g.*, a maximally entangled state of several qubits), which amounts to resource assignment. In the second step, ancilla interacts with data to obtain the signature of the error. In the last step, ancilla are decoded and measured to extract final error information. Once measurements are performed, ancilla qubits are released and can be reused for another syndrome calculations or other tasks. The syndrome calculations subtasks can be scheduled in parallel if sufficient ancilla qubits are available, or (partially) in series otherwise. When fully parallel scheduling is infeasible due to limited availability of ancilla qubits and the resource allocator distributes ancilla such that each one is assigned less than n_{min} ancilla, the schedule can potentially encounter deadlock because none of the subtasks can be completed and no allocated ancilla can be freed for other (sub)tasks. Therefore, a smaller quantum computer can suffer from scheduling deadlock due to inefficient resource allocation.

To address this issue, our resource allocator follows ‘all or nothing’ strategy, where each subtask is scheduled only when enough ancilla are available for its completion. In addition, scheduling of operations within each subtask is prioritized to facilitate quick ancilla release. This is achieved by assigning highest priority to the measurement since they free qubits to be used by other tasks. The same strategy is applied to e-qbt resource management, except that in case of *AvEPR* constraint, the associated tasks are operations local to ELU for each non-local CNOT J . Eisert and Plenio (2000) and can be scheduled with only one e-qbt per ELU. On the contrary, the NP constraint does not create any deadlock since unlike quantum resources which are freed only after measurement of qubits, the classical resource NP is can be released and reused once the current operation (whether it is a gate, preparation or measurement) is completed.

5.3.3 Quantifying Architecture Support for Fault-tolerance

Reliable scheduling of gates in quantum circuit under resource constraints is an important question. The quantum hardware constraints may not support successful scheduling of fault-tolerant computation where maximum parallelism and sufficient supply of ancilla qubits are required. We argue that by imposing an upper limit on qubit idle time to ensure acceptable failure probability for the quantum memory, fault-tolerance can usually be achieved.

When mapped to realistic hardware, typical quantum circuits incur both gate and memory errors. The level of noise in a gate execution determines the failure rate of the gate. Quantum threshold theorem says that fault tolerant quantum computation is possible if the error probability p of each gate is below certain threshold value p_{th} ($p < p_{th}$). In this analysis, we choose a fixed gate error rate of $p = 2 \times 10^{-5}$ which is below the fault tolerant threshold, yet potentially achievable in the ion-trap system. The memory error rate R_{mem} may be translated into an error probability p' per unit time T (e.g., the gate time) by multiplying the rate by the time interval, which is a valid approximation when $p' = R_{mem}T \ll 1$. For trapped ions, the decoherence time of the memory can be as long as $1/R_{mem} \geq 1 - 1,000$ sec. We choose a value of $T_{dec} = 5000\mu\text{sec}$ so that $p' = R_{mem}T_{dec} = 5 \times 10^{-5}$ in our simulations, and employ a simple error model where a memory error is added whenever the qubit idle time exceeds T_{dec} .

5.4 Simulation of Bernstein-Vazirani Algorithm

In this section we shall describe our main simulation results. The circuit chosen for simulation is Bernstein-Vazirani algorithm Bernstein and Vazirani (1993b) shown in Figure 5.3. There are four logical qubits, two in each ELU. All qubits are encoded with Steane $[[7,1,3]]$ quantum error correcting code Steane (1996), so that all gate

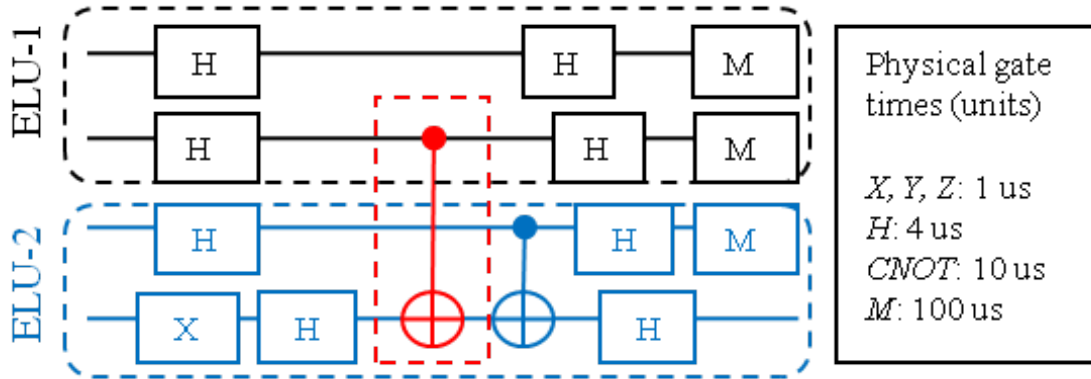


FIGURE 5.3: (Left) The quantum circuit that was simulated. There are two ELUs and the dotted box contains inter-ELU CNOT gate. (Right) Typical execution times of the gate and measurement operations used in the simulation.

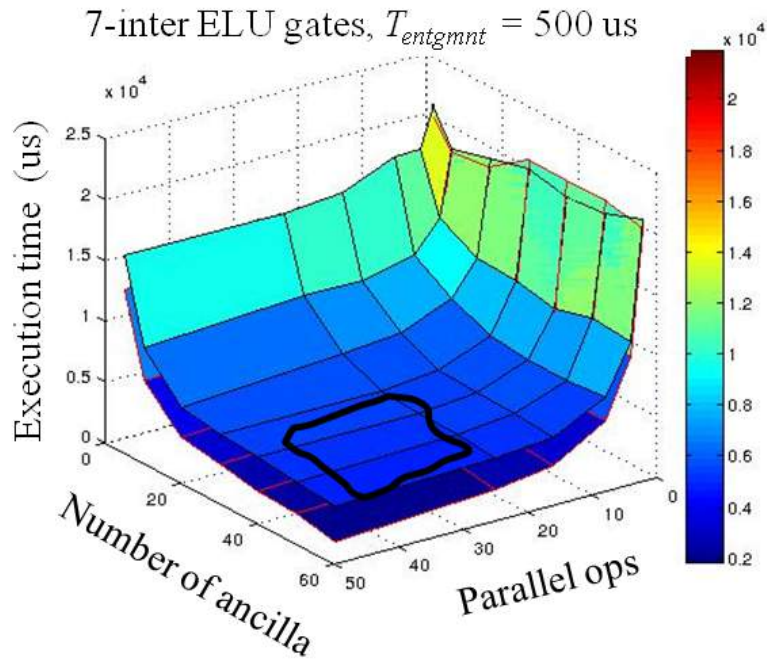


FIGURE 5.4: Simulated execution time as a function of available ancilla qubits and parallel operations. Upper (lower) curve is for $AvEPR = 1$ (7). Region enclosed in the loop is where variation in execution time is minimum ($T_{entgmnt} = 500 \mu\text{sec}$).

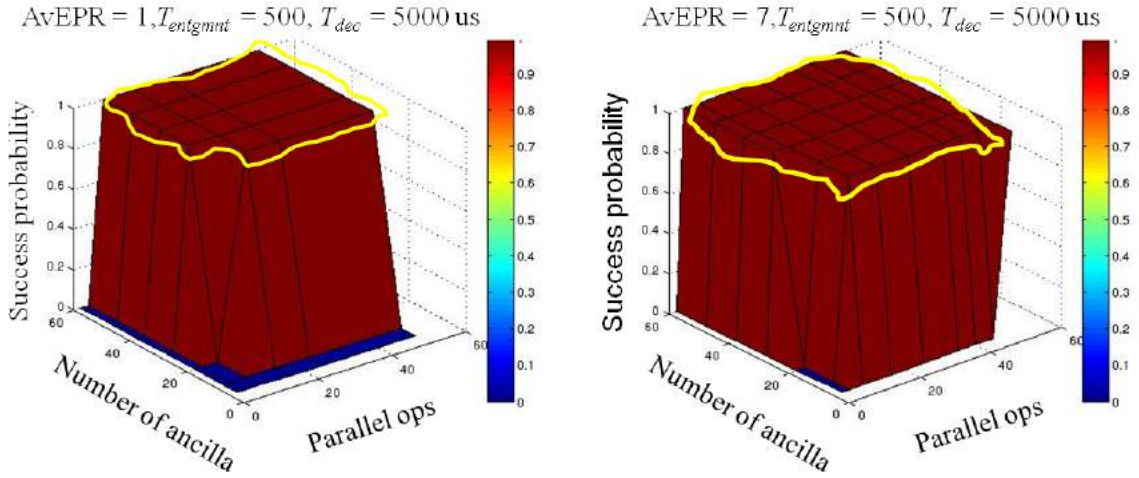


FIGURE 5.5: Success probability variation with ancilla and parallel ops. Left plot is for $AvEPR = 1$ and right one for $AvEPR = 7$. $T_{entgmnt}$ is taken $500\mu s$. The decoherence Time $T_{dec} = 5000\mu s$. Curve inside yellow the boundary within which $P_{success} = 1$

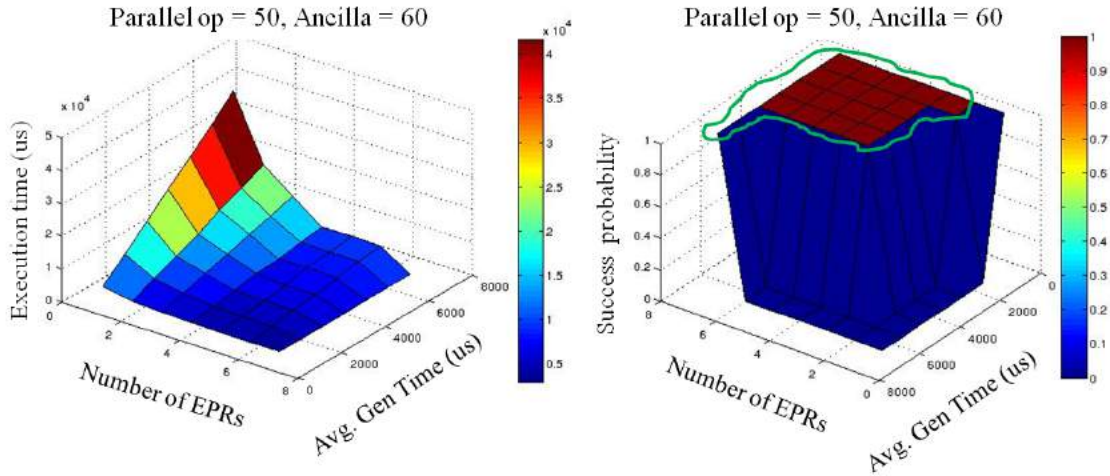


FIGURE 5.6: T_{exec} and $P_{success}$ variation with Number of EPRs ($AvEPR$) and Avg. Gen Time ($T_{entgmnt}$). T_{exec} is shown in left subplot and $P_{success}$ in right subplot on the top. T_{dec} is $5000\mu s$. There is no constraint on number of parallel operations and available ancilla and hence $NAnc$ and NP are set to their maximum values, 60 and 60 respectively. In left subplot, the red curve inside the (blue) boundary is a feasible region for computation since $P_{success}$ is unity.

operations in the circuit are transversal, i.e., they can be performed as a bit-wise gate operation to each constituent qubits within the code block. Transversal gates are automatically fault-tolerant, as the error in one component of the code block do not propagate to other components. A logical inter-ELU CNOT gate shown in (red) dotted rectangle will therefore correspond to seven bit-wise inter-ELU physical CNOT gates. We also assume that error correction is performed after each logical gate and necessary ancilla will always be fetched from the same ELU. Gate and measurement times are taken from Metodi et al. (2005), and shown on the right in Figure 5.3. For error analysis of the circuit, we assume that if qubits sits idle for time greater than T_{dec} , it will acquire Y (both bit and phase flip) error. We keep track of errors in each physical and logical qubits and if any logical qubit has uncorrectable error after all gates have been scheduled, this is counted as failure of entire circuit. Thus $P_{success}$ is 1 if all errors were perfectly corrected and 0 otherwise.

5.4.1 Simulation Results

T_{exec} as a function of $NAnc$ and NP

In this simulation we are interested in observing the effect of $NAnc$ and NP constraint on the execution time of circuit. Intuitively execution time should increase by tightening the constraints as we have fewer resources to exploit parallelism in the circuit. As expected, Figure 5.4 shows that T_{exec} increases as NP and $NAnc$ decrease. We note that there is gentle increase in T_{exec} starting from the regime of least constraints ($NP = 60$, $NAnc = 60$) as NP and $NAnc$ are reduced, until at a certain point a sharp increase is observed.

In the regime of gentle increase, we can find a surface which is ‘almost flat’, shown enclosed in the (black) boundary in Figure 5.4. Execution time remains almost constant at this surface and therefore we can readily trade number of available ancilla with parallel operations. For example there are two points which have roughly

same performance: $T_{exec} = 5605\mu s$ for $NP = 50$, $NAnc = 30$, which is close to $T_{exec} = 5506\mu s$ for $NP = 16$, $NAnc = 60$. On the other hand when constraint variables approach their minimum permissible values, T_{exec} shows rapid increase. Figure 5.4 shows two surfaces, one for $AvEPR = 1$ (upper surface) and another for $AvEPR = 7$ (lower surface). The difference in execution time for these two cases is more prominent when NP and $NAnc$ are sufficiently large, because execution time depends more strongly on $AvEPR$. This is when T_{entgmt} begin contributing to T_{exec} depending upon chosen $AvEPR$ value. As NP and $NAnc$ are reduced, the impact of $AvEPR$ diminishes, as limitations due to NP and $NAnc$ start dominating the execution time T_{exec} .

$P_{success}$ as a function of $NAnc$ and NP

This simulation describes trends in $P_{success}$ when $NAnc$ and NP change for a given $AvEPR$ value. Intuitively, as NP and $NAnc$ decrease, the qubits tend to sit idle waiting for the queues to clear due to limited resources, and eventually experiences a memory error. When large number of qubits simultaneously acquire errors, error correction procedure would fail and fault-tolerant protocols will not work.

Figure 5.5 shows that for decoherence time (T_{dec}) of $5000\mu sec$, errors resulting from most input combinations of constraint variables were corrected, shown by the red portion enclosed in (yellow) boundary. There is a small (blue) portion where the circuit was unable to correct for errors in all logical qubits. Here, we also observe a trade-off similar to the previous simulation for the execution time T_{exec} . For example $P_{success} = 1$ for both $NP = 50$, $NAnc = 20$ and $NP = 16$, $NAnc = 60$ for $AvEPR=1$. However if we supply seven EPRs for concurrent inter-ELU gates (right in Figure 5.5), the region where $P_{success} = 1$ occupies larger area compared to single EPR case (left in Figure 5.5) when we have to reuse these e-qbts to regenerate entanglement six more times.

Table 5.1: Combined data for simulations of Figure 5.4 and Figure 5.5. Each entry of the table contains latency and failure rate pair of the form $(T_{exec}, P_{success})$ for $AvEPR = 1$. T_{exec} is in μs

	NAnc						
NP	60	50	40	30	20	10	5
50	4786,1	5039,1	5431,1	5605,1	6271,1	9489,0	16324,0
25	5115,1	5486,1	5505,1	5705,1	7076,1	10281,0	16108,0
16	5506,1	5841,1	5888,1	6186,1	7544,1	10154,0	16588,0
8	7757,1	7462,1	7540,1	7819,1	9190,0	11145,0	18465,0
4	11765,0	12020,0	11070,0	11807,0	12041,0	13682,0	18456,0
2	21544,0	20407,0	20092,0	20705,0	19930,0	18960,0	21870,0

T_{exec} and $P_{success}$ as functions of $AvEPR$ and $T_{entgmnt}$

Next, we explore performance deterioration due to fewer number of e-qubits ($AvEPR$) and overhead of entanglement generation time $T_{entgmnt}$ when $NAnc$ and NP are set to their maximum value. Both T_{exec} and $P_{success}$ metrics are plotted against different values of $AvEPR$ and $T_{entgmnt}$ in Figure 5.6. Lower $AvEPR$ and higher $T_{entgmnt}$ will lead to increased latency due to costly entanglement links between ELUs. Resulting increase in T_{exec} can raise the idle time of the qubits, thereby contributing to memory errors. However, we note that in several events these memory errors can be fully corrected, *i.e.*, $P_{success}$ is unity for certain combination of $AvEPR$ and $T_{entgmnt}$, opening up an opportunity for trade-off. For example, $T_{exec} = 7014\mu sec$ can be achieved either with $AvEPR = 2$ and $T_{entgmnt} = 1500\mu sec$, or with $AvEPR = 6$ and $T_{entgmnt} = 4500\mu sec$. A consequence of this trade-off is that higher $AvEPR$ values can compensate for higher $T_{entgmnt}$, and lower values of $T_{entgmnt}$ allows successful circuit operation with smaller $AvEPR$.

Table 5.2: Combined data for simulations of Figure 5.4 and Figure 5.5. Each entry of the table contains latency and failure rate value of the form $(T_{exec}, P_{success})$ for $AvEPR = 7$. The T_{exec} is in microseconds

	NAnc						
NP	60	50	40	30	20	10	5
50	1833,1	2404,1	2613,1	3163,1	3787,1	6813,1	13416,1
25	2526,1	2948,1	3208,1	3807,1	4647,1	7039,1	14410,0
16	3187,1	3597,1	3833,1	4387,1	5916,1	7793,1	15486,0
8	5737,1	5854,1	7256,1	6365,1	7136,1	9791,1	15552,0
4	10442,1	11021,1	10549,1	10695,1	10583,0	12511,0	18177,0
2	20732,0	21420,0	21475,0	21447,0	19418,0	18814,0	21048,0

5.4.2 Analysis of Resource Reduction

In order to examine the reduction in the overhead of $NAnc$ and NP , we present combined data of Simulation (1) and (2) in Table 5.1 and Table 5.2 for $AvEPR = 1$ and $AvEPR = 7$, respectively. Based on the discussion in section 5.3.3, we only need to determine if we can recover from memory errors for given budget of $T_{dec} = 5000\mu\text{sec}$ in our case. Hence we are restricted to work in the shaded regions of these tables where $P_{success} = 1$. For $AvEPR = 1$ case (Table 5.1), T_{exec} increases by a factor of 1.6 ($4789\mu\text{sec}$ to $7757\mu\text{sec}$) for 6 fold reduction in NP (from 50 to 8) and by factor of 1.3 ($4789\mu\text{sec}$ to $6271\mu\text{sec}$) for 3 fold (60 to 20) reduction in $NAnc$. For $AvEPR = 7$ case (Table 5.2), T_{exec} increases 5.5 times ($1833\mu\text{sec}$ to $10442\mu\text{sec}$) for 12 fold decrease in NP (from 50 to 4) and by factor of 7.3 ($1833\mu\text{sec}$ to $13416\mu\text{sec}$) for 12 fold reduction in $NAnc$ (60 to 5).

In case of Simulation (3), we will consider shaded portion of the Table 5.3. Recall that in this simulation we put no constraints on NP and $NAnc$ so that latency is a function of $AvEPR$ and $T_{entgmnt}$ alone. In terms of decreasing physical resource usage, $AvEPR$ is the only candidate, since it counts the number of entangling qubits. Table 5.3 shows T_{exec} increases by a factor of 1.76 ($3161\mu\text{sec}$ to $5571\mu\text{sec}$) for 7 fold

Table 5.3: Data for simulations of Figure 5.6. Each entry of the table contains latency and failure rate value of the form $(T_{exec}, P_{success})$. T_{exec} is in μs

Avg. Gen. Time	AvEPR						
	1	2	3	4	5	6	7
6500	41695,0	22063,0	15385,0	9014,0	9074,0	9014,0	3161,1
5500	35653,0	19040,0	13394,0	8134,0	8071,0	8311,0	3201,1
4500	29511,0	16021,0	11574,1	7014,1	7464,1	7014,1	2992,1
3500	23634,0	12830,0	9511,1	6640,1	6000,1	6014,1	2971,1
2500	17631,0	9952,0	7574,1	5200,1	5081,1	5124,1	3071,1
1500	11454,0	7014,1	5344,1	4410,1	4084,1	4004,1	2864,1
500	5571,1	4070,1	3571,1	3464,1	3182,1	3002,1	3151,1

reduction in *AvEPR* (7 to 1). Note that this is the maximum attainable reduction for *AvEPR* since other physical resources are assumed to be unlimited. However once *NP* and *NAnc* values are bounded then the improvement can only decrease due to other constraints.

5.5 Discussions

5.5.1 Tool Testing, Verification and Validation

Verification of tool requires that the implementation of its components meet respective specifications. In our case, we mainly require that our Mapping and Scheduling algorithms:

- Shall handle specified constraints and inter-ELU gates
- Shall minimize total execution time of the circuit.

To check whether algorithms meet these requirements, both Mapper and Scheduler can be verified independently by analyzing our tool design flow, formulation of sub-problems and their optimization objectives.

Validation requires that tool gives correct result, i.e., it works according to our expectation under given settings. In a constraint-centric tool, we can compare the

simulation results with anticipated correct behavior by setting up important test cases. These cases can be chosen either according to our initial specifications, expectations or known results from literature. For example we can validate our tool by verifying the increase in total execution time and failure rate as available resources are reduced as shown in section 5.4. Scheduler can be validated if it can generate a complete schedule with minimum number of ancilla qubits for given task, e.g., five ancilla for simple Steane error correction circuit. Individual components of the tool can also be validated independently. In order to validate the Mapper we can verify if it maps frequently interacting qubits to closely placed ions in the hardware. Similarly, we can further validate Scheduler by performing two tests. First, under no constraints, the execution time of the input circuit should be equal to the sum of execution times of all the gates in the critical path of the circuit. Secondly, if constraint are tightest, (e.g. $NP = 1$) execution time should be at least the sum of execution times of the all the gates in the quantum circuit.

5.5.2 *Running Time and Scalability of the Tool*

Scalability mainly depends on running time, used as a yardstick to measure the tool efficiency. All simulations were done for circuit in Figure 5.3 on a system based on Intel®Core i3 CPU M370@2.4GHz dual-core processor, containing 1GB of available physical memory. We find that some constraint variables can adversely effect the running time while others are more benign.

Figure 5.7 shows that running time is an increasing function of constraints on $NAnc$ and constant in NP under same $AvEPR$; as $NAnc$ decreases, running time increases in general and vice versa. Notice that sharpest increase in running time occurs when we approach the minimum permissible value of $NAnc = 5$. This happens because, as constraint on $NAnc$ becomes tighter, Scheduler may revisit same gate operation several times before dispatching it for execution. For small $NAnc$ values,

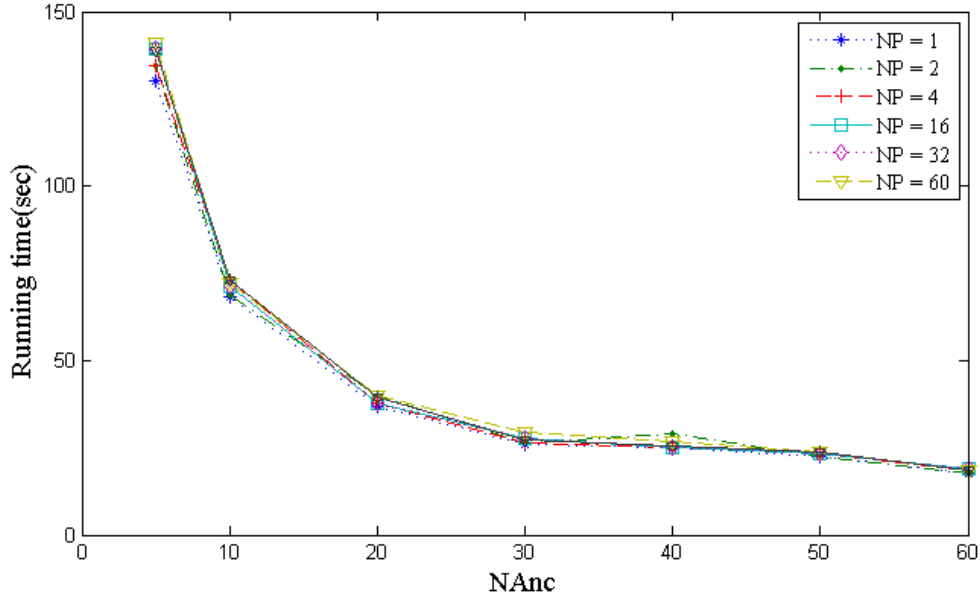


FIGURE 5.7: Tool running time as a function of constraints on $NAnc$ and NP under no limitations on $AvEPR$

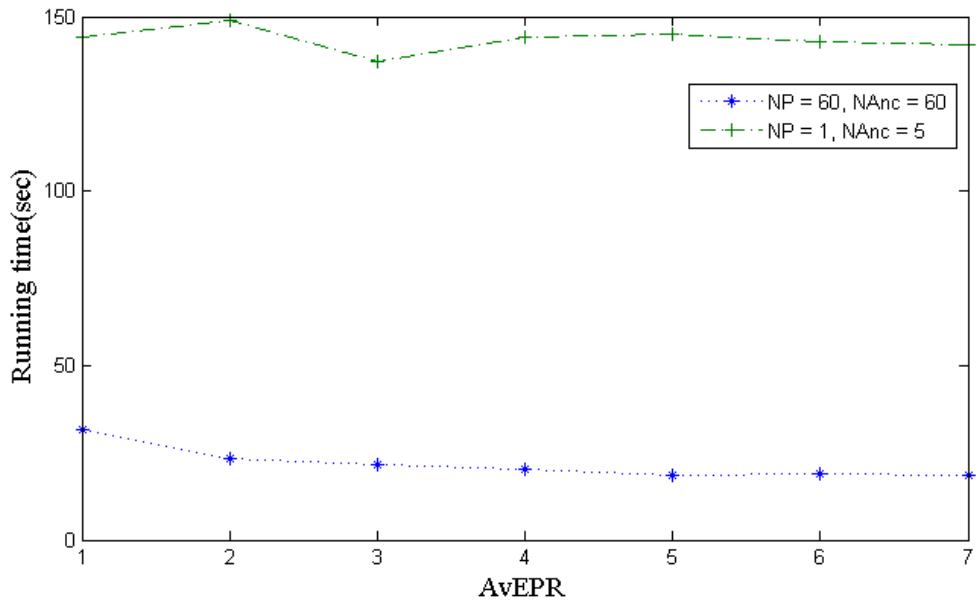


FIGURE 5.8: Tool running time as a function of constraints on $AvEPR$ under loosest (lower curve) and tightest (upper curve) constraints on NP and $NAnc$

very few candidate gates using ancilla are selected for execution in each iteration. In the current implementation of Scheduler, it may optimistically reconsider rejected gates in the previous iteration as candidates for executable gates in current iteration. This leads to several revisits to ancilla gates and dramatic increase in running time. On the other hand, time is roughly constant with NP since Scheduler uses a buffer to store when previously dispatched gates are finishing and successfully finds time slots for all executable gates in the current iteration. Low priority gates competing under NP constraints may be scheduled for execution later but their start time is always known and not rejected in a given iteration. This is evident from the six indistinguishable curves, each corresponding to a different value of NP in Figure 5.7.

Figure 5.8 shows that lower $AvEPR$ value can modestly increase running time when NP and $NAnc$ are unconstrained. Like $NAnc$ case, once available e-qbts are occupied for higher priority inter-ELU gates, Scheduler may not know precisely when these e-qbts will be released, re-entangled and available for other inter-ELU gates. As discussed earlier, this uncertainty increases ‘less productive’ Scheduler iteration, which leads to increased running time. The increase shown in the bottom curve is small, since number of inter-ELU gates in the circuit were few: seven in the whole circuit. The upper curve shows that under smallest permissible values for NP and $NAnc$, the effect of $AvEPR$ on running time seems virtually negligible. This is due to the fact that NP and $NAnc$ values constitute bulk of total resources in the simulation.

The effect of $NAnc$ or $AvEPR$ on the running time of scheduler can be minimized as follows. Rejected gates in previous iteration will keep a flag variable which will help Scheduler avoid revisiting unnecessary gates in subsequent iteration. Scheduler can also be modified to pre-calculate time at which occupied ancilla or e-qbts will be released so that it can generate output more efficiently if calculations are correct. In the next version of our tool we would like to try and make these changes that might

dramatically reduce the running time.

We conclude this section by discussing the efficiency of the Mapper. The placement algorithm in the Mapper scales very well since the main complexity comes from calculating eigenvectors of the adjacency matrix which can be done in polynomial time complexity. However because the problem of linear placement is NP-Complete we can't always guarantee the best solution. The partitioning algorithm used in this chapter was based on integer programming gives exact solution which can scale up to 20-35 logical qubits as tested on our system. For small sized quantum circuit exact solution can be found, however it becomes quickly infeasible as circuit size begins to grow. In this next chapter we use placement algorithm for partitioning problem as described in section 4.5.2. Thus we can substantially speedup our mapping module.

5.6 Summary

While the mapping and scheduling algorithms used in our tool are well-known in classical CAD tools, there are distinct considerations in scheduling quantum circuits arising from the unique properties of the given quantum computer hardware. The vulnerability of qubits to decoherence forces the quantum hardware to be heavily encoded in quantum error correcting codes, that dictates the structure of quantum circuits at low levels. Since qubits cannot be duplicated, the execution of error correction procedures requires fresh ancilla qubits as extra resources. Unlike classical integrated circuits hardware, the ion trap system does not provide high levels of parallel operation in the hardware. Therefore, the efficiency of quantum circuit execution depend strongly on the availability of these hardware resources and constraints.

In this chapter, we presented simple performance simulation for quantum circuits under such limited hardware resources, by considering the mapping and scheduling of quantum circuits on a simple architecture built on MUSIQC hardware. Using this tool, we showed that significant reduction in physical resource was possible at the cost

of modest increase in latency, and potential trade-offs between constraint variables for given performance were identified. These relations could give architects useful insights into appropriate investment of physical resources in the quantum computer hardware. Furthermore, we carefully analyzed the scalability and running time of toolbox and proposed different ways to make the tool scalable, so that large sized quantum circuits could be efficiently analyzed. In the next chapter we describe how to performance simulation sizable benchmark quantum circuit.

Optimization of a Quantum Computer Architecture

In the previous chapter we analyzed the impact of various hardware constraint variables on the performance of quantum algorithm. In order to systematically analyze the larger design space spanned by increasing number of such variables, their adequate categorization becomes inevitable. In this chapter we show that separating physical attributes of device (called device parameters or DPs) from architecture description opens interesting framework for systematically optimize design for larger quantum systems.

We highlight the role of DPs and architecture by quantifying the performance of a fully error-corrected 1,024-bit Quantum Carry Look-Ahead Adder on a modular, reconfigurable architecture based on trapped ions. We extend our simulation tool that estimates the performance and resource requirements for running a quantum circuit on various quantum architectures as a function of the underlying DPs. Using this tool, we found that (1) the latency of the adder circuit execution due to slow entanglement generation process for qubit communication can be adequately eliminated with a small increase in entangling qubits, and (2) the failure probability of the circuit is ultimately determined by the qubit coherence time, which needs to

be improved in order to reliably execute the adders comprising core of the Shor's algorithm.

6.1 Motivation of Study

Performance of computing systems in the early stages of development is dictated by the quality of component devices in the underlying hardware. Current quality of available quantum bits (qubits), and the logic operations and interconnects between them calls for quantum error correction and fault-tolerant construction, which requires substantial resource overhead in implementing a functional quantum computer. A number of scalable architecture designs have been proposed [Metodi et al. \(2005\)](#); [Van Meter et al. \(2008\)](#); [Whitney et al. \(2009\)](#); [Kim and Kim \(2009\)](#); [Monroe et al. \(2014\)](#); [Galiautdinov et al. \(2012\)](#); [Fowler et al. \(2012a\)](#), but a process of optimizing the architectural choices for the best system performance based on rapidly evolving device parameters (DPs) of the hardware platform remains a challenging problem.

The performance of a quantum architecture can be quantified using adequate simulation tool. However, the accuracy of the estimated performance outputted from the tool hinges on the ability of the tool to simulate realistic constraints of the hardware architecture, such as the availability of scarce resources and the ability to transport qubits from one place to another within a large scale quantum system. In this chapter, we extend our previous version of the tool to include flexibility adequate for the use in architectural optimization as the parameters of the component devices change. In particular, this simulator is improved to

- handle reconfigurable quantum hardware, where device parameter (DP) values can be adjusted
- handle reconfigurable quantum architecture, where resource investment scenar-

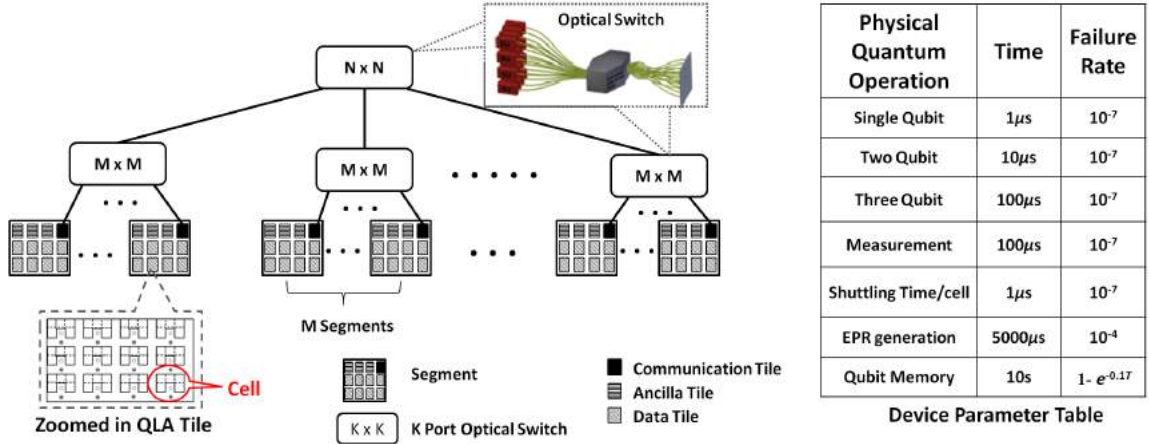


FIGURE 6.1: Overview of the reconfigurable quantum computer architecture analyzed in our performance simulation tool.

ios can be studied

- output concrete breakdown of performance metrics (e.g. total execution time and failure probability) and contents of resource overhead (such as qubits used as ancilla or for communication purposes). This feature allows the user to readily identify performance bottlenecks.

Using the advanced version of our tool, we analyze the performance characteristics and resource consumption of scalable and flexible architecture build on MUSIQC hardware. Our tool allows us to quantify the system performance for a given quantum architecture as a function of the DPs. Given a set of DPs, the architecture space is searched by varying different types of resource investments to improve the performance for a benchmark algorithm, a Quantum Carry Look-Ahead Adder (QCLA) Draper et al. (2006) in our example. This resource-performance trade-off is measured by and optimized for the Qubit-Delay product (QD), a metric which captures the effect of performance gain in execution time per unit resource investment. Once we extract the best performance from the optimized architecture, the result naturally exposes the DPs that impose bottleneck on the system performance. This approach

provides concrete guidelines for the experimental research to improve the critical DPs towards the construction of large scale quantum computers. We organize the rest of the chapter as follows: section 6.2 describes the quantum computer architecture used in our simulation. Section 6.3 briefly details our toolset, and the results and relevant discussions are given in Section 6.4. Section 6.5 concludes the study with possible directions for the future work.

6.2 Quantum Hardware and Architecture Models

In this paper, we refer to ‘hardware’ as the underlying physical device technology to perform quantum computation, such as trapped ions, superconducting circuits, and quantum dots. On the other hand, the term ‘architecture’ describes the allocation and arrangement of different types of quantum resources as larger quantum systems are considered. For example, the size and the bandwidth of quantum communication channel and amount of qubits dedicated to fault tolerant quantum operations are parameters of the architecture.

6.2.1 *Quantum Hardware Model*

The underlying hardware assumed for this study is MUSIQC hardware containing qubits stored in two internal states of the atomic ion (e.g., $^{171}\text{Yb}^+$ ion Olmschenk et al. (2007)), described as a two-level spin system, manipulated by focusing adequate laser beams at the target ion(s). The ion qubits are assumed to be individually addressable Knoernschild et al. (2010), and a single-qubit quantum gate is accomplished by a simple application of laser pulse(s) on the qubit in its original location. The two-qubit gate, on the other hand, requires that both ions are brought in ‘proximity’ before the laser pulse(s) are applied. In our model, there are two ways to achieve this proximity using two different types of physical resources: the ballistic shuttling channel (BSC) and the entanglement link (EL) Monroe et al. (2014). BSC

provides a physical channel through which an ion can be physically transported from its original location to the target location by carefully controlling the voltages of the electrodes on the ion trap chip. In the EL case, an EPR pair is established between designated proxy “entangling ions” (e-ions) that belong to two independent ion trap chips using a photonic channel. The resulting EPR pair is used by the actual operand ions as a resource to perform the desired gate via quantum teleportation between two ions that cannot be connected by BSC Gottesman and Chuang (1999). It should be noted that the generation time for the EPR pairs is currently a slow process due to technology limitation; however, it can be reduced using pipelined EPR Generation (PEG) at an expense of additional resources Monroe et al. (2014). The details of this hardware model is provided in chapter 3.

Since the physical distance between ions grows naturally as the size of the computer increases, the choice of communication method between ions is an important design consideration for both hardware and architecture. In our hardware model, the communication time is proportional to the physical distance between the qubits in the system for the BSC, and almost independent of the distance for the EL as long as the photonic channel can be established (see Sec. 7.3.2 for details). In this study, BSC is used for the short-distance while EL is used for the long-distance qubit communication.

6.2.2 Quantum Architecture Model

Figure 6.1 shows the hierarchical model of the modular scalable universal architecture chosen for our analysis Monroe et al. (2014) supported by MUSIQ hardware. At the bottom of the hierarchy, architecture maps ELUs to the Tiles made of memory cells and BSC regions Metodi et al. (2005). Memory cells provide storage and manipulation of qubits for quantum gates, whereas BSCs allow rapid transportation of qubits to facilitate multi-qubit gates between memory cells. The physical qubits in the Tile

are encoded into a logical qubit using a quantum error correcting code (QECC) of choice (the Steane code is used in our model), and the error correction is performed at regular intervals to preserve the logical qubit from degrading. Multiple layers of encoding can be concatenated to dramatically improve the protection against errors at the expense of increased number of qubits and circuit complexity used for the error correction. For example a Tile supporting two layers of Steane encoding (L2 Tile) has lower failure probability compared to single layer of encoding (L1 Tile) but contains up to seven times more qubits. In addition to the error correction, Tile is equipped to perform other necessary operation which defines its architectural functionality: state preparation (Ancilla Tile), basic fault tolerant quantum gate operations (Data Tile) and long distance communication between two Tiles (Communication Tile). These Tiles are sufficient to construct a scalable architecture.

At the next level of the hierarchy, multiple Tiles are arranged to form an extended ELU called Segment that performs complex fault tolerant circuit level gate operations. For example, a Segment groups necessary set of Tiles to execute a fault tolerant Controlled-Controlled-NOT (CCNOT, or Toffoli) gate, widely used in quantum arithmetic circuits Draper et al. (2006). The Tiles within the Segment communicate through BSC. As more Tiles are assembled into a Segment, the communication among Tiles within a Segment will require constituent qubits to move over longer distances, where shuttling becomes costly. Therefore at the the third level of hierarchy, Segments are connected to each other through a reconfigurable optical cross-connect switch Kim et al. (2003). The entangling ions comprising the Communication Tiles from any pair of Segments within the system can be connected by establishing an EPR pair between them using the photonic channel. This feature makes the cost of inter-Segment communication (and gates between logical qubits in the corresponding Segments) independent of distance, a unique feature of the MUSIQC architecture that provides significant advantages in implementing useful quantum circuits Monroe

et al. (2014).

The performance of the quantum computer for executing a chosen application will depend on the size and composition of the Segments used in the architecture. By varying the architecture parameters (resources) such as number of Segments (N_{Seg}), total Ancilla Tiles (N_{Anc}), and Communication Tiles per Segment (N_{Comm}), we can minimize the impact of DPs on the performance metrics: execution time T_{exec} and failure probability $1 - P_{succ}$. Identifying specific DPs which become performance bottlenecks for various architectures is the main contribution of this work.

6.2.3 Error Model and Baseline Device Parameters

For the simulations, we assume the following as the baseline DPs. A simplified depolarizing channel (equal probability of bit flip, phase flip) is assumed as the underlying error model Nielsen and Chuang (2000), with fixed physical gate error probability of 10^{-7} for both single-qubit and two-qubit gates. Furthermore, qubit memory error is modeled as an exponential decay in its fidelity $F \sim \exp(-at)$, where a ($=1/T_{coh}$) is determined by the coherence time of the qubit assumed to be $T_{coh} = 10$ s, and t is the time between quantum gates over which qubit sits idle (No-op). It is assumed that a single-qubit gate takes $1\mu s$, a two-qubit gate $10\mu s$, and a three-qubit Toffoli gate and measurement each takes $100\mu s$. For BSC, since the dimensions of the state of the art ion-trap cell described in Monroe and Kim (2013) fall in the \sim mm size range, we use $T_{shutt} = 1\mu s$ as the time it takes for an ion to be shuttled through the cell. For a L2 Tile which contains 600 cells arranged in a 2-D grid, the time to shuttle logical qubit through the Tile is taken as $60\mu s$.

For EL, we assume that the entanglement generation time T_{gen} between two Segments using one optical switch is $5,000\mu s$. The size of the optical switch is restricted to 1,000 ports Kim et al. (2003) which can connect up to 20 Segments using 49 optical fibers/L2 Communication Tile. This means that as the number of N_{Seg}

or $NComm$ increases, multiple optical switch ports will be needed to connect the Segments, which are arranged in a tree-like network topology, as shown in Figure 6.1. The time cost of establishing an entangled pair through such a network will depend on the height H of the switch in the tree hierarchy, as the photons will have to propagate through additional switches incurring more photon loss. We adjust new T'_{gen} as $2^H \times T_{gen}$. The resulting EPR pair has non-zero infidelity characterized by EPR_{inf} which will be taken as 10^{-4} , unless otherwise specified. These DP values are optimistic, but most of them are realistically achievable through aggressive technology development in the near future.

6.2.4 Benchmark Application Algorithm

In order to analyze the performance, one must choose a benchmark algorithm against which the performance of various quantum architectures is compared. We select a 1,024-bit quantum carry look-ahead adder (QCLA) as the benchmark algorithm Draper et al. (2006). Such adder circuit forms the elementary building block of a modular exponentiation circuit that underlies the execution time of Shor's prime number factorization algorithm Vedral et al. (1996b). At the logical level, QCLA can be implemented effectively a CNOT and Toffoli gates and provides an efficient logarithmic-depth adder in terms of overall execution time, provided a mechanism for executing these gates between remote qubits is available in the architecture. The EL channels in MUSIQC is an ideal physical process that provides such capability.

6.3 Tool Description

We saw in chapter 4 that our complete tool consisted of two main components: Tile Designer and Performance Analyzer (TDPA), and Architecture Designer and Performance Analyzer (ADPA). The tool components used in previous chapter belonged to the TDPA (Figure 4.1) as they directly laid out flattened circuit on the hardware.

In this study we leverage same TDPA functionalities to map the fault tolerant circuit blocks to the Tile whose performance is then computed and stored as functions parametrized by DPs similar to that described in the chapter 5. This allows TDPA to produce a spectrum of Tiles for ADPA, with different number of physical qubits having different performance levels depending on DP values. For baseline DPs in Figure 6.1, TDPA computes the performance of a unified L2 Tile (which can act as Data, Ancilla and Communication Tile) in Table 6.1.

The main advancement to the tool is the addition of ADPA, indicated by shaded components in Figure 6.2. The ADPA is designed to map application circuits onto the reconfigurable architecture shown in Figure 6.1 described by user-provided architecture parameters and DPs. Based on these parameters, initial map of available Tiles (obtained from TDPA) to be assembled in the Segments is first generated. During this process, the Assembler maximizes the locality among Data Tiles (computed from the connectivity of logical qubits in the circuit) by solving Optimal Linear Arrangement Problem using efficient graph-theoretic algorithm Juvan and Mohar (1992). The Scheduler of ADPA generates the sequence of gates for QCLA execution by solving the standard Resource-Constraint Scheduling problem using ASAP (As Soon As Possible) heuristic. T_{exec} is minimized by reducing the circuit critical path through maximum utilization of available resources. The Scheduler also maximizes P_{succ} by scheduling error correction on Data Tiles at regular intervals when they sit idle. The overall P_{succ} is computed by Error Analyzer as $\prod_{i=1}^{i=N} (1 - P_{Li})$, where P_{Li} is the failure probability of the i -th logical gate and N is the total number of logical operations in the circuit. The failure probability for each logical gate was pre-computed using Fault Path Counting Method Aliferis et al. (2005). Error Analyzer keeps track of the operational source of each P_{Li} [such as Shuttling, Teleportation, Memory (or No-Op) or Gate] while scheduling is in progress. This way, $1 - P_{succ}$ is considered as a sum of errors for Shuttling, Teleportation, Memory and Gate operations.

Table 6.1: L2 Tile Performance Numbers

Logical Operation	$T_{exec}(\mu s)$	$1 - P_{succ}$
Pauli gate (X, Z)	1	7.4×10^{-24}
Hadamard gate	4	7.4×10^{-24}
CNOT gate	18	4.74×10^{-22}
Toffoli gate	108	2.9×10^{-21}
7-qubit cat-state prep.	780	2.67×10^{-22}
Measurement	814	1.32×10^{-17}
Error Correction	4830	6.86×10^{-16}
State prep. ($ \bar{0}\rangle, \bar{\oplus}\rangle$)	5644	6.99×10^{-16}
EPR pair connection	$T_{gen} + 5662$	6.3×10^{-11}

ADPA also performs breakdown of T_{exec} using critical path analysis, which (1) splits the circuit critical path into transportation latency overhead incurred due to the movement of qubits across the hardware (this includes Shuttling overhead T_{ovhd}^{SHUT} and Teleportation overhead and T_{ovhd}^{TEL}), and (2) calculates total number of logical gates comprising critical path N_{ovhd}^{CRT} that captures the magnitude of serialized execution of parallel operations, enforced by fewer shared resources. The splitting of the performance variables using Performance Metrics Decomposer is a unique feature of our tool, which allows us to identify critical DPs and architecture parameters to drive quantum architecture optimization. It is important to note that the basic version of Performance Metrics Decomposer in this study was further enhanced in the next (and final) version of the toolbox. The interested reader can refer section 4.14 for complete details.

6.4 Simulation Results

Our analysis strategy consists of a sequence of simulation sets, each corresponding to a DP set and an architectural construct. Each set provides performance metrics (T_{exec}, P_{succ}) for executing the benchmark algorithm, and identifies bottlenecks that

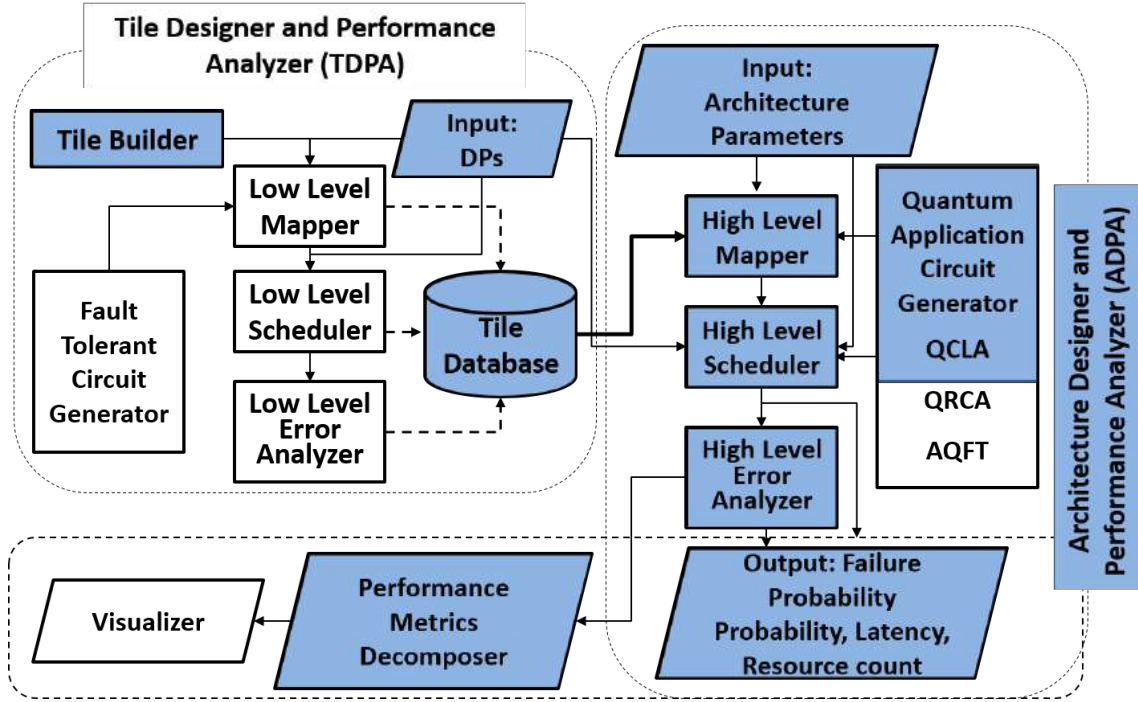


FIGURE 6.2: Shaded blocks representing tool components used in this chapter and their role in the complete toolbox

limit the performance, such as resources or device parameters. Once specific bottlenecks are identified, performance can be improved by adding relevant resources (N_{Anc} or N_{Comm}). Increased resources will enhance performance (T_{exec}) as long as they help mitigate the bottleneck, but cease to impact the performance at some point where the bottleneck is fully removed. Once T_{exec} is reduced to its minimum value, we can tune appropriate device parameters to improve P_{succ} to acceptable values.

However, the search space for the optimal architecture is very large (combinatorial function of N_{Seg} , N_{Comm} and N_{Anc}), and a blind exhaustive search is not an effective approach. Our tool is capable of revealing the detailed breakdown of performance metrics and resource investment for each simulation run. At each stage of the analysis, we can efficiently achieve performance improvement by identifying and then

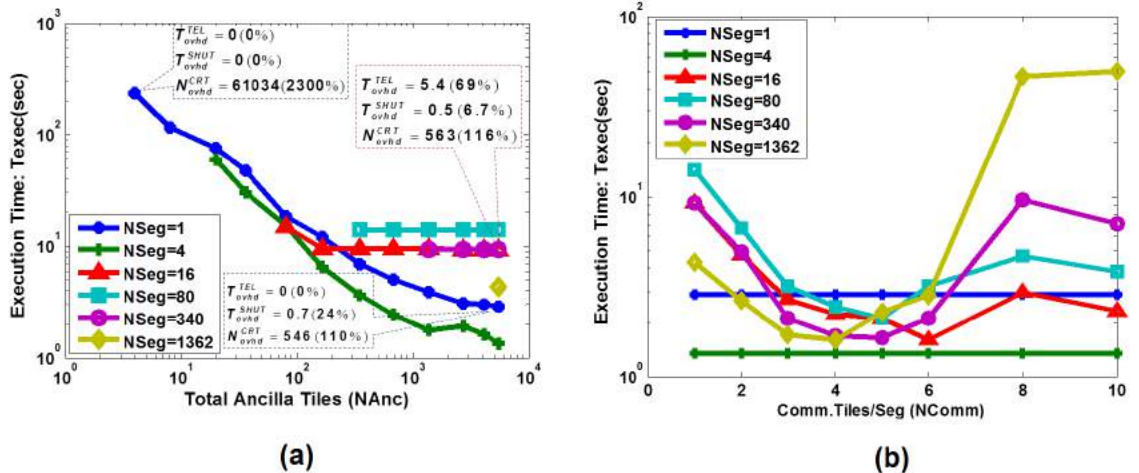


FIGURE 6.3: Execution time plotted as function of (a) $NAnc$ and (b) $NComm$. (a) shows curves for both SA- and Tel- regimes and sample breakdown of T_{exec} for certain points for $NComm = 1$. Each Segment needs to have minimum of 4 Ancilla Tiles for preparation of ancilla state needed for Toffoli gate, so minimum $NAnc$ increases with $NSeg$ and reduces the number of data points for larger $NSeg$. (b) shows T_{exec} as a function of $NComm$ for maximum $NAnc = 5,448$. Note that SA-Regime curves ($NSeg < 16$) remain flat with $NComm$ since T_{ovhd}^{TEL} does not contribute to the execution time.

eliminating specific bottlenecks by adding adequate resources, until the performance is limited by the underlying DPs of the hardware technology. This process significantly simplifies the complicated study of resource-performance trade offs. For the purpose of demonstrating the efficacy of our analysis strategy, we utilize a lumped resource-performance metric QD defined as the product of total number of physical qubits and T_{exec} (qubit-delay product), and look for major trends in architectural choices that minimize this metric. Our search for optimal quantum architecture proceeds in three steps: (1) searching for the architecture space, (2) selecting an architecture, and (3) improving performance through enhanced DPs.

6.4.1 Searching in the Architecture Space

We first search the architecture space for a range of $NSeg$ that represent the “locality” of the quantum architecture (larger $NSeg$ means the computer is more dis-

tributed). For a range of $N\text{Seg}$, the space is traversed by plotting T_{exec} and P_{succ} as a function of $N\text{Anc}$. Figure 6.3(a) shows the total execution time for the benchmark algorithm as additional ancilla qubits are provided to the segments. The architecture space spanned by $N\text{Seg}$ can be categorized into two groups: one group with small $N\text{Seg}$ ($=1$ and 4) in which T_{exec} decreases with $N\text{Anc}$ and the rest with medium to large $N\text{Seg}$ (>16) in which it remains roughly constant after certain amount of $N\text{Anc}$ is provided. The performance of the first set of architectures is limited by the cost of substantial amount of shuttling and limited supply of ancilla, and we call this the Shuttling-Ancilla Regime (SA-Regime). In the second set of architectures, the performance is limited by providing the EL between multiple segments by the slow EPR pair generation process, and is called the Teleportation Regime (Tel-Regime). The curve set for $N\text{Seg} = 16$, in which T_{exec} first decreases with $N\text{Anc}$ and then becomes almost flat, can be considered as a transition point between SA- and Tel-Regimes, and clearly shows a smooth transition between these two regimes.

In the SA-Regime, large number of Data Tiles are clustered into each Segment. Lower $N\text{Anc}$ increases T_{exec} , because a large number of concurrently executable logical Toffoli gates in the QCLA may not be scheduled in parallel due to limited availability of ancilla qubits required for carrying out the logical Toffoli gate Monroe et al. (2014). Increasing $N\text{Anc}$ takes large number of logical Toffoli gates away from the critical path of circuit execution which lowers T_{exec} . In contrast, Tel-Regime features large number of smaller sized Segments leading to decreasing intra-Segment locality among Data Tiles. The inter-Segment non-local gates must be carried out by utilizing EL, and the performance is limited by slow T_{gen} in this regime. The callouts in Figure 6.3(a) highlights the main contributor to the performance bottleneck. For $N\text{Seg} = 1$ case in the SA-Regime, the lack of parallelism due to limited $N\text{Anc}$ is the main bottleneck in the $N\text{Anc}=4$ case, which can be eliminated by increasing $N\text{Anc}$ to 5,448. For the $N\text{Seg} = 16$ case in the Tel-Regime ($N\text{Anc}=5,448$), the main

bottleneck is the teleportation latency overhead due to slow T_{gen} . This teleportation overhead can be mitigated by adding more communication tiles per segment, as shown in Fig 6.3(b). Increasing $NComm$ reduces T_{exec} since EPR pair generation can be parallelized to remove communication latencies on the critical path. In this case, the maximum achievable reduction in T_{exec} is 5.8 (from 9.27 to 1.6) accomplished by increasing total physical qubits by 3.4%. For highly distributed architectures (very large $NSeg$), increasing $NComm$ further decreases the entanglement generation time as the height H of switch tree hierarchy increases, substantially increasing the T'_{eng} .

Figure 6.4 shows the trends of failure probability P_{fail} across the architecture space. We find that P_{fail} in SA-Regime generally increases with $NAnc$, whose dominant component is the error incurred during shuttling, as shown in Table 6.2. In large Segments where most of the data movement is carried out by shuttling, increasing $NAnc$ leads to longer shuttling path lengths due to further distances between the Data Tiles (therefore shuttling error). In Tel-Regime, memory and teleportation errors are dominant components of P_{fail} . Increasing $NAnc$ has minimal impact on P_{fail} in this regime since the contribution of shuttling error is very small. The number of cross-segment teleporations (indicated as Tel in the figure) does not have a big impact on P_{fail} . Table 6.2 shows that although the fraction of teleportation error increases with $NSeg$, the overall P_{fail} remains consistently low for the architectures in the Tel-Regime.

6.4.2 Selecting an Architecture

Once resource bottlenecks are identified for the SA- and Tel- regimes, we identify optimal architectures in each regime by considering both resource budget and performance captured by the Qubit-Delay (QD) product. Figure 6.5 shows the performance (T_{exec} on the left axis) and resources (total number of physical qubits on the right axis) as a function of relevant resource bottlenecks for the SA-Regime [$NAnc$,

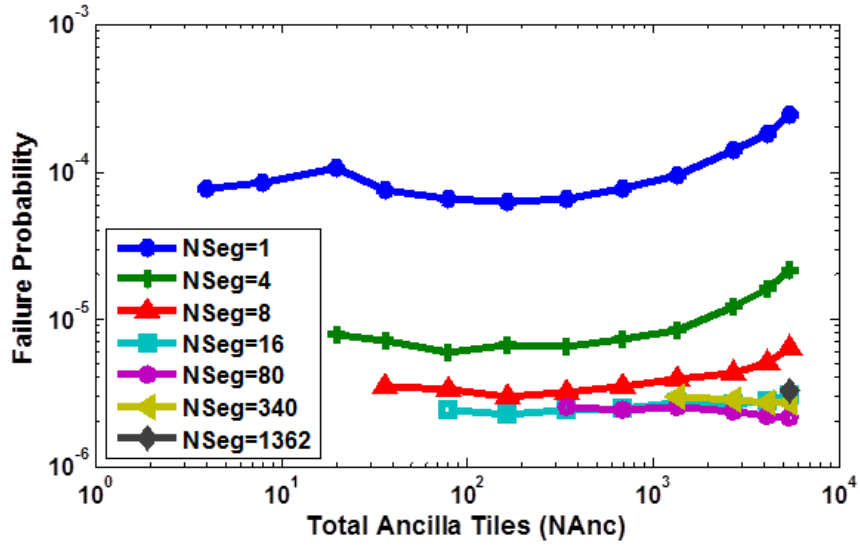


FIGURE 6.4: The failure probability ($1 - P_{succ}$) as a function of $NAnc$. The failure probability shows an overall increasing trend for the curves in SA-Regime, while it remains almost flat in Tel-Regime.

Table 6.2: Breakdown of the $1 - P_{succ}$ for different $NSeg$. The average value is computed over combinations of $NAnc$ and $NComm$.

Segments (NSeg)	% Shuttling Error	%Teleportation Error	% Memory Error	Average $1-P_{succ}$
1	$\geq 96.9\%$	$= 0\%$	$\leq 3.1\%$	1.05×10^{-4}
4	$\geq 66.4\%$	$\leq 0.8\%$	$\leq 33.4\%$	9.9×10^{-6}
8	$\geq 34.4\%$	$\geq 1.33\%$	$\geq 25\%$	6.04×10^{-6}
16	$\leq 48.6\%$	$\geq 2.86\%$	$\geq 48.7\%$	3.1×10^{-6}
80	$\leq 4.3\%$	$\geq 12.5\%$	$\geq 77.7\%$	2.05×10^{-6}
340	$\leq 2.7\%$	$\geq 25\%$	$\geq 69\%$	2.54×10^{-6}
1362	$\leq 0.2\%$	$\geq 36.7\%$	$\geq 45.3\%$	2.7×10^{-6}

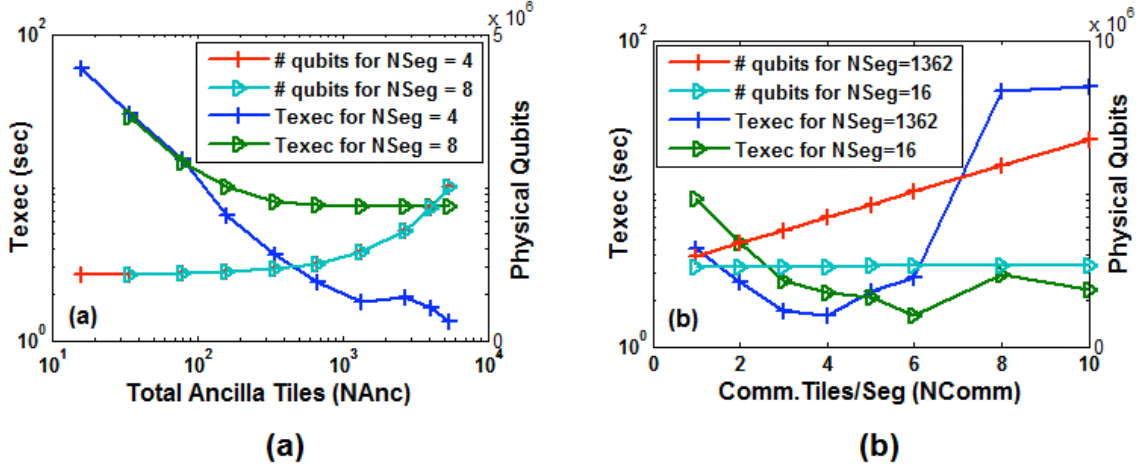


FIGURE 6.5: Comparison of candidate architectures in each regime. Explicit variation of T_{exec} and total number of physical qubits for selected architectures in SA-Regime is shown in (a) while those for Tel-Regime is shown in (b).

Figure 6.5(a)] and Tel-Regime [$NComm$, Figure 6.5(b)]. For clarity, we present two suitable examples from each regime (among all the architectures analyzed) that yield smallest QD minima.

For the architectures in the SA-Regime, we show two representative simulations for $NSeg = 4$ and 8 in Figure 6.5(a). As $NAnc$ increases, T_{exec} for the less distributed case ($NSeg=4$) continues to decrease, whereas the more distributed case saturates as the computation becomes limited by T_{gen} . Since the total number of qubits in both cases are the same, we see that ($NSeg=4$, $NComm=1$, $NAnc=1,362$) is an optimal case that minimizes QD metric in the SA-Regime. This is confirmed by comparing the QD metric for other $NSeg$ values in this regime. The performance metric for running the benchmark algorithm is ($T_{exec} = 2.22s$, $1 - P_{succ} = 8.86 \times 10^{-6}$) for this case.

For the architectures in the Tel-Regime, we show two representative simulations for $NSeg = 16$ and 1,362 in Figure 6.5(b), when sufficient $NAnc$ is provided (5,448). From the comparison of T_{exec} , we see that both architectures reach the minimum

Table 6.3: Failure probability vs. memory error rate and shuttling time. $T_{gen} = 5000\mu s$. $NComm = 1$, $NAnc = 1,362$. #phys. qubits = 1,439,788.

Memory Error Rate: $1 - e^{-\alpha T}$	Shuttling Time : T_{shutt}			
	1us	0.7us	0.4us	0.1us
$\alpha = 0.1$	8.86×10^{-6}	3.7×10^{-6}	2.62×10^{-6}	2.46×10^{-6}
$\alpha = 0.07$	2.01×10^{-6}	9.07×10^{-7}	6.35×10^{-7}	5.98×10^{-7}
$\alpha = 0.04$	2.21×10^{-7}	1.02×10^{-7}	7.35×10^{-8}	6.96×10^{-8}
$\alpha = 0.01$	7.59×10^{-9}	6.78×10^{-9}	6.67×10^{-9}	6.65×10^{-9}
$\alpha = 0.001$	6.75×10^{-9}	6.41×10^{-9}	6.41×10^{-9}	6.41×10^{-9}
T_{exec}	2.22s	1.71s	1.71s	1.71s

computation time of about 1.3s, although at different values of $NComm$ (4 per Segment for $NSeg = 1,362$ case, and 6 per Segment for $NSeg = 16$ case). However, the total number of qubits is much less when $NSeg = 16$, so this is a better architecture in terms of QD metric. For this architecture, T_{exec} decreases about 6 folds for nominal (4.2 %) increase in total number of qubits. By comparing a range of $NComm$ and $NAnc$ values in this regime, we conclude that ($NSeg=16$, $NComm=6$, $NAnc=1,362$) is the optimal design in the Tel-Regime. The performance metric for running the benchmark algorithm is ($T_{exec} = 1.93s$, $1 - P_{succ} = 2.77 \times 10^{-6}$) for this case. Note that in both cases T_{exec} is within a factor of 2 from its minimum value 1.05s, which can be computed by scheduling the benchmark algorithm under no hardware constraints. Adding more qubits to these architectures cannot provide noticeable performance gains, especially for P_{succ} .

Table 6.4: Failure probability vs. memory error rate and physical EPR pair infidelity. T_{gen} reduced to $250\mu s$ using PEG at the expense of additional communication qubits per port. The T_{shutt} is tuned to $0.1\mu s$. $NComm = 6$, $NAnc = 5,448$. #phys. qubits = 2,733,432.

Memory Error Rate: $1 - e^{-\alpha T}$	Infidelity of physical EPR pair (EPR_{inf})			
	10^{-4}	10^{-5}	10^{-6}	10^{-7}
$\alpha = 0.1$	1.47×10^{-6}	1.38×10^{-6}	1.38×10^{-6}	1.38×10^{-6}
$\alpha = 0.07$	4.19×10^{-7}	3.3×10^{-7}	3.3×10^{-7}	3.3×10^{-7}
$\alpha = 0.04$	1.23×10^{-7}	3.62×10^{-8}	3.62×10^{-8}	3.62×10^{-8}
$\alpha = 0.01$	8.77×10^{-8}	9.58×10^{-10}	9.43×10^{-10}	9.43×10^{-10}
$\alpha = 0.001$	8.76×10^{-8}	8.18×10^{-10}	8.03×10^{-10}	8.03×10^{-10}

Table 6.5: Effective execution time (in days) of running 1,024-bit modular exponentiation circuit on optimized architectures

Memory Error Rate $1 - e^{-\alpha T}$	SA-Regime Optimal Architecture				Tel-Regime Optimal Architecture			
	Shuttling Time (μs)				Infidelity of EPR pair (EPR_{inf})			
	1	0.7	0.4	0.1	10^{-4}	10^{-5}	10^{-6}	10^{-7}
$\alpha = 0.04$	886	133.7	112.1	109.7	175.9	104.5	104.5	104.5
$\alpha = 0.01$	106	81.4	81.3	81.3	137.6	89.7	89.7	89.7
$\alpha = 0.001$	105.6	81.2	81.2	81.2	137.5	89.6	89.6	89.6

6.4.3 Improving Performance through Device Parameters

Failure probability for selected architecture can be reduced by improving the dominant sources of errors in respective regimes which can be identified from Table 6.2. In the SA-Regime, the dominant source of error is memory error while waiting for shuttling operation to complete. By either speeding up the shuttling speed or increasing coherence time of the memory qubits, $1 - P_{succ}$ can be reduced dramatically, as shown in Table 6.3. For example, when T_{coh} is increased 10 fold and T_{shut} is accelerated

slightly from $1\mu\text{s}$ to $0.7\mu\text{s}$, $1 - P_{succ}$ falls drastically by 10^3 fold (from 8.86×10^{-6} to 6.78×10^{-9}). Further improvements in the shuttling time improves neither P_{such} nor T_{exec} , indicating that the shuttling time is no longer the performance bottleneck in this limit.

In the Tel-Regime, EPR pair fidelity and memory errors are the main contributors to the failure probability. In order to further reduce the memory errors arising from waiting for the EPR pair generation, one must dramatically reduce T_{gen} , which can be achieved by pipelined EPR generation (PEG) process Monroe et al. (2014). For the numbers shown in Table 6.4, we assume a 20-fold reduction in T_{gen} using the PEG technique, as well as sufficient ancilla qubits ($N_{Anc} = 5,448$). A ten fold increase in T_{coh} provides more than 10^3 fold reduction in $1 - P_{such}$ (from 1.38×10^{-6} to 9.58×10^{-10}) if the infidelity of the EPR pairs is improved to 10^{-5} .

In order to execute 1,024-bit modular exponentiation circuit which comprises bulk of Shor’s algorithm, about 4 million calls to adder is required Van Meter and Itoh (2005). Thus we need each 1,024-bit adder to be successfully executed with failure probability of $\ll (4 \times 10^6)^{-1} = 2.5 \times 10^{-7}$. The shaded regions in Table 6.3 and Table 6.4 provides combination of memory error rates with shuttling time or EPR pair infidelity which achieve $1 - P_{succ}$ required to reliably execute 1,024 bit modular exponentiation. Recall from section 1.3.3, that due to the non-zero failure probability, Shor’s algorithm may be executed multiple times until the input integer is successfully factored. We defined the ‘effective execution time’ by taking into the account, the number of repetition required for the successful execution of quantum algorithm. For 1,024-bit modular exponentiation containing 4 million calls to the adder circuit, each with execution time: T_{exec} and fails with probability P_{fail} , we define effective execution time as: $\frac{4 \times 10^6 \times T_{exec}}{(1 - 4 \times 10^6 \times P_{fail})}$. For the shaded regions in Table 6.3 and Table 6.4, the effective execution time for the 1,024-bit modular exponentiation running on

optimal architectures in SA- and Tel-Regime is shown in Table 6.5. It can be seen that in order to successfully execute bulk of Shor’s algorithm in three months, we need to increase the coherence time to 100 sec ($a = 1/100$) for both architectures. In addition, we need to tune architecture specific critical device parameters to lower the effective execution time below the desired value. For SA-Regime, shuttling time should be $0.7\mu s$ or less whereas for Tel-Regime, the infidelity of EPR pair should be kept 10^{-5} or lower, to ensure that successful execution of entire modular exponentiation circuit takes no more than three months.

6.5 Summary

In this chapter, we highlighted the impact of device parameters and architectures on the performance of quantum systems using a resource-performance simulation tool. This simulation analysis shows that using baseline device parameters, architecturally optimized system can achieve T_{exec} limited by the ancilla in SA-Regime and communication qubits Tel-Regime. However, the error performance was not quite sufficient to execute a 1,024-bit Shor’s algorithm. The coherence time of the memory qubits is the critical device parameters that limits the error performance, and this aspect of the performance cannot be improved by complementing the system with more resources. These DPs were systematically identified by analyzing the breakdown of performance bottlenecks once sufficient resource have been invested in the system. In both SA- and Tel-Regimes, we see that an order of magnitude improvement in the coherence time of the qubits dramatically enhances the error performance (by a factor of 10^3), and ensure successful execution of a practically useful algorithm. The analysis was facilitated by adding advanced flexibilities in the simulation tool that will allow complete analysis of resource-performance trade-off analysis for a given set of technology parameters. We provide an effective and systematic approach to such optimization in a vast architecture-device parameter space.

Designing a Million-Qubit Quantum Computer

In the last chapter we proposed a methodology of designing quantum computer for one type of application circuit. However, many interesting problems such as integer factorization comprise contrastingly structured blocks. Hence it is important to analyze the proposed design method across range of benchmark circuits. In this chapter we analyze all essential building blocks of Shor’s factorization algorithm and identify important architecture and device parameters (DPs) which are crucial for the overall performance. We also investigate the scalability of the chosen design with the increasing problem size and introduce “general-purpose quantum computer” which can be reconfigured to match the resource-performance needs of given benchmark circuit.

The optimal design for such general-purpose quantum computer depends on appropriate balance of different types of hardware units performing their roles within the application. This balance can be evaluated using a simulation tool capable of accurately modeling the scheduling of quantum logic operations on a given quantum hardware. In this chapter we extend our performance simulation tool to be fully capable of analyzing the performance bottlenecks and visualizing resource utiliza-

tion that provides a means to validate the optimality of the architecture. Using this tool, we search for an optimal design of a modular quantum computer architecture based on trapped ions, over several benchmarking circuits crucial for Shor’s factoring algorithm. We demonstrate that an efficient quantum computer capable of factoring a 2,048-bit number can be constructed in three million physical qubits.

7.1 Motivation of Study

Although quantum computers can in principle solve important problems such as factoring a product of large prime numbers efficiently, the prospect of constructing a practical system is hampered by the need to build reliable systems out of faulty components. Fault-tolerant procedures utilizing quantum error correcting codes (QECC) achieve adequate error performance by protecting the quantum information from noise, but come at the expense of substantial resource investment Nielsen and Chuang (2000). The quantum threshold theorem says that a quantum computation of arbitrary size can be performed as long as the error probability of each operation is kept below a certain threshold value and sufficient computational resources such as the number of quantum bits (qubits), can be provided to implement adequate fault-tolerant construction Aharonov and Ben-Or (1997). Although this is an encouraging theoretical result, an accurate estimate of the resource overhead remains an extremely complex task, as it depends on the details of the hardware (qubit connectivity, gate speeds and coherence time, etc.), the choice of protocols (QECC, etc.), and the nature of the target algorithms. Several application-optimized architectures have been proposed and analyzed Metodi et al. (2005); Van Meter et al. (2008); Whitney et al. (2009); Kim and Kim (2009); Monroe et al. (2014); Galiutdinov et al. (2012); Fowler et al. (2012a), yet the accurate quantification of resource-performance scaling for different benchmarks remains a challenging problem.

In this study we also quantitatively define the ‘scalability’ of a quantum computer

architecture to mean that the resource overhead of running a quantum algorithm, while sustaining expected behavior in execution time and success probability (of order unity, $\sim O(1)$), increases linearly with the problem size. We propose a modular ion-trap-based QC architecture and quantify its scalability for three different benchmark circuits crucial for Shor’s factoring algorithm Shor (1997): a quantum carry look-ahead adder (QCLA) Draper et al. (2006), the CDKM quantum ripple-carry adder (QRCA) Cuccaro et al. (2004), and an approximate quantum Fourier transform (AQFT) Fowler and Hollenberg (2004). This architecture features fast and reliable interconnects to ensure efficient access to computational resources, and enables flexible distribution of computational resources to various workload-intensive parts of the quantum system depending on the circuit being executed. By evaluating this architecture for a variety of benchmarks, we show that it can achieve optimal performance by flexible and efficient utilization of given resources over a range of interesting quantum circuits.

To quantify the performance of an architecture as a function of available resources, we utilize our performance-simulation tool similar to those reported in Refs. Svore et al. (2006a); Whitney et al. (2007); Balensiefer et al. (2005b); Whitney et al. (2009) that (a) maps application circuits on to the quantum hardware, (b) generates and schedules the sequence of quantum logic gates from the algorithm operating on the qubits mapped to the hardware, and (c) estimates performance metrics such as total execution time and failure probability. Unlike the tools reported previously, our tool features unique capabilities to (1) simulate performance over varying hardware technology parameters, (2) allow dynamic resource allocation in the architecture, (3) provide detailed breakdown of resource and performance variables, and (4) enable visualization of resource utilization over (5) a range of benchmark applications. By leveraging these unique attributes we search the architecture space for a suitable design while providing valuable insights into the factors limiting performance in a

large-scale quantum computer. It is highly recommended to review chapter 4 which comprehensively covers final version of the tool used in this study.

Our analysis proceeds in four steps. First, we demonstrate the scalability of our proposed architecture in the presence of resource constraints by simulating the resource-performance trend as a function of the size of benchmark circuits. Next, we analyze the dependence of the performance metrics on the available resource across these circuits, and show that the performance of a resource-intensive circuit degrades in the presence of resource constraints. As an example, the advantage of the logarithmic-depth QCLA over the linear-depth QRCA can be sustained only up to a certain problem size in a given hardware, under which the necessary parallel operations can be effectively exploited. We then provide an optimal computer design under the resource constraints, which our tool found by exhaustively searching the large combinatorial space spanned by the set of architecture parameters. Once execution time is optimized, we identify relevant device parameters to achieve minimum failure probability to schedule 2,048-bit Shor’s algorithm.

This chapter is organized as follows: section 7.2 describes benchmark quantum circuits and their characteristics. Section 7.3 describes the underlying quantum hardware technology and the modular, reconfigurable quantum architecture used in our simulation. The section 7.4 details the toolset and its main features. The simulation results along with detailed discussions are given in section 7.5. The section 7.6 concludes with the summary of insights gained from this study.

7.2 Quantum Circuits Revisited

7.2.1 *Universal Quantum Gates for the Steane code*

Quantum circuits consist of a sequence of gates on qubit operands. An n -qubit quantum gate perform a deterministic unitary transformation on n operand qubits. In the terminology of computer architecture, a gate corresponds to an “instruction” and

the specific sequences of gates translate into instruction-level dependencies. Similar to classical computers, it is known that an arbitrary quantum circuit can be constructed using a finite set of gates (called *universal quantum gates*), which is not unique) Nielsen and Chuang (2000). For fault-tolerant quantum computation, one has to encode the qubits in a QECC, and perform logic gates on the encoded block of logical qubits Nielsen and Chuang (2000).

There are two ways of performing gates on a logical qubit: in the first is where the quantum gate on logical qubit(s) is translated into a bit-wise operation on constituent qubits (referred to as a “transversal” gate). Since an error on one constituent qubit in the logical qubit cannot lead to an error in another constituent qubit in the same logical qubit, the error remains correctable using the QECC and therefore a transversal gate is automatically fault-tolerant. For a good choice of QECC, most of the gates in the universal quantum gate set are transversal, and therefore fault-tolerant implementation is straightforward. Unfortunately, for most of the QECC explored to date (the class of QECC called additive codes), it is impossible to find a transversal implementation of all the gates in the universal quantum gate set Zeng et al. (2011). A second, general procedure for constructing such gates involves fault-tolerantly preparing a very special quantum state, called the “magic state”, and then utilizing quantum teleportation to transfer the operand qubit into the magic state to complete the gate operation Zhou et al. (2000). This operation is generally much more resource intensive and time consuming, so minimizing such operations is a crucial optimization process for the fault-tolerant circuit synthesis.

We employ the widely used Steane code which invests seven qubits to encode one more strongly error-protected qubit. For the universal gate set, we utilize $\{X, Z, H, \text{CNOT}, \text{Toffoli}\}$ for the adder circuits (both QCLA and QRCA), and $\{X, Z, H, \text{CNOT}, T\}$ for the AQFT circuit. For a single qubit state $\alpha|0\rangle + \beta|1\rangle$, X and Z correspond to the bit-flip and phase-flip operation that takes the state to

$\alpha|1\rangle + \beta|0\rangle$ and $\alpha|0\rangle - \beta|1\rangle$, respectively, and span a Pauli group of operators. H is the Hadamard operator, which converts the computational basis states $|0\rangle$ and $|1\rangle$ to the equal linear superposition of two states $|\pm\rangle = (|0\rangle \pm |1\rangle)/\sqrt{2}$, and vice versa. CNOT is a two-qubit gate where the state of the second qubit (called the target qubit) is flipped if and only if the state of the first qubit (called the control qubit) is $|1\rangle$. Along with the Pauli operators, H and CNOT span a Clifford group of operators (Clifford gates). In the Steane code, all operators in the Pauli and Clifford group can be implemented transversally. However, in order to complete the set of universal quantum gates, a non-Clifford gate must be added. This could be either the T gate (sometimes called the $\pi/8$ -gate), a single-qubit gate which shifts the phase of the $|1\rangle$ state by $\pi/4$, or the Toffoli gate (sometimes called the Controlled-Controlled-NOT), a three-qubit gate where the state of the third qubit is flipped if and only if the state of the first and second qubits are both $|1\rangle$. Using either gate is equivalent, in the sense that a Toffoli gate can be constructed using several T gates and Clifford gates Nielsen and Chuang (2000). Fault-tolerant implementation of a non-Clifford gate requires magic state preparation when the Steane code is used. Figure 7.1 shows the fault-tolerant implementation of the Toffoli gate, where the Toffoli magic state is prepared with the help of four ancilla qubits followed by the teleportation of the operand qubits into the magic state.

7.2.2 Benchmark Circuits

Figure 7.2 shows that Shor's factoring algorithm consists of an arithmetic calculation called modular exponentiation Van Meter and Horsman (2013); Vedral et al. (1996a); Beckman et al. (1996b) which can be constructed from adder circuits, followed by a quantum Fourier transform Shor (1997). Arithmetic circuits like adders (QCLA and QRCA) can easily be constructed from X , CNOT and Toffoli gates, while AQFT can be constructed more conveniently from T gates. We consider a quantum architecture

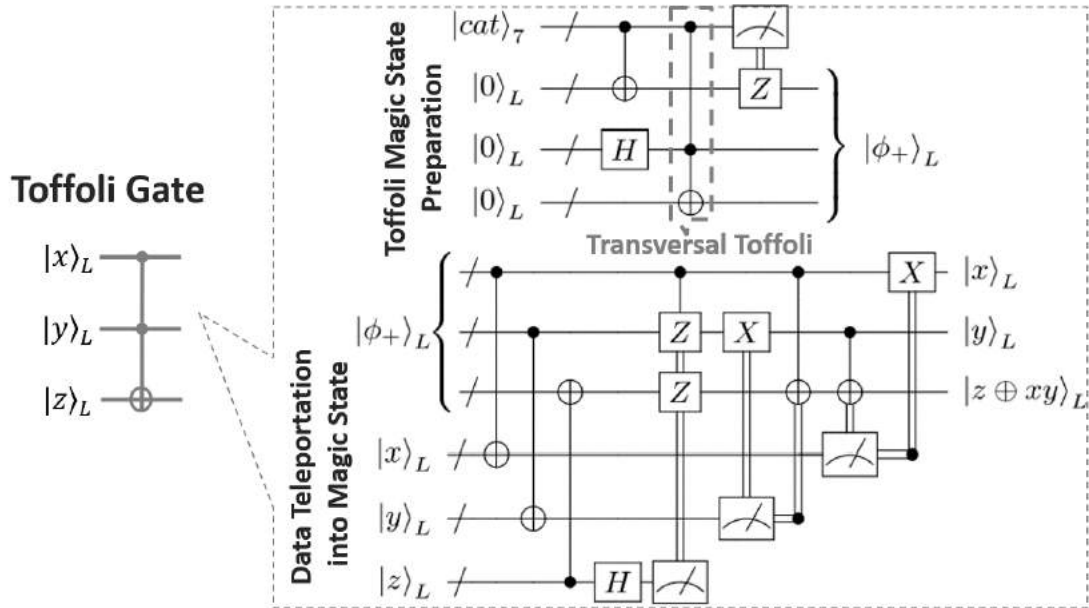


FIGURE 7.1: Fault tolerant circuit for Toffoli gate used in adder benchmarks. $|\alpha\rangle_L$ denotes a logical qubit block representing the state $|\alpha\rangle$.

where both Toffoli and T gates can be executed, and optimize the architecture for executing all required quantum circuits for running Shor's algorithm.

Quantum adders

A large number of quantum adder circuits must be called to complete the modular exponentiation that constitutes the bulk of Shor's algorithm. We select two candidate adders QRCA and QCLA, representing two vastly different addition strategies, analogous to classical adders. QRCA is a linear-depth circuit, containing serially connected CNOT and Toffoli gates: an n -bit addition will consume about $2n$ qubits to perform $2n$ Toffoli and $5n$ CNOT gates Cuccaro et al. (2004). The sequence of these gates is inherently local, and nearest-neighbor connectivity among the qubits is sufficient to implement this circuit. On the other hand, QCLA is a logarithmic-depth $[\sim 4 \log_2 n]$ circuit connecting $4n$ qubits utilizing up to n concurrently executable gates Draper et al. (2006). This circuit roughly contains $5n - 3 \log_2 n$ CNOT and

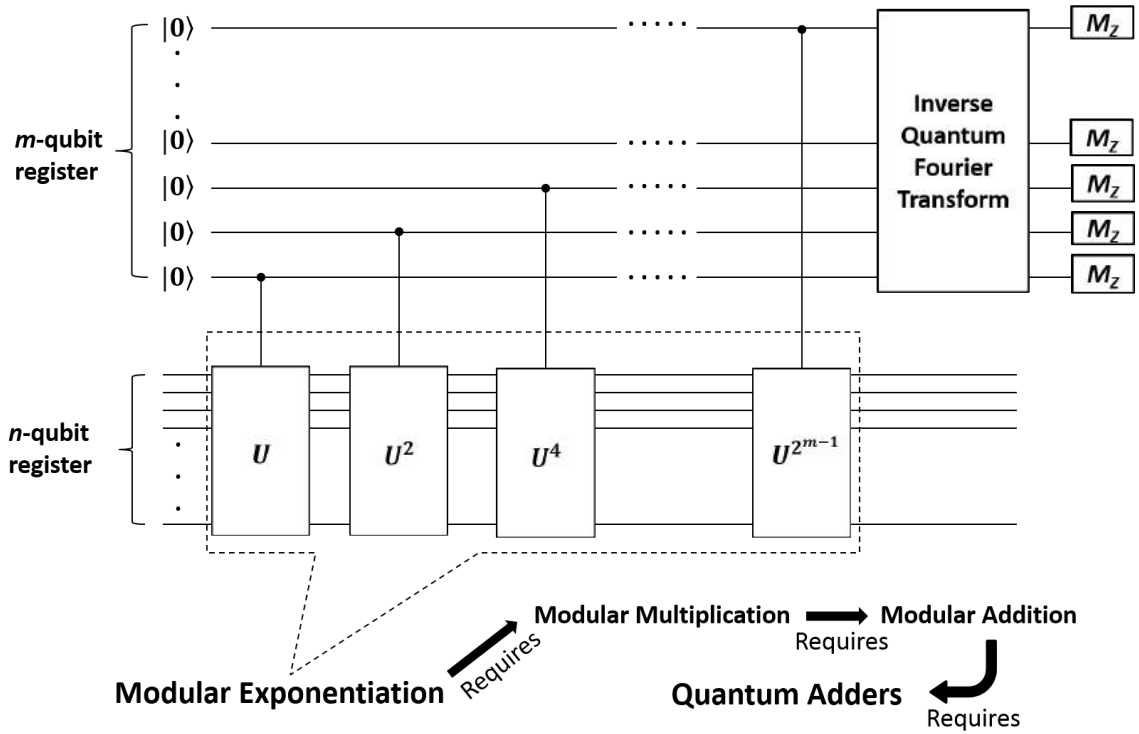


FIGURE 7.2: The block diagram of Shor's algorithm. The Modular Exponentiation (ME) comprise the bulk of circuit. This circuit is constructed from modular multipliers which consists of modular addition requiring quantum adders. The inverse Quantum Fourier Transform is applied as the last step of the algorithm before result is read out.

Toffoli gates for n -bit addition. The exponential gain in performance (execution time) come at the cost of sufficient availability of ancilla qubits and rapid communication channels among distant qubits to exploit parallelism. The quantum hardware model considered here is unique in providing the global connectivity necessary for implementing QCLA. We study the resource-performance tradeoff in selecting QCLA vs. QRCA in Section 7.5.

Approximate quantum Fourier transform

The quantum Fourier transform (QFT) circuit is often used as the order-finding routine in Shor's algorithm Nielsen and Chuang (2000). It contains controlled-rotation gates $R_z(\pi/2^k)$, where the phase of the target qubit is shifted by $\pi/2^k$ for the $|1\rangle$ state if the control qubit is in the $|1\rangle$ state, for $1 \leq k \leq n$, in a n -qubit Fourier transform. Fig. 7.3 shows that the controlled-rotation gates can first be decomposed into CNOTs and single-qubit rotations with twice the angle. These rotation operations are not in the Clifford group for $k > 1$, and must be approximated using gates from the universal quantum gate set Nielsen and Chuang (2000). A recent theoretical breakthrough provides an asymptotically optimal way of approximating an arbitrary quantum gate with a precision of ϵ using only $O(\log(1/\epsilon))$ Clifford group and T gates, and a concrete algorithm for generating the approximation circuit Kliuchnikov et al. (2013); Giles and Selinger (2013).

It has been shown that a QFT circuit can yield the correct result with high enough probability even if one eliminates all small-angle rotation gates with $k > 8$, sufficient to factor numbers as large as 4096-bits Fowler and Hollenberg (2004). The resulting truncated QFT is called approximate QFT (AQFT). The depth of this benchmark circuit is linear in the size of the problem n , and the total number of controlled-rotation gates scales as $16n$. Using the method outlined in Ref. Kliuchnikov et al. (2013), we approximate rotations in our AQFT circuit with a sequence of 375 gates (containing 150 T gates), with a precision of 10^{-16} . The resulting approximation sequence consists of T (or T^\dagger) gates sandwiched between one or two Clifford gates, whose execution time is negligible compared to the T gate. The execution of the T gate proceeds in two steps: preparation of the magic state $T|+\rangle$ and teleportation of data into the magic state. Since state preparation is much longer than teleportation (78 ms vs 12 ms), we can employ multiple hardware units to prepare magic states to

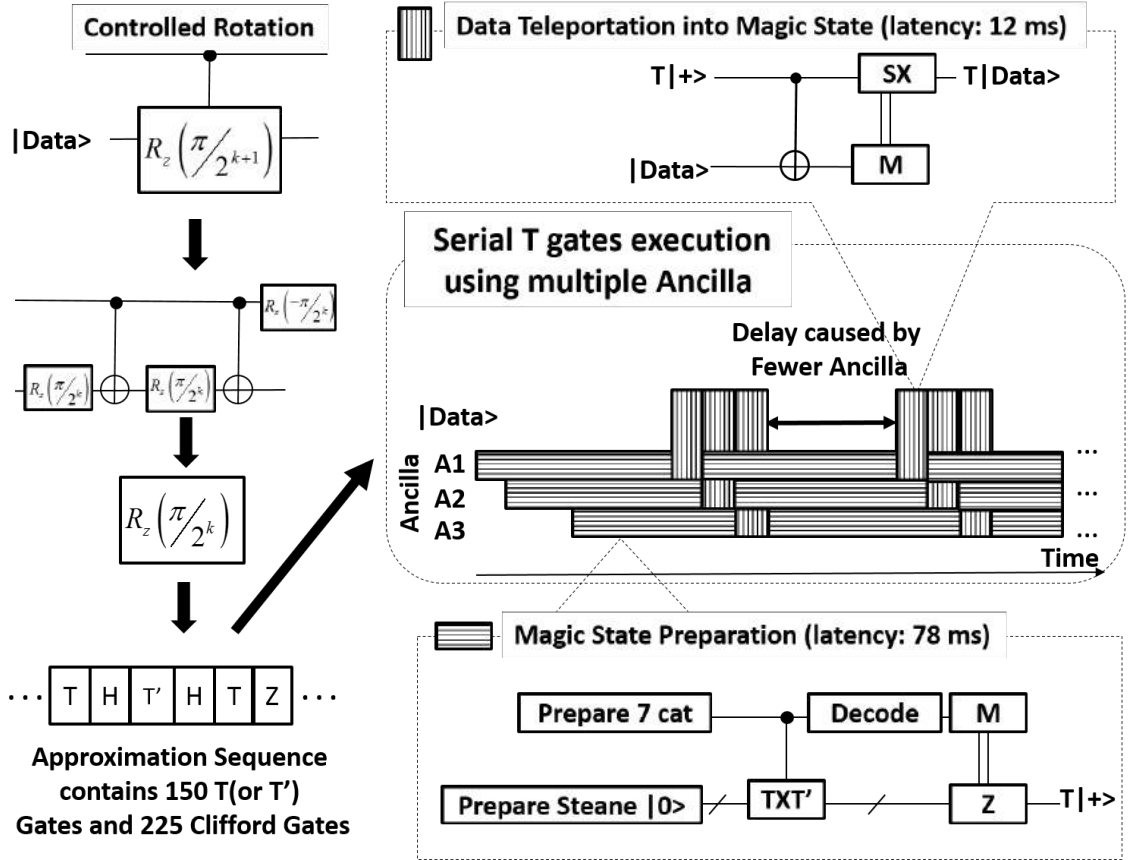


FIGURE 7.3: Fault-tolerant circuit for a controlled-rotation gate used in AQFT benchmark circuit. Small angle rotation gates are approximated by a sequence of T and Clifford gates, and the T gates are performed by magic state preparation and data teleportation.

simulate pipelined execution of T gates. When multiple ancilla qubits are available for the magic state preparation, we can reduce the delay in the execution of the approximation sequence. Using a simple calculation, we can show that the availability of 8 ancilla qubits completely eliminates any delay. When the error correction procedure is inserted in the approximation sequence, its latency can be leveraged to eliminate the preparation delay with even fewer ancilla qubits.

7.3 Quantum Hardware and Quantum Architecture Models

As used in previous chapter, the term *quantum hardware* describes the physical devices used to achieve computation using a specific technology Ladd et al. (2010a) such as trapped ions, atoms, superconductors, or quantum dots. The efficiency and reliability of quantum computation depend on the characteristics of the chosen technology, such as execution time and fidelity of physical gate operations. We describe the physics of the quantum hardware by a set of device parameters. In our simulation, the assumed baseline values for these parameters are optimistic, but can be achieved in the near future through rapid technology advancement. Once quantum hardware technology is specified, we arrange the qubit resources according to their specific roles and their interconnection in order to assemble a large-scale computer. This is captured by the parameters of the quantum architecture. For example, number of qubits dedicated to perform fault tolerant quantum operations and the specification of communication channels are considered architecture parameters.

7.3.1 Quantum Hardware Model

We model quantum hardware based on trapped ions for its prominent properties that have been demonstrated experimentally. Although, trapped-ion technology has been detailed in chapter 3, we would like to facilitate the reader by providing brief summary of the trapped-ion. The qubit in this technology can be represented by two internal states of the atomic ion (e.g., $^{171}\text{Yb}^+$ ion Olmschenk et al. (2007)), described as a two-level spin system, manipulated by focusing adequate laser beams at the target ion(s). The physical ion qubits can be individually accessible for computation Knoernschild et al. (2010); Crain et al. (2014b). These qubits can be reliably initialized to the desired computational state and measured with very high accuracy using standard techniques. Most importantly, by virtue of the very long coherence

time of the ions, qubits can retain their state (memory) for a period of time unparalleled by any other quantum technology. The qubit memory error is modeled as an exponential decay in its fidelity $F \sim \exp(-at)$, where $a (=1/T_{coh})$ is determined by the coherence time of the qubit, and t is the time between quantum gates over which qubit sits idle (no-op). The corruption of the qubit state is modeled using a modified depolarizing channel Nielsen and Chuang (2000) (equal probability of bit flip and phase flip). Arbitrary single qubit gates, CNOT, and measurement can be performed with adequate reliability, making trapped ions a suitable candidate for large scale universal quantum computer.

A single-qubit quantum gate is accomplished by a simple application of laser pulse(s) on the qubit in its original location. A two-qubit gate, on the other hand, requires that both ions are brought in proximity before the laser pulse(s) are applied. As outlined in the previous chapter, there are two ways to achieve this proximity using two different types of physical resources: the ballistic shuttling channel (BSC) and the entanglement link (EL). BSC provides a physical channel through which an ion can be physically transported from its original location to the target location by carefully controlling the voltages of the electrodes on the ion trap chip. This chip can be modeled as a 2-D grid of ion-trap cells as shown in Figure 7.4. The dimensions of the state of the art ion-trap cell described in Monroe and Kim (2013) fall in the \sim mm size range, and we use $T_{shutt} = 1\mu s$ as the time it takes for an ion to be shuttled through a single cell. In the EL case, an EPR pair is established between designated proxy entangling ions (e-ions) that belong to two independent ion trap chips using a photonic channel using heralded entanglement generation Duan et al. (2004). The resulting EPR pair is used by the actual operand ions as a resource to perform the desired gate via quantum teleportation between two ions that cannot be connected by BSC Gottesman and Chuang (1999). It should be noted that the generation time for the EPR pairs is currently a slow process due to technology limitations.

Table 7.1: Device Parameters (DPs)

Physical Operation	Time(μs)	Failure Probability
Single-qubit	1	10^{-7}
Two-qubit (CNOT)	10	10^{-7}
Three-qubit (Toffoli)	100	10^{-7}
Measurement	100	10^{-7}
EPR pair Generation	5000	10^{-4}

This slowness can be compensated for by generating several EPR pairs in parallel using dedicated qubits and hardware. Table 7.1 summarizes the DPs used for all the analyses in this paper.

7.3.2 Quantum Architecture Model

Our architecture model is based on MUSIQ hardware described in chapter 3. The overview of the architecture is given in Figure 7.4. It features a hierarchical construction of larger blocks of qubits (called Segments) from smaller units (called Tiles), which are connected by an optical switch network. We use the Steane $[[7,1,3]]$ code Steane (1996) to encode one logical qubit using 7 physical qubits. Additional (ancilla) qubits are supplied to perform error correction and fault tolerant operations on the logical qubit. We first construct the first layer (L1) logical qubit block containing 22 physical qubits (7 data and 15 ancilla qubits) using Steane encoding. As the size of computation grows, multiple layers of encoding are needed to minimize the impact of increasing noise: with each new layer, the qubit and gate count increase by about a factor of 7. At least two layers of encoding are essential for reliable execution of sizable benchmark circuits analyzed in our simulations. Therefore, we cluster 7 L1 blocks to construct the second layer (L2) logical qubit block containing dedicated qubits to simultaneously carry out error correction operations at L1 level after every L1 gate. We find that the error correction operation at L2 level occurs much less frequently, and a dedicated error correcting ancilla resource at L2 is not

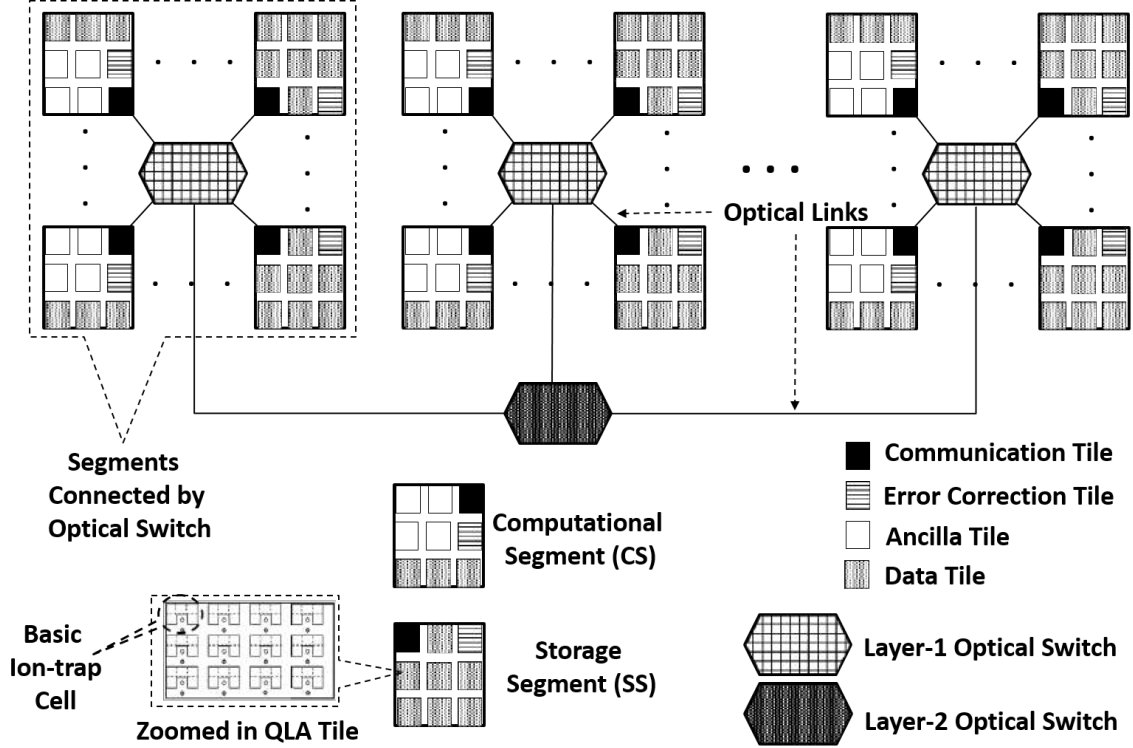


FIGURE 7.4: Overview of the reconfigurable quantum computer architecture analyzed in our performance simulation

necessary for each L2 logical qubit. Therefore, we allocate fewer ancilla qubits at L2 level for error correction, and rely on resource sharing to accomplish fault-tolerance at L2 level.

At L2 level, we construct four different types of logical qubit blocks using L1 logical blocks, called L2 Tiles, that serve different functions in the computation Metodi et al. (2005). Each Tile consists of memory cells that provide storage and manipulation of qubits for quantum gates, and BSCs that allow rapid transportation of ions across the memory cells to support the qubit interaction necessary for multi-qubit gates. Tiles are specified by their tasks, such as data storage (Data Tile), state preparation for non-Clifford gates (Ancilla Tile), error correction (EC Tile), and communication between Segments (Communication Tile). At the highest layer of hierarchy, various L2 Tiles are assembled to construct two types of Segments:

storage Segments (SS) and computational Segments (CS). A segment is the largest unit that is connected by BSCs, and the connections between Segments are carried out using optical interfaces. Each segment must contain at least one Communication Tile, one EC Tile, and several Data Tiles. The SS store qubits when they undergo no-ops. On the other hand, CS contains qubits necessary to perform the complex non-Clifford gates, which requires the ability to prepare the magic states and support the teleportation of data. The magic state is prepared using Ancilla Tiles, which are specific to CS only. The transportation of data between Segments is achieved by teleporting data through the EPR link established by the Communication Tiles of the Segments. A network of optical switches Kim et al. (2003) enables EPR pair generation between any pair of Segments in the system. To generate an L2 logical EPR pair, an L2 CNOT consisting of 49 physical CNOT gates is first enacted using 49 physical EPR pairs J. Eisert and Plenio (2000) and then error correction is applied to improve its fidelity Jiang et al. (2009). The detailed composition of different L2 Tile types is provided in Table 7.2. An L2 Tile can contain up to 600 cells arranged in a 2-D grid, and the time to shuttle logical qubits through the Tile is taken as $60\mu s$.

This architecture scales by adding more qubits to the system in the form of additional Segments, which demands larger optical switch network, at nominal increase in the latency overhead of EPR pair generation. The optical switches can be connected in a tree-like hierarchy such that the height h of the tree scales only logarithmically with the number of Segments. We restrict the size of the optical switches to 1,000 ports Kim et al. (2003), which can connect up to 20 Segments at the lowest level of the optical network tree, using 49 optical ports per L2 Communication Tile. This unique feature enables global connectivity for the entire computer, with the cross-segment communication time almost independent of distance between Segments. The communication time scales with the height h of switch network as 2^{h-1} . We found that $h \leq 3$ is sufficient to connect the maximum number of Segments arising in our

sizable benchmark circuits, and the corresponding maximum communication time is $5\text{ms} \times 2^{h-1} = 20\text{ms}$.

This globally-connected architecture with fast communication channels ensures rapid access to CS where computational resources for non-Clifford groups are available. This allows us to designate a finite number of CS in the overall QC to be shared across the computation. Furthermore, the physical construction of the Data and Ancilla Tiles are nearly identical (one Ancilla Tile can serve as two Data Tiles, and vice versa), so the designation between Data and Ancilla Tiles can be dynamically adjusted during the course of the computation. The total number of Segments and the allocation of Data, Ancilla, EC and Communication Tiles per SS and CS are the architectural parameters of our quantum computer design, denoted by total number of qubits (NTQ) and Segments (NSeg), number of computational Segments (NCS) ($\text{NCS} \leq \text{NSeg}$), number of EC Tiles (NEC) and Communication Tiles (NComm) per segment, and the number of Data (NData) and Ancilla (NAnc) Tiles per CS and SS. Throughout our simulation analysis we assume $\text{NEC} = 1$ since L2 error-correction is applied sparsely to Data Tiles, and a single EC Tile can serve several L2 logical qubits. Hence, a concise description of the architecture contains (1) NCS and (2) configuration of CS specified by three numbers (NData, NAnc, NComm). For SS, we replace NAnc with $2 \times \text{NData}$. Our analysis framework will involve changing architectural parameters and studying their impact on resource-performance trade-offs. The performance metrics consist of execution time T_{exec} and failure probability $P_{fail} = 1 - P_{succ}$ of the circuit, where P_{succ} is the probability that circuit execution yields a correct result.

Our system architecture provides several unique features not considered before, that provide crucial advantage in the resource-performance optimization of the quantum computer design. Firstly, the L2 logical blocks are not identical instantiations of L1 logical blocks: we utilize several L1 logical blocks to construct L2 Tiles with dif-

Table 7.2: Composition of L2 Tile

Tile Type	L1 Tiles	Physical qubits
Data	7	154
Ancilla	15	330
Error Correction	15	330
Communication	22(= 7 + 15)	484 + 49

ferent functionalities. This cross-layer optimization allows efficient utilization of the resources for those tasks (such as error correction at L2), where common resources can be shared (an example is shown in Figure 4.9). Secondly, our system allows dynamic re-allocation of computational resources during the computation (Data vs. Ancilla Tiles) to optimally adapt the architecture to the computational task at hand, analogous to reconfigurable computing using field-programmable gate arrays (FPGAs) in modern classical computing. Lastly, utilizing fully distributed resources (such as CS) is enabled by global connectivity uniquely available in our architecture.

7.4 Tool Description

In this chapter we enhance the version of toolbox used in chapter 6. The newer version is shown by shaded blocks in Figure 7.5. Again, there are two main components: Tile Designer and Performance Analyzer (TDPA) and Architecture Designer and Performance Analyzer (ADPA). Both components share common critical tasks, namely mapping, scheduling and error analysis, but the application of these tasks differs according to their objectives and the constraints.

The TDPA, described in chapter 5 works in the back end of the tool. It simulates the fault-tolerant construction of the logical qubit operations using specified DPs. It builds Tiles using Tile Builder by allocating sufficient qubits which can perform operations specified in Fault-Tolerant Circuit Generator, and maps qubits in the circuit to the physical qubits in the Tile using Low Level Mapper. Then, Low

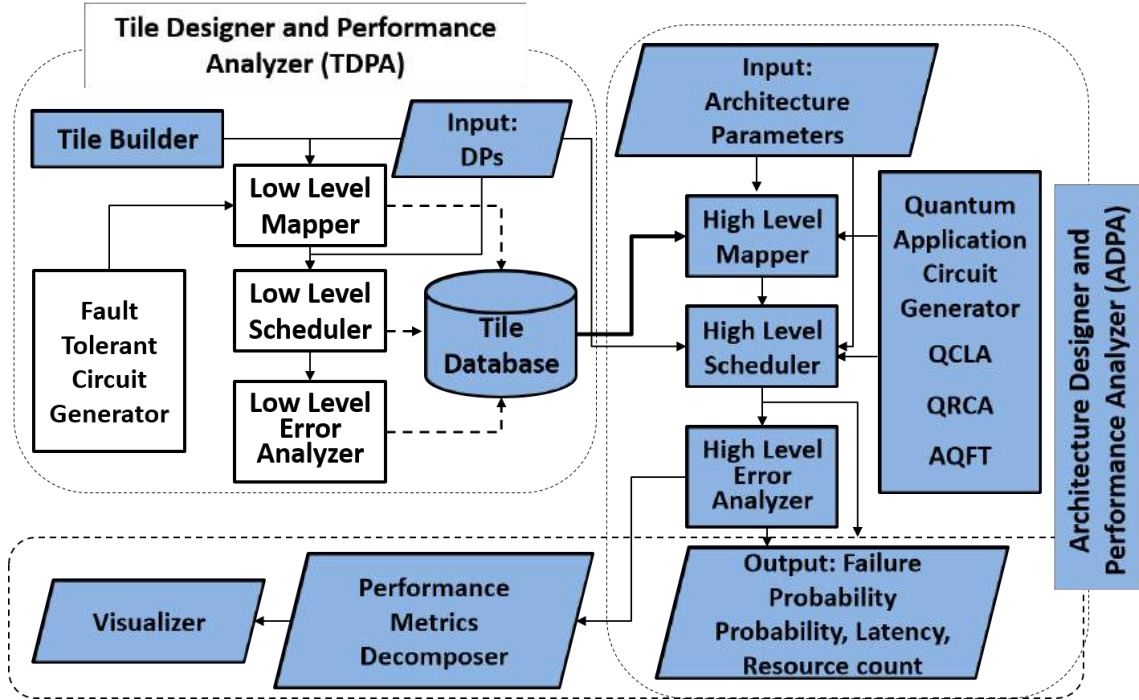


FIGURE 7.5: The shaded components are relevant to the study conducted in this chapter.

Level Scheduler generates the sequence of quantum gate operations to be executed in the circuit, that include transversal gates, magic state preparation for non-Clifford gates, error correction and EPR pair generation. Each logical operation is broken down into constituent physical operations, whose performances are simulated on the Tile by adding up the execution time of each gate subject to circuit dependencies and resource constraints. The Low Level Error Analyzer computes the failure probability of the specified fault-tolerant quantum gates based on the scheduled circuit, by counting the number of ways in which physical errors can propagate to cause logical error in the qubit. The Tile parametrized by DP and the computed performance metrics is stored in Tile Database. Table 7.3 shows the performance of the unified L2 Tile (which can act as Data, EC, Ancilla and Communication Tile) computed by the TDPA for baseline DPs.

ADPA whose bulk was developed in chapter 6 is the front end of the tool that interfaces with the user. It takes architecture parameters specified by the user (e.g., NCS, NEC and NComm) as inputs and (1) builds and connects Segments using Tiles supplied by TDPA to implement benchmark application on hardware configuration, and (2) evaluates performance of the benchmark for given architecture parameters. First, Quantum Application Circuit Generator generates the benchmark circuits from the given algorithms (QCLA, QRCA and AQFT). Then, High Level Mapper maps logical qubits to Tiles in the Segments, maximizing the locality by analyzing their connectivity patterns in the circuit, assigning frequently interacting qubits to the Tiles in the same segment. It generates the initial map of the Data Tiles in the Segments. Using this map, High Level Scheduler generates the sequence of gates for the circuit execution by solving the standard resource-constraint scheduling problem in which resources and constraints are given by architecture parameters. Scheduler minimizes the execution time by reducing the circuit critical path through maximum utilization of available resources (Ancilla and Communication Tiles) in the Segments. The non-Clifford gates require operands to be available in the same CS before being scheduled. Therefore the operand located in remote Data Tiles needs to be teleported into the local Data Tile of the CS, while Ancilla Tiles prepare the magic state for execution. NCS determines how many non-Clifford gates can be scheduled in parallel, while NComm determines how quickly Tiles can be teleported across the Segments. Therefore the delays in gate scheduling depend mainly on architecture parameters. The Scheduler also minimizes P_{fail} by scheduling error correction on Data Tiles at regular intervals when they sit idle. Once a complete schedule of logical operations is obtained, High Level Error Analyzer computes the overall P_{fail} . Since we cannot correct for logical failure of the operation, Error Analyzer simply computes $\prod_{i=1}^{i=N} (1 - P_{Li})$, where P_{Li} is the failure probability of the i -th logical gate and N is the total

Table 7.3: L2 Tile Performance Numbers. The L1 Error Correction takes $687\mu s$ and fails with probability 1.66×10^{-10} .

Logical Operation	$T_{exec}(\mu s)$	$1 - P_{succ}$
Pauli (X, Z)	1	1.15×10^{-18}
Hadamard	4	1.15×10^{-18}
CNOT	10	4.74×10^{-18}
Transversal Toffoli	100	1.06×10^{-17}
7-qubit Cat-State prep.	6,500	3.75×10^{-18}
Measurement	11,900	6.14×10^{-17}
Error Correction	48,900	4.58×10^{-16}
State prep. ($ \bar{0}\rangle, \bar{\mp}\rangle$)	34,500	1.6×10^{-16}
State prep. ($T \bar{\mp}\rangle$)	78,100	4.23×10^{-16}
EPR pair generation	$T_{gen} + 50,800$	1.08×10^{-11}

number of logical operations in the circuit. Error Analyzer also tracks the operational source of each P_{Li} so that with the help of **Performance Metrics Decomposer**, $1 - P_{succ}$ can be broken down into the following important noise components:

- Shuttling Noise P_{SHUT} : Errors due to the qubit shuttling through noisy BSC
- Teleportation Noise P_{TEL} : Errors due to the infidelity of an EPR pair for communication
- Memory Noise P_{MEM} : Errors due to the fidelity degradation of qubit during no-op
- Gate Noise P_{GATE} : Errors due to the noisy quantum gates and measurements

The Performance Metrics Decomposer not only outputs the constituents $1 - P_{succ}$ but also that of T_{exec} by recording various types of latencies which constitute the critical path of the quantum circuit. It rewrites execution time as sum of latencies overhead: $T_{exec} = T_{ANC} + T_{SHT} + T_{TEL} + T_{SWP} + T_{GATE}$ which are explained below:

- T_{ANC} : Latency due to the magic state preparation (fewer NCS or fewer Ancilla Tiles per segment)

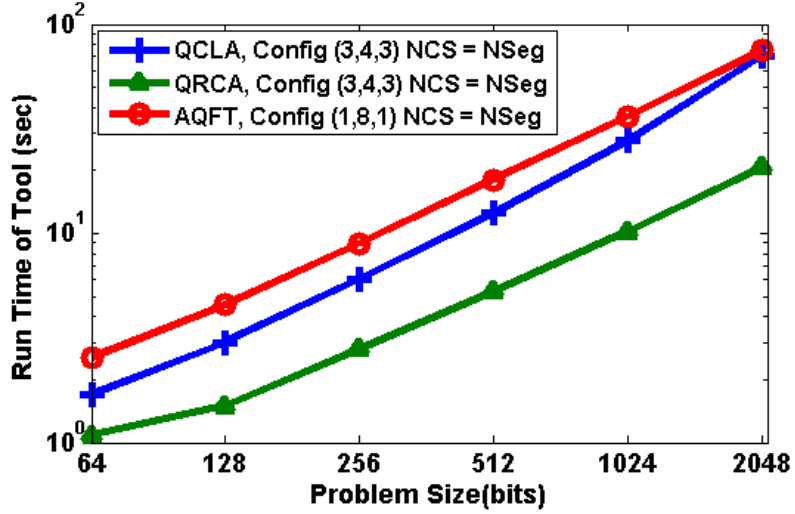


FIGURE 7.6: The running time of ADPA (excluding Visualizer) as a function of benchmark application size. With full Visualization, the running times increases no more than seven times.

- T_{SHT} : Latency due to the transportation of operand qubits of the gate, through BSC inside the segment
- T_{TEL} : Latency due to the logical EPR pair generation for communication (Fewer NComm)
- T_{SWP} : Latency due to the cross-segment swapping (fewer NComm or large number of smaller Segments)
- T_{GATE} : Time spent in performing circuit level quantum operations

The details of performance metrics decomposition can be found in section 4.14. Finally, the Visualizer provides pictorial presentation of circuit scheduling. It allows us to analyze the utilization of system resources which provide deeper insights into the efficacy of chosen design. The complete description of the Visualizer is given in section 4.15. Both Performance Metrics Decomposer and visualizer are used to explain our results in section 7.5.

7.4.1 Tool Validation and Performance

Individual components of the tool can easily be verified for correctness by running these for known circuits and comparing their output with anticipated results. Overall validation can be performed by using visualization and the breakdown of performance metrics for different types of benchmarks. Tool efficiency mainly arises from exploiting the repetitive nature of fault-tolerant procedures and circuit breakdown of universal quantum gates. Performance of low-level circuit blocks is pre-computed and stored in the database, and used for simulating the behavior of high-level circuits. For instance TDPA can be run offline to generate parametrized Tiles, which are used to efficiently run components of ADPA such as High Level Scheduler, Error Analyzer and Visualizer. Similarly, the initial mapping of L2 qubits on these Tiles is generated from a computationally intensive optimization algorithm Juvan and Mohar (1992), but once generated, can be efficiently processed by the High Level Scheduler to generate subsequent gate schedules. Thus, we can run High Level Mapper offline as well. Consequently, the running time of the tool is decided by that of High level Scheduler and Error Analyzer, which mainly depends on architecture resources and benchmark size. The results discussed in Section 7.5 show that the performance improvement saturates once resource investment exceeds a certain value, and the maximum size of the overall system we have to simulate is mainly dictated by the size of the application circuit.

The Figure 7.6 shows the running time of the tool as function of circuit size. The data is collected by running the tool on a computer system containing Intel^(R)Core^(TM) i3 2.4GHz processor and 2GB RAM. To incorporate the dependency of tool running time on the available architecture resources, we choose the configuration containing maximum resources in order to obtain typical worst-case running time of the tool. In this configuration, we allocate maximum Ancilla and Communication Tiles per

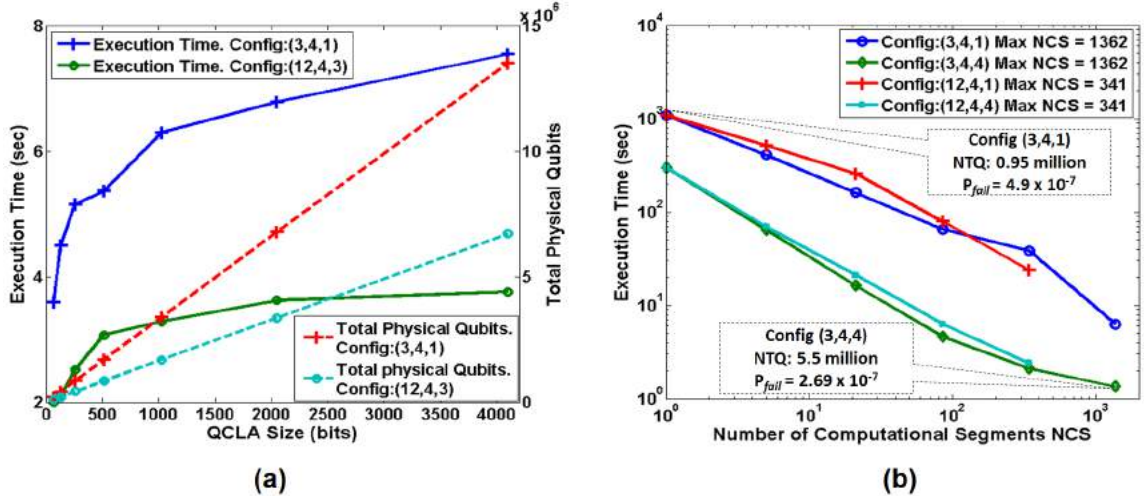


FIGURE 7.7: QCLA execution time (a) scaling under no constraints on physical resources, T_{exec} and total physical qubits (NTQ) consumed are plotted as a function of benchmark size, $NCS = NSeg$. (b) variation with NCS for different NComm, showing trade-offs between resources and T_{exec}

Data Tile and allow all Segments to act as Computational Segments. Under these conditions, Fig.7.6 shows that the performance-simulation of 2,048-bit circuit can be completed in less than 1.5 minutes. Thanks to the efficiency in the performance simulation, our tool can explore a large quantum computer design space in a reasonable amount of time.

7.5 Simulation Results

We first analyze the relationship between resources (qubits) and performance as a function of benchmark size for scalability. We consider the system architecture resource-performance scalable (RPS) if the increase in resources necessary to achieve the expected behavior of the performance (execution time) grows linearly with the size of the benchmark, while maintaining $P_{succ} \sim O(1)$. In the absence of hardware resource constraints, the expected execution time for the QRCA and AQFT grows linearly, while that for QCLA grows logarithmically, as the problem size grows. The execution time could grow much more quickly in the presence of resource constraints,

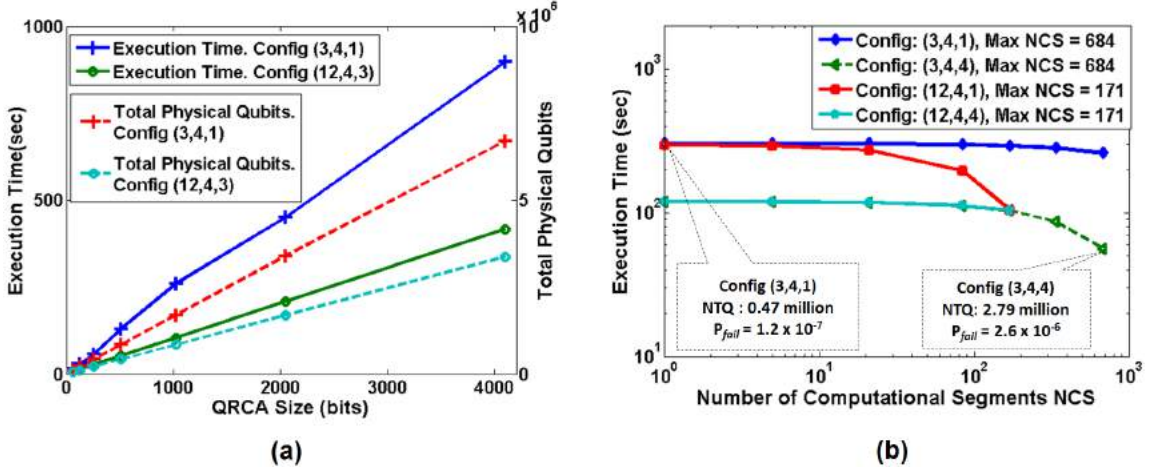


FIGURE 7.8: QRCA execution time (a) scaling under no constraints on physical resources, T_{exec} and total physical qubits (NTQ) consumed are plotted as a function of benchmark size, $NCS = NSeg$. (b) variation with NCS for different NComm, showing trade-offs between resources and T_{exec}

in which case the system is not considered RPS.

In the first step we present a set of simulations to quantify the RPS of the proposed MUSIQC architecture for benchmark circuits and analyze the constituents of performance metrics. In the next step, we study the impact of limited resources and architecture parameters on the performance of fixed size benchmarks. This will provide guidelines to find an optimal design under limited resources. In the last set, optimum designs are obtained under resource constraints, for effectively executing the benchmark circuits.

7.5.1 Resource-Performance Scalability

Figure 7.7(a), 7.8(a) and 7.9(a) show T_{exec} and the total number of physical qubits (NTQ) plotted against benchmark size for QCLA, QRCA and AQFT respectively, and corresponding P_{fail} values are shown in Table 7.4. We consider two adder configurations $(NData, NAnc, NComm) = (3,4,1)$ and $(12,4,3)$, and AQFT configurations $(1,4,1)$ and $(1,8,1)$. When benchmark size increases by factor of x , we expect P_{fail} to increase by at least the same amount (total gate operations increase x -fold) while

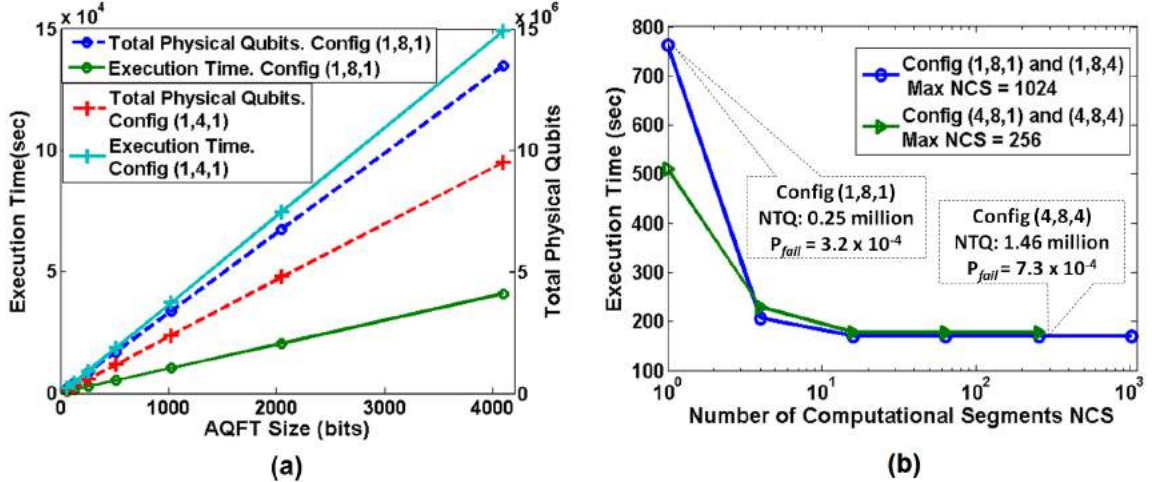


FIGURE 7.9: AQFT execution time (a) scaling under no constraints on physical resources, T_{exec} and total physical qubits (NTQ) consumed are plotted as a function of benchmark size, $NCS = NSeg$. (b) variation with NCS for different NComm, showing trade-offs between resources and T_{exec}

execution time scales as the depth of the circuit. Indeed Table 7.4 confirms this trend for all benchmarks. On the other hand, as problem size doubles, T_{exec} for QRCA and AQFT increases 2 fold [linear curve for T_{exec} in Figure 7.8(a) and 7.9(a)]. For QCLA, T_{exec} increases roughly by a constant amount [logarithmic curve in Figure 7.7(a)] as expected. Since this performance is achieved for the same increase in total qubits as that of the problem size, our architecture shows RPS.

By demonstrating RPS for two different architecture configurations with $NCS = NSeg$, we have presented varying performance levels. The impact of these configurations can be understood by analyzing the breakdown of performance metrics in Table 7.5. The significant contribution of T_{ANC} for adder configuration (12,4,3) and AQFT configuration (1,4,1), shows that magic state preparation is the dominant component of T_{exec} due to insufficient Ancilla Tiles in CS. This overhead can be substantially reduced either by increasing N_{Anc} (configuration (1,8,1) for AQFT) or by increasing the ratio of N_{Data} to N_{Anc} (configuration (3,4,1) for adders). However, the configuration (3,4,1) exposes EPR pair generation overhead captured by

Table 7.4: Failure Probability P_{fail} for corresponding data points of in Figure 7.7(a),7.8(a),7.9(a)

	Failure Probability					
	QCLA (10^{-8})		QRCA(10^{-8})		AQFT(10^{-5})	
Size	Config: (3,4,1)	Config: (12,4,3)	Config: (3,4,1)	Config: (12,4,3)	Config: (1,4,1)	Config: (1,8,1)
64	1.38	0.78	0.62	0.16	1.9	4.3
128	2.81	1.6	1.27	0.35	3.9	8.9
256	5.77	3.4	2.62	0.72	8.04	18.1
512	11.7	7.5	5.91	2.0	16.3	36.2
1024	23.1	15	12.9	4.5	32.7	73.5
2048	47.1	30.3	27.7	7.9	65.4	147.6
4096	93.8	61.1	57.2	18.7	130.1	295.2

Table 7.5: The breakdown of Performance metrics: T_{exec} of Figure 7.7(a),7.8(a),7.9(a) and P_{fail} of Table 7.4 for 1,024-bit benchmark. $NCS = NSeg$

Benchmark	Config	Execution Time Breakdown (%age)					Failure Probability Breakdown (%age)			
		$\%T_{ANC}$	$\%T_{SHT}$	$\%T_{TEL}$	$\%T_{SWP}$	$\%T_{GATE}$	$\%P_{TEL}$	$\%P_{SHUT}$	$\%P_{MEM}$	$\%P_{GATE}$
QCLA	3,4,1	1.8	0	31.0	61.2	5.8	89.3	9.4	0.6	0.5
	12,4,3	41.1	0	13.5	23.9	21.3	86.5	12.0	0.9	0.5
QRCA	3,4,1	6.1	0	80.0	10.7	3.1	80.6	8.3	10.4	0.4
	12,4,3	76.3	0	0.12	0.03	23.5	74.9	10.7	11.9	2.3
AQFT	1, 4, 1	28.9	47.7	0	0	23.2	0	98.7	0	1.2
	1, 8, 1	0.8	22.5	0	0	76.6	0	99.4	0	0.5

T_{TEL} and T_{SWP} . This is due to the large number of cross-segment CNOT and operand swapping required to bring all Toffoli operands to the same segment. Hence for adders, frequent cross-segment communication explains the higher component of teleportation error (P_{TEL}) in the failure probability. For AQFT, configuration (1,8,1) highlights shuttling overhead T_{SHT} as T gates comprise bulk of the operations. The scheduling of T gates in the long approximation sequence leads to a large number

Table 7.6: Optimal Architecture Configurations for the Fig.7.11

	Optimal Architecture Configuration				
	QCLA		QRCA		AQFT
Size	10,000 qubits/ Seg	5,000 qubits/ Seg	10,000 qubits/ Seg	5,000 qubits/ Seg	5000 or 10,000 qubits/Seg
64	(30,8,5) NCS = 8	(5,4,5) NCS = 50	(19,12,6) NCS = 7	(8,8,2) NCS = 17	(1,8,1) NCS \geq 16
128	(30,8,5) NCS = 17	(5,4,5) NCS = 101	(19,12,6) NCS = 14	(8,8,2) NCS = 33	(1,8,1) NCS \geq 16
256	(30,8,5) NCS = 34	(5,4,5) NCS = 203	(19,12,6) NCS = 27	(8,8,2) NCS = 65	(1,8,1) NCS \geq 16
512	(30,8,5) NCS = 68	(12,4,3) NCS = 170	(19,12,6) NCS = 54	(8,8,2) NCS = 131	(1,8,1) NCS = 16
1024	(30,8,5) NCS = 137	(12,4,3) NCS = 245	(19,12,6) NCS = 108	(8,8,2) NCS = 257	(1,8,1) NCS = 16
2048	(48,4,2) NCS = 25	NA	(19,12,6) NCS = 96	(12,4,3) NCS = 244	(1,8,1) NCS = 16
4096	NA	NA	NA	NA	(1,8,1) NCS = 16

of interactions between Data and Ancilla Tiles through BSC in the segment. This intensive localized communication makes (P_{SHUT}) the only noticeable component of P_{fail} . In conclusion, we have shown RPS of the architecture when performance is bottlenecked by different hardware constraints for various benchmarks. This shows that the architecture efficiently utilizes additional resources to achieve adequate performance while running larger size circuits.

7.5.2 Resource-Performance Trade-offs

Now that we have quantified RPS, we examine the impact of reduced resources on the performance by constraining architecture parameters. We fix the size of the benchmarks to 1,024 bits and vary NComm, NCS and the configuration to observe

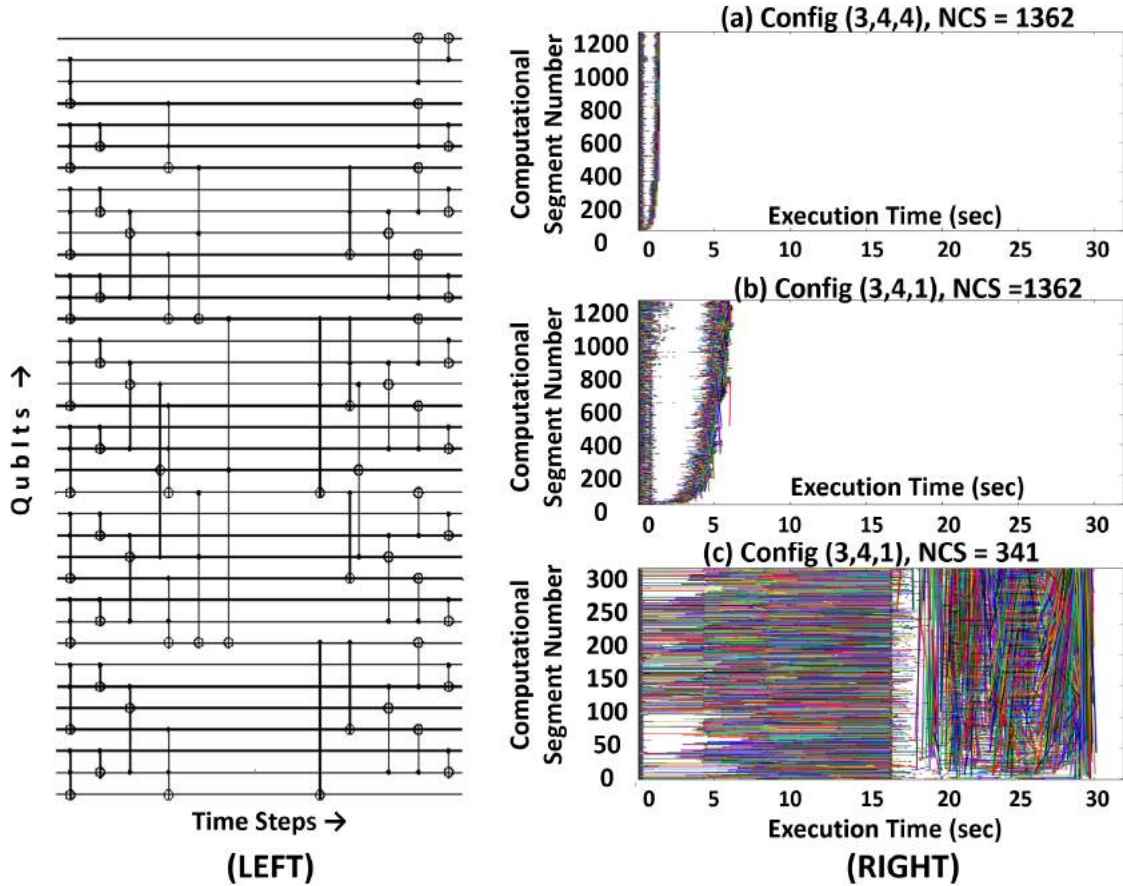


FIGURE 7.10: (LEFT) Sample QCLA circuit (RIGHT) Visual representation of latency overhead for Computational Segments of 1024-bit QCLA architecture with (a) configuration (3,4,4), NCS = NSeg = 1362, (b) configuration (3,4,1), NCS = NSeg = 1362 and (c) configuration (3,4,1), NCS = 341

changes in T_{exec} . Fig. 7.9(b) shows that for AQFT, (1) T_{exec} does not change with NComm since it is not constricted by cross-segment communication resources, and (2) T_{exec} decreases sharply by increasing NCS and becomes flat when $NCS \geq 16$, mainly due to insufficient parallelism in the circuit. The QRCA curves in Fig.7.8(b) remain unchanged as NCS increases until NCS approaches NSeg where T_{exec} shows noticeable decline. However, the overall decrease remains within a factor of only about 3, due to serial gate dependency. In these benchmarks, by comparing curves for different configurations, (12,4,1) vs. (3,4,1) for QRCA and (1,8,x) vs (4,8,x) for

AQFT, we find that clustering more Data Tiles in the CS generally reduces T_{exec} due to fewer delays in cross-Segment operand swapping.

Figure 7.7(b) shows that T_{exec} of QCLA decreases exponentially with NCS. In contrast to QRCA, the large number of concurrently executable Toffoli gates in QCLA demands much higher NCS. Furthermore, T_{exec} also decreases with higher NComm as the large number of cross-segment teleportations consume more communication resources. Figure 7.10(a)-(c) provide pictorial description of these trends. The visualization shows different types of latencies due to resource-constraint scheduling for each CS. The horizontal lines represent magic state preparation, while non-horizontal lines indicate cross-segment teleportation. The corresponding latencies can be derived by projecting these lines on the horizontal axis (execution time). When sufficient NComm resources are provided as in Figure 7.10(a), there is little delay and the circuit execution is fast. However, when NComm is reduced from 4 to 1 as in Figure 7.10(b) teleportation latency caused a 4.5-fold increase in T_{exec} . The same amount of increase occurs in Fig.7.10(c) when NCS is reduced from 1362 to 341. Long horizontal lines indicate delays due to fewer Ancilla Tiles for Toffoli magic state preparation, which increases the overall T_{exec} to over 30 sec.

By comparing Figure 7.7,7.8,7.9 it is easy to conclude that QCLA is the most resource hungry benchmark while AQFT is the least. We also note that P_{fail} values do not tend to improve substantially when we provide more architecture resources. In fact, one can show that a substantial decrease in P_{fail} can only be achieved by improving the relevant DP that contributes to the dominant noise sources. These sources are correctly identifiable once we have invested sufficient resources for scheduling error-correction and chosen optimal architecture configuration which minimizes T_{exec} . In the following subsection we concentrate on T_{exec} only and return to optimizing P_{fail} in the last subsection of the simulation results.

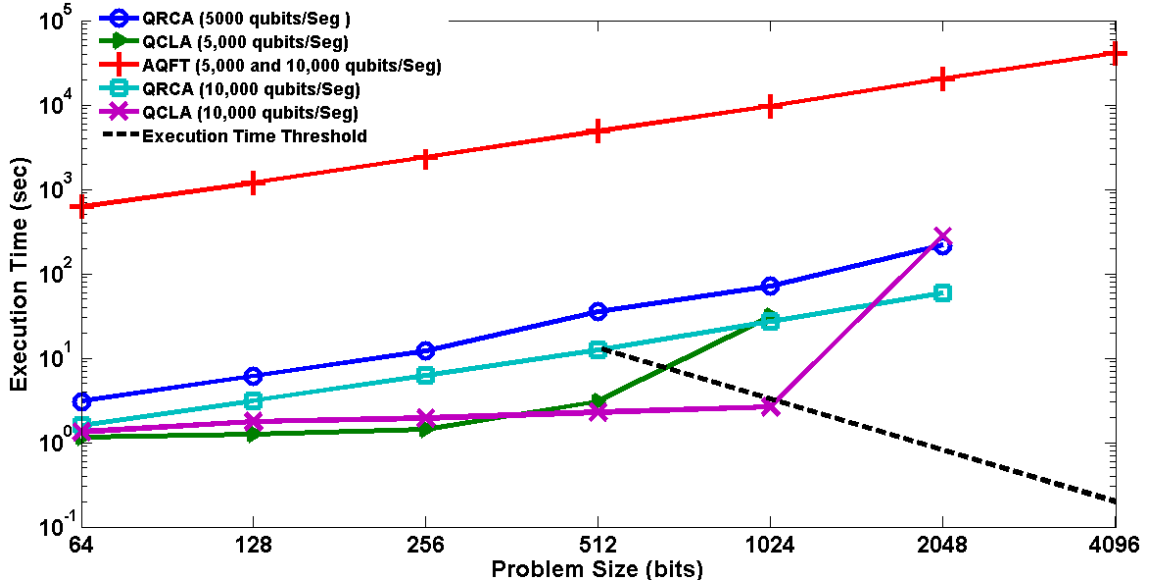


FIGURE 7.11: T_{exec} for optimal architectures plotted against benchmark size for different segment sizes. The Resource budget is 1.5 million physical qubits. Optimal architecture configurations are shown in Table 7.6.

7.5.3 Performance Scaling under Limited Resources

The increase in T_{exec} due to constrained resources can be compensated to some extent by designing an optimal architecture around the root cause that limits the performance. As shown earlier, the choice of optimal architecture configuration varies across benchmarks, since both performance bottlenecks and resource utilization depend strongly on the structure of the application circuit. By plotting optimized T_{exec} against benchmark size using a fixed resource budget, we can determine (1) the largest circuit size which can be scheduled and executed, (2) the trend for optimal performance as a function of problem size, and (3) the choice of configuration which generally obtains optimal performance for specified benchmark circuit. To adequately obtain these insights, we consider an example where we restrict our total qubit resource (NTQ) to 1.5 million physical qubits. We also restrict the number of physical qubits contained in the segment; small segment (SSeg) will contain up to 5,000 physical qubits, while large segment (LSeg) will contain up to 10,000 physical

qubits. For each benchmark we obtain two plots, one for each segment size.

Figure 7.11 shows T_{exec} variation of running the benchmark circuits on this system, as a function of problem size. The T_{exec} for each data point was minimized through optimal design selection by tool-assisted search through all feasible combinations of architecture configurations and NCS parameters. These optimized architecture designs are shown in Table 7.6. It is interesting to compare Ancilla and Communication Tiles invested in the optimal designs. For example, the configurations (19,12,6) and (8,8,2) for QRCA contains a higher Ancilla-to-Communication Tiles ratio, as compared to (30,8,5) and (5,4,5) for QCLA. This indicates that in contrast to the QRCA case, both Ancilla and Communication Tiles are equally vital for the QCLA performance. The AQFT architecture (1,8,1) with sufficient NCS is a natural optimal choice, since the only relevant resource for this circuit is the large number of Ancilla Tiles to schedule the long sequence of T gates necessary to approximate the small-angle rotations.

We observe that regardless of the segment size, T_{exec} of the least resource demanding benchmark of the three, namely AQFT, scales perfectly with problem size at least up to 4,096 bits. This is due to the fact that optimal configuration for AQFT can easily be met within the resource budget. However, in the case of adders, segment size is crucial for an optimal design. Larger Segments open greater search space for optimal design selection for resource intensive circuits. For both QCLA and QRCA, T_{exec} scales more adequately for larger Segments. For smaller Segments, QCLA performance shows significant degradation as the problem size begins to increase. The logarithmic depth of the most resource consuming benchmark (QCLA) is restricted to 256-bit and 1,024-bit for SSeg and LSeg, respectively. After that, T_{exec} shows a sudden rise with problem size and even surpasses the corresponding QRCA performance, as lack of resources leads to substantial delays in executing the parallel gate operations that enable the logarithmic-depth adder. The largest adder

benchmark that can be scheduled on this hardware is the 2,048-bit adder, where QRCA outperforms QCLA by factor of 4.

We have seen how limited resources and problem size impact the choice of adders for quantum arithmetic. Our analysis shows that in a 1.5 million-qubit computer, QCLA is a better adder choice up to 1,024-bits, but larger problems do not fit into the hardware. We can still execute the circuit using QRCA for up to 2,048-bits, although the performance will be substantially slower with this circuit choice.

7.5.4 Design Optimization by Tuning Device Parameters

Reducing the execution time

Quantum adders are used to construct modular exponentiation which comprises the bulk of Shor's integer factorization algorithm. However, in order to demonstrate practical speedup offered by the quantum computer, we need to show that Shor's algorithm in 'reasonable time', can factor integers larger than those factored by conventional computers. The largest reported integer sizes to 768-bit and consumes six months for its factorization Kleinjung et al. (2010). It has been estimated that compared to 768-bit, factorizing 1,024-bit number is 1,000 times harder and larger integers are unlikely to be factored in the near future Kleinjung et al. (2010). Based on these reasons, our criterion of evaluating the practical computational efficiency of quantum computer requires that integers larger than 768-bit can be factored in less than five months.

The total execution time of Shor's algorithm is heavily dominated by the modular exponentiation circuit. To factor integers of size in excess of 512 bits, we require millions of calls to the adders depending upon the size of the problem Van Meter and Itoh (2005). Although these adders can be connected in several ways depending upon resource budget, we restrict to the sequential execution of adders for our analysis. Hence, in order to run Shor's algorithm in less than five months, we can upper bound

the time taken by each adder call depending upon the size of the integer. This bound is shown as execution time threshold represented by dotted line in Figure 7.11. The execution time data points falling on or below this curve show the corresponding adder designs which can be used to construct modular exponentiation part of Shor's algorithm circuit to be scheduled in less than five months.

The Figure 7.11 shows that within 1.5 million qubits 1,024-bit number can be factored within specified time. However, the solution of the 2,048-bit problem cannot meet the time constraint. This is because T_{exec} of 2,048-bit QRCA and QCLA (58.5s and 270s respectively) is significantly higher than the threshold execution time of 0.8s. We first attempt to reduce T_{exec} by increasing our resource budget. With 90% more qubits, the optimal architecture configuration for QCLA changes from (48,4,2), NCS = 25 to (30,8,5), NCS = 273 which shows nearly 100x decrease in T_{exec} (from 270s to 2.76s). This is remarkably higher than the QRCA case which undergoes nominal reduction: from 58.5s to 50.1s. At this point QCLA seems to be the only choice which can meet the threshold execution time criterion if T_{exec} can be further reduced by factor of 4. Unfortunately, we find that further increase in resources (and new architecture configuration) failed to provide meaningful reduction in T_{exec} .

The failure to achieve performance improvement through additional resource highlights the role of DPs in the design space. The set of DPs which affect the speed of quantum circuit include the latency of physical operations. We find that when physical gate (and measurement) time and qubit shuttling latency are reduced by 90% the execution time declines to $0.68s < 0.8s$, meeting our time constraints. The total execution time for 2,048-bit integer factorization can be approximated as the sum of time spent in 16 million calls to the adder ($0.68s \times 16 \times 10^6 \approx 128$ days) and T_{exec} of single run of 4,096-bit AQFT (less than a day). Therefore the proposed quantum computer design can factor 2,048-bit number in less than five months.

Reducing the failure probability

In order to ensure that the entire Shor's algorithm is reliably executed, we require sum of failure probability of 2,048-bit modular exponentiation circuit and 4,096-bit AQFT to be sufficiently low. The Table 7.4 shows already adequate $\sim O(10^{-4}) - O(10^{-3})$ P_{fail} of AQFT when qubit shuttling latency (T_{shutt}) was assigned baseline value of $1\mu s$. When T_{shutt} was reduced to $0.1\mu s$ in our current design to lower the execution time of the adder, the P_{fail} of 4,096-bit AQFT falls well below 10^{-4} . Therefore, the goal of reducing the failure probability of full Shor's algorithm translates into curtailing the failure probability of modular exponentiation. This in turns requires the P_{fail} of each adder call to less than certain threshold value so that overall failure probability falls meet the design criterion. The 2,048-bit integer factorization consumes around 16 million calls to the adder, therefore we require that for each adder execution, $P_{fail} \ll 6.67 \times 10^{-8}$. The P_{fail} for the design optimized for T_{exec} is (30,8,5), NCS = 273 is 2.77×10^{-7} .

The lower the failure probability, we can either add one more layer of encoding or reduce the noise level in the physical device components. Adding a layer of encoding will entail at least an order of magnitude increase in required resources which enormously expands the scale of integration. In addition, the T_{exec} is also significantly inflated (Table 7.3 shows that L2 error correction takes 70x more time than L1 error-correction). Therefore, increasing the layer of encoding does not yield simultaneous reduction in both T_{exec} and failure P_{fail} . On the other hand by reducing the noise level in physical device components, we can decrease P_{fail} without compromising on T_{exec} . We exploit the dependence of P_{fail} on the fidelity of certain physical operation and highlight their role in designing fault-tolerant quantum computer.

The Figure 7.12 shows that original P_{fail} of QCLA can be reduced to 2.37×10^{-9} by improving the fidelity of EPR pair generation. The insight into the choice of this

parameter was obtained by analyzing the breakdown of failure probability. From Figure 7.12(a) it is clear that the contribution of the Teleportation Noise to the initial P_{fail} is more than 99%. The device parameter which directly affects the Teleportation Noise, is the infidelity of EPR pair for cross-Segment communication. By reducing the infidelity from 10^{-4} to 10^{-5} , we gain more than 100x reduction in the failure probability in Figure 7.12(b). It is worth comparing this result with that in chapter 6 where memory coherence time improvement gave us 1000x decrease in the failure probability upfront. In that analysis, the large part of that Memory Noise came from exponential degradation in physical qubit fidelity quickly leading to large Memory Noise in the absence of L2 error correction steps. This is because L2 error-correction is a time intensive operation which may not be always inserted into the time span over which logical underwent no-op. In this study we supplied dedicated resources for L1 error-correction instead of L2 based on the fact that L1 execution time was 75x shorter than that of L2. Therefore, we were able to prevent this exponential decay by fitting repeated L1 error correction during no-op whose time span was too short to allow for L2 error correction. According to Figure 1.2 the introduction of L1 error-correction significantly decelerates the fall in fidelity which eliminates Memory Noise as the primary performance bottleneck in Figure 7.12(a). However, once Teleportation Noise is removed in Figure 7.12(b), the Memory Noise resurfaces along with Gate Noise and further decrease in P_{fail} can be obtained by improving memory coherence time or physical gate fidelity or both.

Conclusively, by tuning the DPs we can lower the failure probability to match our design requirement: P_{fail} of QCLA becomes $2.37 \times 10^{-9} \ll 6.67 \times 10^{-8}$. This gives overall failure probability of modular exponentiation around 3.8%. The P_{fail} of 4,096-bit AQFT is negligible compared to this value, therefore, the overall failure probability does not exceed 4%. Hence we have shown that the optimal adder architecture with the appropriately tuned DPs can be used to construct reliable quantum

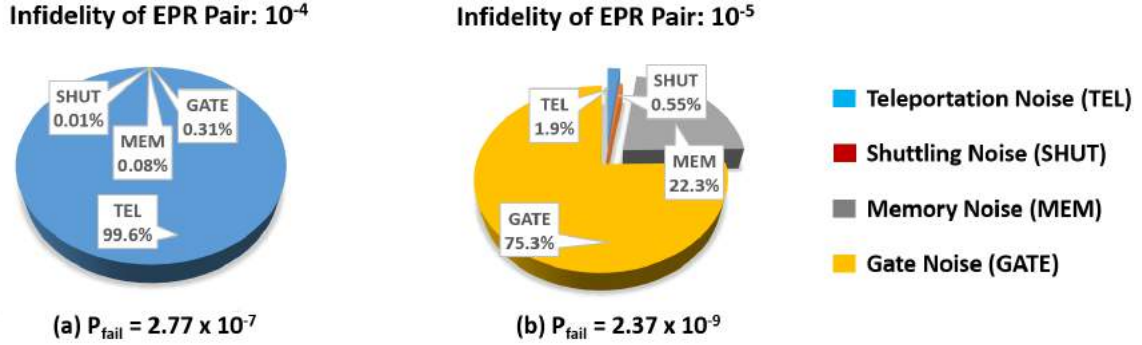


FIGURE 7.12: The breakdown and reduction in the failure probability of 2,048-bit QCLA For the configuration: (30,8,5), NCS = 273 with $T_{shutt} = 1\mu s$ and the baseline physical measurement and multi-qubit gate times reduced by 90%. The original failure probability shown in (a) is reduced from 2.77×10^{-7} to 2.37×10^{-9} in (b) only by decreasing the infidelity of the EPR pair for cross-Segment communication.

computer to execute 2,048-bit Shor’s algorithm.

7.6 Conclusion

In this chapter, we presented a scheme to design Shor’s algorithm using complete performance simulation toolset capable of designing a resource efficient, scalable quantum computer. Our tool was capable of analyzing the performance metrics of a flexible, reconfigurable model, and deepened our insights on the architecture design by providing a comprehensive breakdown of performance metrics and visualization of resource utilization. Using this tool, we were able to quantify, for the first time, the resource-performance scalability of a proposed architecture, featuring unique properties such as (1) dynamic resource allocation, where the functional role of physical qubit blocks could be re-assigned based on the demands of the circuit being executed, (2) cross-layer optimization, where qubit resources providing L2-level functions were shared throughout the system, (3) resource-constrained hardware performance, where optimal architectural design for resource allocation was considered as a function of the problem size, and (4) complete visualization of the resource uti-

lization that provided means to validate the optimality of the performance, (5) over a hardware architecture that provided global connectivity among all the qubits in the computer system.

Moreover, due to the macro-modeling approach used in our tool, we achieved highly efficient runtime for the performance simulation, which allowed us to carry out an exhaustive search for an optimized system design under given resource constraints, over a range of architecture configurations and benchmark circuits. Our benchmarks included crucial building blocks of Shor's algorithm such as the approximate quantum Fourier transform and two types of quantum adders. We found that the optimal designs varied across the benchmark applications depending on the types of gates used, the depth and parallelism of circuit structure, and resource budget. By comparing their performance across these benchmark circuits, we presented a concrete quantum computer design capable of executing 2,048-bit Shor's algorithm in less than five months.

Conclusion and Future Work

8.1 Summary of Important Results

The thesis describes a new performance simulation tool to study trapped-ion based quantum architectures and the role of physical hardware components in designing large-scale quantum computer. The architecture was modeled by set of parameters describing connectivity and allocation of computational qubit resources. The physical characteristics of hardware were incorporated by device parameters indicating the quality of hardware components. The basic tool flow derived from standard classical CAD literature and prior work, was integrated with novel features including detailed breakdown of performance metrics, visualization of resource utilization, flexibility to change design parameters and faster simulations. Using this tool a vast architecture-device parametric design space was efficiently explored, labeled and analyzed and the results provided vital guidelines for the experimentalists and system architects. For the convenience of the reader, these results are summarized as follows:

1. The design space of quantum computer is spanned by the computational resources(e.g., qubits), device and architecture parameters. The performance

can be quantified as execution time and the failure probability (P_{fail}) of the quantum application circuit (see chapter 5 for details).

2. To highlight the role of architecture in the design space, we can fix values for device parameters based on foreseeable advancement in the technology and study the dependence of performance on the resource allocation for various operational tasks in the quantum computer. This type of study generates novel ideas in the field of quantum computer architecture (see chapter 5,6 for details).
3. To highlight the role of device parameter in the design space, we can saturate performance gain by providing enough resources. At this point further enhancement in the performance can be achieved only by tuning the device parameters. This type of study targets experimental efforts and identifies device components which are crucial of the performance of quantum computer (see chapter 6 for details).
4. The total execution time of quantum application directly depends on the latency of physical quantum operations. However for resource intensive application circuits, the decrease in execution time, benefits from increased number of qubits, far greater than from the reduction in the physical operations latencies (see chapter 7 for details).
5. The design space of fully error-corrected (dedicated qubits for error-correction after each logical gate) quantum circuit can be partitioned into two main regimes. In Shuttling-Ancilla regime, execution time is limited by ancilla qubits required for fault-tolerant non-Clifford group gates, whereas, in case of Teleportation regime, it is limited by the qubits dedicated for the long-distance communication (see chapter 6 for details).

6. The overall failure probability of quantum circuit is dictated by the qubit coherence time and the fidelity of long-distance communication channel when gate failure probability falls significantly below threshold of the chosen error-correcting code (see chapter 6 for details).
7. When fidelity of quantum gate is high (i.e. $P_{fail} = 10^{-7}$), the frequency of error-correction across layers of concatenation chooses between (1) memory coherence time and the (2) fidelity of long-distance communication channel to determine the device parameter limiting the overall failure probability of the quantum circuit (compare the architecture assumptions and results of chapter 6 and chapter 7 for details).
8. The performance degradation due to slow and low fidelity communication channels can be adequately neutralized by the reliable qubits and high fidelity quantum gates at the cost of modest increase in resources (see chapter 6 for details). When communication among qubits ceases to be the performance bottleneck, we can define quantum computer architecture which can be efficiently scaled to schedule large size quantum application circuits (see chapter 7 for details).
9. A scalable architecture of the quantum computer shows ‘performance-scalability’ which means that it maintains desired level of performance with the increasing size of the application circuit and proportional raise in the resource supply (see chapter 7 for details).
10. With limited resources, the performance-scalability is achieved when resource allocation matches the computational workload of the application circuit (see chapter 7 for details).
11. A scalable trapped-ion quantum computer can efficiently factor integers of size in excess of 768-bits. By investing three million qubits, 2,048-bit number can

be factored in less than five months if physical gate failure probability is 10^{-7} or less (see chapter 7 for details).

8.2 Directions for Future Research

Although, the vast landscape of design variables makes analysis problem challenging, it also opens new avenues for routing interesting research questions. The work presented in the thesis is a stepping stone to the advanced modeling, computer-aided design and performance simulation framework to study quantum computer architectures. In this chapter we outline possible directions along which presented work can be extended to increase our knowledge about designing quantum computers. Naturally, these directions would entail enhancement to the capabilities of our existing tool. Most of these additions can be accomplished by integrating newer software modules with the predefined comprehensive framework. Therefore, the rest of the chapter is organized in sections, each detailing new problems which can be studied with the specific enhancement of the toolbox.

8.2.1 Introducing Heuristics in Searching Design Space

In chapter 7 exhaustive search was used to find optimal architecture within fixed budget. Based on the state-of-art hardware constraints in the ion trap, the restriction on the total number of qubits inside the trapped-ion chip, pruned non-trivial portion of the search space. In general, the architecture space can grow significantly larger with the advancement in qubit integration technology. In this case, expensive exhaustive search needs to be substituted by adequate heuristics which can efficiently guide us to the optimal or near-optimal design. These valuable heuristics can be derived if sufficient insights into the resource-performance trade-offs can be obtained during the searching process. Fortunately, the performance metrics decomposition feature of our tool can reveal the contribution of specific architecture resources in

defining the performance of design under analysis. By utilizing this feature and the leveraging wealth of Artificial Intelligence literature on designing search algorithm, interesting heuristics can be proposed different to search landscapes for range of quantum architectures.

8.2.2 Mapping and Scheduling Algorithms

The basic mapping and scheduling algorithm (MSA) reported in the thesis describe a way to performance-simulate quantum circuit on the hardware within its constraints. The last chapter testifies the adequacy of these algorithms by preserving the general performance-scaling properties of applications circuits. Among all the tested benchmarks, the QCLA posed the stringiest challenge since both mapping and scheduling techniques needed to optimally avail architecture resources in order to keep execution time growing ‘close to’ logarithmically as a function of circuit size (see Figure 7.7 for details). The results in section 7.5.1 show that our chosen MSA produced desired performance behavior within the specified resource budget for the given problem size. Therefore in the presence of limited resources for computation, performance degradation can be attributed to the said constraint with far greater confidence than to the possible sub-optimality of MSA.

The thesis explored architectures role in high performance quantum computation using one set of mapping and scheduling scheme. An interesting direction for further research is to experiment with other strategies of laying out quantum circuit on the hardware. As noted in chapter 4, many of these strategies for the quantum design can be borrowed from the literature on classical CAD algorithms. One important advantage of using multiple schemes is the freedom to match MSA with the needs of given quantum application circuit and chosen architecture. Moreover, within the framework of selected MSA, there are several places where the management of the pool of resource can benefit from newer schemes. For instance in the selection

of ancilla set for the magic state preparation, error correction and communication qubits for EPR pair generation. Even though the greedy scheduling approach for the resource allocation generally worked satisfactorily for our simulations, there is an opportunity to introduce smarter scheduling schemes to improve the performance of quantum application using CAD techniques.

8.2.3 Support for Other Error Models

The presented simulations assumed depolarizing channel error model wherein a qubit is infected by equally probabilistic X , Y and Z errors. However, it has been observed that in a quantum hardware, Z errors can occur with higher probability than the X errors Evans et al. (2007). Moreover, there are other error models described by varying the contribution the of corresponding error probabilities Nielsen and Chuang (2000). For example, Amplitude or Phase Damping Channels are used to model decoherence in variety of quantum communication systems Preskill (1998a) using unequal probabilities of bit and phase flips. In the absence of fully integrated sizable quantum system, it is difficult to judge which error channel presents an adequate picture of decoherence for given technology. Moreover, the amount of correlation between the occurrence of these error is another open question. Nevertheless, it is widely believed that under the Markovian assumption about the noise, the occurrence of errors can be assumed independent in the quantum systems, meaning that error on a physical qubit is unrelated to the error in the other qubit. Therefore based upon this assumption and the simplicity of analysis, most error correcting codes assume that underlying decoherence process follows independent error model.

In strict sense, errors mutually correlate in space or time Alicki et al. (2006). The challenges of dealing with the correlated noise model has been studied Ben-Aroya and Ta-Shma (2011); Clemens et al. (2004) and schemes have been proposed to extend the capacity of existing error correcting codes to deal with correlation noise Lu and

Marinescu (2007); Chiribella et al. (2011). In this model, the probability of multiple bit or phase flips can be assumed significantly higher than having single error in the logical qubit. In Appendix A, we show that using standard distance-3 Steane $[[7,1,3]]$ and Bacon-Shor $[[9,1,3]]$ code Bacon (2006), it is possible to correct for two X , Y or Z errors, the property that perfectly matches with the need of correlated error model. Similar properties of other codes can be explored in order to obtain deeper understanding of the quantum codes in the context of correlated noise. Meanwhile a tool capable of handling multiple types error models will be highly beneficial to so that the failure probability of quantum algorithm can be computed under more realistic behavior of the hardware.

8.2.4 Improving Models of Physical Device Components

The reliability of the performance-simulation of the quantum mechanical system can be increased if the model of component devices can be enhanced with additional meaningful parameters. The current assumptions about the physical properties of these components can be revised to incorporate complicated issues arising in the real hardware. Some of these intricacies can be handled easily by simply adjusting the values of the parameters. For example, the impact of quantum control protocols on the performance can be subsumed in the overall increase in the gate execution time and the decrease in its failure probability. On the other hands, certain engineering difficulties may need non-trivial modeling techniques. For example, the diffraction of laser beam can inadvertently change the state of the unintended ions located in the vicinity of the target ion. This leads to the non-negligible cross-talk among the ions during the scheduling of quantum gates Crain et al. (2014a). Similarly, neutral atoms quantum system can suffer from physical loss of the qubit Saffman and Walker (2005) due to laser heating. Finally, as more qubits are integrated into the system, the engineering parameters describing the complexity of classical control

system also require careful consideration in the performance evaluation framework. Hence, by upgrading the models of component devices, we can enhance our insights about crucial challenges in designing large scale quantum computer.

8.2.5 Support for Multiple Quantum Technologies

Quantum hardware other than trapped-ion offer alternative device technologies to implement quantum computer. These include Nuclear Magnetic Resonance (NMR) Negrevergne et al. (2006), Single photon Knill et al. (2001), Quantum dots Hanson et al. (2007), superconductors Nakamura et al. (1999) and others. Each technology brings unique realization of qubits and gates as well as their hardware level connectivity relevant physical attributes such as speed and fidelity of operation, immunity to decoherence and feasibility (or in-feasibility) of large scale computation. For example, superconductors feature faster operations but cannot be as easily physically transported as those of the ion qubits. Similarly, photons qubits act as great communication agent but suffer from significantly higher decoherence than other technologies. The quantum dots can benefit from existing fabrication techniques, however, the two-qubit gates can be severely constrained by the distance between operand qubits. Therefore, the accurate modeling of different types of quantum hardware and exploring architecture support within, are both challenging and interesting direction for the future research in the performance simulation. A versatile tool capable of comparing various quantum technologies can be used to determine optimal hardware for selected application.

8.2.6 Support for Multiple Error-correcting Codes

We encoded our benchmark application circuits by Steane code for its marvelous fault-tolerance properties. Using this code, our analysis gave us an estimate of resource overhead to reliably execute the circuits of practically interesting size. In

general, this overhead strongly depends on the threshold of chosen error-correcting code; higher threshold can lower the required amount of resource investment. In the literature we find that several other codes have been proposed with different threshold values Aliferis et al. (2005) with their construction detailed in Lidar and Brun (2013). However, one standout class of code which has received bulk of attention for its extremely high threshold property, is ‘topological quantum code’ Kitaev (2003).

The topological error correction have been shown to achieve very high threshold value, at least 1% Fowler et al. (2009) which is about 10-1,000 times higher than known concatenated codes. Therefore one can expect that considerably fewer resources will be required when application circuit relies on topological code. However, in the presence of realistic hardware constraints, scheduling very large number of simultaneous nearest-neighbor parity check operation for error detection is likely to pose major design challenge. The choice of suitable error correcting code for given quantum hardware and chosen application remains an interesting open problem. Hence, the additional capability of analyzing other quantum codes (e.g., topological code) can serve as valuable extension to our tool. This will add another insightful dimension into the quantum computer design space.

8.3 Summary and Final Word

In this final chapter, we summarized our main results and proposed several extensions to our current tool which can explore avenues of future research. We explained that embarking on these new journeys of investigation would increase our collective vision of steering the quantum computing research in the productive direction. We conclude on an optimistic note that the toolbox contribution, the results and the findings presented in the thesis will bring the attention of unfamiliar community to the exciting field of quantum computing.

Appendix A

Bimodal Error Correction

We identify that certain distance-3 quantum error correcting codes $[[3, 1, 3]]$, $[[7, 1, 3]]$ and $[[9, 1, 3]]$ can correct arbitrary double errors $(X_i X_j, Y_i Y_j, Z_i Z_j)$ even though they were initially designed for single error correction. In case of double error model, an event containing two identical errors on two different qubits of the logical qubit block, maps to same syndrome which apparently contradicts standard condition for quantum error correction. However we show that as long as the product of any two elements of the set acts trivially on the code space, perfect error correction is still possible. This result can be applied to a specific correlated error model Chiribella et al. (2011) where an error in a qubit is somehow linked with the occurrence of same error in the nearby qubit (which belongs to the same logical qubit block) resulting in double error event in the block of encoded qubit. Moreover, since $[[7,1,3]]$ and $[[9,1,3]]$ can always correct arbitrary single qubit errors, their added capacity of correcting arbitrary double error can be used to show the existence of quantum error correcting codes which can be switched between two modes of error correction (single or double). In this Appendix we first describe the theory behind standard error correction and

then extend this idea to double error correction using the $[[3, 1, 3]]$, $[[7, 1, 3]]$ and $[[9, 1, 3]]$ codes.

A.1 Condition for Quantum Error Correction

The necessary and sufficient condition for quantum error correction is follows:

$$\langle i | E_a E_b^\dagger | j \rangle = C_{ab} \delta_{ij} \quad (\text{A.1})$$

Where i and j are code basis (or codewords), E_a and E_b are correctable error operators. C should be hermitian matrix with entries C_{ab} and $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. This condition says that given quantum error correcting code with codeword i, j can correct set of error E_i when either

1. Error E_a acting on codeword $|i\rangle$ should not produce a state which overlaps with one produced when different error E_b acts on codeword $|j\rangle$.

or

2. For codeword $|i\rangle$, the overlap between state produced by Error E_a and E_b is same as that for codeword $|j\rangle$ for same errors.

While condition (1) holds only for non-degenerate codes, condition (2) can extend the error correction condition to degenerate codes. For bimodal error correction, we rely on the special symmetry of the chosen error correcting codes which indirectly satisfies eq.A.1.

A.2 Double Error Correction

Double error can be described by the map $\varepsilon(\rho)$ which transforms initial logical qubit state ρ as follows:

Table A.1: Double error syndromes for $[[3,1,3]]$ bit flip code

Double error type	Error	Syndrome	
		Z_1Z_2	Z_2Z_3
X_1X_2	YES	+1	-1
X_2X_3	YES	-1	+1
X_1X_3	YES	-1	-1
$\forall_{i,j} Z_iZ_j$	NO	+1	+1

$$\varepsilon(\rho) = p\rho + \sum_{i=1,2,\dots,n,j>i} (p_X^{i,j} X_iX_j\rho X_jX_i + p_Y^{i,j} Y_iY_j\rho Y_jY_i + p_Z^{i,j} Z_iZ_j\rho Z_jZ_i)$$

where $p = 1 - \sum_{i=1,2,\dots,n,j>i} (p_X^{i,j} + p_Y^{i,j} + p_Z^{i,j})$ is the probability that code state ρ was unchanged while with probabilities $p_X^{i,j}$, $p_Y^{i,j}$ and $p_Z^{i,j}$ it will undergo double X , Y or Z error respectively. Since Y error can be broken down into X and Z errors on the same qubit, therefore it is sufficient to describe concrete procedures for arbitrary double X and double Z error corrections only.

A.2.1 The $[[3,1,3]]$ code

The $[[3,1,3]]$ code can correct for either single bit flip or single phase flip error. We show that it can also correct for arbitrary double error. To show this property, we assume $[[3,1,3]]$ bit flip code which has following stabilizers:

1. Z_1Z_2
2. Z_2Z_3

Table A.1 shows that each double bit flip error X_iX_j gives distinct syndrome and therefore condition (A.1) is always satisfied. Double Z_iZ_j error is always a stabilizer of the state so it acts trivially on the code space. Thus $[[3,1,3]]$ code is a bimodal error correcting code.

Table A.2: Double X error syndromes for $[[7,1,3]]$ code

Double error type	Syndrome				Product of double error pairs
	S_z^1	S_z^2	S_z^3	S_x^i	
$(X_1X_2), (X_5X_6), (X_4X_7)$	-1	-1	+1	+1	$S_x^1S_x^2, S_x^3, S_x^1S_x^2S_x^3$
$(X_1X_5), (X_2X_6), (X_3X_7)$	+1	+1	-1	+1	$S_x^1S_x^2, S_x^1, S_x^2$
$(X_1X_3), (X_5X_7), (X_4X_6)$	+1	-1	+1	+1	$S_x^1, S_x^1S_x^3, S_x^3$
$(X_1X_4), (X_3X_6), (X_2X_7)$	-1	+1	-1	+1	$S_x^1S_x^3, S_x^2, S_x^1, S_x^2$
$(X_1X_6), (X_3X_4), (X_2X_5)$	-1	-1	-1	+1	$S_x^1S_x^3, S_x^2S_x^3, S_x^1S_x^2$
$(X_1X_7), (X_2X_4), (X_3X_5)$	+1	-1	-1	+1	$S_x^1S_x^2S_x^3, S_x^1, S_x^2S_x^3$
$(X_2X_3), (X_4X_5), (X_6X_7)$	-1	+1	+1	+1	$S_x^2S_x^3, S_x^2, S_x^3$

Table A.3: Double Z error syndromes for $[[7,1,3]]$ code

Double error type	Syndrome				Product of double error pairs
	S_x^1	S_x^2	S_x^3	S_z^i	
$(Z_1Z_2), (Z_5Z_6), (Z_4Z_7)$	-1	-1	+1	+1	$S_z^1S_z^2, S_z^3, S_z^1S_z^2S_z^3$
$(Z_1Z_5), (Z_2Z_6), (Z_3Z_7)$	+1	+1	-1	+1	$S_z^1S_z^2, S_z^1, S_z^2$
$(Z_1Z_3), (Z_5Z_7), (Z_4Z_6)$	+1	-1	+1	+1	$S_z^1, S_z^1S_z^3, S_z^3$
$(Z_1Z_4), (Z_3Z_6), (Z_2Z_7)$	-1	+1	-1	+1	$S_z^1S_z^3, S_z^2, S_z^1, S_z^2$
$(Z_1Z_6), (Z_3Z_4), (Z_2Z_5)$	-1	-1	-1	+1	$S_z^1S_z^3, S_z^2S_z^3, S_z^1S_z^2$
$(Z_1Z_7), (Z_2Z_4), (Z_3Z_5)$	+1	-1	-1	+1	$S_z^1S_z^2S_z^3, S_z^1, S_z^2S_z^3$
$(Z_2Z_3), (Z_4Z_5), (Z_6Z_7)$	-1	+1	+1	+1	$S_z^2S_z^3, S_z^2, S_z^3$

A.2.2 Steane $[[7,1,3]]$ code

Steane code has following six stabilizers:

1. $S_x^1 : X_1X_3X_5X_7$
2. $S_x^2 : X_2X_3X_6X_7$
3. $S_x^3 : X_4X_5X_6X_7$
4. $S_z^1 : Z_1Z_3Z_5Z_7$
5. $S_z^2 : Z_2Z_3Z_6Z_7$
6. $S_z^3 : Z_4Z_5Z_6Z_7$

For double error correction, one can make Steane code behave similar to the degenerate codes; multiple double errors can cause same syndrome but they do not prevent us from fixing the error. Table A.2 shows that each single syndrome value corresponds to a set of three double errors called Clique of Benign Double Errors (CBDE). However notice that product of any two double errors in CBDE is either a X -stabilizer or product of X -stabilizers, both of which leave original encoded state unchanged. For example consider the first CBDE $(X_1X_2),(X_5X_6),(X_4X_7)$ all of which give $(-1,-1,+1)$ when Z -stabilizers are measured. The product of $(X_1X_2),(X_5X_6)$ is $X_1X_2X_5X_6 = S_x^1S_x^2$, that of $(X_5X_6),(X_4X_7)$ is $X_4X_5X_6X_7 = S_3$ and that of $(X_1X_2),(X_4X_7)$ is $X_1X_2X_4X_7 = S_x^1S_x^2S_x^3$. This means that given a syndrome value, we can identify corresponding CDBE and can correct for the double error by applying arbitrary chosen double error from the set without actually identifying original double error.

Table A.3 is for double phase-flip error and constructed in the same way as that for Table A.2 for double X -error. Based on the scheme described for X errors, reader can verify that Z double errors can be corrected as well. Hence we have shown that Steane $[[7,1,3]]$ can correct for arbitrary double error. The key idea behind our scheme is exploitation of induced degeneracy in the code for double errors.

A.2.3 Bacon-Shor $[[9,1,3]]$ code

The Bacon-Shor (BS) code has following four stabilizers:

1. $S_x^1 : X_1X_2X_3X_4X_5X_6$
2. $S_x^2 : X_4X_5X_6X_7X_8X_9$
3. $S_z^1 : Z_1Z_2Z_4Z_5Z_7Z_8$
4. $S_z^2 : Z_2Z_3Z_5Z_6Z_8Z_9$

and gauge operators are shown in Table A.4. Recall that BS-code construction forces gauge operators which act trivially on the code space. For correction, we convert

Table A.4: Gauge operators for $[[9,1,3]]$ code

X-gauge operators			Z-gauge operators		
Gx1	Gx2	Gx3	Gz1	Gz2	Gz3
(X_1X_4)	(X_2X_5)	(X_3X_6)	(Z_1Z_2)	(Z_4Z_5)	(Z_7Z_8)
(X_1X_7)	(X_2X_8)	(X_3X_9)	(Z_2Z_3)	(Z_5Z_6)	(Z_8Z_9)
(X_4X_7)	(X_5X_8)	(X_6X_9)	(Z_1Z_3)	(Z_4Z_6)	(Z_7Z_9)

double error into relevant gauge operator. For example an unknown double X error can be converted into some X gauge operator and likewise for double Z error. In Table A.5 we identify total of 27 possible X double errors, which can be clustered into three sets, each containing 9 double errors. Similar to the case of Steane, each set essentially forms a CBDE. In order to correct double X error, Z syndromes value is determined and corresponding CBDE is identified. Error can be fixed by applying arbitrarily chosen double error from the identified CBDE.

Consider CDBE for $(S_z^1, S_z^2) = (+1, -1)$. Reader can verify that the product of any two double errors in this clique will be one of the X -gauge operator in column Gx1 or Gx2 Table A.4 or a product of two gauge operators chosen from these columns. A similar Table A.6 can be constructed for double Z error and it can easily be verified that all 27 errors can be corrected using this scheme. Hence we have shown that $[[9,1,3]]$ code can correct for arbitrary double error because any pair of double errors in CBDE forms a gauge operator.

Table A.5: Double X error syndromes for $[[9,1,3]]$ code

Double error types	Syndrome			Product of double error pairs
	S_z^1	S_z^2	S_x^i	
$(X_1X_2), (X_1X_5), (X_1X_8),$ $(X_2X_4), (X_2X_7), (X_4X_5),$ $(X_4X_8), (X_5X_7), (X_7X_8)$	+1	-1	+1	Gx1 and Gx2 (Table A.4)
$(X_1X_3), (X_1X_6), (X_1X_9),$ $(X_3X_4), (X_3X_7), (X_4X_6),$ $(X_4X_9), (X_6X_7), (X_7X_9)$	-1	-1	+1	Gx1 and Gx3 (Table A.4)
$(X_2X_3), (X_2X_6), (X_2X_9),$ $(X_3X_5), (X_3X_8), (X_5X_6),$ $(X_5X_9), (X_6X_8), (X_8X_9)$	-1	+1	+1	Gx2 and Gx3 (Table A.4)

Table A.6: Double Z error syndromes for $[[9,1,3]]$ code

Double error types	Syndrome			Product of double error pairs
	S_x^1	S_x^2	S_z^i	
$(Z_1Z_4), (Z_1Z_5), (Z_1Z_6),$ $(Z_2Z_4), (Z_2Z_5), (Z_2Z_6),$ $(Z_3Z_4), (Z_3Z_5), (Z_3Z_6)$	+1	-1	+1	Gz1 and Gz2 (Table A.4)
$(Z_1Z_7), (Z_1Z_8), (Z_1Z_9),$ $(Z_2Z_7), (Z_2Z_8), (Z_2Z_9),$ $(Z_3Z_7), (Z_3Z_8), (Z_3Z_9)$	-1	-1	+1	Gz1 and Gz3 (Table A.4)
$(Z_4Z_7), (Z_4Z_8), (Z_4Z_9),$ $(Z_5Z_7), (Z_5Z_8), (Z_5Z_9),$ $(Z_6Z_7), (Z_6Z_8), (Z_6Z_9)$	-1	+1	+1	Gz2 and Gz3 (Table A.4)

Bibliography

- Aaronson, S. and Gottesman, D. (2004), “Improved simulation of stabilizer circuits,” *Physical Review A*, 70, 052328.
- Aharonov, D. and Ben-Or, M. (1997), “Fault-tolerant quantum computation with constant error Fault-tolerant quantum computation with constant error Fault-tolerant quantum computation with constant error,” in *Proceedings of the 29th Annual Symposium on Theory of Computing*, pp. 176–188.
- Alicki, R., Lidar, D. A., and Zanardi, P. (2006), “Internal consistency of fault-tolerant quantum error correction in light of rigorous derivations of the quantum Markovian limit,” *Physical Review A*, 73, 052311.
- Aliferis, P., Gottesman, D., and Preskill, J. (2005), “Quantum accuracy threshold for concatenated distance-3 codes,” *arXiv:quant-ph/0504218*.
- Andreev, K. and Racke, H. (2006), “Balanced Graph Partitioning,” *Theory of Computing Systems*, 39, 929–939.
- Bacon, D. (2003), “Decoherence, control, and symmetry in quantum computers,” *arXiv preprint quant-ph/0305025*.
- Bacon, D. (2006), “Operator quantum error-correcting subsystems for self-correcting quantum memories,” *Physical Review A*, 73, 012340.
- Balensiefer, S., Kregor-Stickles, L., and Oskin, M. (2005a), “An Evaluation Framework and Instruction Set Architecture for Ion-Trap Based Quantum Micro-Architectures,” *SIGARCH Comput. Archit. News*, 33, 186–196.
- Balensiefer, S., Kregor-Stickles, L., and Oskin, M. (2005b), “QUALE: quantum architecture layout evaluator,” in *Proc. of the SPIE*, vol. 5815, pp. 103–114.
- Ballance, C., Harty, T., Linke, N., and Lucas, D. (2014), “High-fidelity two-qubit quantum logic gates using trapped calcium-43 ions,” *arXiv preprint arXiv:1406.5473*.
- Barends, R., Kelly, J., Megrant, A., Veitia, A., Sank, D., Jeffrey, E., White, T., Mutus, J., Fowler, A., Campbell, B., et al. (2014), “Logic gates at the surface code

- threshold: Superconducting qubits poised for fault-tolerant quantum computing,” *arXiv preprint arXiv:1402.4848*.
- Beckman, D., Chari, A. N., Devabhaktuni, S., and Preskill, J. (1996a), “Efficient networks for quantum factoring,” *Physical Review A*, 54, 1034.
- Beckman, D., Chari, A. N., Devabhaktuni, S., and Preskill, J. (1996b), “Efficient networks for quantum factoring,” *Phys. Rev. A*, 54, 1034–1063, <http://arXiv.org/quant-ph/9602016>.
- Ben-Aroya, A. and Ta-Shma, A. (2011), “Approximate quantum error correction for correlated noise,” *Information Theory, IEEE Transactions on*, 57, 3982–3988.
- Benhelm, J., Kirchmair, G., Roos, C. F., and Blatt, R. (2008), “Towards fault-tolerant quantum computing with trapped ions,” *Nature Physics*, 4, 463–466.
- Bennett, C. H., Brassard, G., Crépeau, C., Jozsa, R., Peres, A., and Wootters, W. K. (1993), “Teleporting an unknown quantum state via dual classical and Einstein-Podolsky-Rosen channels,” *Physical review letters*, 70, 1895.
- Bernstein, E. and Vazirani, U. (1993a), “Quantum complexity theory,” in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pp. 11–20, ACM.
- Bernstein, E. and Vazirani, U. (1993b), “Quantum complexity theory,” in *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pp. 11–20, New York, NY, USA, ACM.
- Blume-Kohout, R. (2010), “Optimal, reliable estimation of quantum states,” *New Journal of Physics*, 12, 043034.
- Bravyi, S. and Kitaev, A. (2005), “Universal quantum computation with ideal Clifford gates and noisy ancillas,” *Phys. Rev. A*, 71, 022316.
- Briegel, H., Dür, W., Cirac, J., and Zoller, P. (1998), “Quantum repeaters: The role of imperfect local operations in quantum communication,” *Physical Review Letters*, 81, 5932–5935.
- Brown, K., Wilson, A., Colombe, Y., Ospelkaus, C., Meier, A., Knill, E., Leibfried, D., and Wineland, D. (2011), “Single-qubit-gate error below 10^{-4} in a trapped ion,” *Physical Review A*, 84, 030303.
- Calderbank, A. R. and Shor, P. W. (1996), “Good quantum error-correcting codes exist,” *Physical Review A*, 54, 1098.
- Chiribella, G., Dall'Arno, M., D'Ariano, G. M., Macchiavello, C., and Perinotti, P. (2011), “Quantum error correction with degenerate codes for correlated noise,” *Physical Review A*, 83, 052305.

- Cirac, J. I. and Zoller, P. (1995), “Quantum computations with cold trapped ions,” *Physical review letters*, 74, 4091.
- Clemens, J. P., Siddiqui, S., and Gea-Banacloche, J. (2004), “Quantum error correction against correlated noise,” *Physical Review A*, 69, 062313.
- Crain, S., Mount, E., Baek, S., and Kim, J. (2014a), “Individual addressing of trapped $^{171}\text{Yb}^+$ ion qubits using a microelectromechanical systems-based beam steering system,” *Applied Physics Letters*, 105, 181115.
- Crain, S., Mount, E., Baek, S.-Y., and Kim, J. (2014b), “Individual addressing of trapped $^{171}\text{Yb}^+$ ion qubits using a microelectromechanical systems-based beam steering system,” *Appl. Phys. Lett.*, 105, 181115.
- Crespi, A., Ramponi, R., Osellame, R., Sansoni, L., Bongioanni, I., Sciarrino, F., Vallone, G., and Mataloni, P. (2011), “Integrated photonic quantum gates for polarization qubits,” *Nature communications*, 2, 566.
- Cuccaro, S. A., Draper, T. G., Kutin, S. A., and Moulton, D. P. (2004), “A new quantum ripple-carry addition circuit,” *arXiv preprint quant-ph/0410184*.
- Dousti, M. J. and Pedram, M. (2012), “Minimizing the latency of quantum circuits during mapping to the ion-trap circuit fabric,” in *Proceedings of the Conference on Design, Automation and Test in Europe*, pp. 840–843, EDA Consortium.
- Dousti, M. J. and Pedram, M. (2013), “Latency estimation for a quantum algorithm mapped to a quantum circuit fabric,” in *Proceedings of the 50th Annual Design Automation Conference*, p. 42, ACM.
- Dousti, M. J., Shafaei, A., and Pedram, M. (2014), “Squash: a scalable quantum mapper considering ancilla sharing,” in *Proceedings of the 24th edition of the great lakes symposium on VLSI*, pp. 117–122, ACM.
- Draper, T. G., Kutin, S. A., Rains, E. M., and Svore, K. M. (2006), “A logarithmic-depth quantum carry-lookahead adder,” *Quantum Inf. & Comput.*, 6, 351–369.
- Duan, L.-M., Blinov, B. B., Moehring, D. L., and Monroe, C. (2004), “Scaling Trapped Ions for Quantum Computation with Probabilistic Ion-Photon Mapping,” *Quant. Inf. Comp.*, 4, 165–173.
- Einstein, A., Podolsky, B., and Rosen, N. (1935), “Can quantum-mechanical description of physical reality be considered complete?” *Physical review*, 47, 777.
- Evans, Z., Stephens, A., Cole, J., and Hollenberg, L. (2007), “Error correction optimisation in the presence of X/Z asymmetry,” *arXiv preprint arXiv:0709.3875*.

- Fowler, A. G. and Hollenberg, L. C. (2004), “Scalability of Shor’s algorithm with a limited set of rotation gates,” *Physical Review A*, 70, 032329.
- Fowler, A. G., Stephens, A. M., and Groszkowski, P. (2009), “High-threshold universal quantum computation on the surface code,” *Physical Review A*, 80, 052312.
- Fowler, A. G., Mariantoni, M., Martinis, J. M., and Cleland, A. N. (2012a), “Surface codes: Towards practical large-scale quantum computation,” *Phys. Rev. A*, 86, 032324.
- Fowler, A. G., Whiteside, A. C., McInnes, A. L., and Rabbani, A. (2012b), “Topological code Autotune,” *Physical Review X*, 2, 041003.
- Galiautdinov, A., Korotkov, A. N., and Martinis, J. M. (2012), “Resonator-zero-qubit architecture for superconducting qubits,” *Phys. Rev. A*, 85, 042321.
- Garcia, H. and Markov, I. (2013), “Quipu: High-performance simulation of quantum circuits using stabilizer frames,” in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pp. 404–410.
- Giles, B. and Selinger, P. (2013), “Exact synthesis of multiqubit Clifford+T circuits,” *Phys. Rev. A*, 87, 032332.
- Gottesman, D. (1997), “Stabilizer codes and quantum error correction,” *arXiv preprint quant-ph/9705052*.
- Gottesman, D. (1998a), “The Heisenberg representation of quantum computers,” *arXiv preprint quant-ph/9807006*.
- Gottesman, D. (1998b), “Theory of fault-tolerant quantum computation,” *Phys. Rev. A*, 57, 127–137.
- Gottesman, D. and Chuang, I. L. (1999), “Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations,” *Nature*, 402, 390 – 393.
- Goudarzi, H., Dousti, M. J., Shafaei, A., and Pedram, M. (2014), “Design of a universal logic block for fault-tolerant realization of any logic operation in trapped-ion quantum circuits,” *Quantum information processing*, 13, 1267–1299.
- Guise, N. D., Fallek, S. D., Stevens, K. E., Brown, K., Volin, C., Harter, A. W., Amini, J. M., Higashi, R. E., Lu, S. T., Chanhvongsak, H. M., et al. (2014), “Ball-grid array architecture for microfabricated ion traps,” *arXiv preprint arXiv:1412.5576*.
- Gutiérrez, M. and Brown, K. R. (2014), “Comparison of a quantum error correction threshold for exact and approximate errors,” *arXiv preprint arXiv:1501.00068*.

- Han, R., Ng, H. K., and Englert, B.-G. (2014), “Implementing a neutral-atom controlled-phase gate with a single Rydberg pulse,” *arXiv preprint arXiv:1407.8051*.
- Hanneke, D., Home, J. P., Jost, J. D., Amini, J. M., Leibfried, D., and Wineland, D. J. (2009), “Realization of a programmable two-qubit quantum processor,” *Nature Physics*, 6, 13–16.
- Hanson, R., Kouwenhoven, L., Petta, J., Tarucha, S., and Vandersypen, L. (2007), “Spins in few-electron quantum dots,” *Reviews of Modern Physics*, 79, 1217.
- Harty, T., Allcock, D., Ballance, C., Guidoni, L., Janacek, H., Linke, N., Stacey, D., and Lucas, D. (2014), “High-Fidelity Preparation, Gates, Memory, and Readout of a Trapped-Ion Quantum Bit,” *Physical review letters*, 113, 220501.
- Isailovic, N., Whitney, M., Patel, Y., and Kubiawicz, J. (2008), “Running a quantum circuit at the speed of data,” in *ACM SIGARCH Computer Architecture News*, vol. 36, pp. 177–188, IEEE Computer Society.
- J. Eisert, K. Jacobs, P. P. and Plenio, M. B. (2000), “Optimal local implementation of non-local quantum gates,” *Phys. Rev. A*, 62.
- Jiang, L., Taylor, J. M., Nemoto, K., Munro, W. J., Van Meter, R., and Lukin, M. D. (2009), “Quantum repeater with encoding,” *Phys. Rev. A*, 79, 032325.
- Juvan, M. and Mohar, B. (1992), “Optimal linear labelings and eigenvalues of graphs,” *Discrete Appl. Math.*, 36, 153–168.
- Kielpinski, D., Monroe, C., and Wineland, D. (2002), “Architecture for a large-scale ion-trap quantum computer,” *Nature*, 417, 709–711.
- Kim, J. (2014), “Trapped Ions Make Impeccable Qubits,” *Physics*, 7, 119.
- Kim, J. and Kim, C. (2009), “Integrated optical approach to trapped ion quantum computation,” *Quantum Inf. & Comput.*, 9, 181–202.
- Kim, J., Nuzman, C., Kumar, B., Lieuwen, D., Kraus, J., Weiss, A., Lichtenwalner, C., Papazian, A., Frahm, R., Basavanthally, N., Ramsey, D., Aksyuk, V., Pardo, F., Simon, M., Lifton, V., Chan, H., Haueis, M., Gasparyan, A., Shea, H., Arney, S., Bolle, C., Kolodner, P., Ryf, R., Neilson, D., and Gates, J. (2003), “1100 X 1100 port MEMS-based optical crossconnect with 4-dB maximum loss,” *IEEE Photon. Technol. Lett.*, 15, 1537–1539.
- Kitaev, A. Y. (1995), “Quantum measurements and the Abelian stabilizer problem,” *arXiv preprint quant-ph/9511026*.

- Kitaev, A. Y. (2003), “Fault-tolerant quantum computation by anyons,” *Annals of Physics*, 303, 2–30.
- Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thomé, E., Bos, J. W., Gaudry, P., Kruppa, A., Montgomery, P. L., Osvik, D. A., et al. (2010), “Factorization of a 768-bit RSA modulus,” in *Advances in Cryptology–CRYPTO 2010*, pp. 333–350, Springer.
- Kliuchnikov, V., Maslov, D., and Mosca, M. (2013), “Asymptotically Optimal Approximation of Single Qubit Unitaries by Clifford and T Circuits Using a Constant Number of Ancillary Qubits,” *Phys. Rev. Lett.*, 110, 190502.
- Knill, E. (2004), “Fault-tolerant postselected quantum computation: Schemes,” *arXiv preprint quant-ph/0402171*.
- Knill, E., Laflamme, R., and Milburn, G. J. (2001), “A scheme for efficient quantum computation with linear optics,” *nature*, 409, 46–52.
- Knoernschild, C., Zhang, X. L., Isenhower, L., Gill, A. T., Lu, F. P., Saffman, M., and Kim, J. (2010), “Independent individual addressing of multiple neutral atom qubits with a micromirror-based beam steering system,” *Appl. Phys. Lett.*, 97, 134101.
- Ladd, T., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C., and O’Brien, J. (2010a), “Quantum computers,” *Nature*, 464, 45–53.
- Ladd, T. D., Jelezko, F., Laflamme, R., Nakamura, Y., Monroe, C., and OBrien, J. L. (2010b), “Quantum computers,” *Nature*, 464, 45–53.
- Laflamme, R., Miquel, C., Paz, J. P., and Zurek, W. H. (1996), “Perfect quantum error correcting code,” *Physical Review Letters*, 77, 198.
- Langer, C., Ozeri, R., Jost, J. D., Chiaverini, J., DeMarco, B., Ben-Kish, A., Blakestad, R., Britton, J., Hume, D., Itano, W., et al. (2005), “Long-lived qubit memory using atomic ions,” *Physical review letters*, 95, 060502.
- Lidar, D. A. and Brun, T. A. (2013), *Quantum error correction*, Cambridge University Press.
- Loss, D. and DiVincenzo, D. P. (1998), “Quantum computation with quantum dots,” *Physical Review A*, 57, 120.
- Lu, F. and Marinescu, D. C. (2007), “Quantum Error Correction of Time-Correlated Errors,” *Quantum Information Processing*, 6, 273–293.

- Mariantoni, M., H., W., Yamamoto, T., Neeley, M., Bialczak, R. C., Chen, Y., Lenander, M., Lucero, E., O’Connell, A. D., Sank, D., Weides, M., Wenner, J., Yin, Y., Zhao, J., Korotkov, A. N., Cleland, A. N., and Martinis, J. M. (2011), “Implementing the quantum von Neumann architecture with superconducting circuits,” *Science*, 334, 61–65.
- Marx, R., Fahmy, A., Myers, J., Bermel, W., and Glaser, S. (2000), “Approaching five-bit NMR quantum computing,” *Physical Review A*, 62, 012310.
- Maslov, D., Falconer, S., and Mosca, M. (2007), “Quantum Circuit Placement: Optimizing Qubit-to-qubit Interactions through Mapping Quantum Circuits into a Physical Experiment,” in *Design Automation Conference, 2007. DAC ’07. 44th ACM/IEEE*, pp. 962–965.
- Maslove, D., Falconer, S., and Mosca, M. (2008), “Quantum circuit placement,” *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27, 752–763.
- Maunz, P., Olmschenk, S., Hayes, D., Matsukevich, D., Duan, L., and Monroe, C. (2009), “Heralded quantum gate between remote quantum memories,” *Physical review letters*, 102, 250502.
- Mazzola, L., Piilo, J., and Maniscalco, S. (2010), “Sudden transition between classical and quantum decoherence,” *Physical review letters*, 104, 200401.
- Meier, A. M., Eastin, B., and Knill, E. (2012), “Magic-state distillation with the four-qubit code,” *arXiv preprint arXiv:1204.4221*.
- Metodi, T., Thaker, D., Cross, A., Chong, F., and Chuang, I. (2005), “A quantum logic array microarchitecture: Scalable quantum data movement and computation,” in *Proc. 38th Annual IEEE/ACM Internat. Symp. on Microarchitecture (MICRO-38)*, pp. 12–23.
- Metodi, T., Thaker, D., Cross, A., Chong, F., and Chuang, I. (2006), “Scheduling physical operations in a quantum information processor,” in *Proceedings of SPIE*, vol. 6244, pp. 210–221, Citeseer.
- Monroe, C. and Kim, J. (2013), “Scaling the Ion Trap Quantum Processor,” *Science*, 339, 1164.
- Monroe, C., Raussendorf, R., Ruthven, A., Brown, K. R., Maunz, P., Duan, L.-M., and Kim, J. (2014), “Large scale modular quantum computer architecture with atomic memory and photonic interconnects,” *Phys. Rev. A*, 89, 022317.
- Monz, T., Schindler, P., Barreiro, J., Chwalla, M., Nigg, D., Coish, W., Harlander, M., Hänsel, W., Hennrich, M., and Blatt, R. (2011), “14-qubit entanglement: Creation and coherence,” *Physical Review Letters*, 106, 130506.

- Mount, E., Baek, S.-Y., Blain, M., Stick, D., Gaultney, D., Crain, S., Noek, R., Kim, T., Maunz, P., and Kim, J. (2013), “Single qubit manipulation in a microfabricated surface electrode ion trap,” *New Journal of Physics*, 15, 093018.
- Nakamura, Y., Pashkin, Y. A., and Tsai, J. (1999), “Coherent control of macroscopic quantum states in a single-Cooper-pair box,” *Nature*, 398, 786–788.
- Negrevergne, C., Mahesh, T., Ryan, C., Ditty, M., Cyr-Racine, F., Power, W., Boulant, N., Havel, T., Cory, D., and Laflamme, R. (2006), “Benchmarking quantum control methods on a 12-qubit system,” *Physical review letters*, 96, 170501.
- Nielsen, M. A. and Chuang, I. L. (2000), *Quantum computation and quantum information*, Cambridge university press.
- Noek, R., Vrijsen, G., Gaultney, D., Mount, E., Kim, T., Maunz, P., and Kim, J. (2013a), “High speed, high fidelity detection of an atomic hyperfine qubit,” *Optics letters*, 38, 4735–4738.
- Noek, R., Kim, T., Mount, E., Baek, S.-Y., Maunz, P., and Kim, J. (2013b), “Trapping and cooling of 174Yb^+ ions in a microfabricated surface trap,” *Journal of the Korean Physical Society*, 63, 907–913.
- Olmschenk, S., Younge, K. C., Moehring, D. L., Matsukevich, D. N., Maunz, P., and Monroe, C. (2007), “Manipulation and detection of a trapped Yb^+ hyperfine qubit,” *Phys. Rev. A*, 76, 052314.
- Patel, Y. (2010), “Communication and Control for Quantum Circuits,” .
- Preskill, J. (1998a), “Lecture notes for physics 229: Quantum information and computation,” *California Institute of Technology*.
- Preskill, J. (1998b), “Reliable quantum computers,” *Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, 454, 385–410.
- Saffman, M. and Walker, T. (2005), “Analysis of a quantum logic device based on dipole-dipole interactions of optically trapped Rydberg atoms,” *Physical Review A*, 72, 022347.
- Schlosshauer, M. A. (2007), *Decoherence: and the quantum-to-classical transition*, Springer Science & Business Media.
- Schwemmer, C., Tóth, G., Niggelbaum, A., Moroder, T., Gross, D., Gühne, O., and Weinfurter, H. (2014), “Experimental comparison of efficient tomography schemes for a six-qubit state,” *Physical review letters*, 113, 040503.

- Shabani, A., Kosut, R., Mohseni, M., Rabitz, H., Broome, M., Almeida, M., Fedrizzi, A., and White, A. (2011), “Efficient measurement of quantum dynamics via compressive sensing,” *Physical review letters*, 106, 100401.
- Shor, P. (1997), “Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer,” *SIAM Journal on Computing*, 26, 1484–1509.
- Shor, P. W. (1995), “Scheme for reducing decoherence in quantum computer memory,” *Physical review A*, 52, R2493.
- Steane, A. M. (1996), “Error correcting codes in quantum theory,” *Physical Review Letters*, 77, 793–797.
- Suchara, M., Faruque, A., Lai, C.-Y., Paz, G., Chong, F. T., and Kubiawicz, J. (2013a), “Comparing the Overhead of Topological and Concatenated Quantum Error Correction,” *arXiv preprint arXiv:1312.2316*.
- Suchara, M., Kubiawicz, J., Faruque, A., Chong, F. T., Lai, C.-Y., and Paz, G. (2013b), “QuRE: The Quantum Resource Estimator Toolbox,” in *Computer Design (ICCD), 2013 IEEE 31st International Conference on*, pp. 419–426, IEEE.
- Svore, K., Cross, A., Aho, A., Chuang, I., and Markov, I. (2004), “Toward a software architecture for quantum computing design tools,” in *Proceedings of the 2nd International Workshop on Quantum Programming Languages (QPL)*, pp. 145–162.
- Svore, K. M., Aho, A. V., Cross, A. W., Chuang, I., and Markov, I. L. (2006a), “A Layered Software Architecture for Quantum Computing Design Tools,” *Computer*, 39, 74–83.
- Svore, K. M., Divincenzo, D. P., and Terhal, B. M. (2006b), “Noise threshold for a fault-tolerant two-dimensional lattice architecture,” *arXiv preprint quant-ph/0604090*.
- Thaker, D., Metodi, T., Cross, A., Chuang, I., and Chong, F. (2006), “Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing,” in *ACM SIGARCH Computer Architecture News*, vol. 34, pp. 378–390, IEEE Computer Society.
- Unruh, W. G. (1995), “Maintaining coherence in quantum computers,” *Physical Review A*, 51, 992.
- Van Meter, R. and Horsman, C. (2013), “A blueprint for building a quantum computer,” *Communications of the ACM*, 56, 84–93.
- Van Meter, R. and Itoh, K. M. (2005), “Fast quantum modular exponentiation,” *Phys. Rev. A*, 71, 052320.

- Van Meter, R., Itoh, K. M., and Ladd, T. D. (2006), “Architecture-Dependent Execution Time of Shor’s Algorithm,” in *Proc. Int. Symp. on Mesoscopic Superconductivity and Spintronics (MS+S2006)*, available as <http://arxiv.org/abs/quant-ph/0507023>.
- Van Meter, R., Munro, W. J., Nemoto, K., and Itoh, K. M. (2008), “Arithmetic on a Distributed-memory Quantum Multicomputer,” *J. Emerg. Technol. Comput. Syst.*, 3, 2:1–2:23.
- Vandersypen, L., Steffen, M., Breyta, G., Yannoni, C., Sherwood, M., and Chuang, I. (2001), “Experimental realization of Shor’s quantum factoring algorithm using nuclear magnetic resonance,” *Nature*, 414, 883–887.
- Vedral, V., Barenco, A., and Ekert, A. (1996a), “Quantum networks for elementary arithmetic operations,” *Phys. Rev. A*, 54, 147–153, <http://arXiv.org/quant-ph/9511018>.
- Vedral, V., Barenco, A., and Ekert, A. (1996b), “Quantum Networks for Elementary Arithmetic Operations,” *Phys. Rev. A*, 54, 147.
- Viamontes, G. F., Markov, I. L., and Hayes, J. P. (2003), “Improving gate-level simulation of quantum circuits,” *Quantum Information Processing*, 2, 347–380.
- Viamontes, G. F., Markov, I. L., and Hayes, J. P. (2009), *Quantum circuit simulation*, Springer Science & Business Media.
- Whitney, M., Isailovic, N., Patel, Y., and Kubiawicz, J. (2007), “Automated generation of layout and control for quantum circuits,” in *Proc. of the 4th Internat. Conf. on Computing Frontiers*, pp. 83–94.
- Whitney, M. G., Isailovic, N., Patel, Y., and Kubiawicz, J. (2009), “A fault tolerant, area efficient architecture for Shor’s factoring algorithm,” *ACM SIGARCH Computer Architecture News*, 37, 383–394.
- Wootters, W. K. and Zurek, W. H. (1982), “A single quantum cannot be cloned,” .
- Zeng, B., Cross, A., and Chuang, I. L. (2011), “Transversality versus universality for additive quantum codes,” *Information Theory, IEEE Transactions on*, 57, 6272–6284.
- Zhou, X., Leung, D. W., and Chuang, I. L. (2000), “Methodology for Quantum Logic Gate Construction,” *Phys. Rev. A*, 62, 052316.
- Zhu, S., Monroe, C., and Duan, L. (2006), “Trapped ion quantum computation with transverse phonon modes,” *Physical review letters*, 97, 50505.
- Zurek, W. H. (2003), “Decoherence and the transition from quantum to classical—REVISITED,” *arXiv preprint quant-ph/0306072*.

Biography

Muhammad Ahsan was born on July 28, 1984 in Lahore, Pakistan. He received his primary education from Crescent Model Higher Secondary School. He completed his intermediate (high school) degree from the Government College and University, Lahore. His B.Sc. degree was earned in Electrical Engineering from University of Engineering and Technology, Lahore. He graduated from Duke university, USA with Masters degree and is currently enrolled as a Ph.D. candidate at the same institute. His general research interests include quantum computing and computer architecture. After receiving his Ph.D. degree he will work for some industrial research lab to obtain further training in research.