THE UNIVERSITY OF CHICAGO

ARCHITECTING QUANTUM COMPUTER SYSTEMS IN THE PRESENCE OF NOISE

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
YONGSHAN DING

CHICAGO, ILLINOIS
AUGUST 2021

To my parents, Genlin and Shuowen.

And to my wife, Meizi.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would not have reached this point without the support from so many people in my life. I am very grateful for the incredible environment of creativity, kindness, and enthusiasm created by my family, mentors, colleagues, and friends.

First and foremost, I would like to thank my advisor, Fred Chong, for being a source of endless support, encouragement, and wisdom. I would walk away from every meeting with Fred thinking differently about one idea and becoming enthusiastic about several more. During my time at UChicago, Fred gave me the most generous support and advice on work and life. For that, I am incredibly grateful.

I also wish to thank Margaret Martonosi and Diana Franklin for serving on my committee. I have always appreciated their strong support throughout my graduate study, whether it is guidance in several projects or simply advice on research and teaching. I want to extend my warmest thanks to Ken Brown. Conversations with Ken are always enlightening. I am very fortunate to have them as mentors and role models during graduate school.

Ryan O'Donnell first introduced me to the field of quantum computing. His course on quantum computing at CMU was so good that it answered the question, "what should I do with my life?". I am grateful to Ike Chuang, Aram Harrow, and Peter Shor for hosting me at MIT in the spring of 2020 and virtually in the summer of 2020. A special thank you to Ike for the discussions on the "art of teaching"; they are highly prized and taken to heart. I also want to thank Umesh Vazirani for an inspiring in-person month-long visit to the Simons Institute at Berkeley in the summer of 2021.

Ryerson Hall and Crerar Library have been a fantastic place to work, whatever the time of day or night, because a fun group of people came here every day and were eager to share, argue, and build. Online Zoom meetings over the last year were never nearly as fun; I miss being around this crowd: Adam Holmes, Casey Duckering, Gokul Subramanian Ravi, Jonathan Baker, Kate Smith, Pranav Gokhale, Reza Jokar, Rich Rines, Ryan Wu, Sophia

Lin, and Yunong Shi. They all have made enormous contributions to the work described in this dissertation. I am grateful to my collaborators for every thought-provoking discussion over the years.

I am saving my most special thanks to Mom, Dad, and my wife. I am extremely grateful for their unwavering support and love, their patience in listening to my research, and their belief in my ability to pull this off. Today I can finally say I did it!

ABSTRACT

Quantum computers may solve some problems far beyond the reach of classical digital computers. However, emerging quantum systems are typically noisy and difficult to control. These noises create significant difficulties for the practical usage of quantum systems, leaving a substantial gap between the requirements of quantum applications and the realities of noisy devices. Bridging this gap is crucial – the dissertation shows that the system software can adapt to the constraints of large applications and noisy hardware. In particular, this dissertation combines systems techniques at three main levels: *(i)*. Gate-level noise mitigation, which enhances the robustness of quantum processors through coordinated control instructions that reduce errors; *(ii)*. Program-level quantum memory management, which addresses problems of qubit allocation throughout a quantum program and implements automated tools to selectively reuse qubits, much like in garbage collection for classical programs; *(iii)*. Architecture-level quantum error correction with low overhead, which seeks to provide correctness guarantees for quantum applications by encoding quantum bits so that errors can be detected and corrected, analogous to classical error-correcting codes.

CHAPTER 1

INTRODUCTION

1.1 Quantum Computing: A Historical Perspective

Paul Benioff began research on the theoretical possibility of building a quantum computer in the 1970s, resulting in his 1980 paper on quantum Turing machines [Ben80]. His work was influenced by the work of Charles Bennett on classical reversible Turing machines from [Ben73a].

In 1982, the Nobel-winning physicist Richard Feynman famously imagined building a quantum computer to tackle problems in quantum mechanics [Fey18, Llo96]. The theory of quantum mechanics aims to simulate material and chemical processes by predicting the behavior of the elementary particles involved, such as the electrons and the nuclei. These simulations quickly become unfeasible on traditional digital computers, which simply could not model the staggering number of all possible arrangements of electrons in even a very small molecule. Feynman then turned the problem around and proposed a simple but bold idea: why don't we store information on individual particles that already follow the very rules of quantum mechanics that we try to simulate?

The idea of quantum computation was made rigorous by pioneers including [Deu85, Deu89] and [Alb83]. Since then, the development of quantum computing has profoundly altered how physicists and chemists think about and use quantum mechanics. For instance, by inventing new ways of encoding a quantum many-body system as qubits on a quantum computer, we gain insights on the best quantum model for describing the electronic structure of the system. It gives rise to interdisciplinary fields like quantum computational chemistry. As recent experimental breakthroughs and theoretical milestones in quantum simulation are made, we can no longer talk about how to study a quantum system without bringing quantum computation to the table.

For computer scientists, the change that quantum computing brings has also been nothing short of astounding. It is so far the only new model of computing that is not bounded by the extended Church-Turing thesis in [BV97, Sim97], which states that all computers can only be polynomially faster than a probabilistic Turing machine. Strikingly, a quantum computer can solve certain computational tasks drastically more efficiently than anything ever imagined in classical computational complexity theory.

It is not until the mid 1990s that the power of quantum computing was getting fully appreciated. In 1993, [BV97] demonstrated a quantum algorithm with exponential speedup over any classical algorithms, deterministic or randomized, for a computational problem named recursive Fourier sampling. Many more astonishing discoveries followed. In 1994, [Sim97] showed another computational problem that a quantum computer has an exponential advantage over any classical computers.

Then in the same year, [Sho94a, Sho99] discovered that more problems, namely factoring large integers and solving discrete logarithms, also have efficient solutions on a quantum computer, far more so than any classical algorithms that are ever known. The implication of this discovery is breathtaking. Existing cryptographic codes encrypt today's private network communications, data storage, and financial transactions, relying on the fact that prime factorization for sufficiently large integers is so difficult that the most powerful digital supercomputers could take thousands or millions of years to compute. But the security of our private information could be under threat, should a quantum computer capable of running Shor's algorithm be built.

In 1996, another algorithm by Lov Grover was discovered in [Gro96a]. Once again, a quantum algorithm is shown to provide improvement over classical algorithms, and in this case Grover's algorithm exhibits quadratic speedup for the problem of *unstructured database search* in which we are given a database and aim to find some marked items. For example, given an unordered set S of N elements, we want to find where $x \in S$ is located in the set.

Classically, we need $O(N)$ accesses to the database in the worst case, while quantumly, we can do it with $O(\sqrt{N})$ accesses.

These are just a few examples of quantum algorithms that have been discovered. When implemented appropriately on a quantum computer, they offer efficient solutions to problems that seem to be intractable in the classical computing paradigm.

Building a quantum computer is, however, extremely challenging. When the idea was first proposed, no one knew how to build such powerful computers. To realize the computational power, we must learn to coherently control and manipulate highly-entangled, complex, physical systems to near perfection.

In the last 30 years or so, technologies for manufacturing quantum chips have significantly advanced. Today, we are at an exciting time where small and intermediate-scale prototypes have been built. It marks a new era for quantum computing, as John Preskill, a long-time leader in quantum computing at Caltech, puts it, "we have entered the *Noisy Intermediate-Scale Quantum* (NISQ) era," in [Pre18] in which quantum computing hardware is becoming large and reliable enough to perform small useful computational tasks. Research labs from both academia and industry, domestic and abroad, are now eager to experimentally demonstrate a first application of quantum computers to some real-world problems that any classical computers would have a hard time solving efficiently.

1.2 Building a Quantum Processor Architecture

A quantum computer implements a fundamentally different model of computation than a modern classical computer does. It would be surprising if the exact design of a computer architecture would extend well for a quantum computer as shown in [CFM17, MR19]. In Figure 1.1, the architectural design of a quantum computer resembles that of a classical computer in the 1950s where device constraints are so high that full-stack sharing of information is required from algorithms to devices. In time, as technology advances and resource

becomes abundant, a quantum computer perhaps will adapt to the modularity and layering models as seen in classical architectures. But in the short term, as long as the NISQ era lasts, it is premature to copy the abstraction layers of today's conventional computer systems to a quantum system.

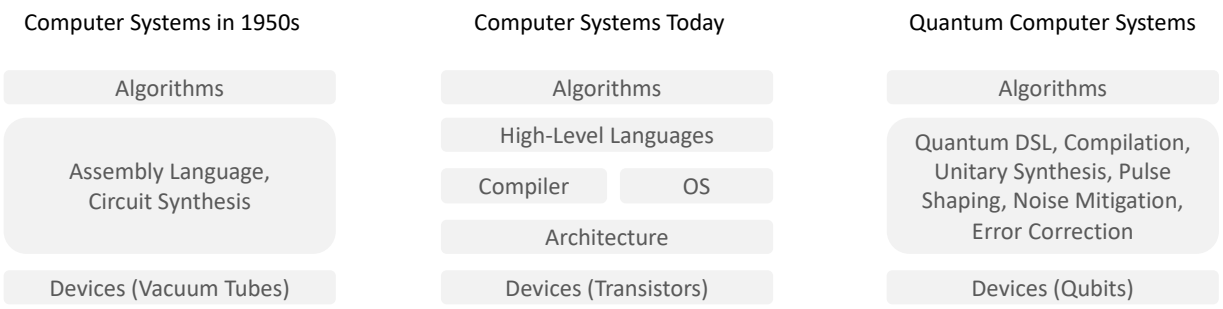


Figure 1.1: Architectural designs of classical versus quantum computers. The abstraction layers for 1950's classical computing, today's classical computing, and quantum computing are compared.

Furthermore, quantum information processing is fundamentally different than what computer engineers are used to. For instance, for conventional computers, engineers go to great lengths in minimizing the noises caused by quantum mechanics in the transistor components. Rather than suppressing its effects, a quantum computer harnesses the power of quantum mechanics. As such, the control apparatus for a quantum computer would look drastically different from that of a conventional computer.

In reality, for successful operation, a quantum computer must implement a well-isolated physical system that encodes a sufficiently large number of qubits, and controls these qubits with extremely high speed and precision in order to carry out computation. The rest of the section describes at a high level the key components in a fully functional quantum computer architecture. The developments of digital quantum computers, for both the NISQ and FT era, still face challenges, which comprise of reliably addressing and controlling qubits and correcting errors. The control complexity becomes overwhelming as the number of qubits scale up, necessitating system-level automation to guarantee the successful execution of

quantum programs as in [SHR18]. As such, classical computers are needed to control and assist the quantum processor. Quantum computers are generally viewed as co-processors or accelerators of classical computers as shown in Figure 2.6.

To some extent, the quantum computer architecture illustrated above arguably resembles in-memory processing or reconfigurable computing architectures. As shown in Figure 2.6, inside a QPU, quantum data are implemented by physical (quantum mechanical) objects such as atoms while quantum gates are control signals such as lasers acting on the data – this "gates-go-to-data" model of computation motivates a control unit close to the quantum data and an interface that talks frequently with the quantum memory and the classical memory.

1.2.1 Mapping Quantum Circuits to Target Architecture

Once the underlying machine technology is defined, the next step is to translate a quantum program to a sequence of gates that a quantum machine recognizes and natively supports. This process include determine the choice of *instruction set architecture* (ISA), quantum gate decomposition/synthesis, data movement, and pulse execution. Systems software that performs such mapping from quantum circuits to target machine code is called a *quantum compiler* or *quantum transpiler*. The mapping must be able to resolve several *architectural constraints*. This means considering the following two aspects:

1) *Native Quantum Gates.* There are certain quantum logic gates that are supported in a given device architecture. In most cases, this gate set is "Clifford+T" gates, comprised of the `CNOT` gate, `NOT` gate (or `X` gate), Hadamard gate (or `H` gate) and `T` gate. This is a common set for most of today's gate-based quantum hardware prototypes, as well as for large-scale fault-tolerant machines (e.g. with surface code error correction). Given a classical reversible circuit, we can replace each gate with its quantum counter-part. In particular, `NOT` gates and `CNOT` gates can be directly implemented as quantum gates. For

`Toffoli` gates, algorithms exist that decompose them into a sequence of Clifford+T gates [AASD16, AMMR13, KMM12, Mas16, WBS14]. At lower level, some instruction sets are proposed to offer direct control over the target hardware [FRR+19].

2) *Qubit communication.* Multi-qubit quantum gates are implemented by interacting the operand qubits with one another. At the physical level, building large-scale quantum machines with all-to-all qubit connectivity is shown to be extremely challenging. The latest effort from [Ion] offers a machine with 11 fully-connected qubits using trapped-ion technology. Superconducting machines, for instance those by [IBM] and [Lar18], typically have *much* lower connectivity. Any scalable proposal would involve an architecture of limited qubit connectivity and a model for resolving long-distance interactions. As a consequence, interacting qubits that are not directly connected would induce communication costs.

Difference between NISQ and FT machines.

Depending on the topology of the architecture and the model for resolving two-qubit interactions, communication costs will differ. In the context of a NISQ machine, the most frequently used approach to resolve a long-distance two-qubit gate is through swaps, where two (physical) qubits are moved closer by performing a chain of swap gates that connects them. Each `SWAP` gate consists of three `CNOT` gates. The time to complete a swap chain is proportional to the length of the chain. In a FT machine, a logical qubit is encoded by a number of physical qubits. A logical operation is specified by a sequence of physical operations on its physical qubits. For instance, for surface code implementation, physical qubits form a 2D grid with every data qubit connected to its four nearest neighbors through stabilizer ancillas. In essence, a logical operation is defined by specifying how the stabilizer ancillas interact with the data qubits. For instance, a logical two-qubit gate can be defined by braiding, which creates a path between logical qubits, where the stabilzer ancillas along the path do not interact with their neighbors [DHJA+18, JAGH+17]. Although it can ex-

tend to arbitrary length and shape in constant time, two braids are not allowed to cross. We refer interested readers to [FMMC12, Got10] for excellent tutorial.

1.2.2 Logic Implementation and Scheduling

Logic Synthesis

Here we aim to address one essential question in quantum compiling, namely how to (efficiently) implement some arbitrary unitary transformation using a given finite set of realizable quantum gates (i.e., primitive instructions). The complexity of the problem varies, depending on the objectives and assumptions. As a result, efficiency and optimality of the solutions varies. So it is important to recognize the different situations being considered in the community, and categorize the known synthesis techniques into classes accordingly. Some example types of synthesis techniques considered in the rest of the section can be summarized as follows: (i) Choice of universal instruction set, (ii) Single-qubit, multi-qubit, and qudit (i.e., d-level quantum logic) synthesis, (iii) Exact and approximate synthesis.

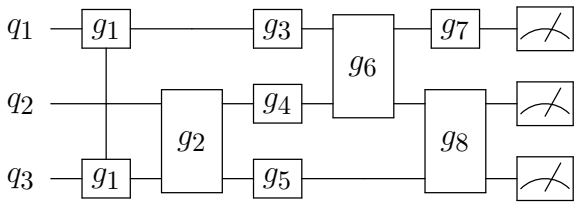
The existence of an efficient synthesis of quantum circuits is largely determined by the choice of instruction set; one can imagine some quantum gates to be more "powerful" than others. Here powerful, which will be defined in subsequent sections, can be informally thought of as being able to cover the entire space of possible unitary gates more quickly. Furthermore, synthesis for multi-qubit unitaries or qudit unitaries is believed to be more difficult in general due to the high dimensionality involved; a general strategy is to decompose the high-dimensional unitary matrices into pieces of one- or two-qubit unitary matrices, for which efficient synthesis methods are known. Lastly, strategies for exactly or approximately synthesizing quantum circuits differ significantly, and consequently, they can have drastically different complexity. For a more comprehensive overview of quantum logic synthesis, we refer readers to [DC20].

For example, if a quantum program is to be executed on a superconducting NISQ com-

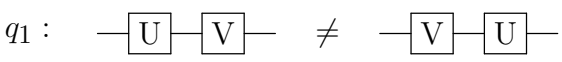
puter (without quantum error correction), then the instruction set would likely consist of single-qubit rotation ($R(\theta)$) gates and two-qubit cross-resonance (CR) gate, for they are easier to implement to high precision. Consequently, the target transformation U of the quantum program is synthesized, exactly or approximately (depending on the precision tolerance). The synthesis procedure typically involves first decomposing the multi-qubit unitary U into sequence of single-qubit unitaries and two-qubit CR gates, and then decomposing the single-qubit unitaries into Pauli rotation gates. The goal would be to synthesize the most efficient circuit (e.g., short in depth and small in number of qubits) to some high precision required by the target computer.

Gate Scheduling

The performance of a quantum circuit also depends on how its gates are scheduled. A *schedule* of a quantum program is a sequence of gate operations on logical qubits. The sequence ordering defines *data dependencies* between gates, where a gate g_2 depends on g_1 if they share a logical qubit and g_2 appears later in the schedule than g_1.



A quantum circuit executes from left to right. A qubit can only be involved in one quantum gate at a time. Data dependencies determine the sequential and parallel execution ordering of the gates in a quantum circuit. For example, two sequential gates back to back in general have strict ordering constraints:

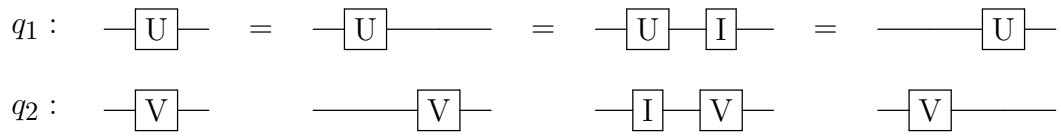


For two arbitrary unitary operators U and V, swapping the order of the two generally yields different results. One can quickly verify by writing down the matrix multiplications for two unitary matrices:

$$U \cdot V \neq V \cdot U$$

Although not always equivalent, unitary matrices *can* sometimes be reordered in special cases. When that happens, we say the two matrices *commute* with each other. We will elaborate on the *commutation relations* later in the section.

Two parallel gates side by side in general has no ordering constraints:



Temporal ordering of parallel gates does not matter because we can check the corresponding tensor products and verify:

$$U \otimes V = (U \otimes I) \cdot (I \otimes V) = (I \otimes V) \cdot (U \otimes I)$$

The impact of gate scheduling can be quite significant in quantum circuits, and many algorithm implementations rely upon the execution of gates in parallel in order to achieve substantial algorithmic speedup. Gate scheduling in quantum algorithms differs from classical instruction scheduling, as gate commutativity introduces another degree of freedom for schedulers to consider. Compared to the field of classical instruction scheduling, quantum gate scheduling has been relatively understudied, with only few systematic approaches being proposed that incorporate these new constraints.

Some common scheduling strategies include:

- *ALAP (As-Late-As-Possible) Scheduling.* One of the simplest scheduling algorithms is the ALAP (As-Late-As-Possible) scheduler. In essence, it starts with the end of the circuit and schedules the last gates needed to be completed, and goes backward to their previous gates. The advantage of ALAP scheduler, as opposed to ASAP (as-soon-as-possible), is the qubits are initialized only when absolutely needed. Since qubits have limited lifetime, it is beneficial to initialize them as late as possible.

- *LPF (Longest-Path-First) Scheduling.* When programs have more complex control flow, prioritization is needed between different parallel modules. The LPF (longest-path-first) scheduler tries to avoid increasing the critical path of the program, thus reducing the circuit depth. [HPJ$^+$15a, JAGH$^+$17] are some example LPF schedulers.

- *Communication-Aware Scheduling.* More advanced schedulers take into account costs of communication, due to two-qubit gates between operands that are far apart. Communication cost varies for different architectures, so does their scheduling strategies. Here we highlight a general technique commonly used for reducing communication: *barrier insertion*. When a two-qubit gate is known to be causing high communication cost (such as long swap distance), separating it from other gates along the critical path can be effective. It is shown that iterative algorithms can benefit from the introduction of barriers as well, as inserting a barrier at the end of each round creates clean divisions between the rounds. [DHJA$^+$18].

- *Adaptive Scheduling.* When there are multiple implementations of the same gate, it is possible to let the scheduler choose the best one based on gate time, qubit fidelity, gate noises, etc. Due to our limited understanding of the noise characteristics of NISQ machines, this strategy remains a challenge.

Scheduling can be done at the logical level or the physical level. In NISQ machines, we

just have the physical level as there is no error correction. While in error-corrected machines, we have both logical (fault-tolerant) and physical level. In the latter, it makes sense to apply these optimizations at both levels, that is to schedule the application program as well as to schedule the error correction routine [HDJA$^+$19, DHJA$^+$18].

1.2.3 Qubit Allocation and Memory Management

Two convenient tools in analyzing qubit mapping are the *device connectivity graph* and the *program interaction graph*. A device connectivity graph, where each node is a qubit and two qubits are connected if the two qubits are allowed to directly interact. For example, in a superconducting device, this means whether or not two qubits are linked by circuit wires (through a coupler such as a capacitor); in a trapped ion device, this means whether or not laser beams can simultaneously address the two qubits. Connectivity graphs for superconducting device is commonly of the 2D mesh/lattice type, as shown in Figure 1.2. In contrast, trapped ion devices typically have much dense connectivity graphs, thanks to flexibility in performing two-qubit gates. Figure 1.3 shows some examples of trapped ion device connectivity graphs.

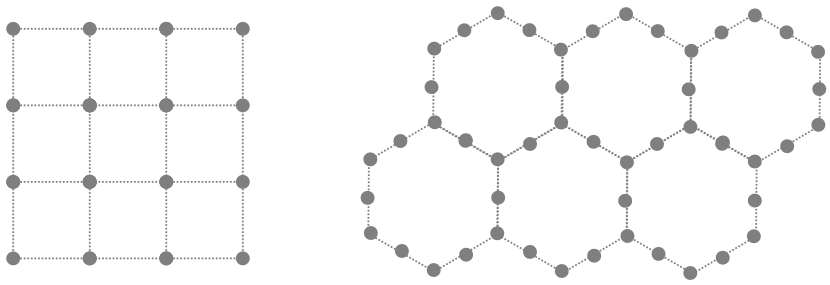


Figure 1.2: Different connectivity graph for superconducting devices. *Left:* 2D square lattice. *Right:* 2D (heavy) hexagonal lattice. The choice of connectivity graphs is typically based on hardware constraints such as wiring bandwidth and noises of circuit components.

Given a quantum program, we can define a program interaction graph as a graph $G = (V, E)$ where V is a set of logical qubits present in the computation, and E is a set of two-

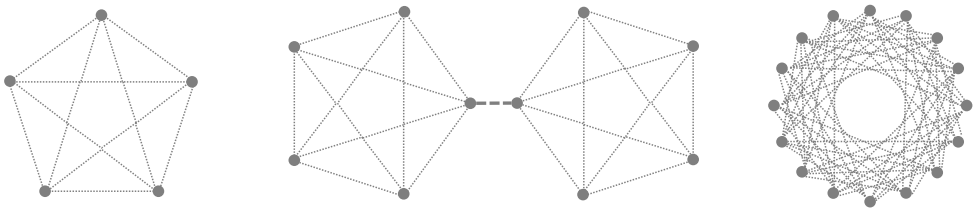


Figure 1.3: Different connectivity graph for trapped ion devices. *Left:* Complete (Clique) connectivity for small number of ions in one trap. *Center:* Weakly connected cliques for multiple traps. *Right:* A long chain of ions in one trap in a tape-like architecture.

qubit interaction gates contained in the program (e.g., CNOT gates). By analyzing this graph, we can perform an optimized *mapping*, which assigns a physical location for each logical qubit $q \in V$. The program interaction graph of the toy circuit can be constructed from enumerating all two-qubit gates, as shown in Figure 1.4.

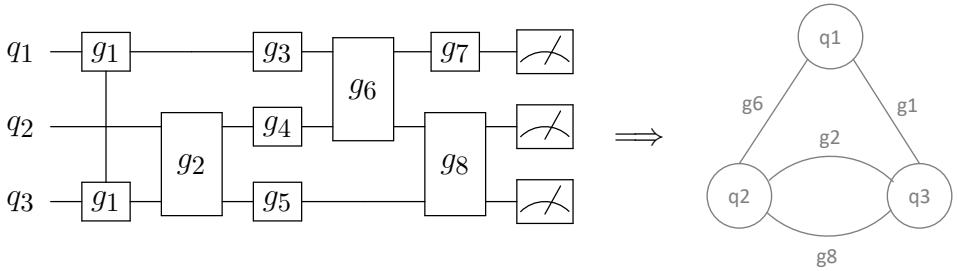


Figure 1.4: The program interaction graph for an example quantum circuit.

The goal of a *quantum memory manager* can be described as to *most efficiently embed a program interaction graph in a device connectivity graph*. Here efficiency means objectives such as minimizing communication and avoiding noisy qubits and links, etc. Once the program interaction graph is embedded, each edge (two-qubit interaction) is weighted by the cost of completing such interaction on the mapped qubits.

In the context of NISQ machines, long-distance interactions are typically resolved by moving or swapping qubits. For simplicity, we call these as "routing qubits". Some architectures (such as trapped ion devices) support shuttling/transporting qubits directly, others (such as superconducting devices) bring two qubits together through a chain of swaps. Moving or swapping qubits not only costs time but also introduces errors.

Having the device connectivity graph is very convenient for analyzing these communication costs. We can model the overhead of long-distance interactions as weights on the edges of the connectivity graph. For example, the weighted distance from qubit q_i to qubit q_j is used to represent the cost of moving from location q_i to location q_j. Most existing algorithms follow a three-step approach to memory management: *i*) Allocate qubits. *ii*) Route long-distance interactions. *iii*) Reusing qubits.

We will dedicate Chapter 4 for advanced strategies of memory management involving qubit reuse.

1.3 Motivation: Working with Quantum Noises

Current Noisy Intermediate-Scale Quantum (NISQ) computers [Pre18, AAB+19, WBD+19, IBM, PVDC+20] aim to isolate and control a non-trivial quantity of quantum bits (qubits) with high precision. Scaling up a quantum computer requires improvements in both the quality of qubits (with longer lifetime) and the quality of gates (with higher fidelity).

For a quantum computer to work, it takes extreme precision to isolate and control qubits. In particular, qubits are very short-lived due to the interactions with the environment. Quantum logic gates can have small drifts when pulses are out-of-focus or out-of-tune. Classical controls and calibrations may have a hard time keeping up in scale, speed, and power. If there were no strategies to overcome the aforementioned examples of noises, they would accumulate and eventually lead to critical failure in computation. Physical noises in current devices put stringent limitations on their computing capabilities. Indeed, this dissertation aims to address the central issue: can we protect information from the adverse impacts of noise with systems techniques, keeping the overheads at a manageable level? We present some affirmative answers in Section 1.4.

One cannot really talk about building scalable quantum computer systems without bringing the discussion of an architectural support for the increasingly complex control and mem-

ory modules to the table. As we enter the NISQ era and beyond, we will need to orchestrate the simultaneous quantum operations on hundreds or thousands of qubits. What kind of microarchitecture can keep up with the speed and bandwidth of quantum information processing? How do we build a reliable interface between classical control/feedback signals and quantum data? Can we efficiently translate and synchronize machine pulses from gate instructions? This may be feasible by hand for small-scale devices with 5-10 qubits, but it will soon become intractable without an automated, robust control system as the size of the devices scales up.

1.3.1 Optimizing Program Performance

To demonstrate practical quantum computation, we must learn to implement applications in an error-tolerant fashion and make the best use of a relatively small number of quantum bits and operations. Compilation tools will play a critical role in achieving these goals. The job of a quantum compiler is to efficiently translate a high-level quantum program into native instructions recognizable by the hardware, through a series of transformations and optimizations. Traditional wisdom from compilation for classical computers can occasionally be inherited or adapted to the quantum case, such as resolving control flows and allocating registers. We put particular emphasis on the aspects of compilation that are unique to quantum computers. Notably, compilation under strict resource constraints has proven challenging, and optimization will have to break traditional abstractions and be customized to algorithmic and device characteristics in a manner never before seen in classical computing. We call attention to a number of important steps specialized for quantum compilation to help ensure the efficiency and correctness needed. To name a few, unitary synthesis focuses on exactly or approximately expressing arbitrary unitary transformations (such as single qubit rotations by an arbitrary angle) in a sequence of elementary gates. The goal of gate scheduling is to utilize commutation relations to determine the ordering of the (possibly parallel)

operations, and to use circuit equivalence to simplify quantum programs. Qubit mapping is another challenge, in that we aim to strategically assign the variables in a quantum program to the qubits available in the system, under multiple constraints such as limited connectivity between qubits, fluctuations in the reliability of qubits and links, and potential opportunities for reclamation and reuse of qubits, etc.

1.3.2 Optimizing Program Reliability

Intertwined with program performance (e.g., resource efficiency) is program reliability. Noise mitigation is one of the biggest challenges facing the QC community. Without strategies to reduce or get around the physical noises, any execution of a quantum program is almost always doomed to fail under such stringent conditions. When noise mitigation techniques are integrated in the systems software, we can make qubits more robust with coordinated control elements to prolong their lifetimes or performing more accurate gate operations with a composition of pulses to improve their fidelities, etc. Current NISQ computers [Pre18] lack the ability to isolate and control a sufficiently large number of quantum bits (qubits) with high precision. There is an urgent need for a new systems stack that provides guarantees for performance and reliability. On one hand, we want to equip a quantum computer with as many qubits as possible to accommodate large quantum applications. On the other hand, we want each qubit to be as long-lived and controllable as possible to run quantum programs fast and reliably.

1.4 Research Challenges and Contributions

This is an incredibly exciting time for quantum computing – research institutions and technology companies worldwide are racing toward practical-scale, fully programmable quantum computers (QC). Quantum systems are inherently noisy, in that one error can occur every few thousand quantum operations. Central to the design of quantum computer systems is

to understand and mitigate those errors. Because of such pervasive impact of noise, it is still a central open question on whether or how QC will adopt the modularity and layering approach that helped scale modern computer systems. To some degree, the current state of QC systems design resembles circuit synthesis and architectural design from the early stage of classical computing. Unlike the old times, however, QC development can benefit from the lessons learned in classical computing. More importantly, we can now use powerful classical computers to optimize QC or improve information processing with a hybrid quantum-classical model. I give three examples from my recent contributions to quantum computer systems design, showing the kinds of problems where architectural solutions are particularly promising.

1.4.1 Engineering Better Qubits — Systematic Noise Mitigation

Noises in quantum systems pose a significant obstacle to scaling up quantum processors. For example, crosstalk error, a form of unwanted interactions between qubits, is prevalent in today's superconducting transmon architecture. Noises like crosstalk are challenging hardware problems due to systematic drifts and inaccurate controls. One of the major thrusts of my research is to discover computer science solutions to managing physical systems.

Recent contributions: In recent work at MICRO [4] with Gokhale et al., I found that crosstalk error can be significantly reduced with software. The key observation is that qubits can be calibrated/tuned dynamically during the execution of quantum programs. By doing so, quantum logic gates (orchestrated by real-time calibration signals) are executed at a significantly higher fidelity. This work proposed a compiler framework for mitigating hardware noises on a superconducting transmon quantum architecture. I showed that software techniques could greatly simplify the hardware complexity necessary to reduce crosstalk in quantum machines. This insight influenced the design methodology and eventual technological course of commercial quantum machines. Prior compilation solutions either are unaware

of hardware noise or perform passive noise mitigation by avoiding some qubits and couplings when noises are present [5, 6]. We demonstrated how to actively tune system parameters (e.g., qubit frequencies) according to input programs to mitigate crosstalk, trading parallelism for higher gate fidelity when necessary. Our methodology has a notable feature: it reduces a complex physical problem to several subproblems, including graph coloring and constraint satisfaction problem (CSP).

1.4.2 Putting Qubits to Work — Quantum Memory Management

In a recent ACM SIGARCH blog [7] co-written with Prof. Fred Chong, I talked about the key challenges and opportunities in quantum memory management due to qubits' unique properties, such as entanglement and no-cloning. The entanglement property of quantum memory imposes intrinsic correlation among separate reads/writes. The no-cloning theorem prevents us from making copies/caches of quantum states during computation. Hence, the design of quantum memory systems must adapt to these constraints.

Recent Contributions: In an ISCA paper [8] with Wu et al., I demonstrated that garbage collection in the presence of hardware noise could be done efficiently on a quantum computer. This paper is the first to show that garbage collection of scratch qubits using uncomputation (i.e., reversing part of the computation to reset qubits) is, surprisingly, often cheaper than moving new scratch qubits across a quantum chip. It changed how quantum systems allocate and manage qubits, greatly extending their practical use for a given machine size. During the uncomputation process, I utilize information such as program structures, data locality, and, most importantly, qubit/gate noise. Fortunately, I can directly adapt a rich set of techniques from classical compilation, including LLVM instrumentation, register allocation, and parallel programming model. In this work, I promote the "active quantum volume" as a crucial compiler design metric, which characterizes the workload of noisy qubits over the course of a program. This work also provides the syntactic and architectural supports that

allow programmers to easily express the program structure and verify its correctness.

Quantum memory readouts (i.e., measurements) are a critical part of quantum computation. These measurements are, however, typically slow and prone to error in near-term devices. In a sequence of recent papers with Gokhale et al. [9, 10], we identified parallel memory readout opportunities by grouping measurements into mutually commuting sets. As a result, we achieved asymptotically lower measurement costs for promising near-term applications such as variational quantum eigensolver (VQE). Such improvement on measurement cost is crucial for near-term quantum devices. Our work is recognized by the IBM Q Best Paper Award in 2019 and QCE Best Paper Award in 2020.

1.4.3 Scaling Up Systems — Quantum Error Correction

In the long run, scaling up to fault-tolerant systems is critical if quantum computers are to realize their full potential. Conventional quantum error-correcting codes usually have significant resource costs in terms of system size and operation count, partly because they are designed to correct general errors regardless of noise sources. My research seeks to implement quantum error correction codes by adapting to the underlying architectures and target applications.

Recent Contributions: In an error-corrected system, some logical operations (e.g., Clifford gates) are easy, while others are difficult to implement with high accuracy. A leading approach to improving the difficult operations' accuracy is to assist each operation with a special resource state. The accuracy of the operation is thus determined by the fidelity of the resource state, which can be prepared by a filtering procedure called magic state distillation. My research contributed an essential step towards lowering the cost of magic state distillation on a quantum computer. In a MICRO paper [11] and its follow-up [12] with Holmes et al., I proposed magic state functional units, taking the distillation procedure from a proof-of-concept quantum circuit to a scalable microarchitectural solution. We proposed a novel

organization and data-path for an error-corrected quantum processor. Specifically, we map the distillation circuit and computation circuit onto different processor regions and schedule the preparation and distribution of magic states for computation. We found significant performance gain by uniting knowledge across multiple disciplines. We utilize architectural design concepts (e.g., ASIC chip layout optimization and register renaming), graph theoretical concepts (e.g., graph partitioning and community clustering), and physical concepts (e.g., force-directed annealing).

1.5 Dissertation Outline

The rest of the thesis is organized as follows. Chapter 2 introduces the background information on quantum logic/programming and quantum computer systems design. Chapter 3 presents a software technique to reduce hardware crosstalk, which is the dominant source of gate error for systems like transmon architectures. After an exploration of the hardware architecture design space, we demonstrate how software techniques, like graph coloring and SMT solvers, can greatly simplify hardware complexity needed for scaling up quantum systems. Chapter 4 presents a application- and device adapted compiler framework to perform quantum memory management with high efficiency and high reliability. Chapter 5 extends the scheduling and mapping techniques to future large-scale fault-tolerant architectures. It quantifies and reduces the resource overhead for quantum error correction, thus shedding lights on a feasible path towards practical fault tolerance. Finally, Chapter 6 discusses some exciting future research directions raised by the work in the thesis.

Bibliographic Note

This dissertation is based on the author's published research work, including a book and a series of papers. Most of the research was performed jointly with other researchers. The dissertation author is the primary contributor for the work of which he is the first author.

- Yongshan Ding and Frederic T Chong. *Quantum Computer Systems: Research for Noisy Intermediate-Scale Quantum Computers*, volume 15. Morgan & Claypool Publishers, 2020

- Yongshan Ding, Pranav Gokhale, Sophia Fuhui Lin, Richard Rines, Thomas Propson, and Frederic T Chong. Systematic crosstalk mitigation for superconducting qubits via frequency-aware compilation. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 201–214. IEEE, 2020

- Yongshan Ding, Xin-Chuan Wu, Adam Holmes, Ash Wiseth, Diana Franklin, Margaret Martonosi, and Frederic T Chong. Square: strategic quantum ancilla reuse for modular quantum programs via cost-effective uncomputation. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 570–583. IEEE, 2020

- Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 828–840. IEEE, 2018

- Adam Holmes, Yongshan Ding, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T Chong. Resource optimized quantum architectures for surface code implementations of magic-state distillation. *Microprocessors and Microsystems*, 67:56–70, 2019

During the course of his doctoral study, the author has co-authored additional research articles that are not included in this dissertation.

- Weilong Cui, Yongshan Ding, Deeksha Dangwal, Adam Holmes, Joseph McMahan, Ali Javadi-Abhari, Georgios Tzimpragos, Frederic Chong, and Timothy Sherwood. Charm:

A language for closed-form high-level architecture modeling. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 152–165. IEEE, 2018

- Pranav Gokhale, Olivia Angiuli, Yongshan Ding, Kaiwen Gui, Teague Tomesh, Martin Suchara, Margaret Martonosi, and Frederic T Chong. Minimizing state preparations in variational quantum eigensolver by partitioning into commuting families. *arXiv preprint arXiv:1907.13623*, 2019

- Pranav Gokhale, Yongshan Ding, Thomas Propson, Christopher Winkler, Nelson Leung, Yunong Shi, David I Schuster, Henry Hoffmann, and Frederic T Chong. Partial compilation of variational algorithms for noisy intermediate-scale quantum machines. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 266–278, 2019

- Adam Holmes, Mohammad Reza Jokar, Ghasem Pasandi, Yongshan Ding, Massoud Pedram, and Frederic T Chong. Nisq+: Boosting quantum computing power by approximating quantum error correction. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 556–569. IEEE, 2020

- Yunong Shi, Pranav Gokhale, Prakash Murali, Jonathan M Baker, Casey Duckering, Yongshan Ding, Natalie C Brown, Christopher Chamberland, Ali Javadi-Abhari, Andrew W Cross, et al. Resource-efficient quantum computing by breaking abstractions. *Proceedings of the IEEE*, 108(8):1353–1370, 2020

- Pranav Gokhale, Olivia Angiuli, Yongshan Ding, Kaiwen Gui, Teague Tomesh, Martin Suchara, Margaret Martonosi, and Frederic T Chong. O (N**3) measurement cost for variational quantum eigensolver on molecular hamiltonians. *IEEE Transactions on Quantum Engineering*, 1:1–24, 2020

- Xin-Chuan Wu, Dripto M Debroy, Yongshan Ding, Jonathan M Baker, Yuri Alexeev, Kenneth R Brown, and Frederic T Chong. Tilt: Achieving higher fidelity on a trapped-ion linear-tape quantum computing architecture. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 153–166. IEEE

CHAPTER 2

PRELIMINARIES

2.1 Quantum Logic and Programming

The theory of quantum computation can be formulated as a neat branch of mathematics. If we consider classical computation as operations under the laws of *boolean algebra*, then quantum computation operates under the rules of *linear algebra*. The probabilistic nature of quantum states adds another layer of complexity to understanding the behavior of quantum computers. Nonetheless, all of it can be beautifully captured in four simple postulates, describing quantum states, composition of quantum systems, measurements, and quantum gates, respectively. This mathematical formulation allows us to reason about how a quantum system behaves under our manipulation, i.e., *quantum logic* in a quantum computer. Throughout this section, basic linear algebra and probability theory concepts are reviewed or referenced when necessary.

Together, the four postulates describe how information is stored and manipulated in a quantum system. In particular, a quantum computer works with a finite set of computational objects called quantum bits (or *qubits*). The *quantum state postulate* defines the superposition state of each qubit. The *composition postulate* then generalizes it to represent a system of multiple qubits, and provides a formal, mathematical definition of the entanglement property. The *measurement postulate* is used to describe how much information can be read out from a quantum system, as well as the consequence of the measurement action to the system. Finally, the *quantum gate postulate* defines the logical operations that transform a quantum system.

2.1.1 States of Quantum Systems

Quantum States

Definition 2.1.1 (Superposition)**.** A single-qubit quantum state $|\psi\rangle$ can be defined as a (column) vector of two complex numbers:

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Here α and β are called the *amplitudes* of the quantum state. It is called a superposition state because we can rewrite it as a *linear combination* of the basis states $|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ as follows:

$$|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}.$$

Lastly, we write the conjugate transpose of $|\psi\rangle$ as a row vector

$$\langle\psi| = (|\psi\rangle)^\dagger = \begin{pmatrix} \alpha^* & \beta^* \end{pmatrix}$$

We can check the condition on the amplitudes by the inner product

$$\langle\psi|\psi\rangle = \alpha\alpha^* + \beta\beta^* = |\alpha|^2 + |\beta|^2 = 1.$$

More generally, we can extend to a *qudit system*: a d-dimensional qudit system is defined

as a superposition of d basis states:

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle + \cdots + \alpha_{d-1} |d-1\rangle ,$$

where $|\alpha_0|^2 + \cdots + |\alpha_{d-1}|^2 = 1$. In theory, we can construct a qudit system using qubits. However, in practice, many quantum systems are intrinsically a multi-level system. For example, a superconducting transmon has infinite levels among which the first few levels are easily accessible. A three-dimensional qudit system is sometimes called a *qutrit* system.

Composition of Quantum systems

In classical computing, when moving from a single-bit system to a system consisting of n number of bit, we use a string of bits to represent the 2^n possible states that the system could be in, for there are exactly 2 choices for each bit. Take two bits, there are four possible states, namely 00, 01, 10 and 11. Naturally, intuition from the superposition principle tells us that, in a quantum computer, the *joint state* of a two-qubit system should be a linear combination of the *four* possible basis states, i.e., $|\psi\rangle = \alpha |00\rangle + \beta |01\rangle + \gamma |10\rangle + \delta |11\rangle$. Indeed, we can build a bigger quantum state from small quantum states using a *tensor product*.

Definition 2.1.2 (Composition)**.** The joint state of two separate quantum systems $|\psi_0\rangle = \sum_j \alpha_j |a_j\rangle$ and $|\psi_1\rangle = \sum_k \beta_k |b_k\rangle$ is represented as the *tensor product* of the components. That is,

$$|\psi\rangle = |\psi_0\rangle \otimes |\psi_1\rangle = \sum_j \sum_k \alpha_j \beta_k (|a_j\rangle \otimes |b_j\rangle)$$

where $|a_j\rangle \otimes |b_j\rangle$ can often be shortened as $|a_j b_k\rangle$.

2.1.2 Quantum Operations

Measurements

How much information can we store in or get out of a single qubit? The amplitudes of a qubit state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ take complex coefficients. The *quantum measurement postulate* in quantum mechanics states that, the only way to read out information from a quantum system is by interacting with the system via measurement, from which we obtain a probabilistic outcome. Formally, we can define the following process:

Definition 2.1.3 (Measurement)**.** When we measure a qubit $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ we observe the basis state $|0\rangle$ with probability $|\alpha|^2$ and the basis state $|1\rangle$ with probability $|\beta|^2$.

The process of measurement is irreversible and probabilistic, meaning once measurement has occurred, the state $|\psi\rangle$ *collapses* into one of the two basis states ($|0\rangle$ or $|1\rangle$) and the original quantum superposition cannot be recovered.

Example 2.1.4. *If $MeasZ$ is the measurement operator (along the computational axis), the measurement outcome for each qubit $MeasZ|\psi\rangle$ will be either $|0\rangle$ or $|1\rangle$ depending on its state. In the Bloch sphere picture, the $MeasZ$ outcome is related to the* latitude *of the quantum state – global phase (longitude) does not matter (see Table 2.1).*

For completeness, we describe the *general measurement rules* for (pure) quantum states. To start with, we pick a measurement basis set, which can be written as a set of matrices $\{M_i\}_i$ satisfying the completeness condition

$$\sum_i M_i^\dagger M_i = I.$$

For instance, for the computational basis measurement, we take $M_1 = |0\rangle\langle 0|$ and $M_2 =$

Initial State	Readout	Final State	Probability
$\|\psi\rangle = \|0\rangle$	0	$\|0\rangle$	100%
$\|\psi\rangle = \|1\rangle$	1	$\|1\rangle$	100%
$\|\psi\rangle = \|+\rangle$	0	$\|0\rangle$	50%
	1	$\|1\rangle$	50%
$\|\psi\rangle = \|-\rangle$	0	$\|0\rangle$	50%
	1	$\|1\rangle$	50%
$\|\psi\rangle = \frac{1}{\sqrt{2}}\|00\rangle + \frac{1}{\sqrt{2}}\|11\rangle$	00	$\|00\rangle$	50%
	11	$\|11\rangle$	50%
$\|\psi\rangle = \alpha\|00\rangle + \beta\|01\rangle + \gamma\|10\rangle + \delta\|11\rangle$	00	$\|00\rangle$	$\|\alpha\|^2$
	01	$\|01\rangle$	$\|\beta\|^2$
	10	$\|10\rangle$	$\|\gamma\|^2$
	11	$\|11\rangle$	$\|\delta\|^2$

Table 2.1: Example measurement outcomes by $MeasZ$ on initial state $|\psi\rangle$.

$|1\rangle\langle 1|$. Upon measurement, we obtain the outcome "i" with probability

$$\mathbf{Pr}[\text{observe}i] = |M_i|\psi\rangle|^2 = \langle\psi|M_i^\dagger M_i|\psi\rangle,$$

which results in a quantum state

$$|\psi'\rangle = \frac{M_i|\psi\rangle}{|M_i|\psi\rangle|} = \frac{M_i|\psi\rangle}{\sqrt{\langle\psi|M_i^\dagger M_i|\psi\rangle}}.$$

Quantum Gates

What kind of quantum logic operations can we achieve? How do we transform from one quantum state to another? Mathematically, this process is defined as a "norm-preserving linear transformation", in other words, a unitary transformation. Transforming from $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ to $|\varphi\rangle = \alpha'|0\rangle + \beta'|1\rangle$, we must have $|\alpha|^2 + |\beta|^2 = |\alpha'|^2 + |\beta'|^2 = 1$ to ensure that both $|\psi\rangle$ and $|\varphi\rangle$ are valid quantum states. If we represent a quantum state $|\psi\rangle$ as a column vector as in Definition 2.1.1, then we can represent the quantum logic gate on the state vector by a linear operator U given by a matrix.

Definition 2.1.5 (Transformation)**.** A valid logical transformation must map a quantum

state to another quantum state. That is, for $U : |\psi\rangle \rightarrow U|\psi\rangle$, we require $|\langle\psi|\psi\rangle|^2 = 1 = |\langle\psi|U^\dagger U|\psi\rangle|^2$. Formally, this means that U is represented by a *unitary matrix* (i.e., $U^\dagger U = I$).

Unlike measurement operators which are irreversible and probabilistic, such logical transformation is *reversible* (since unitary matrix U is always invertible) and *deterministic* (since U maps any fixed initial state $|\psi\rangle$ to a unique final state $U|\psi\rangle$). Physically, it means that the transformation is energetically coherent and we can always undo this process by inverting the action. From an information theory perspective, it means that no information is destroyed (or leaked to the environment) under unitary transformations. In other words, knowing the output and what transformation it underwent, we can always recover the input. Notice that this is not always the case in classical boolean logic. Take a common logic gate, the AND gate, as an example - knowing that we obtained the bit 0 from an AND operation of two bits x and y, i.e., $\text{AND}(x, y) = 0$, we cannot tell if we started with $(x, y) = (0, 0)$ or $(0, 1)$ or $(1, 0)$. Hence, we call the AND gate an irreversible gate. An example of nontrivial classical reversible gate is the NOT gate, which negates the two states 0 and 1. Transformations via quantum logic gates, however, are all reversible. It is important to point out that the transformation principle does not account for the effect of noise. For instance, a qubit, when perturbed by the environment, can decohere to a classical state. Such a process is incoherent and not reversible. For simplicity, this chapter will assume an ideal, noise-free situation.

Example 2.1.6. Quantum logic gates *define the set of elementary operations that we can perform in a quantum computer. Let's start with the simplest example, namely quantum gates on a single qubit. A single-qubit gate can be viewed as a transformation that takes one point on the Bloch sphere to another by rotating by an arbitrary angle along a certain axis. Table 2.2 below shows a few examples of single-qubit operations.*

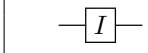
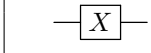
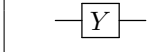
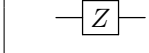
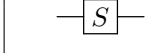
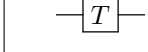
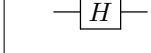
Quantum Gate	Circuit Form	Matrix Form	Truth Table
Identity gate (I)		$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$	$\|0\rangle \mapsto \|0\rangle$ $\|1\rangle \mapsto \|1\rangle$
Not gate (X)		$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$	$\|0\rangle \mapsto \|1\rangle$ $\|1\rangle \mapsto \|0\rangle$
Y gate (Y)		$Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$	$\|0\rangle \mapsto \|i\rangle$ $\|1\rangle \mapsto \|-i\rangle$
Z gate (Z)		$Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$	$\|0\rangle \mapsto \|0\rangle$ $\|1\rangle \mapsto -\|1\rangle$
Phase gate (S)		$S = \begin{pmatrix} 1 & 0 \\ 0 & i \end{pmatrix}$	$\|0\rangle \mapsto \|0\rangle$ $\|1\rangle \mapsto i\|1\rangle$
T gate (T)		$T = \begin{pmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{pmatrix}$	$\|0\rangle \mapsto \|0\rangle$ $\|1\rangle \mapsto e^{i\frac{\pi}{4}}\|1\rangle$
Hadamard gate (H)		$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$	$\|0\rangle \mapsto \|+\rangle$ $\|1\rangle \mapsto \|-\rangle$

Table 2.2: Example quantum gates and their different representations.

For example, when a qubit is in a superposition state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ then the operation applies to each of the basis states, e.g.,

$$H|\psi\rangle = \alpha(H|0\rangle) + \beta(H|1\rangle) = \alpha|+\rangle + \beta|-\rangle = \frac{\alpha+\beta}{\sqrt{2}}|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}|1\rangle$$

X gate, Y gate, and Z gate are π (or 180°) rotations about the x-axis, y-axis, and z-axis of the Bloch sphere, respectively. S gate performs a $\frac{\pi}{2}$ rotation about the z-axis, thus we have $S^2 = Z$. T gate performs a $\frac{\pi}{4}$ rotation about the z-axis, thus $T^2 = S$. H (Hadamard) gate is a π rotation about an axis diagonal in the x-z plane.

Example 2.1.7. *It is usually convenient to include generic single-qubit rotation gates (e.g., R_x, R_y, R_z gates) along the Pauli axes in our gate set. We write $R_x(\theta)$ to indicate a rotation of θ angle about the x-axis. Several of the gates we've already discussed are just examples of the $R_z(\theta)$ gates, specifically the Z, S, and T gates which rotate by a π, $\frac{\pi}{2}$, and $\frac{\pi}{4}$ angle, respectively. Formally, the rotation gate can be written in their matrix forms as follows:*

$$R_x(\theta) = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}X = \begin{pmatrix} \cos\frac{\theta}{2} & -i\sin\frac{\theta}{2} \\ -i\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$R_y(\theta) = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Y = \begin{pmatrix} \cos\frac{\theta}{2} & -\sin\frac{\theta}{2} \\ \sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix}$$

$$R_z(\theta) = \cos\frac{\theta}{2}I - i\sin\frac{\theta}{2}Z = \begin{pmatrix} e^{-i\frac{\theta}{2}} & 0 \\ 0 & e^{i\frac{\theta}{2}} \end{pmatrix}$$

Example 2.1.8. Two-qubit gates *take two qubits as inputs. They typically have an "entangling" effect – the operation applied to one qubit is dependent on the state of the other qubit, in other words, they are conditional gates. Among the most common two-qubit operations are the controlled-not gate (or CNOT gate), and the controlled-phase gate (or CZ gate), as shown in Table 2.3 below.*

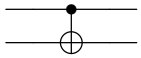
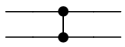
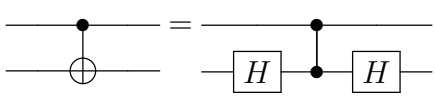
Quantum Gate	*Circuit Form*	*Matrix Form*	*Truth Table*
CNOT gate		$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$	$\|00\rangle \mapsto \|00\rangle$ $\|01\rangle \mapsto \|01\rangle$ $\|10\rangle \mapsto \|11\rangle$ $\|11\rangle \mapsto \|10\rangle$
CZ gate		$CZ = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}$	$\|00\rangle \mapsto \|00\rangle$ $\|01\rangle \mapsto \|01\rangle$ $\|10\rangle \mapsto \|10\rangle$ $\|11\rangle \mapsto -\|11\rangle$

Table 2.3: Example measurement outcomes by $MeasZ$ on initial state $|\psi\rangle$.

In the example, the CNOT gate is a two-input two-output gate which performs a NOT operation on the second (target) qubit only when the first (control) qubit is $|1\rangle$*. Similarly for CZ gate, if the control qubit is* $|1\rangle$*, then we apply a Z gate to the target qubit. But looking at the truth table of the CZ gate, we notice that, in fact, it makes no distinction between the first and the second qubits - a phase is accumulated for the* $|11\rangle$ *basis. Hence the CZ gate has a symmetric circuit symbol. One can in fact implement a CNOT gate with a CZ gate and vice versa. For example, CNOT is equivalent to a CZ gate with two Hadamard gates on both sides, since* $HZH = X$*:*

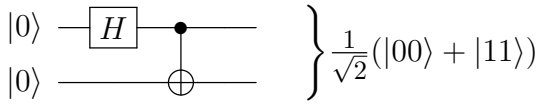


The fact that these gates are conditional gates can also be observed from their matrix representations. In general, we may construct a controlled version of any gate U. Notice

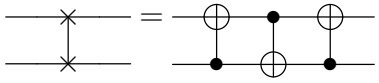
that controlled-U gate can be written as the sum of two terms, namely, when the first qubit is $|0\rangle$, nothing happens to the second qubit, and when the first qubit is $|1\rangle$, then we apply U gate on the second qubit:

$$\text{controlled-}U = \Lambda(U) = |0\rangle\langle 0| \otimes I + |1\rangle\langle 1| \otimes U = \left(\begin{array}{c|c} I & 0 \\ \hline 0 & U \end{array}\right)$$

where the notation $\Lambda(\cdot)$ stands for a controlled version of a gate. One can quickly verify that these controlled gates usually have an *entangling* effect. In particular, they can transform a product state as input into an entangled state as output. For example, the following circuit produces the Bell state:



Another gate, important in architectures which require qubits to be adjacent in order to perform multi-qubit operations, is the SWAP gate, which switches the states of two qubits, which is equivalent to interleaving three CNOT gates:



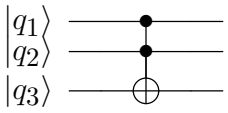
It can be shown that single qubit gates and two qubit gates are *universal* for arbitrary quantum logic. In other words, any unitary gate on multiple qubits can be decomposed into a sequence of one and two-qubit gates. One example of a universal gate set found commonly in the literature is:

$$\mathcal{G} = \{H, T, CNOT\}$$

Physically, realizing a multi-qubit gate is extremely challenging. So finding an efficient decomposition of a unitary gate into a sequence of smaller unitary gates from a chosen gate

set is critical to the success of executing a quantum circuit. This problem is often referred to as *quantum compilation*. We will revisit exactly this problem but in much greater detail in Chapter 4.

Example 2.1.9. Three-qubit gates. *These gates may be controlled on more than one qubit. One of the most famous examples is the Toffoli Gate (CCNOT or Controlled-Controlled-Not). It has the following circuit*



The Toffoli gate can be used to achieve irreversible classical operations like AND and OR in quantum computing in a reversible manner.

2.1.3 Quantum Assembly Programs

At a lower level, a quantum hardware is controlled by instructions signaled by a classical host processor. The quantum assembly language (QASM) is a direct translation from a quantum circuit to a sequential description of quantum instructions for executing a quantum program. Although some existing low-level quantum languages are developed primarily with device-independent and software portability in mind, more and more attention is paid to exposing device specifics, such as the hardware native gates, device connectivity, and noise models, to the language itself and to its software toolchain.

One of earliest low-level quantum languages is called *Quantum Assembly* (QASM) by [Chu]. In the QASM language, a quantum program is described as a linear sequence of gate instructions. For example, the EPR pair creation circuit is written as shown in Figure 2.1.

Due to its root from quantum circuits, sequential QASM language suffers from its limitation on modeling complex classical control, such as "repeat-until-success" procedure and other non-trivial branching. To remedy this, and to improve its expressiveness and coverage,

```
qubit q0
qubit q1
H q0
CNOT q0,q1
Measure q0
Measure q1
```

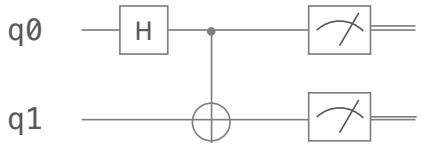


Figure 2.1: The QASM code and circuit diagram for creating an EPR pair with measurements.

a number of extensions to QASM have been developed. In these extensions, basic constructs (commonly used in classical programming) such as loops, subroutine calls, barriers, and classical feedback control are added. For example, the OpenQASM [CBSG17] backend has been developed by IBM Q, and ARTIQ [Bea] by the trapped ion community.

Many have argued for a more expressive language to support full control over the physical properties of the machine, such as pulse features. For example, OpenPulse [MAB$^+$18] developed by IBM is one of the efforts in that direction. OpenPulse is a set of tools for building experiments out of pulses. The performance of the experiments replies heavily on the programmer's understanding of the physical system.

A leading question for computer architects is: what should be included in the instruction set architecture of a quantum processor? This dissertation is a step in this direction – we propose an extended ISA that includes both quantum assembly and qubit tuning.

These low-level languages are naturally more closely tied to hardware specifics. Hence, optimizations for compiling to such languages must be tailored to the specific characteristics of their supported hardware, including device topology and noise rates etc. An efficient low-level software tool can allow quantum algorithms to be successfully executed on resource-constrained machines, such as NISQ computers.

2.2 Quantum Compiling

A quantum compiler aims to efficiently express a high-level quantum program using instructions that a quantum machine recognizes and natively supports, balancing practical *architectural constraints*.

A quantum algorithm is implemented in a *quantum domain-specific language* (QDSL). The quantum compiler translates the high-level program into *quantum assembly code* (QASM) that can be executed on a target hardware. This is accomplished through a series of transformations and optimizations on a *quantum intermediate representation* (QIR) of a program. Finally at the lowest level, machine-level instructions that orchestrate the hardware control pulses are scheduled and optimized.

For a program to be realizable on a given hardware, a number of architectural constraints must be satisfied. This typically means considering the following practical aspects:

- *Instruction set.* There are certain limited number of quantum instructions that are supported in a given architecture. A compiler should aim to translate high-level quantum programs using the supported instruction set. In most cases, this instruction set is "Clifford+T" gates, comprised of the CNOT (controlled-NOT) gate, X (NOT) gate, H (Hadamard) gate, and T ($\pi/8$-phase) gate. This is a common set for most gate-based NISQ machines, as well as large-scale FT machines (e.g., with surface code error correction). Some NISQ compilers choose to target directly the physical analog pulses for improved hardware control.

- *Qubit communication.* A quantum algorithm is hardly interesting if it can be implemented with only single-qubit gates, as two-qubit gates (or multi-qubit gates) provides the entangling power between qubits. Two-qubit gates are implemented by qubit-qubit interaction/communication. Qubit communication has different meanings in the NISQ versus the FT contexts – usually for a NISQ machine, not all qubits can directly inter-

act with each other. two qubits interact by moving closer to one another via a chain of swap gates until they are directly connected hence allowed to interact. The time to complete a swap chain is proportional to the length of the chain. In FT machines, qubit interactions are accomplished through fault-tolerant operations depending on the error correcting codes (such as braiding and lattice surgery in surface-code error-corrected devices [1]). With today's technology, building large-scale quantum machines with all-to-all qubit connectivity is shown to be extremely challenging. Any scalable proposal would involve an architecture of limited qubit connectivity and a model for resolving long-distance interactions, hence inducing communication costs. This constraint is sometimes referred to as "device topology".

- *Hardware noise.* Another important consideration for compiling quantum programs is to minimize errors caused by hardware noise. Errors under consideration typically include *memory errors* (caused by decoherence of qubits) and *gate errors* (caused by imprecise control of gates). In general, the longer the program runs, the higher the chance that the qubits experience decoherence. The more gates are applied, the lower the chance that the program succeeds at the end. In today's technology, a two-qubit gate proves be challenging, hence it is one of the dominant sources of error. A compiler normally aims to express a quantum program in fewer qubits, or fewer number of gates, or shorter circuit depth, etc. Note that these targets are non-exclusive, sometimes conflicting, in which case the compiler would need to balance between the constraints. More advanced noise-aware compilers have also been proposed. For example, in NISQ machines, some qubits are more robust then the others, so picking the longer-lived qubits to perform important computation can improve the overall success rate.

1. Braiding and lattice surgery are techniques that implement gates between logical qubits on the surface code lattice. Details are omitted as they are out of the scope of the thesis; we refer the interested reader to Chapter 5 and other tutorials [Got97, DKLP02, FMMC12] for basics of quantum error correction and topological quantum codes.

- *Available parallel control.* Depending on the technology that implements the qubits, a compiler can be constrained by the available parallelism. The parallelism limitation is usually the consequence of hardware control mechanism, or error mitigation protocols. For instance, the width of the tunable laser beams in a trapped-ion NISQ machine limits the number of independently controlable qubits, and thus the number of parallel single-qubit gates. Some error mitigation protocols dictate that no parallel gates are allowed when they are physically located close to each other, reducing crosstalk errors between them.

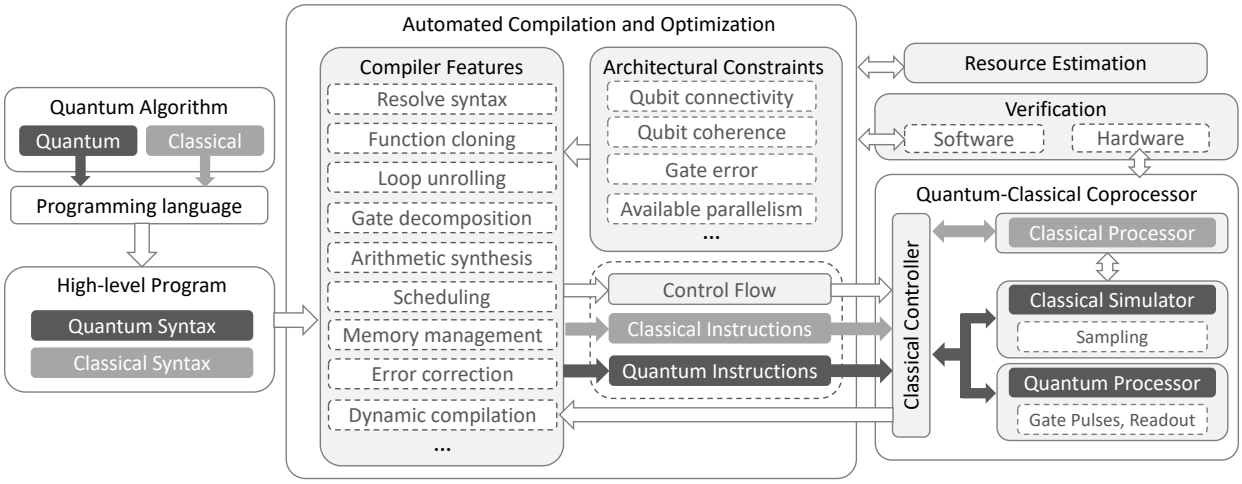


Figure 2.2: A detailed quantum compilation flow outlining the transformations and optimizations involved in a generic compiler.

Figure 2.2 illustrates a typical quantum compilation toolflow. At its core, the quantum compiler passes a high-level quantum program through a series of optimizations, generating the most efficient and robust low-level executable (i.e. sequence of classical and quantum instructions) for the target hardware, balancing different architectural constraints. For historic reasons, more recent work typically targets NISQ-era architectures, but older work targets large FT architectures. Nonetheless, most techniques we introduce here generally apply to the different architectures. As suggested earlier, compilation for quantum machines is very similar to classical circuit synthesis. In the classical setting, we take some high-level

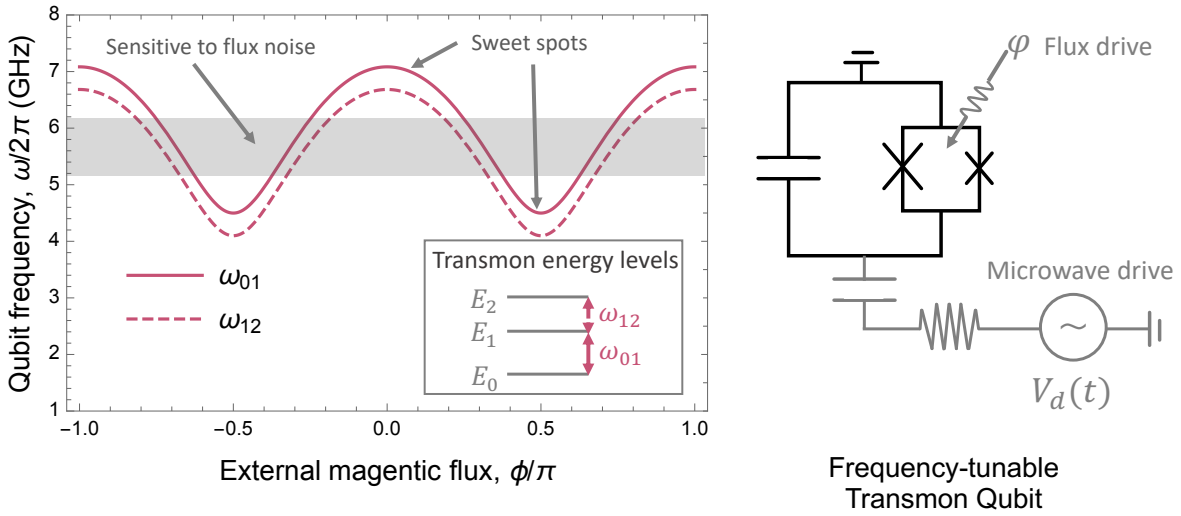


Figure 2.3: *Left:* Qubit frequencies as a function of external magnetic flux. The first three levels of the transmon, ω_{01} and ω_{12}, are plotted. Shaded area is where the qubit is sensitive to flux noise. *Right:* Circuit diagram for a frequency-tunable (asymmetric) transmon qubit (highlighted in black), consisting of a capacitor and two asymmetric Josephson junctions. Highlighted in gray are two control lines: the external magnetic flux control φ and microwave voltage drive line $V_d(t)$ for each transmon qubit.

language (C-like, Verilog, etc) and compile it all the way down to instructions for transistors. This similarity is not pure coincidence; after all, the quantum circuit model of computation is generalized from the Boolean circuit model.

2.3 Device Technology

2.3.1 Basics of Superconducting Qubits

We start with a brief overview of superconducting qubits and how they are manipulated for computation. Transmon-like variety of superconducting qubits [DCG+09, BKM+14b, KBF+15, HHL+17] are among the most widely deployed quantum computer architectures [AAB+19, ROT+18, KKY+19]. The discussions in this work are centered around techniques for frequency-tunable transmons [BKM+14b, KBF+15, BSL+16b, BKM+13b], but some general principles will be applicable to all types of superconducting architectures. We

refer interested readers to an excellent survey of the prospects superconducting qubits by [GPK+21].

A superconducting transmon *quantum bit* (qubit), as shown in Figure 2.3, is by design a multi-level quantum system made out of lithographically printed circuit elements, configured such that they exhibit atom-like energy spectra. The lowest two levels are used as the bit 0 and 1 for computation. The ground energy level represents the state $|0\rangle \equiv [1\ 0]^T$, and the first excited energy level represents the state $|1\rangle \equiv [0\ 1]^T$. Unlike a classical bit, a qubit can be in a linear combination of 0 and 1: $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle = [\alpha\ \beta]^T$, where α, β are complex coefficients satisfying $|\alpha|^2 + |\beta|^2 = 1$.

When a transmon gets accidentally excited to the second (or higher) energy level, e.g. $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle + \gamma\,|2\rangle$, for $\gamma \neq 0$, we call this process "leakage". This can happen due to imprecision in quantum control. The energy gap between the ground state $|0\rangle$ and the first excited state $|1\rangle$ is known as the *qubit frequency*, i.e. $\omega_q \equiv \omega_{01} = E_{01}/h$, where h is the Planck's constant. Hence, we will sometimes use the terms energy and frequency interchangeably. More generally, ω_{01} is referred to as the (first-level) qubit frequency and ω_{02} is the second-level qubit frequency, defined as the gap between the ground state $|0\rangle$ and the second excited state $|2\rangle$. The frequency of a transmon qubit can be changed by applying external magnetic flux through the transmon loop, as shown in Figure 2.3. In this case, there are two frequency sweet spots, i.e. frequency values that are relatively stable against flux noise [KKY+19]. As such, choosing operating frequencies around the sweet spots is desirable for tunable architectures.

2.3.2 Operations and Noises

In QC systems, computation is accomplished by applying a sequence of instructions/operations called *quantum gates*, which take one quantum state to another through unitary transformations, i.e. $|\psi\rangle \to U\,|\psi\rangle$, where U is a unitary matrix. These primitive transforma-

tions are implemented by driving the qubits via *i*) microwave voltage signals, and *ii*) local magnetic flux pulses. The control mechanism for each qubit is illustrated in Figure 2.3.

A *quantum compiler* takes a quantum program written in a high-level programming language, performs a series of transformations and optimizations on the intermediate representations (IR) or quantum circuits, and finally outputs low-level control pulses for driving the qubits. At the end, results of the application are obtained by readout operations (called measurements) on the qubits, which collapse each qubit's quantum state to a classical bit $|0\rangle$ or $|1\rangle$.

Single-qubit Gates and Decoherence Noise

In superconducting transmon systems, single-qubit gates are implemented by driving the target qubit via: *i*) a microwave drive line (feeding time-dependent voltage signals $V_d(t)$) through a capacitor connected to the qubit, and *ii*) a flux drive line (with time-dependent magnetic flux pulses) [KKY+19]. For example, `Rx` and `Ry` rotation gates are implemented by sending microwave voltage signals in-phase (I) and out-of-phase (Q) through the drive line, respectively. Other single-qubit gates, such as Hadamard gate (`H`), can be accomplished by a combination of `Rx` and `Ry` gates.

Qubits naturally decay due to perturbations from the environment. Such decay can happen in two ways: *i*) T_1 *relaxation* (i.e. spontaneous loss of energy causing decay from $|1\rangle$ to $|0\rangle$), and *ii*) T_2 *dephasing* (i.e. loss of relative quantum phase between $|0\rangle$ and $|1\rangle$). We can model both decays in a combined *decoherence error*: $\epsilon_q(t) = (1 - e^{-t/T_1})(1 - e^{-t/T_2})$, where t is time, and T_1, T_2 are constants characterizing the speed of the decays, for some qubit q.

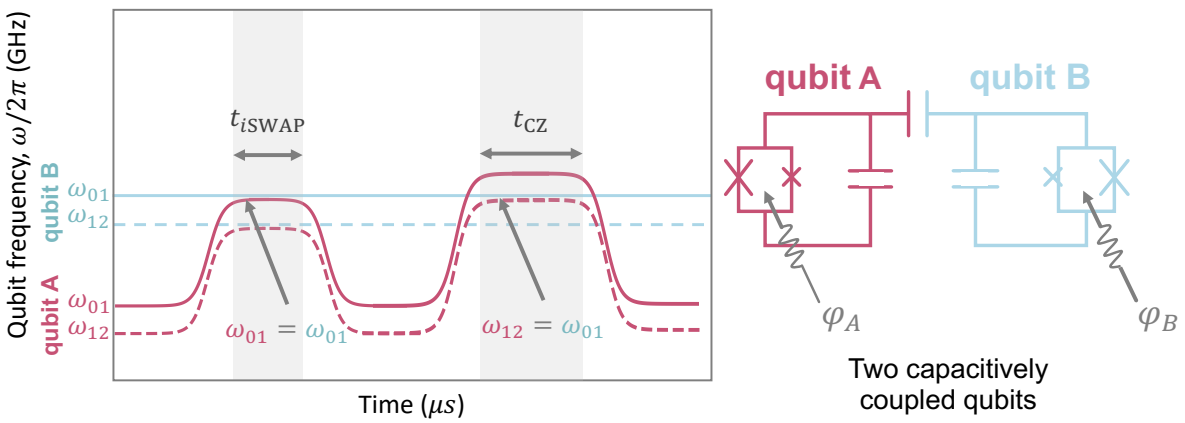


Figure 2.4: Two-qubit interactions for two capacitively coupled transmons. *Left:* Two-qubit gates are implemented with resonance of qubit frequencies. Shown here are how qubit frequencies are tuned for i`SWAP` gate and `CZ` gate. *Right:* Circuit diagram of two capacitively coupled transmon qubits.

Two-qubit Gates and Crosstalk Noise

Two-qubit gates play important roles in quantum computation, as they implement entangling operations, that is, transformations of one qubit conditioned on the state of the other qubit [CDR+18]. Some commonly used two-qubit gates include `CNOT` (controlled-not) gate and `SWAP` gate. Despite their simple forms in the unitary matrix representations, these gates are not typically supported directly in the target architecture. For example, they need to be *decomposed* into primitive gates, such as `iSWAP` gate and `CZ` (controlled-phase) gate, for tunable transmon architectures. The matrix forms for the `iSWAP` gate and the `CZ` gate are:

$$\texttt{iSWAP} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -i & 0 \\ 0 & -i & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \quad \texttt{CZ} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix}.$$

These gates are implemented by tuning the frequencies of the two interacting qubits to

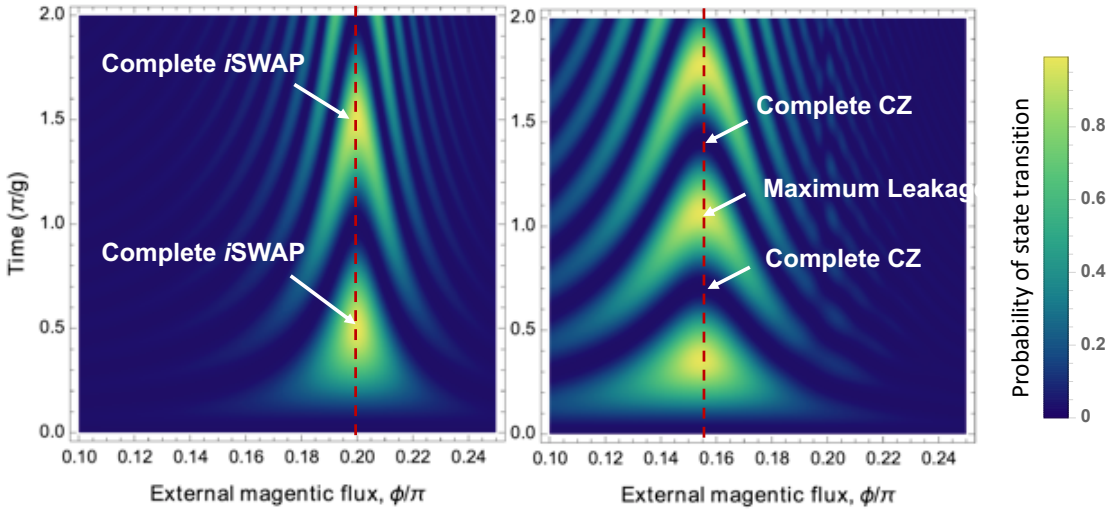


Figure 2.5: *Left:* Probability of state transition between $|01\rangle$ and $|10\rangle$, as a function of external magnetic flux and time. *Right:* Probability of state transition between $|11\rangle$ and $|20\rangle$, as a function of external magnetic flux and time.

some desired operating point, denoted *interaction frequencies*. Then, the qubits are held at that frequency for a duration of time t, depending on the interaction strength g between the two qubits. Figure 2.4 depicts this process.

In the most general sense, *crosstalk* (i.e. unwanted interaction) happens when two qubits are accidentally tuned on (or close to) resonance. Interaction strength varies with closeness of frequencies, $\delta\omega = |\omega_A - \omega_B|$. Gate time t is shorter when g is higher (i.e. when $\delta\omega$ is small). Two-qubit gate error can therefore be modeled as a function of qubit frequencies and time: $\epsilon_g(\omega, t)$, for any gate g (see below for details). For example, crosstalk can occur when a pair of two-qubit gates (on connected qubits simultaneously) happened to use very close interaction frequencies. Chapter 3 illustrates in details how to understand and mitigate these types of crosstalk error.

For frequency-tunable transmon qubits, two-qubit gates are accomplished via resonance. Depending on the energy levels that the resonance occurs, we can implement i`SWAP` and `CZ` gates. In Figure 2.5, we plot the probability of state transitions as we tune the local magnetic flux of one of the qubit (along x-axis) and as the time spent on that operating

point is increased (along y-axis). Let $\delta\omega = |\omega_A - \omega_B|$ be the frequency difference of two adjacent qubits, with residual coupling strength [KKY+19]:

$$g'(\delta\omega) = \frac{g_0^2}{\hbar^2 \delta\omega}, \tag{2.1}$$

where g_0 depends on the effective coupling capacitance C_{qq}. The coupling strength determines how fast and strong the state transitions undergo. When brought on resonance, the two states $|01\rangle$ and $|10\rangle$ will undergo Rabi oscillation, giving rise to a periodic exchange of energy population. The transition probability is $\mathbf{Pr}[t] = \sin(gt)^2$, where g is the coupling strength. Following [BQP+19], the crosstalk error (for idle qubits) is

$$\epsilon_g(\delta\omega, t) = 1 - \sin(g'(\delta\omega)t)^2. \tag{2.2}$$

For iSWAP gate operations, we want a complete exchange of population, predicted at $t = \frac{\pi}{2g}$. We note that for $t = \frac{\pi}{4g}$, it results in another important operation relevant to this work, the $\sqrt{i\text{SWAP}}$ gate. The CZ operation is implemented by resonance of $|11\rangle$ and $|20\rangle$. Due to the higher photon number, the coupling strength is scaled by a constant factor, $\sqrt{2}g$. A complete CZ happens when exchanged from $|11\rangle$ to $|20\rangle$, and back to $|11\rangle$, in other words, CZ gate time is $t = \frac{\pi}{\sqrt{2}g}$.

2.4 Classical-Quantum Co-Processor

Quantum computing hardware is being envisioned as a hardware accelerator for classical computers, as shown in Figure 2.6. To some extent, it is like a processing unit specialized in dealing with quantum information, in a way similar to a GPU (graphics processing unit) that specializes in numerical acceleration of kernels, including creation of images for display. For this reason, the QC hardware is referred to as a QPU (*quantum processing unit*). Unlike a GPU, which can perform arithmetic logic and data fetching at the same time, a QPU does

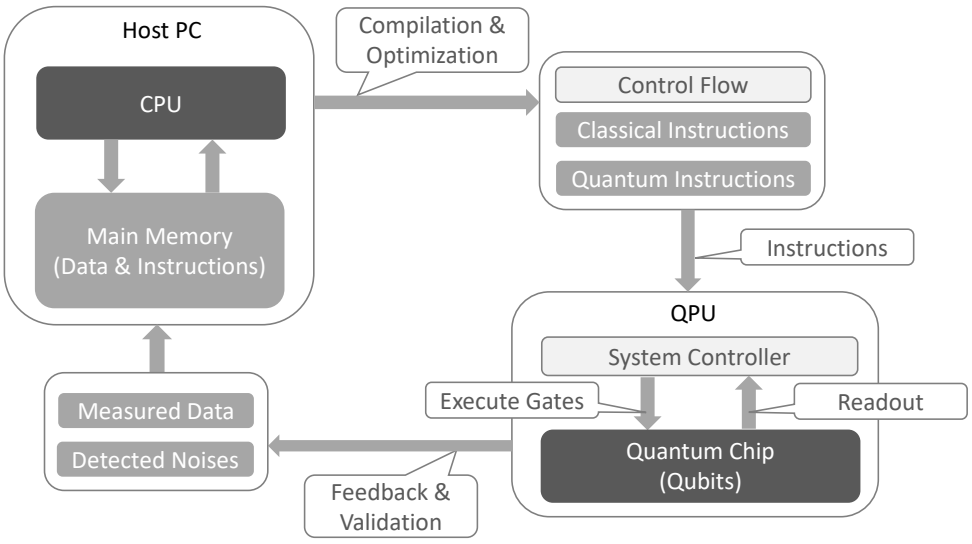


Figure 2.6: A QPU (quantum processor unit) and how it interacts with classical computers.

not fetch data or instructions on its own. A host processor controls every move of the QPU. Let us now dive deeper into the architectural design of a QPU.

It is often times misunderstood that a quantum computer is going to replace all classical digital computers. A quantum computer should *never* be viewed as a competitor with a classical computer. In fact, classical processing and classical control play vital roles in quantum computing. On one hand, a quantum algorithm generally involves classical pre- or post-processing. On the other hand, efficient classical controls are needed for running the algorithm on hardware. As such, a better way of regarding the QC hardware is as a co-processor or an accelerator, that is a QPU (quantum processing unit), as opposed to direct replacements of classical computers.

CHAPTER 3

SYSTEMATIC NOISE MITIGATION

3.1 Quantum Software-Hardware Co-Design

The job of a software systems stack is to bridge the gap between the requirements of the quantum applications and the realities of the quantum hardware. One promising approach is to build a reliable and scalable quantum systems stack via *software-hardware co-design*.

Traditionally, when we design a software tool to map quantum applications to hardware, we consider the application as a black box (like a quantum algorithm is just a sequence of quantum gates), and the hardware as a black box (with a certain number of qubits). However, if the requirement of the quantum algorithm exceeds the resource given by the device, then we will not be able to successfully run this application. Typically resulting in a low success rate and high resource cost.

However, if we open up the boxes, we will see what is inside of an application and how we can change an application to adapt to the hardware. For example, we can synthesize quantum circuits differently or adjust control flow/parallelism differently according to hardware. We can also see what can be changed on the hardware side to serve the target application better, such as qubit selection or measurement reduction based on the application. The software system stack in between is responsible for performing this cross-layer optimization in an automated and systematic fashion. The key here is picking the right amount of information from each side and optimize together.

Quantum computing is at a similar stage of development as classical computing in the 1950's. Today's QC systems consist roughly of three essential components, namely the three layers in quantum computer architecture: application layer, systems software layer, and hardware layer, as shown in Figure 3.1. Today's classical computer systems manage highly complex hardware and software through layering abstractions. Going up through the

systems stack, each layer hides some implementation details and expose a manageable set of controls for the next layer.

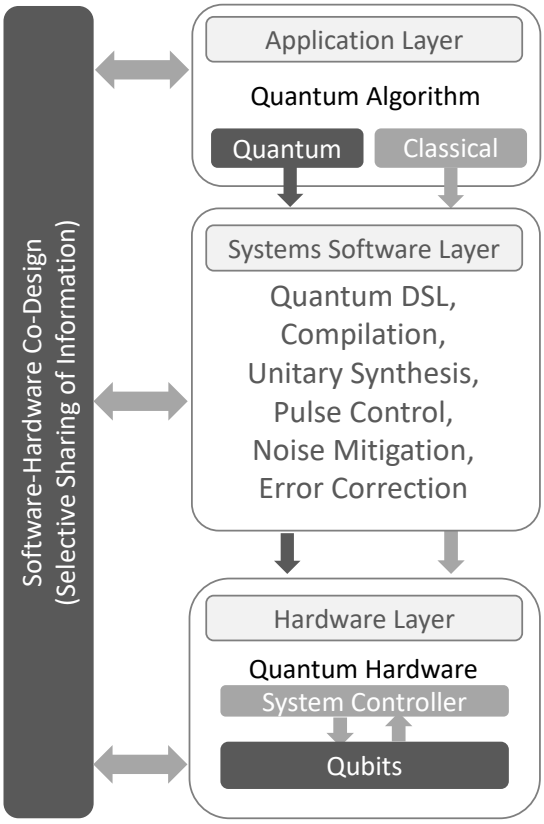


Figure 3.1: Selective sharing of information allows algorithms to use limited resource in NISQ hardware most efficiently.

In contrast, the development of quantum computer systems is still at its nascent stage. This is great for researchers, because there are so many interesting problems to be solved. It also means that resources are very scarce and that we are motivated to *break abstractions* and pay for efficiency with greater software complexity. Even classical computing is backsliding a bit towards less abstraction as the end of Dennard scaling puts pressure on architectures to become more efficient. How much of what we learn in the next 5 years will carry forward to a future of much larger quantum machines? Perhaps more than we might think, as it would be hard to imagine a future in which qubits and quantum operations are not costly. A functional quantum computer requires painstaking attention to the isolation and control

over many qubits. Some physical details may always be exposed. The experience and lessons we learn about how to manipulate qubits in NISQ computers, be it at the algorithmic, systems, or hardware level, will pave the way for larger fault-tolerant quantum devices in the future. Noise resilience is not only for experimentalists who build the hardware to worry about; opportunities are ubiquitous in the entire systems stack. In fact, it is crucial for algorithm designers, systems architects, and software developers to take responsibilities in tackling this challenge together. It is expected that, in the NISQ era, a QC toolchain must break the traditional abstraction layers and use aggressive optimizations throughout the full systems stack. The key to successful execution of quantum algorithms on NISQ devices is to selectively share information across layers of the stack (from device specifics to application characteristics) such that programs can use the limited qubits most efficiently.

To illustrate this co-design methodology, we take a motivating example of mitigating crosstalk noise in a superconducting quantum architecture.

Quantum systems are typically fragile and difficult to stabilize [DC20]. In particular, unwanted crosstalk between neighboring qubits is one of the dominant sources of noise in superconducting quantum computers. To this end, our primary goal is to find computer systems solutions to address such key experimental challenges. In our MICRO paper [1], *we demonstrate a novel approach of frequency-aware compilation and real-time calibration to systematically mitigate crosstalk.*

Crosstalk can be viewed as an unwanted interaction between qubits. Such error is prevalent in today's superconducting transmon architecture due to drifts in the systems and inaccuracies in the controls. As such, crosstalk poses a major obstacle for scaling up quantum processors. *Fortunately, we found that such crosstalk error can be significantly reduced with software.* The key observation is that qubits can be calibrated/tuned dynamically over the

1. "Systematic Crosstalk Mitigation for Superconducting Qubits via Frequency-Aware Compilation." Yongshan Ding, Pranav Gokhale, Sophia Fuhui Lin, Richard Rines, Thomas Propson, and Frederic T. Chong. In Proceedings of the 53rd International Symposium on Microarchitecture (MICRO), Pages 201-214. October 2020.

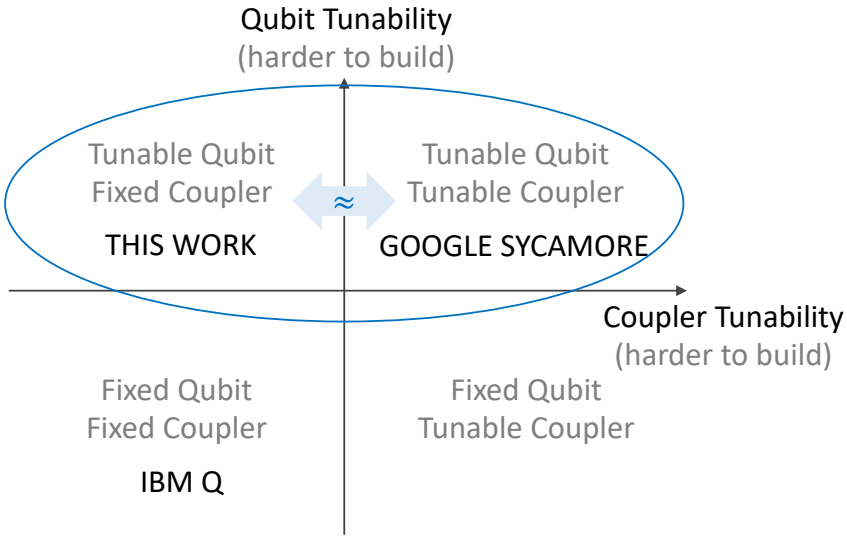


Figure 3.2: Motivating example: key benefit of a software-hardware co-design methodology on crosstalk mitigation. Our software toolflow reduces crosstalk on tunable qubits via frequency-aware compilation and real-time calibration. As a result, this work makes the fixed-coupler (simpler hardware) architecture as crosstalk-resilient as the tunable-coupler architecture.

course of the quantum program execution. By doing so, quantum logic gates (orchestrated by real-time calibration signals) are executed at a significantly higher fidelity. We show that such an efficient software exists, and is applicable to any input quantum programs and to any low- and medium-density device connectivity. As in Figure 3.2, we showed that software techniques can greatly simplify the hardware complexity necessary to reduce crosstalk in quantum machines, altering the design methodology and eventual technological course of commercial quantum machines.

Crosstalk is the one of the leading limitations for fast, high-fidelity quantum gates. *This work identifies crosstalk as an microarchitecture design issue, and calls out for a software-hardware co-design approach to suppress crosstalk.*

At the hardware level, it is sometimes beneficial to implement device with high tunability so that the software has more control over the systems. Higher tunability, however, typically leads to higher manufacturing complexity and higher sensitivity to control noise. At the

software level, the complexity of learning optimal systems parameters grows as systems become larger and more tunable. For a depth-d quantum circuit on n qubits, the overall cost for scheduling such circuit can be estimated at $O(nd)$. Assuming a conventional 2-D grid nearest-neighbor connectivity, the search space for the optimal combination of qubit idle frequency and gate interaction frequency is $O(c^n)$, where $c \sim 100$ is the number of frequency values allowed.

Prior efforts have been mostly focusing on optimizations on device connectivity, program scheduling, fabrication-time frequency assignment. Our work takes a step further – it optimizes compilation, scheduling, and dynamic frequency tuning jointly. Independently with our work, a calibration optimizer were developed, but specialized for single device or application [KKMN20].

3.2 Understanding Hardware

3.2.1 Motivating Example: Frequency Crowding and Crosstalk Noise

Crosstalk mitigation is one of the major challenges in scaling up superconducting quantum architectures. Each qubit has a frequency ω_q^{01}, as well as its associated higher-level excitation frequency ω_q^{12}, which is slightly smaller than ω_q^{01}. For qubit A and qubit B connected by a capacitor:

(i) when qubits are non-interacting (i.e. during `Identity` or single-qubit gates), their *idle frequencies* should have sufficient separation (e.g. $\omega_A^{01} \neq \omega_B^{01}$, $\omega_A^{01} \neq \omega_B^{12}$, and $\omega_A^{12} \neq \omega_B^{01}$);

(ii) when implementing two-qubit gates, they should be placed on resonance at *interaction frequency* (e.g. $\omega_A^{01} = \omega_B^{01}$ for `iSWAP` gate, and $\omega_A^{01} = \omega_B^{12}$ or $\omega_A^{12} = \omega_B^{01}$ for `CZ` gate).

To avoid crosstalk, every pair of connected qubits must be fabricated or tuned to idle frequencies that satisfy the above constraints. However, each qubit can choose from a lim-

ited range [2] of frequency spectrum. Furthermore, every two-qubit gate needs an interaction frequency far enough from those of its neighboring gates. This issue is termed *frequency crowding*, because the frequencies grow increasingly crowded and the above constraints become harder to satisfy, as systems scale up and as programs use more parallelism. It is critical to determine the assignment of frequencies that minimizes unwanted crosstalk.

We can further understand the issue of crosstalk crowding at the system level by focusing on two types of graphs: *i*) the device connectivity graph, and *ii*) the crosstalk graph. For each of these two graphs, we will define formally and illustrate how coloring them can effectively reduce crowding of qubit frequencies.

Idle Frequencies and Connectivity Graph

Qubit connectivity is an important characteristic of a quantum device, as it describes the pairs of qubits between which a two-qubit gate can be directly performed. For completeness, we revisit the definition of a connectivity graph: In a connectivity graph G_c, each vertex is a qubit, every edge is a coupling between the two qubits, e.g. a capacitor in the frequency-tunable transmon architecture.

When the qubits are idle (i.e. not interacting with any other qubits), we want to avoid collision of frequencies for every pair of connected qubits. Therefore, we park the qubits at "idle frequencies". To avoid collisions in idle frequencies, it is equivalent to coloring the connectivity graph where no two end-points of an edge share the same color. If a connectivity graph is colorable by c colors, then we need only c frequency values $\{\omega_0, \omega_1, \ldots, \omega_{c-1}\}$ to keep idle qubits from interacting If the separation between the c frequencies are large enough (i.e. any $|\omega_i - \omega_j|$ sufficiently larger than the anharmonicity), then the higher-energy excitation frequencies are also well separated from the other frequencies, reducing interactions through the leakage channel as well. This strategy works well for simple connectivity graphs like the

2. For example, in a typical frequency-tunable transmon architecture, each qubit can be tuned to frequency around 5 GHz to 7 GHz [AAB+19].

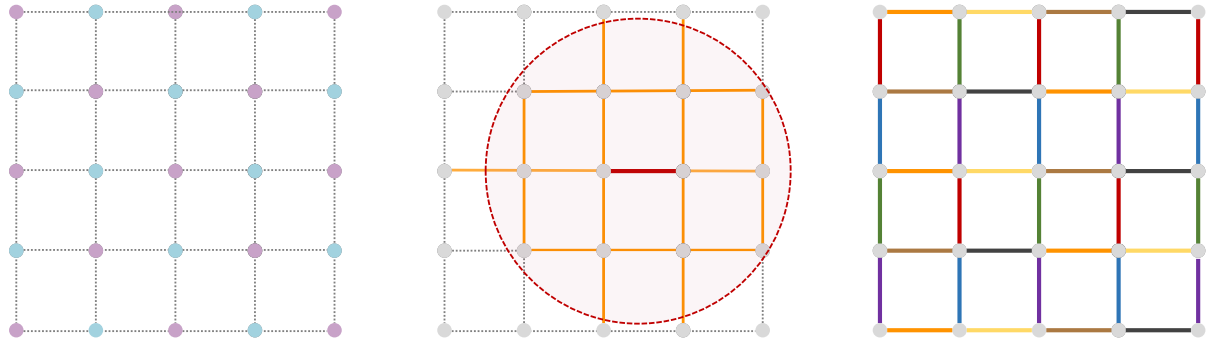


Figure 3.3: *Left:* the connectivity graph for a 5×5 mesh of qubits; 2 colors (highlighted in blue and purple) are needed to color the nodes of the graph. The colors map to idle frequencies of the qubits. *Center:* when the two qubits at the center choose an interaction frequency (highlighted in red) all qubits within the crosstalk range must be tuned off resonance from this interaction frequency. *Right:* A non-crosstalking edge coloring of the 2-D mesh, resulting from coloring the crosstalk graph. 8 colors are required to avoid crosstalk among maximum simultaneous operations. Notably, fewer colors will suffice for program-specific compilation that utilizes circuit slicing and subgraph coloring.

2-D mesh, because the 2-D mesh is bipartite and thus 2-colorable. We also test the general applicability of our algorithm on different choices of device connectivity.

Interaction Frequencies and Crosstalk Graph

Two-qubit gates are implemented by bringing the two qubits on resonance at some "interaction frequency". Any other qubits nearby should be tuned off-resonance from that frequency to avoid unwanted interactions. We define the crosstalk graph to exactly match this constraint. The crosstalk graph G_x of a connectivity graph G_c represent the potential crosstalk that could happen between qubits, which must be addressed by frequency tuning. Here we describe how to construct the crosstalk graph G_x:

(i) Derive the line graph[3] G_L of the connectivity graph G_c.

3. A line graph of a graph G maps each edge in G to a vertex, and two vertices are connected if the two edges in G share a same vertex. [HN60]

(ii) Connect two vertices in G_L if the corresponding two edges in G_c is distance[4] one apart.

To elucidate the structures behind the crosstalk graph, we use a 5×5 quantum chip as an example. Consider the middle edge highlighted in red in the center panel of Figure 3.3. Every orange edge either shares a common vertex with the red edge or is connected to the red edge by a third edge. Thus in the crosstalk graph, the vertex corresponding to the red edge in G_c is connected with the vertices corresponding to all orange edges. If we tune the qubits on the red edge to an interaction frequency ω_{int}, then during the gate time, none of the orange edges should share that frequency.

The crosstalk graph for a 2-D mesh can be colored by 8 colors as shown at the right of Figure 3.3. This coloring is general for any $N \times N$ 2-D mesh, and 8 is the minimum number of colors needed.

We report an important observation here: for a device with 2-D mesh connectivity, crosstalk due to frequency crowding is *mostly localized*. In other words, the frequency space does not become more and more crowded as we increase the size of the mesh. To understand how localized is it, we extend our discussion on nearest-neighbor crosstalk to *next*-neighbor crosstalk.

Generalization to Higher Distance

So far, we have been discussing crosstalk between directly coupled qubits (i.e. nearest-neighbor crosstalk). One could imagine the residual coupling between a qubit and its next-neighbor could result in crosstalk as well. We introduce a generalization to the crosstalk graph to higher distance d, denoted as $G_x^{(d)}$: The distance-d crosstalk graph $G_x^{(d)}$ of a connectivity graph G_c has a vertex for each edge in G_c, and two vertices are connected if the two edges in G_c share a common vertex or are connected by a path of length d.

4. Distance between two edges equals the length of the shortest path that connects the two edges.

3.3 Widening the Architecturally Visible State

In a traditional instruction set architecture (ISA) for a quantum processor, a program can manipulate the logic state of the qubits of the system by performing quantum logic gates. In other words, the *architecturally visible state* is the quantum state of the processor and a quantum assembly code can modify its quantum state with quantum gates. In a more realistic setting, however, various properties of a processor beyond its qubit state can influence the evolution of the state of the processor as well. One notable example in a superconducting architecture is the qubit frequencies as shown above. As such, we extend the architecturally-visible state to include qubit frequencies. As a result, the extended description of the instructions for the ISA includes not only the quantum gates but also frequency tuning operations.

We now demonstrate that the extended ISA allows *systematic software optimizations* to mitigate crosstalk that would not have been possible given a traditional ISA. Such optimizations utilize a variety of microarchitecture tunability features. These features (such as different degree of tunability in qubits themselves and their couplers) allow the hardware to be *dynamically configured* to avoid crosstalk as program executes. We propose frequency-aware systems software that reduces the chances of both decoherence and crosstalk, via strategic frequency tuning and instruction scheduling.

3.3.1 Frequency Tuning and Instruction Scheduling

To remedy this frequency crowding issue, we present a systematic scheme that dynamically tunes the device and schedules instructions according to input programs. Consider the toy program in Figure 3.5 as an example – we found that *a general recipe for avoiding crosstalk between two parallel gates is to create sufficient separation: i) either in frequency, ii) or in time.*

In order to understand and mitigate the impact of crosstalk, we begin with two simple

observations: *i*) Every qubit (when not interacting with others) needs to pick a 0-1 excitation frequency sufficiently far apart from the 0-1 or 1-2 excitation frequencies of its neighbors. *ii*) The extend of tunability is limited and there are few preferred operational frequencies for each qubit. These two constraints are naturally in tension with each other. The key is to balance the two.

To the best of our knowledge, this work is the first to study strategies for systematically tuning qubit frequencies in a program-aware fashion.

Throughout, we explore crosstalk on a flux-tunable transmon architecture with 2-D mesh-like connectivity. Nonetheless, the input to our algorithm can be any arbitrary device topology; hence the crosstalk mitigation techniques we introduce here are applicable to all types of device connectivity, as showed quantitatively in Section 3.7.3.

3.4 Frequency-Aware Compilation

3.4.1 Resolving Frequency Crowding via Graph Coloring

Figure 3.4 is an overview of our approach. This work aims to provide means for understanding and mitigating the impact of crosstalk, from a software optimization perspective. Recent work by architects have demonstrated that software optimizations can lead to efficient noise mitigation, effectively providing the equivalent of months of hardware progress. For example, [TQ18, LDX19, MBJA$^+$19a] show how to improve qubit utilization and [SLG$^+$19] shows how to optimize pulses to speedup gates. We demonstrate that quantum programs can be optimized to reduce the chance of crosstalk and decoherence by scheduling instructions at the right operational frequency and time step, preventing *spectral* and *temporal* collisions, respectively. To do so, we define a type of graph called the *crosstalk graph*; our mitigation technique maps the frequency-aware compilation problem to the coloring of crosstalk graph. Furthermore, the diversity of gate decomposition gives us an extra degree of freedom in

scheduling. In sum, our main contributions include:

- An efficient compilation algorithm that mitigates the impact of crosstalk and decoherence via program-specific frequency tuning and instruction scheduling, making tunable-qubit, fixed-coupler systems a competitive, scalable design.
- A systematic analysis of device tunability and sensitivity to provide insights on the advantages and disadvantages of different architectural designs, such as IBM's fixed-frequency systems and Google's tunable-coupler systems.
- Evaluations of our crosstalk mitigation algorithm on a variety of NISQ benchmarks including `BV` [BV97], `QAOA` [FGG14], `QGAN` [LW18], `ISING` [BSL$^+$16b], and `XEB` [AAB$^+$19] circuits.

3.4.2 Using A Compiler to Mitigate Hardware Noise

Now we illustrate the key steps in our crosstalk mitigation algorithm – the inputs to the algorithm include device characteristics (e.g. qubit number, connectivity, transmon tunability), program characteristics (e.g. a scheduled quantum circuit), and optimization level (e.g. crosstalk distance). An illustrative example is shown in Figure 3.5.

Finding optimal (idle and interaction) frequency configurations based on device and program characteristics is a high-dimensional optimization problem; *we break the problem into multiple scalable sub-problems*. As shown in Figure 3.4, we begin by constructing a crosstalk graph for the input device. Next, the input program is decomposed into primitive gates and sliced into layers (time steps). Then, we produce *a feasible coloring of an active subgraph* of the crosstalk graph for each layer of the circuit. From the colors, we thereafter map to the idle and interaction frequencies via a Satisfiability Modulo Theory (SMT) solver [BPF15, DMB08]. Lastly, we produce a feasible schedule of the program (i.e. gate instructions and qubit frequencies for each time step), throttling parallelism if necessary. Algorithm

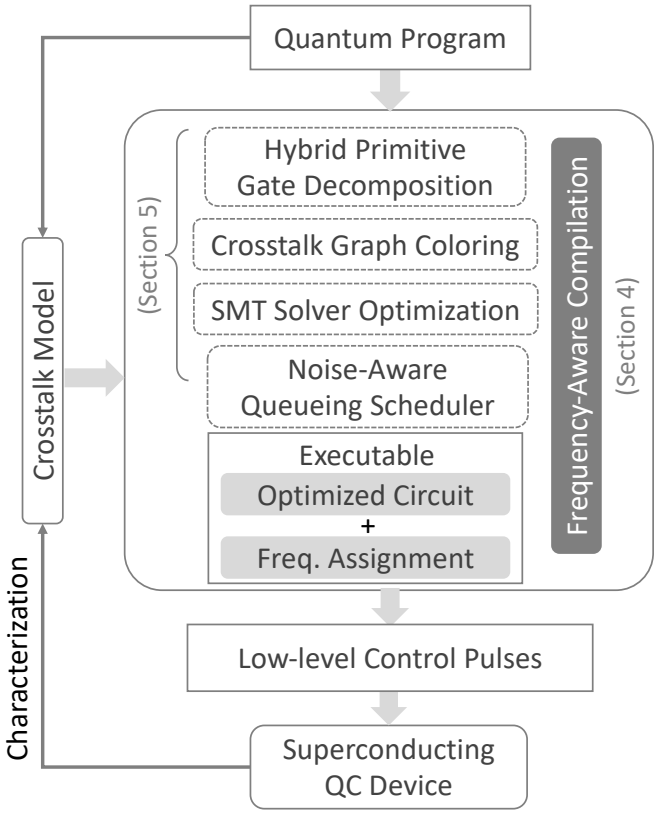


Figure 3.4: Flow of our crosstalk mitigation software for tunable superconducting QC systems. We develop a frequency-aware compilation algorithm that systematically reduces crosstalk and decoherence.

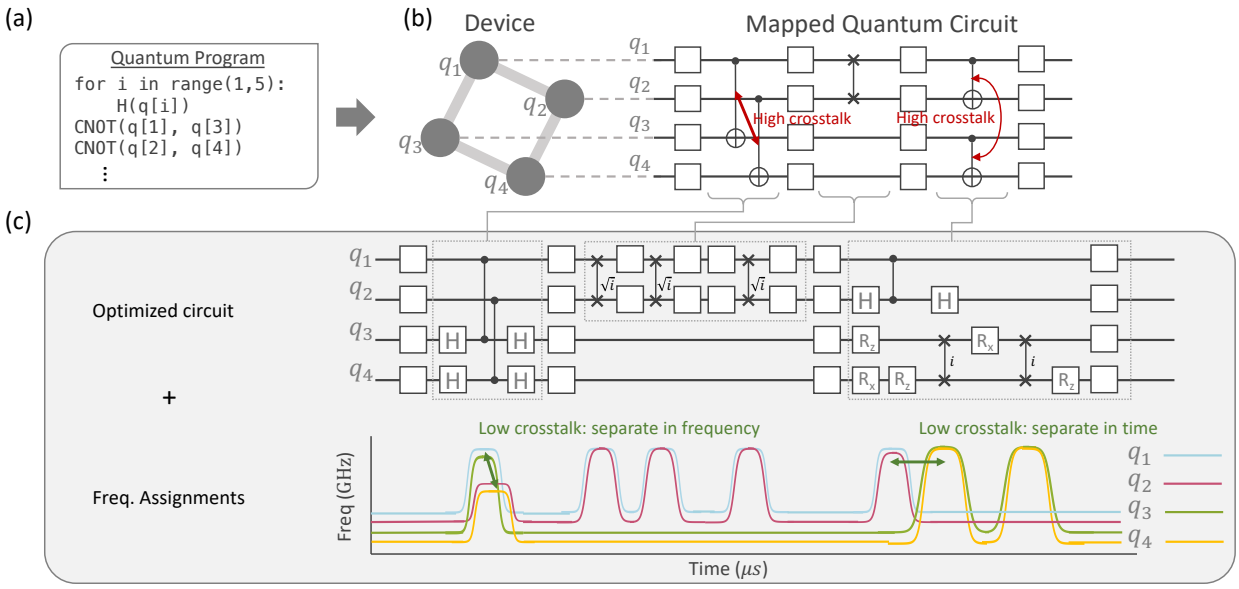


Figure 3.5: (a) An example quantum program on four qubits. (b) The quantum program is mapped to a QC system of 2×2 qubits with nearest-neighbor connectivity. In a quantum circuit, qubits are lines; gates are applied to the qubits from left to right. Highlighted in red are the parallel quantum gates with high likelihood of crosstalk. (c) The optimized circuit and frequency assignment resulting from our compilation algorithm. Crosstalk is mitigated by avoiding spectral and temporal collisions in the those gates.

1 is the main algorithm outlining this process. Specifically, line 10-16 is the queueing schedule in Section 3.4.3; line 17-19 is the coloring step in Section 3.4.3; line 20-22 corresponds to the SMT solver optimization in 3.4.3.

3.4.3 Optimization Details

This section is dedicated to explaining the key ingredients of the algorithm in greater detail. Through a series of optimizations, our frequency-aware compilation algorithm drastically reduces the chance of crosstalk and scales favorably with systems sizes, making it a viable long-term solution to frequency tuning for superconducting qubits.

Algorithm 1 Frequency-Aware Compilation

$d \leftarrow$ crosstalk distance parameter
$G_c \leftarrow$ connectivity graph of the device D
$G \leftarrow$ gen_crosstalk_graph(D, d)
$C_c \leftarrow$ coloring(G_c)
$\Omega_c \leftarrow$ colors in C_c are mapped to parking frequencies
$P \leftarrow$ decompose input program P into primitive gates
$S \leftarrow$ first layer (time step) of program P
$Q \leftarrow \varnothing$
while S non empty **do**
    $I \leftarrow \varnothing$
    $S \leftarrow$ sort S by criticality
    for gate in S **do**
        if not noise_conflict(gate, I) **then**
            $I \leftarrow I \cup \{gate\}$
        end if
    end for
    $E \leftarrow$ collect relevant two-qubit gates in I
    $H \leftarrow$ subgraph(G, E)
    $C \leftarrow$ coloring(H)
    $\Omega \leftarrow$ smt_find(C)
    $S \leftarrow (S \setminus I) \cup \{$next layer of $P\}$
    $F \leftarrow$ qubit frequencies for this cycle based on Ω_c and Ω
    $Q \leftarrow Q \cup \{(I, F)\}$
end while
return Q

Crosstalk Graph Construction

In Section 3.4.1, we outlined how the crosstalk graph is constructed; the steps are made rigorous in the following Algorithm 2. By abstracting all possible crosstalk channels between pairs of qubits as graph theoretical objects, we are now equipped to quantitatively analyze and systematically mitigate crosstalk errors due to frequency crowding.

Circuit Slicing and Subgraph Coloring

One of the major advantages of our approach is in producing a *dynamic* frequency assignment tailored for each input program. This wins over a static (program independent) frequency assignment because frequencies are substantially less crowded when only considering a subset

Algorithm 2 gen_crosstalk_graph

$G_c \leftarrow$ connectivity graph of the device D
$G \leftarrow$ networkx.line_graph(G_c)
$S \leftarrow \varnothing$
for pair of nodes (e_1, e_2) in G_ℓ **do**
    $(u_1, v_1) \leftarrow$ pair of qubits for e_1
    $(u_2, v_2) \leftarrow$ pair of qubits for e_2
    cond $\leftarrow dist(u_1, u_2) \leq d$ or $dist(u_1, v_2) \leq d$
    cond $\leftarrow$ cond or $dist(v_1, u_2) \leq d$ or $dist(v_1, v_2) \leq d$
    if cond **then**
        $S \leftarrow S \cup \{(e_1, e_2)\}$
    end if
end for
G.add_edges_from(S)
return G

of couplings between qubits that are "active" for a given time step. Here active couplings refers to only those pairs of qubits currently involved in two-qubit gates.

We identify the active subgraph H of the crosstalk graph G, by profiling the two-qubit gates in one time step. The (vertex) coloring of H, denoted as C, is an assignment of labels (called colors) for the vertices of H such that no two adjacent vertices share the same color, while minimizing the number of colors in total. Graph coloring is known to be an NP-complete problem; section 3.7.1 shows how we maintained efficiency. In our optimization, we apply a polynomial-time greedy approximation, the Welsh-Powell algorithm [WP67], to color the active subgraph.

As a result, a feasible coloring of H yields a set of non-colliding interaction frequencies for the two-qubit gates. Qubits that undergo `Identity` or single-qubit gates are parked at idle frequencies, determined by coloring the device connectivity graph. In the next section, we describe how to map from a coloring to a frequency assignment via a SMT solver.

SMT Solver Optimization

The mapping from colors C to frequencies Ω is reduced to a constrained optimization problem. The objective is to assign $|C|$ frequencies within some range $[\omega_{lo}, \omega_{hi}]$, satisfying the

crosstalk constraints in Section 3.2.1. We use a SMT solver to find a feasible solution with the following constraints.

$$\forall c \in C, \omega_{lo} \leq x_c \leq \omega_{hi}, \tag{3.1}$$

$$\forall x_{c_i}, x_{c_j}, |x_{c_i} - x_{c_j}| \geq \delta, \tag{3.2}$$

$$|x_{c_i} + \alpha - x_{c_j}| \geq \delta, \tag{3.3}$$

where α is the anharmonicity, and δ is a threshold. Then, `smt_find` uses a simple binary search to find the maximum threshold δ, for which a feasible solution exists. We ensure the efficiency of the procedure by keeping $|C|$ small.

Once the optimal solution is found, a one-to-one mapping from C to Ω is enforced by a total ordering, motivated by the fact that higher interaction frequency value would yield faster gate time, i.e., $t_{gate} \sim 1/\omega$ [KKY+19]. In particular, let us denote $n(c)$ as the number of times c appear in C and $\omega(c)$ as the frequency value to which c maps. We dictate that, for any $c_i, c_j \in C$, if $n(c_i) \geq n(c_j)$ then $\omega(c_i) \geq \omega(c_j)$. The following section details how the frequency ranges are determined.

Frequency Partitioning

We partition the range of tunable frequency spectrum into three regions: interaction region, exclusion region, and parking region. Similar partitioning strategies has been studied for surface code error correction circuits [VPK+17]. This allows us to decouple the idle frequency assignment from that of the interaction frequency. For a realistic frequency-tunable transmon, the tunable range is typically just a few GHz. So a reasonable design would use a partition with 1 GHz interaction region, 0.5 GHz exclusion region, and 1GHz parking region. By this design, no frequency is assigned in the exclusion region (which are most sensitive to flux noise), preventing idle qubits from interacting with iswap/cphase qubits.

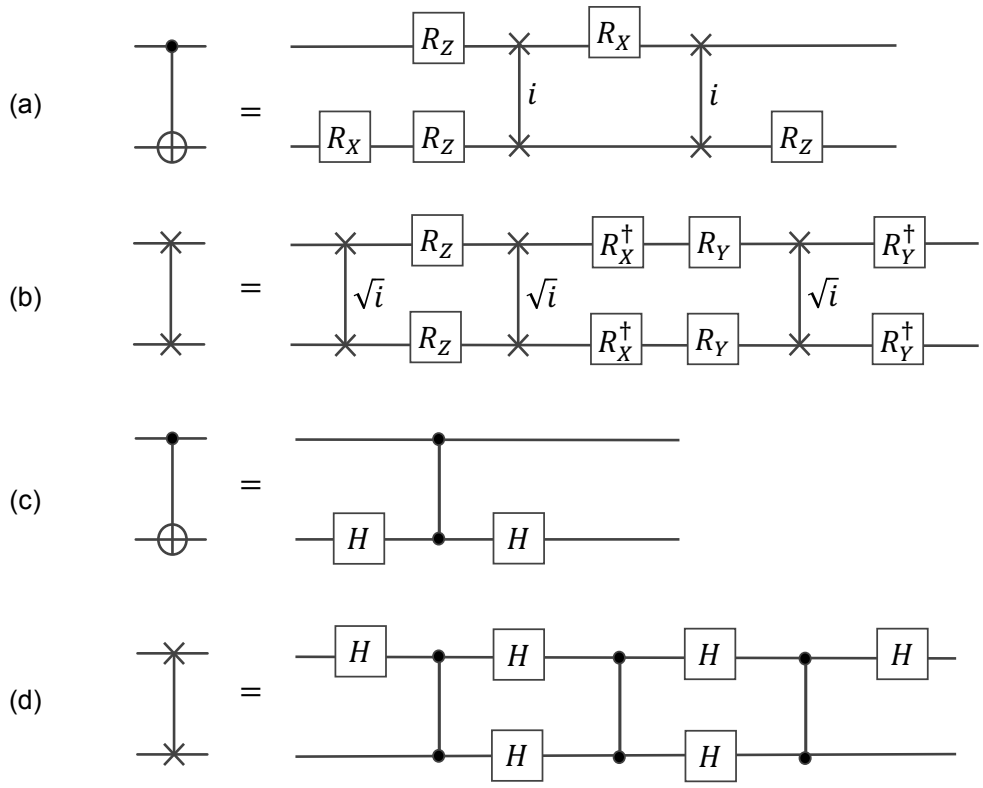


Figure 3.6: *(a)*: The `CNOT` gate, decomposed with i`SWAP`. *(b)*: The `SWAP` gate, decomposed with $\sqrt{i\texttt{SWAP}}$. *(c)*: The `CNOT` gate, decomposed with `CZ`. *(d)*: The `SWAP` gate, decomposed with `CZ`.

The interaction frequencies are determined using the coloring C for H. This is a two-step process. First, each coupling in H (that is a pair of qubits performing a two-qubit gate) gets assigned a color $c \in C$ corresponds to an interaction frequency. Second, qubits that appear in its complement $G \setminus H$ remain in their parking frequencies.

Hybrid Circuit Decomposition

To implement a two-qubit gate that is not directly supported by the frequency-tunable transmon architecture, we need to decompose it into a series of native gates. Two commonly used two-qubit gates in quantum programs are the `CNOT` gate and the `SWAP` gate, because they implement relatively simple Boolean logic. Fig. 3.6 shows that they can be decomposed into `iSWAP` (or $\sqrt{i\texttt{SWAP}}$) and `CZ` gates.

The strategy for circuit decomposition can affect performance. Compared to decomposing all the two-qubit gates in a circuit with one type of native gates, hybrid strategies can help achieve better fidelity. A simple hybrid strategy is to decompose `CNOT` gates with `CZ`, and `SWAP` gates with $\sqrt{i\texttt{SWAP}}$. As depicted in Figure 3.6, this strategy is advantageous because `CNOT` (`SWAP`) is cheaper to implement with `CZ` ($\sqrt{i\texttt{SWAP}}$) gates than with $\sqrt{i\texttt{SWAP}}$ (`CZ`) gates.

Noise-Aware Queueing Scheduler

Finally, parallelism is another crucial concern in our algorithm – on one hand, parallelism helps shorten the circuit execution time, reducing chances of decoherence; on the other hand, it crowds the interaction frequency range, increasing chances of crosstalk. Our noise-aware queueing scheduler finds a sweet spot by strategically serializing gates that are likely to cause crosstalk. In algorithm 1 (line 9-16), gates are delayed based on their criticality and potential noise conflicts. Criticality of a gate is its position along the program critical path, calculated by profiling the input program during circuit slicing on line 7. Function `noise_conflict` predicts potential crosstalk: when scheduling g (e.g. `CNOT(q1,q2)`), if too many of its neighbors in the crosstalk graph are already in I, then their interaction frequencies are likely very close, so we postpone g for the next time step. Serialization is done conservatively while maintaining minimal impact on the critical path length of the program (that is the circuit depth). This *greedy* scheduling approach is shown to be effective in balancing crosstalk and decoherence.

3.5 Case Study: Balancing Software Control and Hardware Complexity

High-performance quantum systems require extremely precise control to execute quantum logic gate and frequent calibration to keep qubits stable. This work breaks conventional

wisdom and includes both quantum logic gates and qubit frequency tuning as microarchitectural instructions. Quantum logic gates, orchestrated by calibration signals, are executed with significantly higher fidelity. As a result, *it gives a general architectural solution for considering compilation and calibration as a joint optimization process.*

Can an architectural solution change the technological course of quantum machines? This work gives an encouraging affirmative answer. Having an optimized software indeed changes perspective on the hardware design space. As shown in Figure 3.2, different choices of tunability offer different levels of software control and hardware manufacturability. Looking at the state-of-the-art superconducting quantum computing systems design spaces, we can find roughly three different approaches among the leading industry vendors:

1. IBM is most conservative; they prioritize manufacturability, and have many machines deployed 24/7.

2. Rigetti is more aggressive, in that they make machines with tunable qubits, which offer more degrees of freedom in software control. Fewer machines have been deployed.

3. Google is most aggressive, in that an exemplar machine (with tunable qubits and couplers) are built with extreme care in controls. Since the machine requires fine tuning, it is not available for public use. Additionally, more complex fabrication has led to fewer machines manufactured.

Our results suggest that the middle road here, with appropriate software support, may be the best approach, balancing flexibility with complexity. It is tempting to invent growingly sophisticated hardware components to tackle each hardware issue; however, that is not necessarily the best approach when it comes to manufacturability and scalability.

Crosstalk is one of the biggest engineering hurdles in scaling up quantum processors; here is why we believe our software methodology is **a viable long-term solution**:

- *Complexity.* With support of software, the hardware complexity necessary for crosstalk

Algorithms	Microarch. Features
Baseline N	Tunable transmon, fixed coupler, Qiskit [AAA+19] scheduler
Baseline G	Tunable transmon, tunable coupler, tiling scheduler
Baseline U	Tunable transmon (with single interaction frequency), fixed coupler, serial scheduler
Baseline S	Tunable transmon, fixed coupler, crosstalk-aware scheduler
ColorDynamic	Tunable transmon, fixed coupler, crosstalk-aware scheduler

Table 3.1: List of algorithms used in our evaluation.

mitigation is significantly reduced. This is an important advantage in the long run, because as systems scale up, manufacturing cost and control complexity will be much more demanding.

- *Generality.* To the best of our knowledge, our work is the first proposed software to evaluate and mitigate crosstalk across a large array of applications and over a broad spectrum of device connectivity, achieving a record level of protection against crosstalk noise. For future large-scale devices, this is an essential feature.

- *Scalability.* Our software also has a favorable scaling. It gives a competitive edge on performance (time-to-solution) with state-of-the-art optimizers for frequency calibration and scheduling. A typical near-term program instance can be optimized within 5 seconds, and future large-scale instance within 30 seconds, far faster than systems drift.

3.5.1 Tuning and Scheduling Baselines

We test the performance of our frequency-aware compilation algorithm (i.e. *ColorDynamic*) in comparison to four baselines, *Baseline N* (naive), *Baseline G* (gmon), *Baseline U* (uniform), and *Baseline S* (static), shown in Table 3.1; they represent strategies of frequency

tuning and instruction scheduling from leading industry architectures.

Baseline N: Naive Compilation. A conventional crosstalk-unaware compilation algorithm. Qubits are assigned with separated idle and interaction frequencies.

Baseline G: Gmon with Tunable Coupler. This baseline has advanced hardware requirements to activate couplers – the "gmon" architecture, implemented in Google's recent Sycamore quantum architectures [AAB+19], takes advantage of both tunable qubit and tunable coupling features to mitigate crosstalk. On the flip side, the flux-tunable coupler would incur fabrication overheads, and introduce extra sensitivity to flux noise. We reconstruct and evaluate a gmon-like architecture where the couplers are activated following the same pattern used for Sycamore, and idle and interaction frequencies match exactly the reported values in [AAB+19].

Baseline U: Uniform Frequency with Serialization. This baseline relies on serialization to avoid crosstalk, similar to [IBM, MMMJA20]. All two-qubit gates share one common interaction frequency ω_{int}, demonstrating the impact of serialization.

Baseline S: Static Frequency-Aware Compilation. Baseline S optimizes the idle and interaction frequencies independent of input programs, producing a static set of optimized values. Most crosstalk-aware optimizers perform this type of static optimization [VPK+17, AAB+19].

ColorDynamic: Program-specific Frequency-Aware Compilation. This is the pinnacle of our work. Instead of finding a static interaction frequency solution for all programs, ColorDynamic returns optimized frequencies for each time step of a program. It combines all optimizations in Algorithm 1, including circuit slicing, strategical decomposition and serialization, graph coloring, and SMT solvers.

3.5.2 Experimental Setup

Benchmarks: We study the performance of our algorithm through a variety of NISQ benchmarks, shown in Table 3.2. These benchmarks are among the best known applications for near-term quantum machines. We also include circuits for benchmarking simultaneous quantum gates to demonstrate the impact of crosstalk on the fidelity of those gates [AAB+19].

In our evaluation, we vary number of qubits $n = 4, 9, 16, 25$. These circuits are of most interest, because the range of crosstalk is typically localized, as shown in Figure 3.3.

Benchmarks	Descriptions
`BV(n)`	Bernstein-Vazirani (BV) algorithm on n qubits [BV97]
`QAOA(n)`	Quantum Approximate Optimization Algorithm (QAOA) [FGG14] for MAX-CUT on an Erdos-Renyi random graph with n vertices
`ISING(n)`	Linear Ising model simulation of spin chain of length n [BSL+16b]
`QGAN(n)`	Quantum Generative Adversarial Network (QGAN) with training data of dimentsion 2^n [LW18]
`XEB(n, p)`	Cross entropy benchmarking circuit for calibrating two-qubit gates on n qubits with p cycles [AAB+19]

Table 3.2: List of benchmarks used in our evaluation.

Software implementation: Our compilation algorithms are implemented in `Python` 3.7, interfacing the IBM `Qiskit` software library [AAA+19]. The graph coloring optimization uses `greedy_coloring` in `NetworkX` library [HSSC08], and the SMT optimization uses `Z3 solver` [DMB08] through the `Z3py` APIs. All compilation experiments use Intel E5-2680v4 (2.4GHz, 64GB RAM).

Architectural features: We consider a 2D grid of $N \times N$ asymmetric frequency-tunable transmons, each having maximum frequencies ω_q (in GHz) sampled from Gaussian distribu-

tion: $\Omega \sim \mathcal{N}(\omega, 0.1)$, with nearly constant aharmonicity $\alpha/2\pi = (\omega_{12} - \omega_{01})/2\pi \approx 200$ MHz, to account for realistic variation in fabrication and initial detuning. Any pair of nearest-neighbor qubits are directly connected with a capacitor; the coupling strength g depends on the frequencies of the qubits, which is typically around $g/2\pi \approx 30$ MHz. For gmon-like experiments, qubits are connected by flux-tunable couplers, each with its own independent external magnetic flux control. These parameters are set to realistic values in line with experimental data from the literature [KSG$^+$20].

Metrics: For our compilation experiments, we need to efficiently compute the program success rate – we define a heuristic for efficiently estimating the *worst case* success rate of a program under crosstalk and decoherence noises.

$$P_{success} = \Pi_{g \in G}(1 - \epsilon_g) \cdot \Pi_{q \in Q}(1 - \epsilon_q) \tag{3.4}$$

where ϵ_g is the crosstalk gate error, and ϵ_q is qubit decoherence error. Details on ϵ_g can be found in Chapter 2, Equation 2.2; ϵ_q is captured by modeling T_1 and T_2 during idle or gate time, as studied in [KSG$^+$20]. A similar metric to $P_{success}$ is used in [ZBL20, AAB$^+$19].

Besides being efficiently computable, this heuristic has useful operational significance – we can understand and mitigate the worst-case impact of crosstalk and decoherence on the systems during compile-time or run-time of quantum programs. Of course, to gain full knowledge of the crosstalk and decoherence errors, we need full noisy circuit simulation, which quickly becomes intractable as circuit size grows beyond tens of qubits. Hence, we validate the heuristic estimator on small-scale circuits, for which noisy circuit simulation is possible.

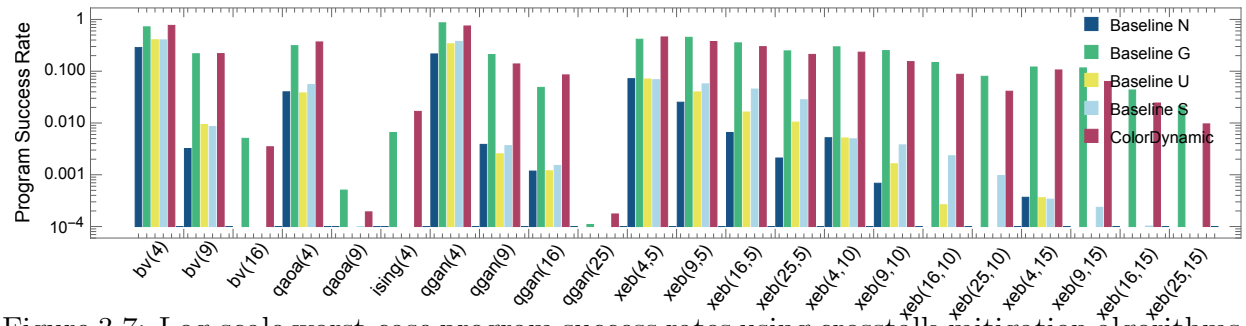


Figure 3.7: Log-scale worst-case program success rates using crosstalk-mitigation algorithms, estimated by heuristics. Higher success rate is better. Across the benchmarks, ColorDynamic performs consistently well compared to other algorithms. In particular, it matches the crosstalk resilience of baseline G (with tunable-qubit, tunable coupler), but on fixed-coupler hardware which is more robust to external noise. Results for qaoa(16) and ising(16) are omitted due to high circuit depth and qubit decoherence.

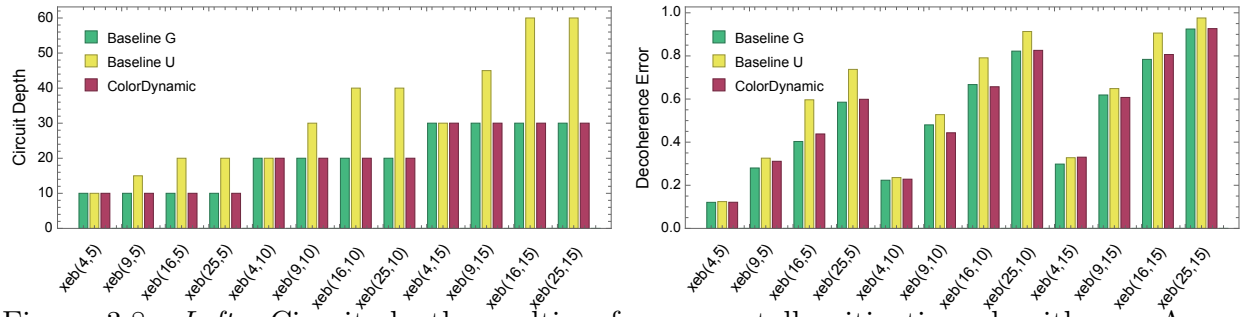


Figure 3.8: *Left:* Circuit depth resulting from crosstalk-mitigation algorithms. Across the benchmarks, ColorDynamic avoids crosstalk without incurring significant serialization. *Right:* Decoherence errors resulting from crosstalk-mitigation algorithms. Lower is better.

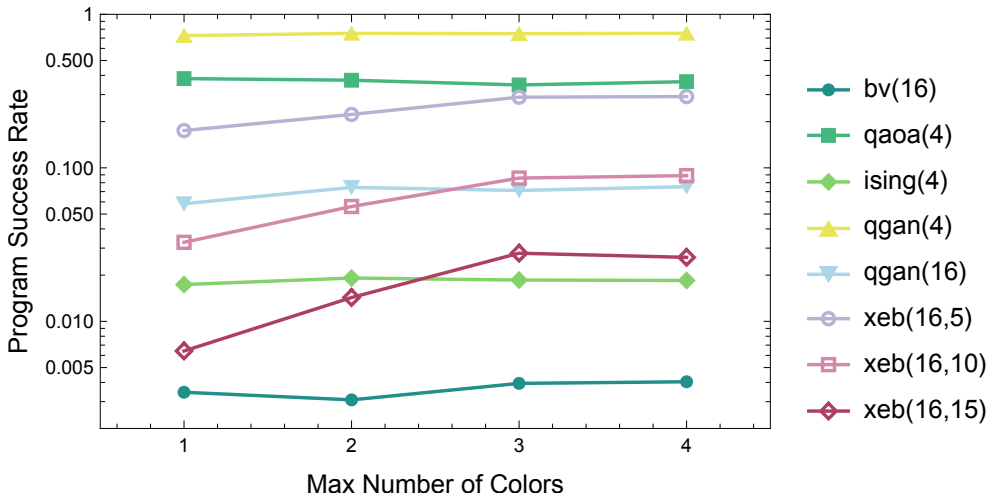


Figure 3.9: Finding sweet spot of tunability. More than three colors (i.e. frequencies) are typically *unnecessary* for NISQ benchmarks.

3.6 Performance Results: Evaluating on Noisy Machines

3.6.1 Program Success Rate

Figure 3.7 shows worst-case overall success rate, estimated using our heuristic Equation 3.4. Note that statistics, such as those from `qaoa(16)` and `ising(16)` circuits, are excluded from the analysis due to their estimated success rates being lower than 10^{-4}. Baseline N is crosstalk-unaware; as a result, crosstalk has detrimental impact on program success rates for any circuit with parallel two-qubit gates on adjacent qubits, as shown in Figure 3.7. ColorDynamic achieves *comparable performance to Baseline G but with simpler hardware (no tunable couplers)*. Results for Baseline G in Figure 3.7 is a conservative estimate, assuming couplers can be deactivated *perfectly*. We study the effect of residual coupling in Figure 3.10. Compared to Baseline U (with serialization), ColorDynamic consistently outperforms, *achieving 13.3x better success rate on average*. Compared to Baseline S, across all benchmarks, ColorDynamic outperforms static strategies because it is able to exploit program structures and assign frequencies tailored for every layer of the program.

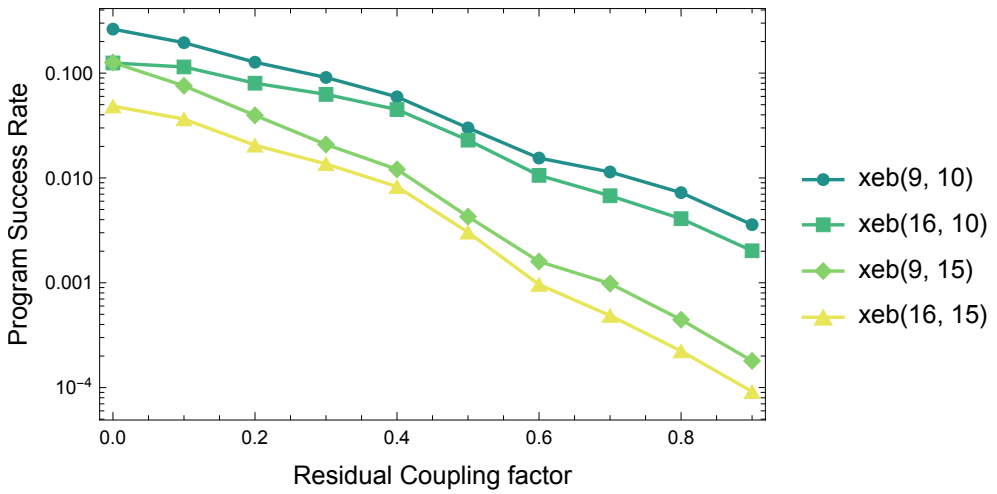


Figure 3.10: Log-scale success rate by strength of residual coupling. Baseline G success rate decays exponentially as residual coupling increases.

3.6.2 Impact on Serialization

Figure 3.8 compares the resulting program depth and decoherence error across algorithms. Although serialization can effectively prevent gates from crosstalk (commonly adopted such as for IBM's fixed-frequency qubits), it results in deeper circuits (i.e. longer execution time), which consequently implies higher qubit decoherence. Overall, baseline U requires the most amount of serialization. ColorDynamic produces *1.02x average decoherence error*, compared to baseline G, and *0.90x average decoherence error*, compared to baseline U. Lower decoherence error is desirable when executing on NISQ hardware.

3.7 Broader Applicability

3.7.1 Software Scalability and Complexity

Globally optimizing for the best frequency configuration based on device and program characteristics is challenging; our approach breaks the optimization problem into multiple scalable sub-problems. ColorDynamic keeps the complexity of each sub-problem small, trading off program parallelism for optimization complexity when necessary. In particular, the leading

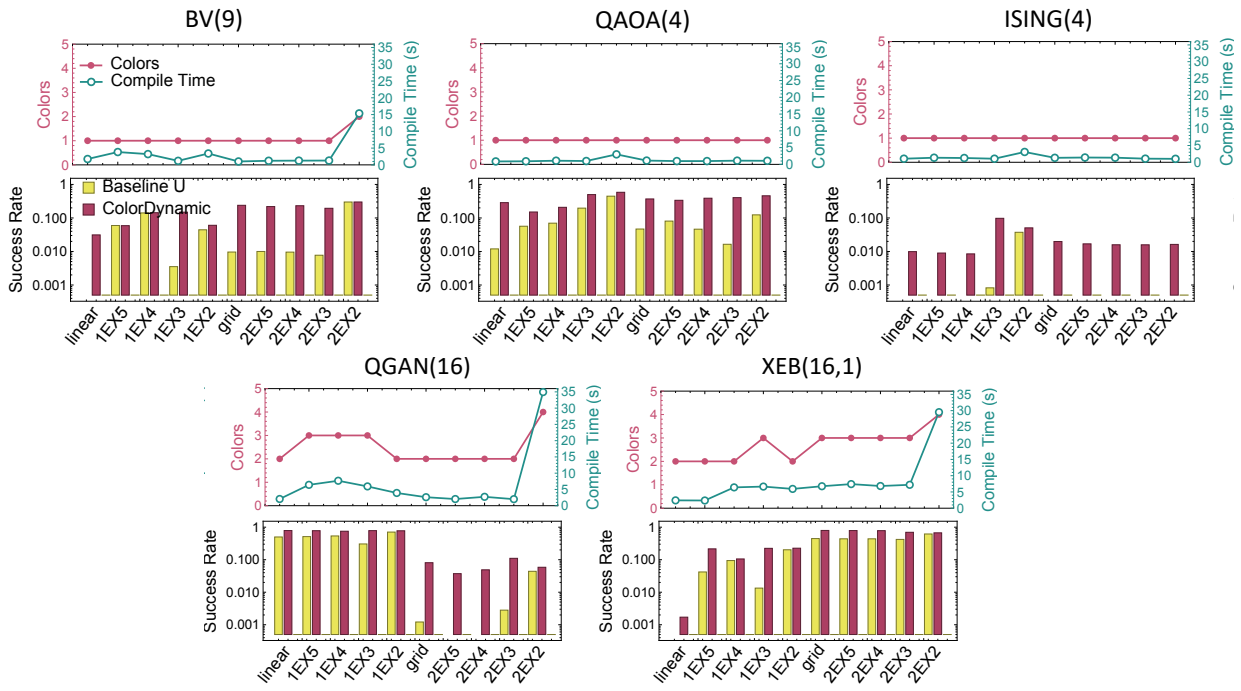


Figure 3.11: Results on general device connectivity across benchmarks. For each panel: *Top:* Number of colors (for interaction frequency) and compilation time of ColorDynamic. *Bottom:* Log-scale program success rate for Baseline U and ColorDynamic. Denser connectivity from left to right along x-axis. n-EX-k is an n-ary express cube [Dal91] with inserted connections every k nodes.

costs stem from coloring of crosstalk graphs and application of SMT solvers.

The greedy coloring algorithm takes time polynomially in the graph size, which is kept small thanks to circuit slicing and strategic serialization. The number of variables/constraints in the SMT solver is proportional to the number of colors obtained from coloring; in the next section, we demonstrate that the number of colors remains small. Empirically, we report the *number of colors and compilation time* of ColorDynamic across benchmarks in Figure 3.11. Compilation time *remains* < 30 *seconds on systems up to 81 qubits* for a highly parallel benchmark such as `XEB`.

3.7.2 Finding Optimal Tunability

In ColorDynamic, we can limit the maximum number of colors used for assigning qubit frequencies. To guarantee low crosstalk, fewer colors implies more serialization. In Figure 3.9, we examine the balance between spectral and temporal optimizations, and find the best tunability for each benchmark. In general, we observe optimal operating point at 1 or 2 colors, depending on the initial parallelism of the benchmark. This result has significant hardware implications – such program-specific optimization shows that frequency-tunable qubits with 2 frequency sweet spots are good candidates for near-term algorithms, hence building qubits with more sweet spots will only give diminishing returns.

3.7.3 General Device Connectivity

To demonstrate the general applicability of our algorithm with respect to device connectivity, we perform a systematic study shown in Figure 3.11. Denser connectivity for superconducting device is challenging [KSB+20], due to limitations such as coupling and addressing qubits. As such, we target a class of connectivity graphs with increasing density while incurring minimal wiring overhead, namely the "express cubes" [Dal91] designed for interconnection networks. In particular, we augment an increasing number of connections to a 1-D linear path and a 2-D grid, denoted as 1EX-k and 2EX-k graphs respectively, where k stands for inserting a connection every k nodes[Dal91].

ColorDynamic consistently *improves program success rate by 3.97x in geometric mean* across all benchmarks, compared to baseline U. Depending on applications, best performance is usually found on connectivity not too sparse or denser than grid. Compilation time of ColorDynamic is kept low ($\sim$ 10 seconds) in practice, because the number of colors remains small, as argued in Section 3.7.1 and Figure 3.9. Empirically, we see some increase in the extreme cases with unrealistically dense connectivity, but still within a desirable range.

3.8 Related Work

A number of hardware features have been proposed to help mitigate crosstalk: *i*) connectivity reduction, *ii*) qubit frequency tuning, and *iii*) coupler tuning. In addition to these hardware features, some software constraints are usually imposed to effectively reduce crosstalk; for example, certain operations may be prohibited to occur simultaneously.

Connectivity reduction works by building devices with sparse connections between qubits, hence reducing the number of possible crosstalk channels. This greatly increases the circuit mapping and re-mapping overhead for executing a logical circuit, since many `SWAP` gates are needed. Moreover, this model necessitates an intelligent scheduler to serialize operations to avoid crosstalk [MMMJA20]. This strategy is commonly deployed for fixed-frequency transmon architectures, e.g. from [IBM]. Because of their non-tunable nature, these architectures have stringent constraints on the initial qubit frequency; a number of optimizers are proposed for this issue [BCH$^+$18, LDX20].

A second class of techniques rely on actively *tuning qubit frequencies* to avoid crosstalk, featured in some prototypes [HHL$^+$17] and by Google [BQP$^+$19]. Software can decide when to schedule an instruction and which frequency to operate the instruction at. In this class, [VPK$^+$17] found a frequency assignment for the surface code circuit; [HMF$^+$09] suggests a sudoku-style pattern of frequency assignment for cavity grid.

A third class builds not only frequency-tunable qubits but also *tunable couplers* between qubits, termed "gmon" architectures [CNR$^+$14]. Without resorting to permanently reducing device connectivity in hardware, a different subset of connections are activated (via flux drives to the couplers) at different time steps. As such, a schedule for when to activate couplers is needed. For instance, Google proposes a tiling pattern in [AAB$^+$19].

Most previous studies on quantum program compilation [SLG$^+$19, GDP$^+$19] have largely targeted short program execution time (i.e. low circuit depth), and neglected the impact of gate errors such as crosstalk. Optimizations are performed at the gate level, typically

involving strategic qubit mapping and instruction scheduling. Recent efforts [MMMJA20, LDX20] are among the first to explore architects' role in mitigating crosstalk.

Our work here shows that *frequency-tunable architecture without connectivity reduction and without tunable couplers (but with our software crosstalk mitigation) is competitive against other architectures.* The frequency-tunable but untunable coupler architecture is an optimization sweet spot. On one end of the spectrum, fixed-frequency architectures have a relatively constrained space for software optimization. On the other end of the spectrum, requiring both qubit frequencies and couplers to be tunable introduces higher overhead in fabrication and higher control noises.

3.9 Chapter Summary

In this chapter, we introduced a systematic approach to software mitigation of crosstalk due to frequency crowding. Our approach allowed fixed coupler architectures to compete with tunable coupler architectures in reliability, potentially simplifying the fabrication of quantum machines. It is a timely demonstration that an architectural solution achieves record level of gain in software performance and hardware stability. Such compilation methodology is likely to guide the technological design space of quantum computing hardware for years to come.

CHAPTER 4

QUANTUM COMPILATION AND MEMORY MANAGEMENT

4.1 Rethinking Garbage Collection in Quantum Systems

Current Noisy Intermediate-Scale Quantum (NISQ) computers and forward-looking Fault-Tolerant (FT) quantum computers have fundamentally different constraints such as data locality, instruction parallelism, and communication overhead. As such, it is not uncommon that we re-think about the architectural design choices we made for classical computers, under the unique constraints in quantum computer systems. In this chapter, we illustrate how re-thinking about memory has led to interesting new research directions across the systems stack.

4.1.1 Think Quantumly About Memory

Quantum memory management is critical to any quantum computer system where the quantum program (sequence of quantum logic gates and measurements) requires the bulk of resources from the system, e.g., demanding nearly all of the qubits available. This problem is pervasive in quantum compiling, since typical quantum algorithms require a significant number of qubits and gates, as they usually make use of a key component called quantum oracles, which implements heavy arithmetic with extensive usage of ancilla qubits (scratch memory).

Much like a classical memory manager, the goal of a quantum memory manager is to allocate and free portions of quantum memory (qubits) for computation dynamically. But, unlike a classical memory manager, its quantum counterpart must respect a few characteristics of quantum memory, such as:

- Data has limited lifetime, as qubits can decohere spontaneously.

- Data cannot be copied in general, due to the quantum no-cloning theorem.
- Reading data (measurements) or incoherence noises on some qubits can permanently alter their data, as well as data on other qubits entangled with them.
- Data processing (computation) is performed directly in memory, and quantum logic gates can be reversed by applying their inverses.
- Data locality matters, as two-qubit gates are accomplished by interacting the operand qubits.

These characteristics of quantum memory profoundly influence the design of quantum computer architectures in general, and complicate the implementation of a memory manager.

A Special Type of Shared State

One of the fundamental limitations of a quantum computer system is the inability to make copies of an arbitrary qubit. This is called the no-cloning theorem due to Wootters and Zurek. In classical computing, we are used to making shared state of data when designing and programming algorithms. The no-cloning limitation prevents us from directly implementing a quantum analog of the classical memory hierarchy, as caches require making copies of data. Hence, current quantum computer architecture proposals follow the general principles that transformations are applied directly to quantum memory, and data in memory are moved but not copied. However, we are allowed to make an entangled copy of a qubit. This type of shared state has the special property that the state of one part of the memory system cannot be fully described without considering the other part(s). Measurements (Reads) on such systems typically result in highly correlated outcomes. As such, reads and writes on entangled states must be handled with care. In classical memory systems, reads and writes must follow models of cache coherence and memory consistency to ensure correctness on shared states. In a quantum system, we can no longer easily read from or write to the

quantum memory, as read is done through measurements (which likely alter the data) and write generally requires complex state preparation routines.

4.1.2 Challenges of Memory Management in Quantum Systems

We raise a series of questions regarding quantum memory: How to allocate and reuse qubits most economically? How to clean up garbage qubits? How to prevent error from propagating through entangled qubits?

Relying on programmers to keep track of all usage of qubits is hardly scalable nor efficient. So design automation plays a crucial role, in order to make programming manageable and algorithms practical. Several techniques have been developed to save the number of qubits used by quantum programs. The effectiveness of memory manager can impact the performance of programs significantly; a good memory manager fulfills the allocation requests of a high-level quantum circuit by locating and reusing qubits from a highly constrained pool of memory.

When a program requires new allocations of qubits, the memory manager faces a decision: whether to assign brand-new unused qubits or to reuse reclaimed qubits. It may seem that reusing reclaimed qubits whenever available is the most economical strategy, for it minimizes the number of qubits used by a program. However, qubit reuse could potentially reduce program parallelism. Operations that could have been performed in parallel are now forced to be scheduled after the last usage of the reclaimed qubits. This additional data dependency could potentially lengthen the overall time to complete the program. Hardware constraints, such as reliability and locality, can also impact the allocation decisions. Some qubits might be more reliable than the others. It could be beneficial to prioritize qubits that are more reliable and balance the workload on each qubit. Some qubits might be closer than the others. Multi-qubit operations performed on distant qubits can therefore induce communication overhead. Ideally, an efficient qubit allocator must make decisions based on

program structures and hardware constraints and reuse qubits discretely.

Techniques to reuse qubits generally fall in to three categories:

- Measurement and Reset — When some qubits are disentangled from the data qubits, we can directly reclaim those qubits by performing a measurement and reset. We can save the number of qubits, by moving measurements to as early as possible in the program, so early that we can reuse the same qubits after measurement for other computation. Prior art has extensively studied this problem and proposed algorithms for discovering such opportunities. This measurement-and-reset (M&R) approach has limitations. First, today's NISQ hardware does not yet support fast qubit reset, so reusing qubits after measurement could be costly or, in many cases, unfeasible. The state-of-the-art technique for resetting a qubit on a NISQ architecture is by waiting long enough for qubit decoherence to happen naturally, typically on the order of milliseconds for superconducting machines, significantly longer than the average gate time around several nanoseconds. Fault-tolerant (FT) architectures have much lower measurement over- head (that is roughly the same as that of a single gate operation), and thus are more amenable to the M&R approach. Second, qubit rewiring works if measurements can be done early in a program, which may be rare in quantum algorithms—measurements are absent in many program (such as arithmetic subroutines) or only present somewhere deep in the circuit. Unlike the uncomputation approach, M&R does not actively create qubit reuse opportunities.

- Qubit Borrowing — Another strategy for reusing qubits involve temporarily borrowing a qubit for computation and return the qubit to its original state when completed. This technique is sometimes called the "dirty borrowing" of qubits, because the qubits we borrow can be in an arbitrary unknown quantum state; this is to be contrasted with the uncomputation technique we will introduce next, in which the qubits to reuse are always clean ancilla (i.e., qubits initialized to a known state such as 0). Dirty borrowing

opportunities depends highly on the structures in quantum circuits; the reason is two-fold. First, we need to return the borrowed qubits to their original states timely, otherwise the original computation has to be stalled. Second, computation of the borrowing circuit is restricted — as it performs computation on borrowed qubit without knowing its exact state. For example, one typically cannot perform measurements on the borrowed qubits, due to their entanglement with other qubits. This technique has been applied to speed up implementations of arithmetic circuits.

- Uncomputation — is to recycle ancilla qubits for future reuse through a process called "uncomputation". This can be thought of as analogous to the concept of *garbage collection* in classical computing. Reclamation comes with a gate cost, as it is accomplished by first storing the output to a safe space and then undoing part of a computation so as to reset the ancilla qubits to their initial value and remove the entanglement relationship with the output qubits. It is critical to perform this uncomputation step, otherwise directly reusing ancilla could also change the value stored in the output qubits. Identifying the appropriate points for reclamation is not an easy task, especially for programs with hierarchical structures. Finding the optimal strategy for programs with general data dependency graphs are shown to be PSPACE complete. The reversible pebbling game strategy demonstrates that uncomputation gate cost can be reduced if a divide and conquer approach is used. In a more realistic setting, a heuristic approach strategically fulfills qubit allocation and reclamation, taking into account information such as program structures, data locality, and qubit/gate noise.

4.1.3 Garbage Collection Done Strategically

For the remaining of the chapter, we consider the motivating example of qubit reuse/garbage collection via "uncomputation" [Ben73b] (i.e., undoing parts of computation to reset scratch qubits to its initial value).

In practice, uncomputation must be done strategically – too much uncomputation could result in significant gate cost while too little uncomputation could cause qubits to run out. An optimal approach would find a middle road, strategically fulfilling qubit allocation and reclamation. We found that considering information such as program structures, data locality, and qubit/gate noise entirely changes the perspective – when evaluated at systems level, strategic uncomputation is found to be beneficial. *The most counter-intuitive, novel discovery of this paper* is that adding gates for uncomputation can usually reduce total number of operations, hence improving the fidelity of a program rather than impairing it. One explanation is that adding uncomputation allows us to create ancilla with better locality, thus reducing the communication cost.

Finally, an advantage of the research study in this chapter is that it casts a wide net – we demonstrate the applicability of our techniques not only in near-term NISQ setting but also in long-term fault-tolerant setting. For the former, we quantify the impact of garbage collection to program fidelity, by performing extensive noisy circuit simulations. For the latter, we introduces the concept of qubit "liveness" quantified by a new metric (active quantum volume) to evaluate and guide optimal resource allocation.

4.2 Reversible Logic Preliminaries

Quantum computers are devices that harness quantum mechanics to store and process information. For this paper, we highlight three of the basic rules derived from the principles of quantum mechanics:

- *Superposition rule:* A quantum bit (qubit) can be in a quantum state of a linear combination of 0 and 1: $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, where α and β are complex amplitudes satisfying $|\alpha|^2 + |\beta|^2 = 1$.

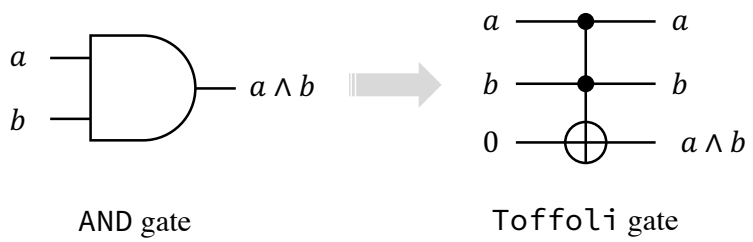


Figure 4.1: Circuit diagram for the irreversible `AND` gate and the reversible `Toffoli` gate.

- *Transformation rule:* Computation on qubits is accomplished by applying a unitary quantum logic gate that maps from one quantum state to another. This process is *reversible* and *deterministic*.

- *Measurement rule:* Measurement or readout of a qubit $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$ collapses the quantum state to classical outcomes: $|\psi'\rangle = |0\rangle$ with probability $|\alpha|^2$ and $|\psi'\rangle = |1\rangle$ with probability $|\beta|^2$. This is *irreversible* and *probabilistic*.

Reversibility constraints.

The above three rules give rise to the potential computing power that quantum computers possess, but at the same time, they impose strict constraints on what we can do in quantum computation. For example, the transformation rule implies that any quantum logic gate we apply to a qubit has to be *reversible*. The classical `AND` gate in Figure 4.1 is *not* reversible because we cannot recover the two input bits based solely on one output bit. To make it reversible, we could introduce a scratch bit, called *ancilla*, to store the result out-of-place, as in controlled-controlled-`NOT` gate (or `Toffoli` gate) in Figure 4.1. Note that we use the terminology "ancilla" in its most general sense–it is not limited to error correction ancilla, but rather, any (physical or logical) qubits used as scratch space for computation. As the arithmetic complexity scales up when tackling difficult computational problems, we quickly see extensive usage of ancilla bits in our circuits due to this *reversibility constraint*.

4.2.1 Synthesizing Reversible Arithmetic

For small arithmetic logic, algorithms exist to directly synthesize reversible circuits from the truth-table of the desired function [GWDD09, MMD03, SRWDM17] and with templates [MDM05]. This typically works well for small low-level combinational functions, but not for functions with internal states [PRS15]. As the complexity of the arithmetic in an algorithm scales up, *modularity* quickly becomes convenient, and in many cases necessary. That is, to construct high-level arithmetic, we need to build up from small modular subroutines.

In reversible logic synthesis and optimization, besides making our circuit for the reversible function contain as few gates as possible, we would also like to minimize the amount of scratch memory (i.e. number of ancilla bits) used in the circuit. Fortunately, there is a way to recycle ancilla bits for later reuse. For a circuit that makes extensive use of scratch memory, managing the allocation and reclamation of the ancilla bits becomes critical to producing an efficient implementation of the function.

Role of reversible arithmetic in quantum algorithms.

Reversible arithmetic plays a pivotal role in many known quantum algorithms. The advantage of quantum algorithms is thought to stem from their ability to pass a superposition of inputs into a classical function at once, whereas a classical algorithm can only evaluate the function on single input at a time. Many quantum algorithms involve computing classical functions, which must be embedded in the form of reversible arithmetic subroutines in quantum circuits. For example, Shor's factoring algorithm [Sho99] uses classical modular-exponentiation arithmetic, Grover's searching algorithm [Gro96a] also implements its underlying search problem as an oracle subroutine, and the HHL algorithm for solving linear system of equations contains an expensive reciprocal step [HHL09a]. These reversible arithmetic subroutines are typically the most resource-demanding computational components of the entire quantum circuit.

4.2.2 Reusing Qubits

Reclaiming Ancilla Qubits via Uncomputation

Reclaiming qubits is the process of returning them to their original $|0\rangle$ state for future reuse. Due to entanglement, this process could be costly; ancilla qubits that are entangled with data qubits will alter the data qubits' state if they are reset or measured. Fortunately, *uncomputation*, introduced by Bennett [Ben73b], is the process for undoing a computation in order to remove the entanglement relationship between ancilla qubits and data qubits from previous computations. Figure 4.2 (left pane) illustrates this process. In that circuit diagram, the U_f box denotes the circuit that computes a classical function f. The garbage produced at the end of U_f is cleaned up by storing the output elsewhere and then undoing the computation.

This uncomputation approach has two potential limitations: firstly, if uncomputation is not done appropriately, we need to pay for the additional gate cost, and secondly, it only works if the circuit U_f implements classical reversible logic - i.e. can be implemented with `Toffoli` gate alone, optionally with `NOT` gate and `CNOT` gate. Quantum algorithms contain non-classical gates such as Hadamard gate, phase gate and T gate; this work focuses on the part in quantum algorithms that computes classical functions (usually arithmetics) which can be implemented without those gates. As discussed in Section 4.2.1, classical reversible logic plays a large part of most quantum algorithms.

Related work on optimization of qubit allocation and reclamation in reversible programs dates back to as early as [Ben89, BTV01], where they propose to reduce qubit cost via fine-grained uncomputation at the expense of increasing time. Since then, more [CLNV15, FKJ99, Kni95, KSS18] have followed in characterizing the complexity of reclamation for programs with complex modular structures. Recent work in [ARS17, PRS15] show that knowing the structure of the operations in U_f can also help identify bits that may be eligible

for cleanup early. A more recent example [MSR+19] improves the reclamation strategy for straight-line programs using a SAT-based algorithm. Some of the above work emphasizes on identifying reclamation opportunities in a flat program, whereas our focus is on coordinating multiple reclamation points in a larger modular program.

Reclaiming Qubits via Measurement and Reset

If ancilla qubits have already been disentangled from the data qubits, we can directly reclaim them by performing a measurement and reset. We can save the number of qubits, by moving measurements to as early as possible in the program, so early that we can reuse the same qubits after measurement for other computation. Prior art [PFW19, PWD16] has extensively studied this problem and proposed algorithms for discovering such opportunities.

This measurement-and-reset (M&R) approach also has limitations: firstly, a near-term challenge for NISQ hardware is to support fast qubit reset. Without it, reusing qubits after measurement could be costly or, in many cases, unfeasible. The state-of-the-art technique for resetting a qubit on a NISQ architecture is by waiting long enough for qubit decoherence to happen naturally, typically on the order of milliseconds for superconducting machines [IBM], significantly longer than the average gate time around several nanoseconds. FT architectures have much lower (logical) measurement overhead (that is roughly the same as that of a single gate operation), and thus are more amenable to the M&R approach. Secondly, qubit rewiring as introduced in [PWD16] works only if measurements can be done early in a program, which may be rare in quantum algorithms – measurements are absent in many program (such as arithmetic subroutines) or only present somewhere deep in the circuit. M&R of a qubit is allowed only after all entangled results are no longer needed, whereas uncomputation can be done partially for any subcircuit. As such, unlike the uncomputation approach, M&R does not *actively create* qubit reuse opportunities.

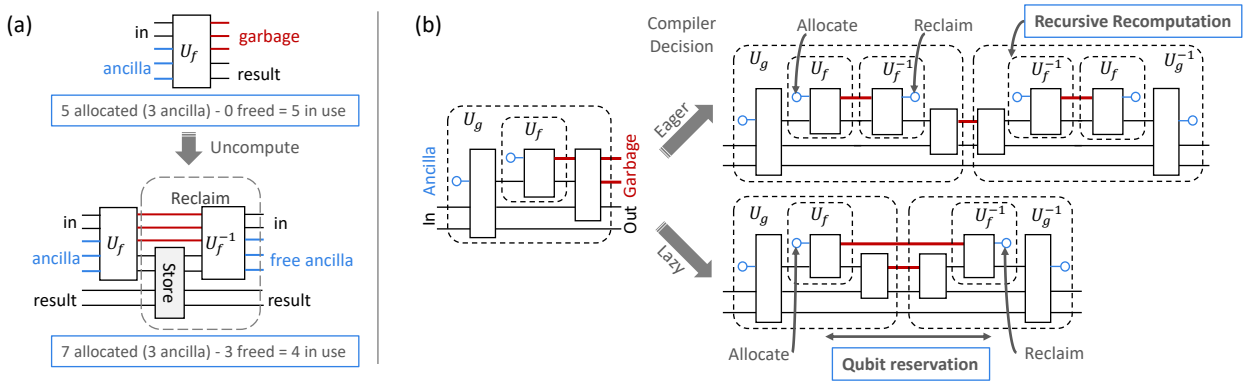


Figure 4.2: (a) Ancilla qubit reclamation via uncomputation. Each horizontal line is a qubit. Each solid box contains reversible gates. Qubits are highlighted red for the duration of being garbage. (b) Illustration for Eager and Lazy strategies with their respective issues – recursive recomputation and qubit reservation. Each dashed box denotes a function call containing the enclosed gates. The allocation and reclamation points have been marked as blue circles in the circuit.

4.2.3 Connection to Classical Compilation

Some similarities can be seen in register allocation in classical computing. In that setting, we assign program variables to a limited number of registers in the CPU for fast access. Variables that are not stored in register may be moved to and from RAM, as a process called "register spilling". The analysis of live variable and register reuse can be very similar to that of qubits. For instance, our heuristic-based methodology is inspired by register allocation in GPU/distributed systems where communication cost needs to be minimized, and by the technique "rematerialization" that reduces the register pressure (i.e. number of registers in use at any point in time) by recomputing some variables instead of storing them to memory. But the trade-offs in qubit allocation and reclamation are unique, which we will introduce as "recursive recomputation" and "qubit reservation" in Section 4.4.2. Finding the optimal strategy for register allocation, and similarly for qubit reuse, is known to be a hard problem [BDGR06, CLNV15]. Luckily, we are able to transfer some general insights from the rich history of classical register allocation optimization to solve the problems in qubit allocation and reclamation.

The connection made between qubit reuse and classical register allocation [BCT94, Cha82, PS99] allows us to inherit some of the intuitions from a wealth of classical literature. Nonetheless, the uncomputation/reuse/locality trade-offs we face are fairly unique. Indeed, rematerialization [BCT92] is very much like qubit reclamation, in that they both aim to lower active registers/qubits at the expense of computation, yet it does not exhibit the same exponential recomputation cost, nor is the increase in the live-range of variables from the recomputing step the same as qubit reservation caused by not uncomputing. We also gained general insights from numerous techniques in code scheduling[GH14, Pin93], and thread-level parallelism[XLL+15].

4.3 Motivation: Qubit and Gate Overheads

One major challenge, however, facing the QC community, is the substantial resource gap between what quantum computer hardware can offer and what quantum algorithms (for classically intractable problems) typically require. Space and time resources in a quantum computer are extremely constrained. Space is constrained in the sense that there will be a limited number of qubits available, often further complicated by poor connectivity between qubits. Time is also constrained because qubits suffer from decoherence noise and gate noise. Too many successive operations on qubits results in lower program success rates.

Due to space and time constraints, it is critical to find efficient ways to compile large programs into programs (circuits) that minimize the number of qubits and sequential operations (circuit depth). Several options have been proposed [BAN+19, CH17, CFM17, HC18, PPND17, SHT18, WS14a]. Among the options, one approach not yet well studied is to coordinate allocation and reclamation of qubits for optimal reuse and load balancing [SHR18]. Reclaiming qubits, however, comes with a substantial operational cost. In particular, to obey the rules of quantum computation, before recycling a used qubit, additional gate operations need to be applied to "undo" part of its computation.

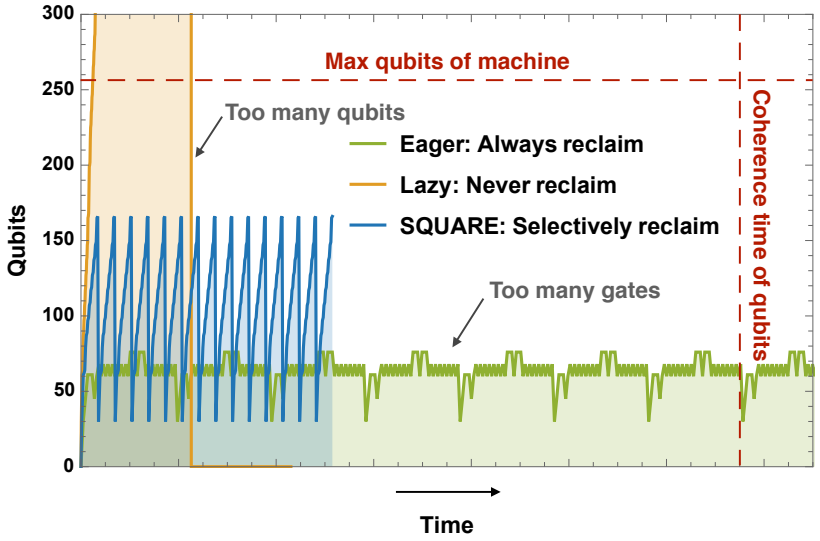


Figure 4.3: Qubit usage over time for Modular Exponentiation. The shaded area under the curve corresponds to the active quantum volume of this application. The blue curve, representing a balance between qubit reclamation and uncomputation, has the lowest area and is the best option.

Optimally choosing reclamation points in a program is crucial in minimizing resource consumption. This is because reclaiming too often can result in significant time cost (due to more gates dedicated to uncomputation). Likewise, reclaiming too seldom may require too many qubits (e.g. fail to fit the program in the machine). For example, Figure 4.3 shows how qubit usage changes over time for the modular exponentiation step in Shor's algorithm [Sho99]. Unfortunately, finding the optimal points in a program for reclaiming qubit could get extremely complex [Ben89, Kni95]. An efficient qubit reuse strategy will play a pivotal role in enabling the execution of programs on resource-constrained machines.

4.3.1 Defining Workload: Active Quantum Volume

To precisely estimate the workload of a computational task, we propose a resource metric called "*active quantum volume*" (AQV) that evaluates the "liveness" of qubit during the lifetime of the program, which we will formally introduce in Section 4.3.1. This is inspired by the concept of "quantum volume" introduced by IBM [BBC+17], a common measure for

the computational capability of a quantum hardware device, based on parameters such as number of qubits, number of gates, and error probability. AQV is a metric that measures the volume of resource required by a program when executing on a target hardware, which can therefore serve as an minimization objective for the allocation and reclamation strategies.

To quantify the workload of a program, we define the active quantum volume (AQV) of a program as:

$$V_A = \sum_{q \in Q} \sum_{(t_i, t_f) \in T_q} (t_f - t_i)$$

where Q is the set of all qubits in the system, and T_q is a sequence of pairs

$$\{(t_i^0, t_f^0), (t_i^1, t_f^1), \ldots, (t_i^{|T_q|-1}, t_f^{|T_q|-1})\}$$

Each pair corresponds to a qubit usage segment, that is we denote t_i^k and t_f^k as the allocation time and reclamation time of the k^{th} time that qubit q is being used, respectively. AQV is high when a large number of qubits stay "live" (in-use) during the execution; thus, the higher the AQV, the more costly it is to execute on that target machine.

The key to this metric is in the term "active". In particular, we exclude the time that a reclaimed qubit spends in the heap from volume calculation, because it has been restored to the $|0\rangle$ state (ground state), which does not suffer from the decoherence noise as an excited state does. Hence, AQV serves as a *minimization objective* in SQUARE. There are a few practical advantages for using AQV over other resource metrics:

1. AQV is a better measure of the exposure to errors than the space×time metric (i.e. number of qubits times circuit depth) [FMMC12, HDJA$^+$19]. The more time a qubit stays live, the more susceptible it is to noise from its surroundings. We show lower AQV yields higher success rate in Section 4.6.2.

2. Unlike qubit count, gate count, or circuit depth, AQV allows us to more accurately model "liveness" of qubits on a machine (i.e. which qubits are actively carrying information and performing computation as opposed to staying in ground state unused). [MBJA+19b] and [TQ18] shows that keeping a preferred subset of qubits live can boost program success rate.

3. IBM's quantum volume (QV) [BBC+17] characterizes the amount of computational resource a quantum device offers, AQV measures the portion of resource being actively utilized by a program on the device.

4.4 SQUARE Compiler Tool: A Full-Stack Approach

4.4.1 Instrumentation-Driven Compilation

In this section, we illustrate a number of advantages of the instrumentation-drive approach over the conventional pass-drive approach used in most quantum compilers.

The traditional pass-driven approach for compiling and optimizing quantum programs is done by sending a high-level quantum program through multiple layers of transformations, each of which completes a different task. For instance, we have transformations to resolve classical control structures (e.g. loop unrolling and module inlining), explore circuit optimizations (e.g. commutativity and parallelism), satisfy architectural constraints (e.g. qubit connectivity), assign qubit mappings, and perform gate scheduling, etc. One of the potential limitations in this approach is that each transformation performs independently, and in some cases even conflicts with each other [GH14]. So it is very hard to jointly optimize for some correlated problems such as mapping and scheduling. Techniques such as feedback loops could in some cases work well in practice.

Two main reasons that the instrumentation-driven approach may be a more natural fit for our purpose are: the dynamic nature of our optimization and compilation time scalability.

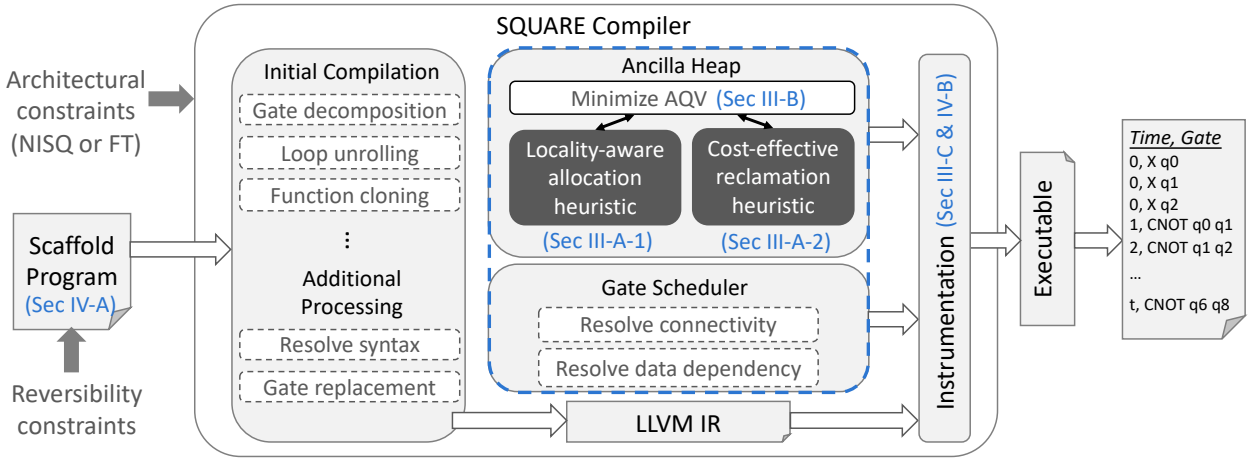


Figure 4.4: Our Strategic Quantum Ancilla Reuse (SQUARE) compilation flow. SQUARE takes as input a Scaffold [JAPK+14] program (see sample code in Figure 4.6) and produces an executable that simulates the dynamics of qubit allocation/reclamation and gate scheduling, which can then prints out an optimized schedule of quantum gate instructions.

Recall from Section 4.4.2, our compilation tool flow produces an executable that allows us to dynamically optimize for the allocation and reclamation of qubits in reversible programs with parallel and modular structures.

4.4.2 Compilation Tool Flow of SQUARE

Most existing qubit reuse algorithms [Ben89, BTV01, PRS15] emphasize on the asymptotic qubit savings, and commonly make an ideal assumption that machines have all-to-all qubit connectivity (i.e. no locality constraint). Since all qubits are considered identical, a straightforward way to keep track of qubits is to maintain a global pool, sometimes referred to as the *ancilla heap*. Ancilla qubits are pushed to the heap when they are reclaimed, and popped off when they are allocated, for instance in a last-in-first-out (LIFO) manner. In this ideal model, we can simply track qubit usage by counting the total number of fresh qubits ever allocated during the lifetime of a program. However, leading proposals of NISQ and FT quantum architectures have far stricter locality constraints.

Our Strategic QUantum Ancilla REuse (SQUARE) algorithm is highly motivated by the

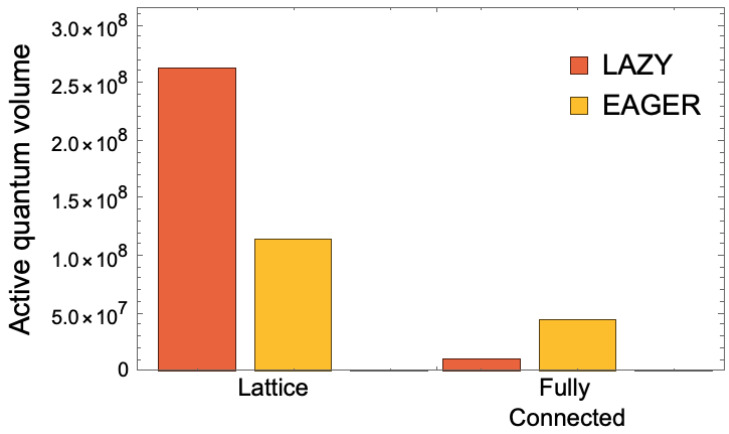


Figure 4.5: Locality constraint changes the desired reclamation strategy. Results are based on a synthetic benchmark "Belle". Lower active quantum volume (defined in Section 4.3.1) is better. Belle performs better on a lattice machine with Eager strategy, while preferring Lazy when operating on a fully-connected machine.

lesson that communication can be a determinant factor for qubit allocation and reclamation. Take the NISQ architecture as an example. We make the following two novel observations. Firstly, *same algorithm needs different strategies for different machine connectivity.* In Figure 4.5, a benchmark named Belle (whose details can be found in Section 4.6.1) prefers Eager strategy on a 2-D lattice topology (with swaps), but Lazy strategy on a fully-connected topology (without swaps). Secondly, and most counter-intuitively, *adding uncomputation gates can improve overall circuit fidelity, if done properly.* With careful allocation and reclamation, the expense of additional uncomputation gates is compensated by the reduction of communication cost. This is because uncomputation allows us to create ancilla with better locality, resulting in fewer swap gates and less overall gate noise.

In a nutshell, our SQUARE compilation algorithm takes as input a Scaffold program [JAPK+14] and produces an optimized schedule of all of its quantum instructions. This is accomplished through what is known as the "instrumentation-driven" approach, also used in [HPJ+15b], which allows us to pre-simulate the control flow in a quantum program. This works because all inputs are known at compile time for most quantum programs, so we can use their known control flow to simulate resource usage.

Figure 4.4 illustrates in detail the compilation flow for SQUARE. It consists of three main components: 1) an easy-to-use syntactical construct compatible within the Scaffold language, 2) a qubit allocation heuristic, and 3) a qubit reclamation heuristic.

Under the hood, an input program first goes through an initial compilation step, where each `Allocate()` and `Free()` instruction is replaced by a classical function call (such as in C/C++) that implements the heuristic algorithm. Each quantum gate is replaced by a classical function that resolves the connectivity constraints of its operand qubits and then schedules the gate to the earliest time step possible. As a result, we have obtained an executable for the classical control flow of the quantum program. The compiler maintains an ancilla heap (i.e. pool of reclaimed qubits) that stores all the reclaimed ancilla qubits. Future allocations can therefore choose to pop from the ancilla heap or initialize brand new qubits. One of the key contributions of our work is a heuristic that makes such decisions.

4.4.3 Compilation Complexity

SQUARE is a heuristic-based greedy algorithm. It makes allocation and reclamation decisions as they appear in program order. As a result, it takes time that scales linearly to the number of reclamation points in a program. Consider a program with nested functions – all decisions in the callees are made prior to that of the caller, so when deciding for the caller function, the cost of uncomputation is deterministic and easy to estimate. On the flip side, we could end up in a sub-optimal situation where callee's decisions are made neglecting the potential burden for uncomputing its caller.

The computational complexity of qubit reclamation via uncomputation has been studied. It has been shown that, for programs with linear sequential dependency graph, we can use the reversible pebbling game to approach this problem [MSR+19]. However, finding the optimal points in a program with hierarchical structure is PSPACE complete [CLNV15]. For a program with ℓ levels and d callees per function, there can be as many as d^ℓ possible

reclamation points in the worst case. We could be dealing with 2^{d^ℓ} different combinations of reclamation decisions. So clearly, the naïve way for finding the optimal strategy by exhaustively enumerating all possible decisions is far from efficient.

4.5 Case Study: Using SQUARE to Perform Garbage Collection

Now we discuss how SQUARE finds the strategies for allocation and reclamation.

Locality-Aware Allocation (LAA).

We present the Locality-Aware Allocation (LAA) heuristic in the SQUARE algorithm that prioritizes qubits according to their locations in the machine. At a high level, LAA chooses qubits from the ancilla heap by balancing three main considerations – communication, serialization, and area expansion.

When deciding which qubits to allocate and reuse, our heuristic-based algorithm assigns priorities to all qubits. The priorities are weighted not only by the communication overhead of two-qubit interactions but also by their potential impact to the parallelism of the program. Reusing qubits adds data dependencies to a program and thus serializes computation (which is similar to how reusing register names could lead to false data dependencies and serialization), but not reusing qubits expands the area of active qubits and thus increases the communication overhead between them. Recall from Section 1.2.1, communication have different tradeoffs under NISQ and FT architectures. We will make this distinction in terms of our heuristics clearer in Section 4.5.2.

Cost-Effective Reclamation (CER).

The Cost-Effective Reclamation (CER) heuristic makes uncomputation decisions with a simple *cost-benefit analysis*: at each potential reclamation point, we estimate and compare

two quantities:

- C_1: cost of uncomputation and reclaiming ancilla qubits;
- C_0: cost of no uncomputation and leaving garbage qubits.

CER balances the cost of recursive recomputation and qubit reservation as discussed in Section 4.2.2. To do so, we need an efficient way to accurately estimate the C_1 and C_0 quantities.

In particular, the decision of child function affects not only the cost of itself, but also the cost of parent function. If a child function decides to uncompute, the additional gate costs need to be duplicated should its parent decide to uncompute as well. This was illustrated in Section 4.4.2 as the phenomenon we called "recomputation". Thus, we should take the level of the function into account when we make the decision. The total cost of a uncomputation, C_1, can be expressed as:

$$C_1 = N_{active} \times G_{uncomp} \times S \times 2^{\ell} \tag{4.1}$$

where N_{active} is the number of active qubits, G_{uncomp} is the number of gates for uncomputation (including those in all children functions), ℓ is the level of the child function in the program call graph, and S is the communication factor. Details can be found in Section 4.5.3.

Now, suppose a function does *not* uncomputing/reclaiming ancilla, the next chance to reclaim them is when its parent function uncomputes. Thus, we want to estimate the cost of holding the ancilla live until the parent's uncompute block is executed. The cost, C_0, can be approximated as:

$$C_0 = N_{anc} \times G_p \times S \times \sqrt{(N_{active} + N_{anc})/N_{active}} \tag{4.2}$$

```
#include "qalloc.h"

void fun1(qbit* in, qbit* out) {
  qbit anc[1];
  Allocate(anc, 1);
  Compute {
    Toffoli(in[0], in[1], in[2]);
    CNOT(in[2], anc[0]);
    Toffoli(in[1], in[0], anc[0]);
  }
  Store {
    CNOT(anc[0], out[0]);
  }
  Uncompute{
    // Invoke Inverse() to populate
    // Or write out explicitly:
    Toffoli(in[1], in[0], anc[0]);
    CNOT(in[2], anc[0]);
    Toffoli(in[0], in[1], in[2]);
  }
  Free(anc, 1);
}

int main () {
  qbit new[4]; // declare name
  Allocate(new, 4); // allocate qubits
  fun1(new, &new[3]);
  return 0;
}
```

Figure 4.6: Format of *compute-store-uncompute* construct for qubit allocation and reclamation. Shown here an example function (`fun1`) that allocates and reclaims an ancilla qubit.

where N_{anc} is the number of ancillae held by the function, G_p is the number of gates from the current function to the parent's uncompute function. The term under the square root sign captures the effect of 'area expansion", which we will discuss in greater detail in Section 4.5.3.

In the remaining of the section, we describe the implementation details of the components of SQUARE algorithm, including the expressive syntactical construct in the Scaffold programming language [JAPK+14] that exposes the optimization opportunities, the instrumentation-driven LLVM [HPJ+15b] that translates the quantum program into a classical executable, and details of the locality-aware allocation heuristics and the cost-effective reclamation heuristics that we left out from Section 4.4.2.

4.5.1 Syntactical Construct

In order to express the opportunities for qubit allocation and reclamation optimizations, we augment the high-level Scaffold [JAPK+14] programming language with an additional syntactical construct: *Compute-Store-Uncompute Code Blocks*. As shown in Figure 4.6, the keywords "Allocate" and "Free" are used to express the locations of qubit allocation and reclamation respectively. To enable automation in the optimizations, the compiler needs additional information about the code structure. By writing a `Compute` code block, the program now has explicitly specified the set of instructions that belong to forward computation. Optionally, programmer can choose to automatically generate the content of the `Uncompute` block by invoking `Inverse()`.

Under the hood, the compiler will replace each `Allocate` and `Free` instruction with an invocation to our heuristic algorithms. Depending on the reclamation decision, it will either execute or skip the uncomputation step accordingly.

4.5.2 Allocation Policy Details

The allocation policy is most concerned about the communication overhead of two-qubit operations in a program.

- Under NISQ architecture, communication between two qubits is accomplished by move one qubit to another via a series of swaps. So swap distance is a direct measure of the locality. The higher the distance, the longer it takes for a chain of swaps to complete.

- The concept of locality can be trickier in a FT architecture. Communication is accomplished via braiding. Braids can have arbitrary length or shape, but they are not allowed to cross. As [DHJA+18] shows, average braid length and average braid spacing are both strongly correlated with the number of braid crossings. So we can reduce communication overhead by moving interacting qubits closer and moving non-interacting

Algorithm	Description
Eager	Reclaim qubits whenever possible, as shown in Section 4.6.1.
Lazy	Only reclaim qubits from the top level in the program call graph, as shown in Section 4.6.1.
SQUARE	Combines Locality-aware allocation (LAA) and Cost-effective reclamation (CER). See Section 4.4.2.

Table 4.1: List of compiler configurations.

qubits far apart.

When there are fewer qubits available than requested (due to either the maximum qubit constraint or a shortage in the ancilla heap), we mark the allocation as pending, and proceed to schedule all non-dependent, parallel computation and reclamation. Allocation requests are not fulfilled until sufficient ancilla qubits have been reclaimed.

4.5.3 Reclamation Policy Details

The reclamation policy dictates what and when ancilla qubits get recycled. The decisions rely heavily on three main considerations: qubit savings, uncomputation gate count, and communication overhead. In Section 4.5, we have discussed how SQUARE balances between qubit savings and gate count. Now we present further details on how to estimate the communication factor in Equation 4.1 and 4.2.

- NISQ architecture: We use the average swap-chain length per gate as the estimate for S. This is obtained from the history of swap chains during the compile time simulation – we keep a running average of the number of swaps for the gates we scheduled, and use it as an estimate for the subsequent gates in the same module.

- Fault-tolerant (FT) architecture: We use average braiding conflicts per gate as the estimate for S. The communication latency due to braid routing is estimated (similarly as [DHJA$^+$18]) by factoring in the average braid length, average braid spacing, and number of crossings.

Since "qubit reservation" causes the active qubit area to be expanded, leading to higher communication overhead, the multiplicative factor $\sqrt{(N_{active} + N_{anc})/N_{active}}$ aims to estimate the swap or braid length increase due to the expansion.

Algorithm 3 and 4 are pseudo-code of SQUARE, implementing LAA and CER respectively. Procedures under namespace LLVM are functions that operate on the LLVM IR. In particular, *get_interact_qubits()* obtains the set of qubits with which the allocated qubits interact by looking ahead in the code block. *gen_uncompute_block()* and *rm_uncompute_block()* conditionally expands or deletes the code block under `Uncompute{}` (as shown in Figure 4.6). *closest_qubit_in_heap()* and *closest_qubit_new()* look for available qubits to reuse from the heap and from new qubits, respectively. Both functions return the candidate qubits and scores. The scores are calculated based on the communication, serialization, and area expansion, as described in Sec 4.5.2. We select the qubits with minimum scores until the requested n qubits are allocated.

Algorithm 3 Allocate: *Locality-Aware Allocation*

Input: Number of qubits n

Output: Set of qubits $\mathcal{S}$

$\mathcal{I} \leftarrow$ LLVM::get_interact_qubits()
$\mathcal{S} \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** n **do**
    $q_1, score_1 \leftarrow$ closest_qubit_in_heap($\mathcal{I}$)
    $q_2, score_2 \leftarrow$ closest_qubit_new($\mathcal{I}$)
    if $score_1 \leq score_2$ **then**
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{q_1\}$
    else
        $\mathcal{S} \leftarrow \mathcal{S} \cup \{q_2\}$
    end if
end for

Algorithm 4 Free: *Cost-Effective Reclamation*

Input: Number of qubits n, Set of qubits $\mathcal{S}$

$C_1 \leftarrow$ cost of uncomputation
$C_0 \leftarrow$ cost of no uncomputation
if $C_1 \leq C_0$ **then**
    LLVM::gen_uncompute_block()
    heap_push($n, \mathcal{S}$)
else
    LLVM::rm_uncompute_block()
    LLVM::transfer_to_parent($n, \mathcal{S}$)
end if

4.6 Evaluation Methodologies: From Noisy Machines to Fault-Tolerant Machines

4.6.1 Experimental Setup

Benchmarks

Table 4.2 lists the QC benchmarks and brief description in our study. These benchmarks are reversible arithmetic functions or applications that use ancilla qubits. Since ancilla qubits are expensive in both NISQ and FT architectures, it is crucial to reuse ancilla qubits and improve the success rate of a program. The first 4 benchmarks (RD53, 6SYM, 2OF5, and ADDER4) are small arithmetic functions suitable for executing on NISQ systems (10 - 100 qubits). The rest of the benchmarks are medium to large functions that are more demanding in computational resources than current NISQ systems can offer. The number of qubits they use, for instance, is on the order of hundreds or thousands. For the last three benchmarks, we construct random synthetic circuits (Jasmine, Elsa, and Belle) with different characteristics

Name	Description
RD53	Input weight function with 5 inputs and 3 outputs.
6SYM	Function with 6 inputs and 1 output.
2OF5	Output is 1 if number of 1s in its input equals two.
ADDER4	4-bit in-place controlled-addition[1].
Jasmine-s	Small and shallow instance of synthetic benchmark Jasmine.
Elsa-s	Small and shallow instance of synthetic benchmark Elsa.
Belle-s	Small and shallow instance of synthetic benchmark Belle.
ADDER32	32-bit in-place controlled-addition.
ADDER64	64-bit in-place controlled-addition.
MUL32	32-bit out-of-place controlled-multiplier.
MUL64	64-bit out-of-place controlled-multiplier.
MODEXP	Modular exponentiation function[2].
SHA2	Cryptographic hash function[3].
SALSA20	Stream cipher core function[4].
Jasmine	Shallowly nested synthetic function[5].
Elsa	Heavy workload and shallowly nested synthetic function.
Belle	Light workload and deeply nested synthetic function.

Table 4.2: Program characteristics used in the benchmark suite.

in their program structures. In particular, a benchmark is parameterized by the size and shape of its program call graph using 5 variables: number of nested levels, max number of callees per function, max number of input qubits per function, max number of ancilla qubits per function, maximum number of gates per function.

1. The adders are based on the Cucarro adder [CDKM04, MS12].

2. Modular exponentiation is an important subroutine used in Shor's factoring algorithm[Sho99].

3. SHA2 contains multiple rounds of in-place modular additions and bit rotations, based on the implementation from [PRS15]. When used as an oracle in Grover's algorithm[Gro96a], we can find hash collisions more efficiently, and thereby reduce the security of the hash function.

4. Salsa20 involves 20 rounds of 4 parallel modules. Each module modifies 4 words with modular additions, XOR operations, and bit rotations. The Salsa20 stream cipher uses the Salsa20 core function to encrypt data. [Ber08] Salsa family functions have been popularly adopted for TLS in places like the Chrome browser and OpenSSH.

5. Qubits and gates are randomly assigned.

Baselines

This paper focuses primarily on reusing qubits via uncomputation, and discusses the significance of our proposed strategy in current noisy intermediate-scale quantum (NISQ) and future fault-tolerant (FT) architectures. Prior work such as [PRS15] follows two basic strategies: "Eager" cleanup and "Lazy" cleanup, as illustrated in Figure 4.2.

Baseline 1 "Eager": Recursive Recomputation. Eager reclaims qubits at the end of every function. In the example of Figure 4.2, Eager performs uncomputation at the end of both U_f and U_g. When reclaiming ancilla qubits in such programs with nested functions, the uncompute step of the caller would have to repeat *everything* inside of its callee, including the callee's uncompute step. This hierarchical structure will consequently lead to re-computation of the callees, as marked in Figure 4.2. More formally, for a hierarchical program with ℓ levels, in the worst case, recomputation causes the number of steps to increase by a factor of 2^ℓ. We call this exponential blowup phenomenon "recursive recomputation". That is why the 2-level program in Figure 4.2 has roughly 4 times more steps as the original circuit. This factor will play a crucial role in our heuristic design.

Baseline 2 "Lazy": Qubit Reservation. Lazy reclaims qubits only at the top-level function. In Figure 4.2, this means only U_g is uncomputed, but not U_f. Lazy can sometimes be a preferred strategy because it avoids the wasted recomputation[6]. In other words, it is sometimes beneficial to temporarily leave the garbage of callees, and uncompute the garbage by their callers. This is equivalent to *inlining* the callee into the caller, and letting the caller handle the reclamation of all ancilla qubits. However, with the benefit of the avoided recomputation comes the cost of "qubit reservation". The ancilla qubits from callee are *reserved* or *blocked out* from any reuse until the end of the caller. This can be seen at the bottom right of the example in Figure 4.2. The garbage qubit from U_f stays as garbage until

6. There are exceptions, such as recursive Fourier sampling, where recomputation cannot be avoided and is required for correctness [Aar03].

almost the end of U_g^{-1}, whereas in the Eager case, it is cleaned up right away.

Simulation Setup

All compilation experiments are carried out on Intel Core i7-3960X (3.3GHz, 64GB RAM), implemented in the quantum compiler framework ScaffCC [JAPK+14] version 4.0. Noise simulations use Intel E5-2680v4 (28-core, 2.4GHz, 64GB RAM), performed using the IBM `Qiskit` software [AAA+19]. Table 4.1 lists the ancilla reuse algorithms in our study. *Eager* and *Lazy* are two baselines that appear commonly in prior work. *SQUARE* is our Strategic QUantum Ancilla REuse algorithm.

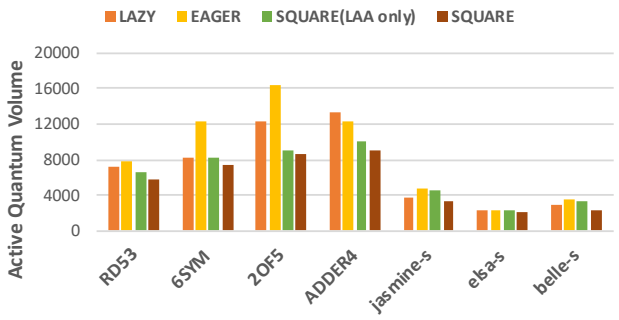
The rest of the section are divided up into three main parts – experimental results on NISQ architecture (Section 4.6.2) with 2D lattice of physical qubits and nearest-neighbor connectivity as commonly used in [IBM, HJG+18, SSP14], on NISQ-FT boundary architecture (Section 4.6.3) with same architecture model but on larger benchmarks, and on FT architecture (Section 4.6.4) with surface code error corrected logical qubits [BKM+14b, DHJA+18].

4.6.2 NISQ Experiments

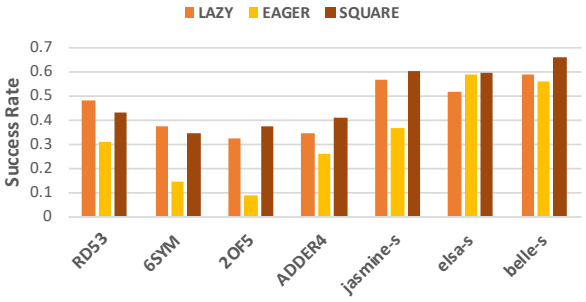
Although SQUARE was initially designed to improve the performance of large-scale applications, we find that reclaiming ancilla reduces program footprint and thus swap count due to communication on NISQ machines. In this section, we give analytic and noise simulation results that quantify the fidelity gains from this reduced swap count. To make noise simulation tractable, we focus on small benchmarks and introduce small versions of our 3 synthetic benchmarks as in Table 4.2.

AQV Analysis

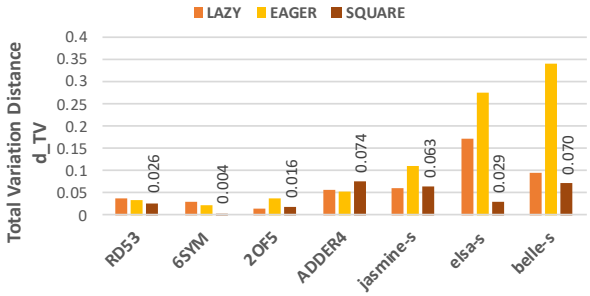
For our NISQ benchmarks, Figure 4.7a and Table 4.3 show the characteristics of the compiled QC programs with different compiling policy. With the Eager compiling policy, the programs



(a) Active quantum volume. (Lower AQV is better.)



(b) Worst-case analytical model. (Higher success rate is better.)



(c) Realistic noise simulation using IBM `Qiskit Aer` simulator. (Lower total variation distance is better.)

Figure 4.7: Impact of SQUARE optimizations on NISQ applications. All benchmarks use fewer than 20 qubits; SQUARE stands out as a strategy that uses substantially fewer qubits while maintaining high application success rate.

use the fewest qubits, but it may cost too many gates to reuse the ancilla qubits. SQUARE finds the balance between qubit uses and gate costs. We show the AQV comparison in Figure 4.7a. The AQV is reduced when we apply LAA that allocates the closest qubits, reducing the number swaps. When full SQUARE is applied, AQV is further reduced because of reduction in uncompute cost.

Program Success Rate by Analytical Model

Program success rates in our evaluation are estimated by a worst-case analysis using qubit decoherences and gate errors. Multiplying the single-qubit/two-qubit gate success rates and the probability of qubit coherence from Table 4.4, we observe an average improvement by

Benchmarks	Policy	# Gates	# Qubits	Circuit Depth	# Swaps
RD53	Lazy	536	19	395	462
	Eager	1064	10	878	633
	SQUARE	932	11	635	370
6SYM	Lazy	648	19	456	654
	Eager	1293	11	1279	1247
	SQUARE	1078	12	731	520
2OF5	Lazy	708	18	723	759
	Eager	1410	8	2374	1728
	SQUARE	1176	10	952	385
ADDER4	Lazy	656	18	787	725
	Eager	1184	12	1139	748
	SQUARE	920	14	715	421
Jasmine-s	Lazy	275	16	232	73
	Eager	1226	5	1055	327
	SQUARE	510	8	427	128
Elsa-s	Lazy	163	15	787	725
	Eager	501	8	438	163
	SQUARE	254	13	223	85
Belle-s	Lazy	220	14	202	69
	Eager	712	6	574	113
	SQUARE	294	9	266	89

Table 4.3: NISQ benchmarks compilation results. Here # Gates does not include swap gates (listed in a separate column).

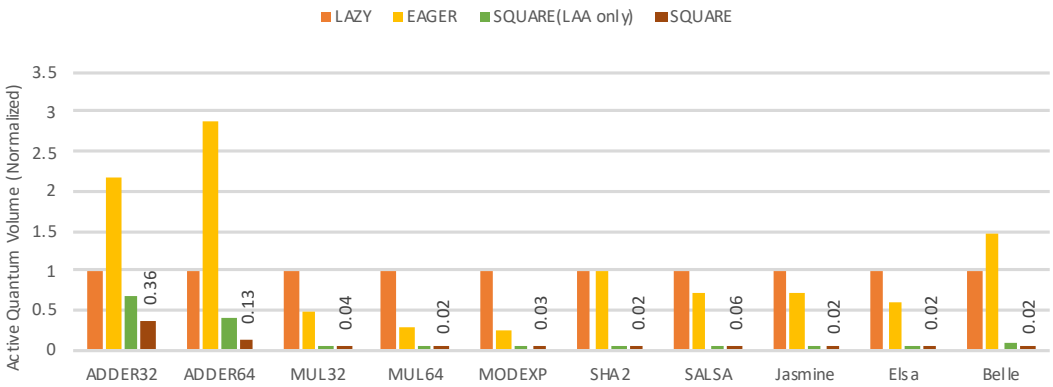


Figure 4.8: AQV results on medium-scale non-error-corrected quantum systems. Numbers on the chart correspond to the normalized AQV values of the SQUARE algorithm.

1.47X w.r.t Eager and 1.07X w.r.t. Lazy. With strategic uncompuptation by SQUARE, programs use fewer qubits and improve overall chance of success. In reality, this worst

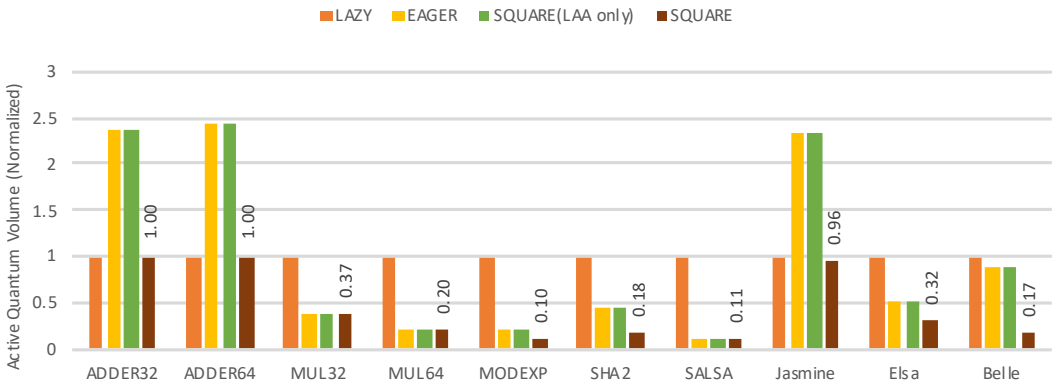


Figure 4.9: AQV results on fault-tolerant quantum systems.

	# Qubits	ϵ_{single}	ϵ_{two}	T1 (μs)	T2 (μs)
IBM-Sup [IBM, C]	20	$< 1\%$	$< 2\%$	55	60
IonQ-Trap [Ion]	79	$< 1\%$	$< 2\%$	$> 10^6$	$> 10^6$
Our Simulation	< 20	0.1%	1%	50	70

Table 4.4: Error rates on real devices and noise models on our simulation.

case analysis may neglect program structures and noise cancellation. Results are even more positive in the next section where we perform noise simulation.

Noise Simulations

All simulations in our evaluation use IBM `Qiskit Aer` simulator [AAA+19] with noise models from the `qiskit.providers.aer.noise` library – `depolarize_noise` for single-qubit and two-qubit gate noises, and `thermal_relaxation` for T1/T2 relaxations to account for qubit decoherence. Table 4.4 shows the parameters in our simulation, compared against those in real devices. Figure 4.7c shows the results from simulation; each data point is obtained from 8192 shots of noisy circuit simulation. We use total variation distance d_{TV}, to compare measurement outcomes of noisy circuits with those of ideal ones; it's a common measure for QC experiments [BOW19, LBR17, AAB+19]. We observe that SQUARE achieves lowest distance for almost all benchmarks compared to Eager or Lazy.

Applicability of SQUARE to NISQ Machines

Table 4.3 and Figure 4.7b together show the impact of uncomputation on circuit fidelity. SQUARE finds a balanced middle-ground between qubit savings and gate costs by strategically uncomputing its functions. Surprisingly, when comparing Lazy with SQUARE, the additional gates for uncomputation *reduces* the total number of operations, thanks to a substantial reduction in swap gates, as ancilla qubits with better locality are actively reclaimed and reused. Uncomputation also dis-entangles garbage qubits from output qubits, preventing noise from propagating. Furthermore, SQUARE retains most of the qubit savings as Eager does. Overall, SQUARE achieves high success rate using fewer qubits than Lazy.

4.6.3 NISQ-FT Boundary Experiments

The boundary between NISQ and fault-tolerant architectures are far from clear. For completeness, we analyzed the performance of the SQUARE algorithm assuming medium-scale machines (with 100-10000 qubits) is built without error correction. Figure 4.8 shows the AQV results with different compiling policies, and the normalized AQVs of SQUARE are labeled. We observe significant AQV savings by SQUARE, reducing the AQV by a factor of 6.9X on average when compared to the Lazy policy.

4.6.4 Fault-Tolerant (FT) Experiments

The FT experiments share the set of benchmarks used in the NISQ-FT experiments, but use braiding for communication. To do so, we build and integrate a braid simulator in SQUARE to precisely calculate the communication overhead for executing a program on a surface-code error-corrected architecture.

Following prior work [FMMC12, JAGH+17, DHJA+18], we assume logical qubits on the surface are laid out in a 2-D array, with sufficient distance between qubits. The separation between qubits serves as channels, allowing other qubits to braid through. So in our simu-

lator, we associate one site per qubit and channels wide enough for a single qubit to braid through. Furthermore, different single-qubit gates have different time cost.

We substitute the swap-chain generation procedure in the SQUARE's gate scheduler with a braid generation procedure. In particular, when a `CNOT` gate is scheduled, we first find a route between the operand qubits, and then check if it crosses with other ongoing braids. It is queued until its route has been cleared.

As shown in Figure 4.9, SQUARE significantly reduces AQV in all applications under the FT system environment. Comparing to Lazy policy, SQUARE achieve 44.08% AQV reduction on average, and up to 89.66% reduction.

4.7 Chapter Summary

State-of-the-art quantum computing systems offer growing memory size (qubit count) and reliability (decoherence time). To realize their full power, however, much attention needs to be paid in the quantum compiling process. Among all compiler optimizations, quantum memory management is one of the places where huge benefits can be obtained. In short, memory management performs optimal qubit allocation to make the best out of a limited set of qubits – e.g., to compute with high success probability and low resource cost. This work demonstrates *how garbage collection can be done systematically on a quantum computer*, by adapting conventional systems techniques (e.g. LLVM instrumentation, register allocation, and thread-level parallelism) to key constraints in quantum systems.

4.7.1 From Theory to Practice

This work is the first system-level quantum memory management paper, and showed the practical value of garbage collection by uncomputation at both NISQ and FT scales. For the first time, the theory of uncomputation is integrated into a full-stack compilation software toolchain, and being optimized jointly across the stack. *Such end-to-end thinking allows us*

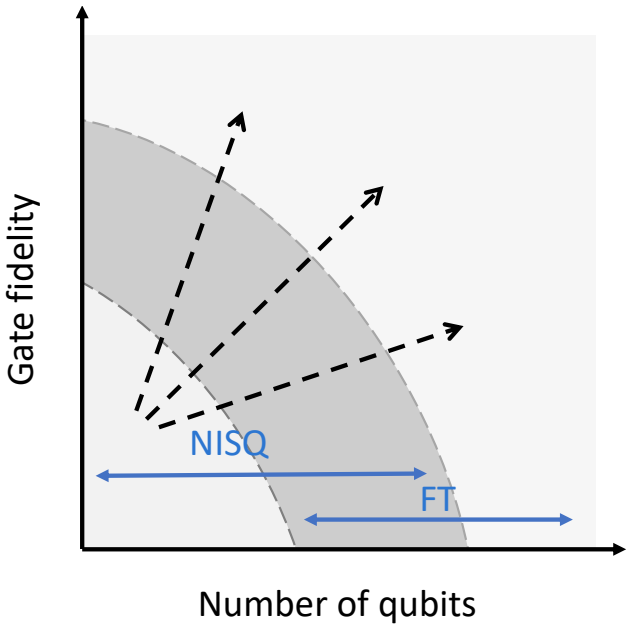


Figure 4.10: Possible paths (dashed arrows) forward from NISQ to FT systems. Architecture designs can guide the course of such paths.

to realize numerous practical implications, which are otherwise neglected. This allows us to accurately evaluate the impact of uncomputation on data locality, program parallelism, and program fidelity, etc.

Our work is likely to alter industry perspectives on best course of quantum hardware and software development. On the hardware side, SQUARE offers insights on how to scale up from NISQ to FT machines. Is increasing qubit count more important than gate fidelity, or vice versa? As shown in Figure 4.10, there are multiple likely paths forward. A full-stack tool like SQUARE is needed to quantitatively study the pros and cons of these paths, since SQUARE can interpolate between qubit saving and gate costs and is amenable to both NISQ and FT machines.

On the software side, there is a growing attention and general interests in quantum memory management, in particular qubit reuse, among leading industry vendors of quantum machines and services. Late this year, Honeywell (a leading trapped-ion quantum device vendor) and QC-ware (a quantum software and service provider) have both published about

the substantial benefits of integrating qubit reuse into the compilation process. Their approach, however, relies on a hardware requirement of mid-circuit measurements which is still challenging and is not always applicable while maintaining correctness in many quantum circuits. We demonstrate that the alternative software solution (with uncomputation) is more general, and practically worthwhile.

CHAPTER 5

FAULT-TOLERANT QUANTUM ARCHITECTURE

5.1 Quantum Error Correction Preliminaries

Quantum error correction (QEC), first developed in [CS96, Ste96b], achieves fault tolerance by repeatedly discretizing continuous errors into digital errors and using many redundant qubits to flag any errors that have occurred to the quantum state, an idea that is not so distant from memory refresh and classical coding theory. It allows us to track and correct errors in real-time while executing a quantum program. QEC is an extraordinary discovery – it not only explains why we can detect and correct a quantum error, but also provides a recipe for doing so systematically. It is a blueprint for how to build a future large-scale quantum computer fault-tolerantly. Since the focus of the book is on near-term NISQ research, we have been postponing discussions on QEC until now. A single section in the book does not justify how remarkable QEC is; nonetheless, we will demonstrate a selection of fundamental concepts in QEC, first via an example then via a generalized principle. For details about quantum error correction, we refer the readers to [Kit97, Got97, DKLP02, Got02, Ter15, NC02].

5.1.1 Basic Principles of QEC

As previously discussed, quantum systems are not ideal. There are many variables that have an impact on the outcome of a computation. The fidelity rates and coherence times are some factors posing challenges. Quantum gate operations and control signals are not perfect. And all of these errors build up to non-negligable amounts. That is why we need a way to correct accumulating errors. This is the motivation behind quantum error correction. Simply put, the purpose of quantum error correction can be summarized as protecting quantum circuits from noise.

Quantum Error vs. Classical Error

Classically, we are using bits, so the information is stored in 0's and 1's. Whenever there is an error, the bit is the opposite of what it is supposed to be (i.e., a bit flip). Because classical errors are just accidental bit flips, they are digitized. However, quantum errors are continuous. This continuous error can be mathematically modeled as follows:

$$|0\rangle \xrightarrow{\text{X gate}} \sqrt{\epsilon}\,|0\rangle + \sqrt{1-\epsilon}\,|1\rangle\,. \tag{5.1}$$

Even though physicists do their best to reduce this effect, it sometimes is not enough. There are a lot of questions that rise from this situation. Are we able to detect and measure how big/small this error ϵ is, or even if we can, is it better to correct it right now or later? One of the hard to questions to answer is at what point do we decide to attempt to correct ϵ?

Key Ingredients in Quantum Error Correction

There are two main ideas that make quantum error correction possible. One idea is to use redundant encoding of information, just like in QR codes. This way, effects of noise in certain parts of the system can be tolerated and will not end up corrupting the state of the system. Another main idea is to digitize quantum errors, since we know how to deal with digitized errors, as they resemble the classical case.

- Redundancy to encode information
- Digitizing quantum error

Quantum Error Correction Code (QECC)

Quantum error correction code is a mapping from k logical qubits to n physical qubits. Here, we must emphasize that n is strictly greater than k, as it takes many physical qubits

to realize one logical qubit. The idea is to use n physical qubits to encode (protect) k qubits of information. Exactly $n - k$ qubits are used for redundancy. This mapping can be shown as follows:

$$|0_L\rangle = |000\rangle \tag{5.2}$$

$$|1_L\rangle = |111\rangle \tag{5.3}$$

In the above example, $|0_L\rangle$ stands for the "logical" $|0\rangle$ state of the qubit, and it is realised by 3 physical qubits. The string 000 and 111 are called the logical *codewords* of the code. Now suppose that with some small probability p, one of the physical qubits flipped, and we got $|001\rangle$. The original "logical" qubit can still be recovered, for example through a majority vote of qubits. We would conclude that the third qubit flipped, and the actual qubit was $|0_L\rangle$.

How to locate a bit flip?

Continuing the above example and representation, locating bit flips can be accomplished by looking at output sequences of a 2-qubit operator. These operators are ZZI and IZZ, and each of the operators act on only one qubit in order. For example, ZZI means a Z gate is applied to both the first and the second qubit and the third qubit is left untouched. Recall that,

$$Z\,|0\rangle = |0\rangle \tag{5.4}$$

$$Z\,|1\rangle = -\,|1\rangle \tag{5.5}$$

Now, for a state $|\psi\rangle$ we can look at what the eigenvalues of these 2-qubit operators are. And if we apply both of these 2-qubit gates consecutively, we can determine which bit flipped. Now suppose that $|\psi\rangle = |100\rangle$. This means,

$$ZZI\,|100\rangle = -\,|100\rangle \tag{5.6}$$

$$IZZ\,|100\rangle = |100\rangle \tag{5.7}$$

The eigenvalues observed (in order) are (-1, +1). This sequence tells us that it's the first qubit that is flipped. For instance, if the second qubit was flipped, we would instead observe a sequence that is (-1, -1). Similarly, we would see (+1, -1) if the third qubit was flipped.

If one wishes to compute the phase flip of a qubit, then all Z gates should be replaced by X gates, and all $|0\rangle$ and $|1\rangle$ should be replaced by $|+\rangle$ and $|-\rangle$. This preserves the stabilizer formalism, as the X gate gives (+1, -1) as eigenvalues when it acts on ($|+\rangle$, $|-\rangle$). Everything else, just remains the same.

Check Matrix Formalism

In the literature, the extension of how to locate bit flips to a more generalised case comes through the check matrix formalism. The idea of a check matrix is to create a set of qubit operations using the stabilizer formalism, with enough permutations sequences of eigenvalues to determine which qubit is flipped. Each row in the check matrix is a gate operation that needs to be applied to the system, and each column is representative of physical qubits. For example, the check matrix formalism for the above example would contain two rows, one for IZZ, and one for ZZI. It would also contain three columns, as there are three physical qubits in that system. The check matrix would be:

$$\begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \tag{5.8}$$

where 1's stand for Z (for bit flip) or X (for phase flip) gates, and 0's for the identity matrix. This matrix shows that first, ZZI must applied, followed by IZZ. A more complicated example where 8 physical qubits are used would be

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \end{pmatrix} \tag{5.9}$$

where we see that a series of 3 gate operations is necessary to encode enough sequences so that one can distinguish which qubit is flipped.

9-qubit Shor Code

Of course, using physical qubits to protect against bit flips would be no use, if one doesn't also protect against phase flips, and vice versa. Unfortunately, each of the physical qubits individually need to also be protected by a second layer of concatenated physical qubits in this case. This gives rise to what is called the 9-qubit Shor Code, a 2-layer, 3x3 physical qubit set that protects against both phase and bit flips and encodes one logical qubit. This way, one layer protects against phase flips and the other against bit flips. The logical qubit encoded this way gives us:

$$|0_L\rangle = \left[\frac{1}{\sqrt{2}}|000\rangle + \frac{1}{\sqrt{2}}|111\rangle\right]^{\otimes 3} \tag{5.10}$$

$$|1_L\rangle = \left[\frac{1}{\sqrt{2}}|000\rangle - \frac{1}{\sqrt{2}}|111\rangle\right]^{\otimes 3} \tag{5.11}$$

In this case, operators to check whether phase flips or bit flips occurred changes. We need 3 sets of bit flip checks, and 2 sets of phase flip checks. These gates are given below:

$$Z_1Z_2, Z_2Z_3 \tag{5.12}$$

$$Z_4Z_5, Z_5Z_6 \tag{5.13}$$

$$Z_7Z_8, Z_8Z_9 \tag{5.14}$$

$$X_1X_2X_3, X_4X_5X_6 \tag{5.15}$$

$$X_4X_5X_6, X_7X_8X_9 \tag{5.16}$$

Projective Measurements

We now describe how to *implement the stabilizer operators* in a quantum circuit. The stabilizer operators (including the example of ZZI in the previous section) are implemented as projective measurements. To see why, we consider the circuit below by calculating the quantum state at each time step in the circuit in Figure 5.1.

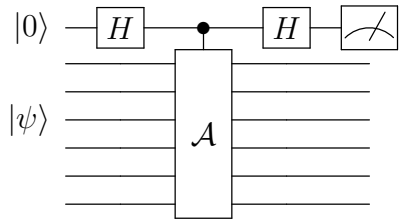


Figure 5.1: The circuit describing a projective measurement for operation A on state $|\psi\rangle$. Here, A can mean any stabilizer n-qubit gate. For example, it can mean IZZ described above. There needs to be an additional ancilla qubit for this process.

1. $|0\rangle |\psi\rangle$
2. $|+\rangle |\psi\rangle$
3. $\frac{1}{\sqrt{2}}(|0\rangle |\psi\rangle + |1\rangle A |\psi\rangle)$
4. $\frac{1}{2}[(|0\rangle + |1\rangle) |\psi\rangle + (|0\rangle - |1\rangle)A |\psi\rangle] = |0\rangle \frac{I+A}{2} |\psi\rangle + |1\rangle \frac{I-A}{2} |\psi\rangle$

where operators $\frac{I+A}{2}$ and $\frac{I-A}{2}$ are called "projectors". It can be shown that any arbitrary state $|\psi\rangle$ can be decomposed into orthogonal states as follows:

$$|\psi\rangle = \alpha |\psi_+\rangle + \beta |\psi_-\rangle , \tag{5.17}$$

where $|\psi_+\rangle$ and $|\psi_-\rangle$ are the eigenstates of A with eigenvalues (+1, -1) respectively. In other words,

$$A\,|\psi_+\rangle = |\psi_+\rangle\,, A\,|\psi_-\rangle = -\,|\psi_-\rangle\,. \tag{5.18}$$

One can think of these states as "no error" and "error" states as well. Since when we have no error, stabilizer operators give us an eigenvalue of +1, and when we have error, it's -1. Therefore, we can see further that:

$$\frac{I+A}{2}(\alpha\,|\psi_+\rangle + \beta\,|\psi_-\rangle) = \alpha\frac{1+1}{2}\,|\psi_+\rangle + \beta\frac{1-1}{2}\,|\psi_-\rangle = \alpha\,|\psi_+\rangle\,, \tag{5.19}$$

which shows us that we recover the original "no error" state $|\psi_+\rangle$ with probability α and the "error" state $|\psi_-\rangle$ with probability β. So if $\alpha\ \sqrt{1-\epsilon}$ and $\beta\ \sqrt{\epsilon}$ where $\epsilon \ll 1$ (i.e., the error is small), we recover the "no error" state with high probability. This procedure shows that "projectors" actually transform the arbitrary state $|\psi\rangle$ into one of the two states, the "no error" and the "error" states.

For instance, measuring the stabilizer operator $X_1X_2X_3$ means that we perform a projective measurement using an ancilla qubit as shown in Figure 5.2.

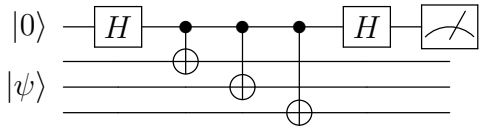


Figure 5.2: Syndrome measurement for the stabilizer operator $X_1X_2X_3$.

5.1.2 Stabilizer Codes

Using the stabilizer formalism defined earlier, we can construct a family of quantum error correction codes, defined by . Thanks to the simplicity in the formalism, we are able to borrow the concepts from linear codes in classical coding theory.

Definition 5.1.1. An $[n, k]$ *stabilizer code* $C(S)$ is defined as the vector space stabilized

by the operators from the abelian subgroup $S = \langle g_1, g_2, \ldots, g_{n-k} \rangle$, where $g_i \in P_n \setminus -I$ is a stabilizer from the n-qubit Pauli group, represented as a length-n Pauli string. k is the number of logical qubits that $C(S)$ encodes.

$$C(S) = \{|\psi\rangle \in \mathcal{H}, \text{s.t. } g\,|\psi\rangle = |\psi\rangle \; \forall g \in S\}$$

The 9-qubit Shor code can therefore be written as a $[9, 1]$ stabilizer code, which uses 9 physical qubits to encode 1 logical qubit where the stabilizers are Equation 5.12-5.16.

With this definition of a quantum error correction code, we have the following theorem showing the set of errors that can be corrected by the stabilizer code.

Theorem 5.1.2. *Given a set of Pauli errors $\mathcal{E}$, if for all $E_i, E_j \in \mathcal{E}$, $\exists g \in S$, s.t. $E_i^\dagger E_j g = -g E_i^\dagger E_j$, then the set of errors $\mathcal{E}$ is correctable by the stabilizer code $C(S)$.*

The proof of this theorem can be found in [NC02]. Effectively, if a Pauli error anti-commute with a stabilizer, then the stabilizer can detect and correct an occurrence of the error. This is because, upon projective measurement of g, we can observe that $E_i\,|\psi\rangle$ is projected to the -1 eigenstate of g, indicating the occurrence of error. The series of projective measurement outcomes are called the *syndrome*. As such, each error will leave a signature in the syndrome. To tell two errors apart, we need their syndromes to be distinct. This process is called *decoding* of the syndromes. We can thus correct the errors appropriately.

5.1.3 Transversality and Eastin-Knill Theorem

Once the error correction code is defined, the next step is to define how to implement logical operations on the codewords of the code fault-tolerantly. After all, we need to prevent errors from propagating through computation. For each error correction code, there is a class of gates whose logical gate operations (i.e., encoded gates) are easy to implement fault-tolerantly, namely the *transversal quantum gates*. To prevent the propagation of error during

a logical operation, we can impose the requirement that each physical gate for the logical gate acts on at most one physical qubit in each of the n-qubit code block using the $[n, k]$ code. For example, if logical Hadamard gate consists of physical Hadamard gate on each of the n physical qubits: $\tilde{H} = \bigotimes_{i=1}^{n} H$, it would be considered as a transversal Hadamard. In the case of stabilizer codes, for example, the logical X (i.e. $|0\rangle_L \rightarrow |1\rangle_L$)and logical Z (i.e. $|1\rangle_L \rightarrow -|1\rangle_L$) operations can be derived by finding the operator $h \in P_n \setminus S$ but commutes with all $g \in S$. Other logical operations are potentially more difficult to implement.

Transversal gates are preferable because the noisy, physical gates are localized in each code block, preventing errors from spreading uncontrollably through computation. However, the *Eastin-Knill theorem* states that no quantum error correction code can transversally implement a universal gate set. So we have to circumvent the theorem using other techniques to implement fault-tolerant quantum gates. We motivate a class of such techniques by the Knill's error correction picture using gate teleportation.

5.1.4 Knill's Error Correction Picture

The Knill's error correction picture differs from conventional error correction in many ways; we highlight one of the differences, namely the concept of error correcting teleportation [Kni05], which generalizes from gate teleportation [GC99b]. In this picture, error correction is combined with logical gate into one step, instead of the conventional syndrome-based scheme discussed earlier. The error correcting teleportation circuit, uses a generalization of the teleportation circuit to use encoded states and encoded gates, as shown in Figure 5.3.

The key observation in Knill's error correction picture is that a stabilizer projection on the encoded qubits before teleportation is equivalent to a stabilizer projection after the teleportation (up to Pauli modification due to the recovery operator at the end of the teleportation circuit) as shown in Figure 5.4.

The argument follows similarly from that of the gate teleportation technique for unitary

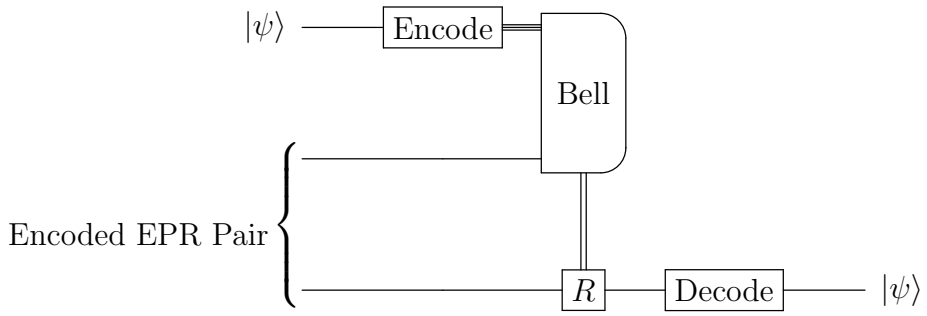


Figure 5.3: An encoded teleportation circuit. The input to the teleportation circuits is the encoded qubits and the encoded EPR pair; the circuit consists of the encoded Bell measurement and encoded recovery Pauli operators.

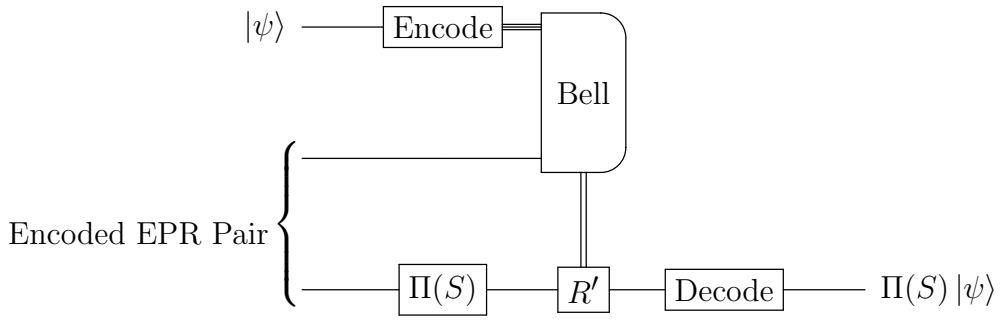


Figure 5.4: Circuit diagram for teleporting a projective measurement $\Pi(S)$. The output of the circuit is $R\Pi(S)\,|\tilde{\psi}\rangle = \Pi(S)R'\,|\tilde{\psi}\rangle$.

gates, but generalized to projective measurements. Consequently, the syndrome of the input state can be deducted from the teleportation Bell measurement [Kni05].

5.2 Motivation: Overheads of Fault Tolerance

Quantum computers of intermediate scale are now becoming a reality. While recent efforts have focused on building Noisy Intermediate-Scale Quantum (NISQ) computers, the long-term goal is to build large-scale fault-tolerant machines [Pre18]. In the context of these machines, typical quantum workloads will be dominated by error correction [TMN$^+$17]. On machines implementing *surface code* error correction, fault-tolerant operations known as *magic-state distillation* will make up the majority of the overhead. The problem is two-fold: 1) useful quantum applications are dominated by magic-state distillation, and 2) their

support is extremely expensive in both physical area and latency overhead. In this paper we are going after *the* obstacle facing large scale quantum computation.

What is required is the preparation (i.e. *distillation*) of high-fidelity logical qubits in a particular state, which can enable the execution of these fault-tolerant instructions. These states require expensive, iterative refinement in order to maintain the reliability of the entire device.

5.2.1 Qubit Overhead: Surface Code Encoding

Quantum states decohere over time which can result in performance loss and failure to produce the correct output. In order to maintain the advantage that quantum computation offers while balancing the fragility of quantum states, quantum error correction codes (QECC) are utilized to protect quantum states undergoing a computation. One of the most prominent quantum error correcting codes today is the *surface code* [DKLP02, Fow12b, JVMF$^+$12a]. These codes are a family of quantum error correcting codes that encode logical qubit states into the collective state of a lattice of physical qubits utilizing only nearest neighbor interactions between qubits designated as *data* and *ancilla* qubits. For a comprehensive introduction see an excellent tutorial in [Fow12b].

An important parameter of the surface code is the *code distance* d. The surface code can protect a logical state up to a specific fidelity P_L, which scales exponentially with d. More precisely, $P_L \sim d(100\epsilon_{in})^{\frac{d+1}{2}}$, where ϵ_{in} is the underlying physical error rate of a system [FDJ13a]. We will use "rotated lattice" surface codes, so that each logical qubit is made up of d^2 physical qubits [HFDVM12a].

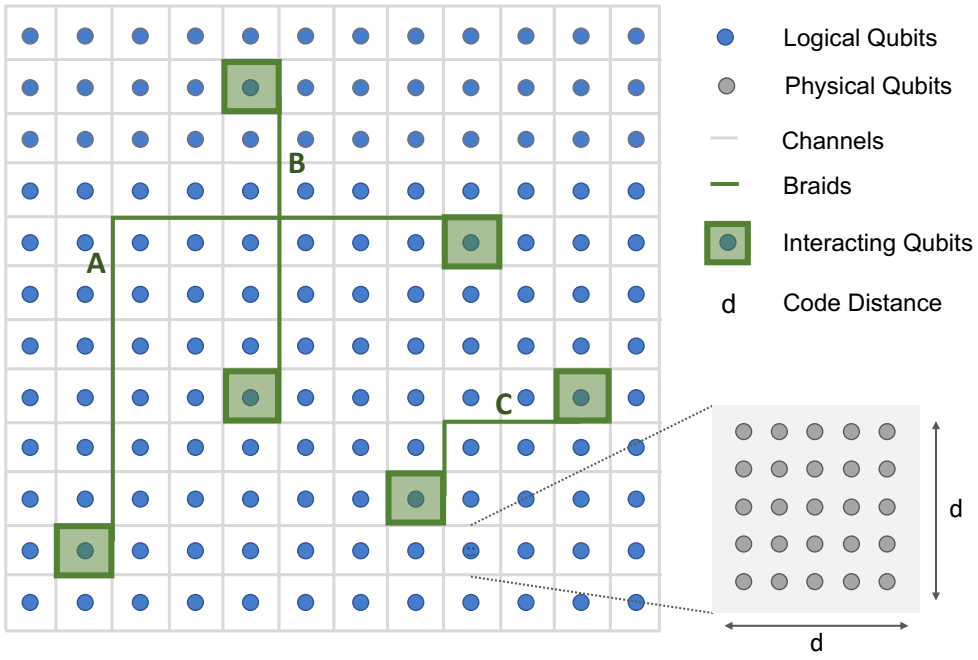


Figure 5.5: An array of (blue) logical qubits in a quantum processor. Highlighted lines indicate *braids* implementing two qubit interactions. These braids must exist spatially and temporally as pathways between qubits. This introduces communication congestion that depends upon specific architectural designs. Braid A and B are *crossing* braids, which cannot be executed simultaneously, while braid C is isolated and free to execute.

5.2.2 Operation Overhead: CNOT and Non-Clifford Gates

CNOT Gate from Qubit Braiding

A *braid* is a path in the surface code lattice, or an area where the error correction mechanisms have been temporarily disabled and where no other operations are allowed to use. In other words, braids are not allowed to cross. In braiding, a logical qubit is entangled with another if the pathway encloses both qubits, where enclosing means extending a pathway from source qubit to target qubit and then contracting back via a (possibly different) pathway. These paths can extend up to arbitrary length in constant time, simply by disabling all area covered by the path in the same cycle.

T Gate from Gate Teleportation

We can use this picture of error correction to motivate a technique that aims to remedy the Eastin-Knill theorem, namely *magic state distillation*. One of the advantages of the error correcting teleportation picture [GC99a, Kni05] is that the difficulty in performing a logical gate U fault-tolerantly on the encoded $|\psi\rangle$ is shifted to the difficulty in preparing an encoded resource state (also known as magic states) fault-tolerantly. The latter is in general easier, because the preparation of specific magic states is generally easier than performing an operation on an unknown state: the magic state can be prepared offline (i.e., prior to the computation), and we can discard a resource state and start over in case that the preparation circuit fails.

Magic state distillation, first proposed by Bravyi and Kitaev [BK05a], is precisely the process for preparing a resource state fault-tolerantly that corresponds to a non-transversal gate for the error correction code.

A widely studied magic state is the resource state for T gate (i.e., $\pi/8$ gate). For example, we can input the following resource state

$$|A\rangle = T|+\rangle = \frac{1}{\sqrt{2}}\left(e^{-i\pi/8}|0\rangle + e^{i\pi/8}|1\rangle\right)$$

to the (single-qubit version) teleportation circuit as shown in Figure 5.6.

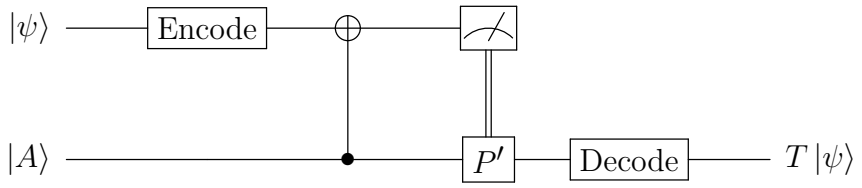


Figure 5.6: The role of magic states in the Knill error correction picture. The magic state $|A\rangle$ is teleported to implement a T gate fault-tolerantly. Here, $P' = TXT^\dagger = XS^\dagger$. All gates (i.e., CNOT gate, P' gate, and measurement) are implemented fault-tolerantly.

As a result, the problem of implementing the non-transversal T gate fault-tolerantly is reduced to the problem of preparing (in advance) high-fidelity magic states.

T rotation gates are important operations in many useful quantum algorithms. When the number of T gates in an application is low, the circuit is in fact able to be efficiently simulated classically [BG16]. T gates have been shown to comprise between 25% and 30% of the instruction stream of useful quantum applications [TMN$^+$17]. Others claim even higher percentages for specific application sets, of between 40% and 47% [IWPK08].

For an estimate of the total number of required T gates in these applications, take as an example the algorithm to estimate the molecular ground state energy of the molecule Fe_2S_2. It requires approximately 10^4 iteration steps for "sufficient" accuracy, each comprised of 7.4×10^6 rotations [WBC$^+$14]. Each of these controlled rotations can be decomposed to sufficient accuracy using approximately 50 T gates per rotation [KMM12]. All of this combines to yield a total number of T gates of order 10^{12}.

As a result, it is crucial to optimize for the resource overhead required by the execution of T gates at this scale to ensure the successful execution of many important quantum algorithms.

5.2.3 Resource Overhead: Magic-State Distillation

T Magic States

T gates, while necessary to perform universal quantum computation on the surface code, are costly to implement under surface code. The number of T gates present in an algorithm is the most common metric for assessing how difficult the algorithm is to execute [Sel13, AMM14]. To achieve fault-tolerance, an ancillary logical qubit must be first prepared in a special state, known as the *magic state* [BK05b]. A distilled magic-state qubit is interacted with the data to achieve the T gate operation, via a probabilistic circuit involving 2 CNOT braids in expectation. For simplicity, because of their rotation angle relationship, we assume all S gates will be decomposed into two T gates.

These ancillary quantum states are called magic states because they simultaneously sat-

isfy two properties: preparing them would yield universal quantum computation, and they can themselves be prepared using other (simpler) quantum operations. Since the task of preparing these states is a repetitive process, it has been proposed that an efficient design would dedicate specialized regions of the architecture to their preparation [Ste97b, JVMF$^+$12a]. These *magic state factories* are responsible for creating a steady supply of low-error magic states. The error in each produced state is minimized through a process called *distillation* [BH12].

Bravyi-Haah Distillation Protocol

Distillation protocols are circuits that accept as input a number of potentially faulty raw magic states, use some ancillary qubits, and output a smaller number of higher fidelity magic states. The input-output ratio, denoted as $n \to k$, assesses the efficiency of a protocol. This work focuses on a popular, low-overhead distillation protocol known as the Bravyi-Haah distillation protocol [BH12, FDJ13a].

To produce k magic states, Bravyi-Haah state distillation circuits take as input $3k + 8$ low-fidelity states, use $k+5$ ancillary qubits, and k additional qubits for higher-fidelity output magic states, thus denoted as the $3k + 8 \to k$ protocol. The total number of qubits involved in each of such circuit is then $5k + 13$, which defines the area cost of the circuit module.

The intuition behind the protocol is to "make good magic states out of bad ones". Given a number of low-fidelity states, the protocol uses a syndrome measurement technique to verify quality, and discards states that are bad. Then, the circuit will convert the subset of good states into a single qubit state. If the filtering and the conversion follows a particular pattern, the output magic states will have a suppression of error. Notably, if the input (injected) states are characterized by error rate ϵ_{inject}, the output state fidelity is improved with this procedure to $(1 + 3k)\epsilon^2_{\text{inject}}$. Due to the filtering step, the success probability of the protocol is, to the highest order, given by $1 - (8 + 3k)\epsilon_{\text{inject}}$.

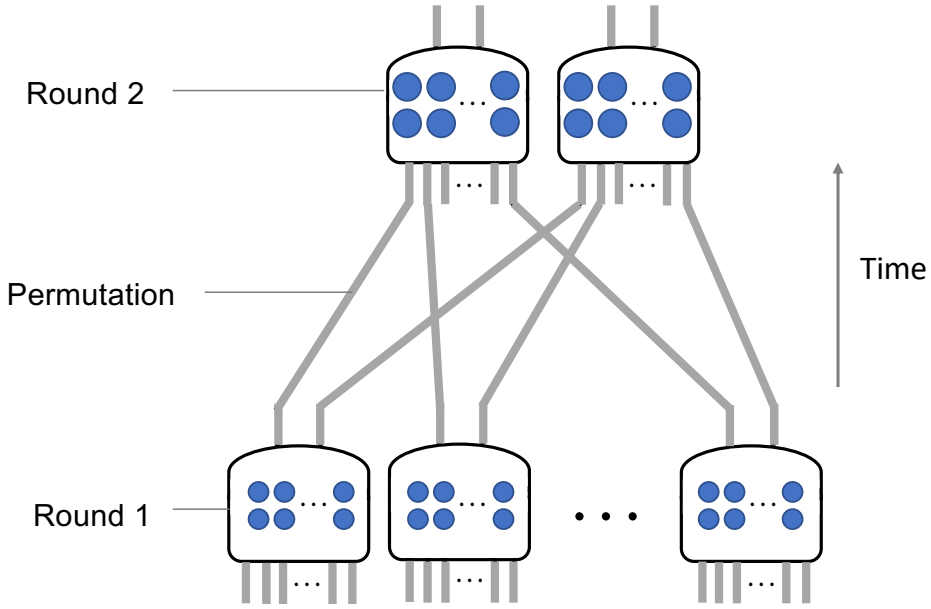


Figure 5.7: The recursive structure of the block code protocol. Each block represents a circuit for Bravyi-Haah $(3k+8) \to k$ protocol. Lines indicate the magic state qubits being distilled, and dots indicates the extra $k+5$ ancillary qubits used, totaling to $5k+13$. This figure shows an example of 2-level block code with $k = 2$. So this protocol takes as input $(3k+8)^2 = 14^2$ states, and outputs $k^2 = 4$ states with higher fidelity. The qubits (dots) in round 2 are drawn at bigger size, indicating the larger code distance d required to encode the logical qubits, since they have lower error rate than in the previous round [OC17].

Block Codes

In order to discover the structure of the state distillation circuits, we need to take a closer look at their construction. Often times one iteration of the distillation procedure is not enough to achieve the desired logical error rate for a given program. In these cases, clearly squaring the input error rate will not achieve the required logical error rate to execute the program. Instead, we can *recursively* apply the Bravyi-Haah circuit a number ℓ times, in order to achieve the desired error rate [Jon13]. Constructing high fidelity states in this fashion is known as *block code* state distillation.

As Figure 5.7 illustrates, ℓ level implementations of this procedure can be constructed recursively that support k^ℓ total output states at fidelity $\sim \epsilon_{\text{inject}}^{2^\ell}$, while requiring $(3k+8)^\ell$

input states.

The structure of the multi-level block code distillation protocol requires that each module takes in at most one state from each module from the previous round. This is because the magic states produced by one module may have correlated errors. So in order to avoid having correlated error in the inputs to the next round, each magic state from one module must be fed into a different module.

At the end of each individual module, error checking is required, and the success of the procedure relies on the correct measurement outcomes in the ancillary states. Additional quality checks were proposed by [OC17], which inserts a checkpoint at the end of each level of the factory. This checkpoint discards groups of modules when it detects failure within any of the modules in a group.

Within any particular round r of an ℓ-level magic state factory, the number of physical qubits required to implement that round defines the *space* occupied by the factory during round r. Because the output error rates will lower after each round, the required code distance will increase accordingly. By the "balanced investment" technique shown in [OC17], each logical qubit in round r will be constructed using $\sim d_r^2$ physical qubits, where each d_r varies with each round. The idea is to use a smaller code distance to encode a logical qubit in earlier rounds of distillation to minimize area overhead.

In general, any particular round requires several modules, each comprised of several Bravyi-Haah circuits implementing a protocol. Defining the number of modules and number of protocols per module as n_ℓ, k_ℓ, we can see that the number of physical qubits q_r required to implement a round r scales exponentially with total number of levels ℓ as: $q_r = k_\ell^{r-1} n_\ell^{\ell-r}(5k+13)d_r^2$.

5.3 Towards A Hierarchical Resource-Based Architecture

The following study is based on a joint work with Adam Holmes, who contributed equally to this project.

In order to minimize the quantum volume (space×time) spent on multilevel magic state distillation, our approach is to take advantage of the unique characteristics of the state distillation circuitry, and to decompose the problem into two aspects - scheduling gate operations and mapping qubits into 2-D mesh. These two are intertwined, as the schedule determines what pairs of qubits *need* to interact, and mapping influences which subset of them *can* interact in the same cycle. An important tool used to perform these optimizations is the program interaction graph, from which circuit structure can be extracted. In particular, we combine the fact that these state distillation circuits are characterized by natural subdivisions between levels of the factory, with the ability of graph partitioning embedding techniques to nearly-optimally map small planar subgraphs of the program. We exploit this information to design a procedure that decomposes state distillation into components that are independently optimized.

Levels of the factory are joined by a specific permutation of the output states exiting from previous rounds of distillation, which appears to impose significant overhead on the whole distillation process. To address this, a force-directed annealing algorithm is used in conjunction with ideas inspired by Valiant intermediate-destination routing for permutation networks [LPV81] to reduce the latency of these permutation steps between block code levels.

The next few sections describe the scheduling and mapping optimizations decoupled from one another, in order to show the specific strengths and weaknesses of each. Section 5.4 then synthesizes these optimizations into a single procedure.

5.3.1 Error Correction Procedures: Scheduling and Mapping

This section describes the impact of instruction level optimizations: gate scheduling and qubit reuse. A *schedule* of a quantum program is a sequence of gate operations on logical qubits. The sequence ordering defines *data dependences* between gates, where a gate g_1 depends on g_0 if they share a logical qubit and g_1 appears later in the schedule sequence than g_0.

Gate Scheduling

The impact of gate scheduling can be quite significant in quantum circuits, and many algorithm implementations rely upon the execution of gates in parallel in order to achieve substantial algorithmic speedup. Gate scheduling in quantum algorithms differs from classical instruction scheduling, as gate commutativity introduces another degree of freedom for schedulers to consider. Compared to the field of classical instruction scheduling, quantum gate scheduling has been relatively understudied, with only few systematic approaches being proposed that incorporate these new constraints [GP17].

In exploring these effects applied to Bravyi-Haah state distillation circuits, we find that these optimizations are limited in their effectiveness. While intuitively the modularity of the block code construction would allow for early execution of gates arising in late rounds of the distillation procedure, the *checkpoints* required to implement module checking as described in section 5.2.3 limit the magnitude of gate mobility.

The structure of the block code circuitry only allows for a small constant number of gates to by executed early, outside of the rounds from which they originate. Because of this, the maximum critical path extension by the introduction of a *barrier* preventing gate mobility outside of the originating round is equal to this small constant multiplied by the number of block code iterations. Barriers in these circuits can be inserted by adding a multi-target CNOT operation into the schedule, controlled by an ancilla qubit initialized into a logical

$|0\rangle$ state, and targeting all of the qubits that the schedule wishes to constrain.

Additionally, gate scheduling order has significant impacts on network congestion. Scheduling these small constant number of gates early therefore runs the risk of causing congestion with previous round activity. This can in fact extend the circuit latency, even though the circuit has executed gates earlier in the schedule.

Overall, the insertion of a barrier appears to not significantly alter the schedule of circuit gates. It does, however, change the interaction between the schedule and a particular physical qubit mapping. This relationship will be explored in more detail in section 5.4.

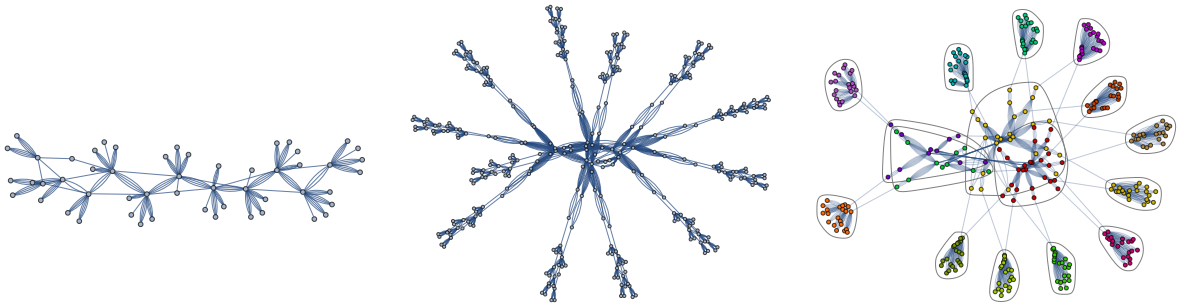
Qubit Reuse

We show in this section that an important schedule characteristic of the block protocol to leverage is the hierarchical structure of the distillation circuit. Between two rounds of the procedure, all ancillary qubits will be measured for error checking at the end of the previous round, and reinitialized at the beginning of the next round. This type of data qubit sharing (which we call "sharing-after-measurement") is a *false dependency*, because they can be resolved by qubit renaming. Now this naturally leads to the question: (how) should we reuse the qubits between multiple rounds?

The first approach we explore is to prevent any false sharing of the qubits, at the cost of larger area, by always allocating new data qubits for different rounds. This removes all dependencies due to ancillary qubits, leaving only true dependencies on qubits generated in the previous round. This minimizes execution time at the cost of extra qubits (and space).

The second approach is to strategically choose which qubits from the previous round to be reused for the next. This approach directly reduces the area needed for the entire factory, at the cost of introducing false dependencies.

In order to make intelligent decisions on which set of ancillary qubits to reuse, it requires us to have information about the topological mapping of the qubits, since mapping and



(a) Planar interaction graph of a capacity 8, single level factory

(b) Non-planar interaction graph of a capacity 4, two level factory

(c) Multi-level factory interaction graph with community structure

Figure 5.8: Interaction graphs of single and two level factories, and community structure of a capacity 4 two level factory. Each vertex represents a distinct logical qubit in the application, and each line represents a required two (or more) qubit operation. (a) shows that the single level distillation circuit has planar interaction graph, so mapping vertices to physical location in quantum processor is relatively simple. Each level in a multi-level factories like (b) have these planar substructures, but the permutation edges between rounds destroy the planarity of the two-level interaction graph. (c) shows that we can leverage the planarity within each level by exploring community structure of the interaction graph, as shown in section 5.3.1

reuse decisions together significantly influence the congestion overhead of the circuit. We will discuss the subtleties of the tradeoff in more detail later in Section 5.4.

Mapping

This section describes the impacts of qubit mapping decisions on the overall circuit overhead. Given a schedule we can define a *program interaction graph* as a graph $G = (V, E)$ where V is a set of logical qubits present in the computation, and E is a set of two-qubit interaction gates contained in the program (e.g. CNOT gates). By analyzing this graph, we can perform an optimized *mapping*, which assigns a physical location for each logical qubit $q \in V$.

Two examples of these graphs are provided in Fig. 5.8a and Fig. 5.8b. These interaction graphs depict a single level and a two level factory, respectively, and distinct graph properties are available to analyze for each. For instance, the single level factory is a planar graph, and while the two level factory is constructed using many instances of the same single level

factory, the requirement for states to be permuted between levels breaks the planarity of the resulting interaction graph. This has significant consequences, and we will leverage them in section 5.4.

In order to execute state distillation most efficiently, we must minimize both the area required for the factory as well as the latency required to execute the circuit. Braid operations, while latency insensitive, still cannot overlap with one another. If an overlap is unavoidable, then one operation must stall while the other completes. As a consequence, we aim to minimize the number of these braid "congestions".

5.3.2 Motivating Example: Gate Congestion

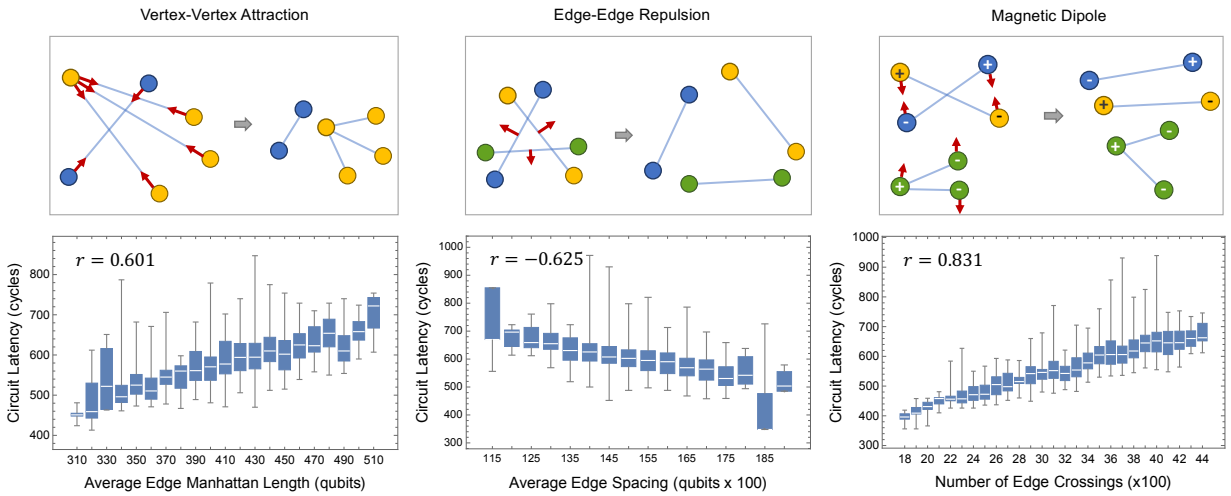


Figure 5.9: Depiction of the heuristics and procedures used in our stitching method. From left to right, edge length is minimized by vertex-vertex attraction, edge spacing is minimized by repulsion forces on the midpoints of edges, and edge crossings are minimized by applying rotational forces to edges emulating a magnetic dipole moment. For each heuristic, the correlation coefficient (r-value) is calculated across a series of randomized mappings of a distillation circuit, and latency is obtained through simulation, shown in bottom figures. The r-values of heuristics with latency are $r = 0.601$, -0.625, and 0.831, respectively. The underlying intuition is that shorter edge length, larger edge spacing and fewer edge crossings will result in fewer braid conflicts and shorter overall latency.

Three common heuristics which we will analyze for minimizing network congestion are: edge distance minimization, edge density uniformity, and edge crossing minimization. We

see that they each correlate in varying degrees with actual circuit latency overhead for these quantum circuits, as shown in Fig. 5.9.

Edge Distance Minimization

The edge distance of the mapping can be defined as the Euclidean distance between the physical locations of each endpoint of each edge in the interaction graph. Intuitively, in classical systems network latency correlates strongly with these distances, because longer edges require longer duration to execute. As discussed in section 5.1, by using surface code braiding operations, there is no direct correspondence between single edge distance and single edge execution latency. However, longer surface code braids are more likely to overlap than shorter braids simply because they occupy larger area on the network, so minimizing the average braid length may reduce the induced network congestion.

Edge Density Uniformity

When two edges are very close to each other, they can share the same channel on the network and cause congestion. Ideally, we would like to maximize the spacing between the edges and distribute them on the network as spread-out and uniformly as possible. This heuristic therefore aims to maximize the spacing between braid operations across the machine.

Edge Crossings

We define an edge crossing in a mapping as two pairs of endpoints that intersect in their geodesic paths, once their endpoint qubits have been mapped. These crossings can indicate network congestion, as the simultaneous execution of two crossing braids could attempt to utilize the same resources on the network. While this heuristic is tightly correlated with routing congestion, minimizing it has been shown to be NP-hard and computationally expensive [GJ83]. An edge crossing in a mapping also does not exactly correspond to induced

network congestion, as more sophisticated routing algorithms can in some instances still perform these braids in parallel [CMS11]. Some algorithms exist to produce crossing-free mappings of planar interaction graphs, though these typically pay a high area cost to do so [Sch90].

Fig. 5.9 summarizes the correlations between each of these three metrics on surface code circuit latency.

5.3.3 Optimizing Performance

With these metrics in mind, we explore two procedures designed to optimize mappings. First, we employ a local, force-directed annealing optimization technique designed to transform the optimized mappings of Fowler [FDJ13a] discussed in section 5.6, specifically targeting optimization of the aforementioned heuristics. Next, we compare this to a mapping procedure based upon recursive graph partitioning and grid bisection embedding.

Force-Directed Annealing

The full force-directed (FD) procedure consists of iteratively calculating cumulative forces and moving vertices according to these forces. Vertex-vertex attraction, edge-edge repulsion, and magnetic dipole edge rotation are used to calculate a set of forces incident upon each vertex of the graph. Once this is complete, the annealing procedure begins to move vertices through the mapping along a pathway directed by the net force calculation. A cost metric is used to decide whether or not to complete a vertex move, as a function of the combination of these heuristics. The algorithm iteratively calculates and transforms an input mapping according to these force calculations, until convergence in a local minima occurs. At this point, the algorithm alternates between higher level *community* structure optimizations that either repulse all nodes within distinct communities away from one another, or attract all nodes within a single community together, which breaks the mapping out of the local min-

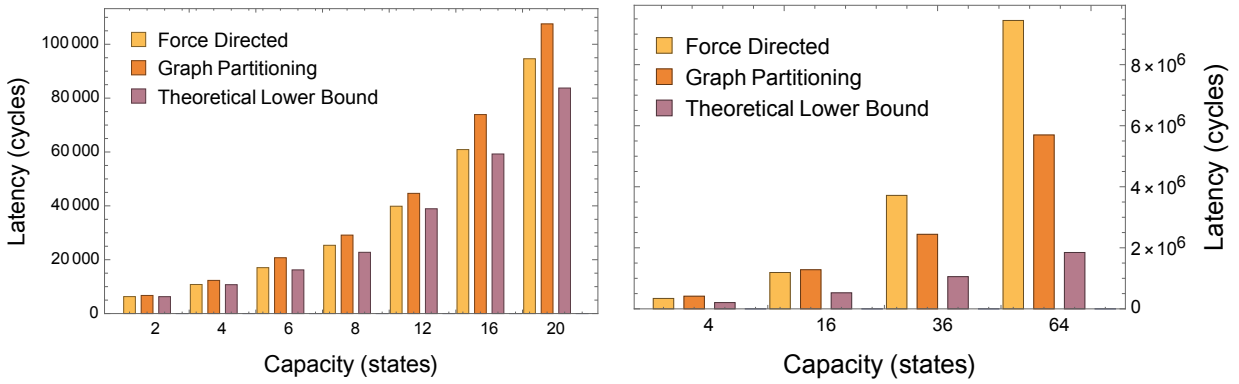
imum that it has converged to. This procedure is repeated until reaching a pre-specified maximum number of iterations.

Within an interaction graph, subsets of qubits may interact more closely than others. These groups of qubits can be detected by performing *community detection analysis* on an interaction graph, including random walks, edge betweenness, spectral analysis of graph matrices, and others [DH73, GN02, Fie73, Fis84, BGLL08, DA05]. By detecting these structures, we can prioritize embeddings that preserve locality for qubits that are members of the same community, thereby reducing the average edge distance of the mapping and localizing the congestion caused by subsets of the qubits.

To minimize the overall edge distance of the mapping, the procedure calculates the *centroid* of each vertex by calculating the effective "center of mass" of the neighborhood subgraph induced by this vertex, i.e. the subgraph containing only the vertices that are connected to this vertex, along with the corresponding edges. The center location of this set is calculated by averaging the locations of all of the neighbors, and this is assigned as the centroid for this vertex. This creates an attractive force on this vertex that is proportional in magnitude to the distance between the vertex and the centroid, and forces of this type are standard in graph drawing techniques [Hu05].

In an attempt to optimize and uniformly distribute the edge density of the mapping, repulsion forces are created between each pair of distinct edges on the graph. For all pairs of edges, repulsion forces are created on their endpoints of magnitude inversely proportional to the square of the distance between the midpoints of the edges. This force law is reflected in many typical graph drawing techniques as well, that aim to uniformly distribute graph vertices and edges [LY12, FR91].

Even though directly minimizing edge crossings in a graph is in general a difficult task to perform, we can approximate it by modeling each edge as a magnetic dipole moment, and the rotational forces applied on each edge will prefer (anti-)parallel orientations over intersecting



(a) In single-level factories, both techniques can nearly optimally execute these circuits, even as capacity increases.

(b) In two-level factories, the difference between the theoretical lower bound and the attained circuit latencies widens.

Figure 5.10: Overall circuit latency obtained by graph partitioning embedding on single and two level distillation factories. Theoretical lower bounds are calculated by the critical path length of the circuits, and may not be physically achievable.

ones, as shown in Fig. 5.9. North and south poles are assigned to every vertex in the graph, and attractive forces are created between opposing poles, while repulsive forces are added between identical poles. The assignment of the poles is done by producing a 2-coloring of the interaction graph. Notice that the graph is not always 2-colorable, and it usually is not. However, within each time step in the schedule, a vertex (qubit) can have degree at most 2, and is always acyclic. This is because we have a schedule that contains only 2-qubit gates and single-control multi-target CNOTs. Any two gates cannot be performed on the same qubit simultaneously, and the multi-target CNOTs will look like a vertex-disjoint path.

To respect the proximity of the vertices in a community that we detected using algorithms described earlier, we break up our procedure into two parts: firstly, impose a repulsion force between two communities such that they do not intersect and are well separated spatially; secondly, if one community has been broken up into individual components/clusters, we join the clusters by exerting attracting forces on the clusters. In particular, we use the KMeans clustering algorithm [KMN$^+$02, AV07] to pinpoint the centroid of each cluster within a community and use them determine the scale of attraction force for joining them.

Recursive Graph Partitioning

To compare against the local force-directed annealing approach, we analyzed the performance of global grid embedding technique based upon graph partitioning (GP) [KL70, Bar82, KK00]. In particular, we utilized a recursive bisectioning technique that contracts vertices according to a heavy edge matching on the interaction graph, and makes a minimum cut on the contracted graph. This is followed by an expanding procedure in which the cut is adjusted to account for small discrepancies in the original coarsening [KK95, PR96]. Each bisection made in the interaction graph is matched by a bisection made on the grid into which logical qubits are being mapped. The recursive procedure ultimately assigns nodes to partitions in the grid that correspond to partitions in the original interaction graph.

The primary difference between these two techniques is that the former force-directed approach makes a series of local transformations to a mapping to optimize the heuristics, while the graph partitioning approach can globally optimize the metrics directly.

Scalability Analysis

Let's now compare the computational complexity of the two graph optimization procedures. Suppose we have an interaction graph of n vertices and m edges. Each iteration of the force-directed annealing procedure consists of three steps, vertex attraction, edge repulsion, and dipole moment rotation. In the worst case, the attraction forces are computed along each edge in $O(m)$ time; the repulsion force computation requires $O(m^2)$ time; rotations are calculated first by a DFS-style graph coloring and then by forces between vertices with $O(n^2)$.

Graph partitioning requires recursively finding minimum weight cut, and partition the graph along the cut. Specifically, it requires $\log_2(n)$ recursive iterations, each of which is a min-cut algorithm on partitions of the graph that requires $\mathcal{O}(n+m)$ time, combining to $\mathcal{O}((n+m)\log_2(n))$ [KK95].

Performance Comparison

Figures 5.10a and 5.10b indicate that, while both techniques perform well for single level factories, the global technique is much better at optimizing higher level factories. This is likely due to the local nature of the force-directed procedure, which is being used to transform the linear hand-optimized initial mapping of the factory. For higher level factories, this hand-optimized mapping incurs high overheads, and the local optimizations are only able to recover a portion of the performance proportional to the original mapping.

It is important to note that while the global graph partitioning technique works well in comparison with the local procedure, there is a widening performance gap between the resulting mapping and the critical resource volume, as factories grow in capacity and levels. This likely indicates that while the procedure is able to very effectively optimize small planar graphs, it has a more difficult time as the size and complexity of the graphs increase. In fact, single level factories have planar interaction graphs, and graph partitioning is able to optimize the mapping of these graphs nearly up to critical resource volume.

5.4 Case Study: Implementing Multi-Round Distillation Protocols

We here present the outline of the iterative, synthesized optimization procedure that make use of the scheduling and mapping techniques we established earlier. To take advantage of the facts that most global optimization techniques (such as graph partitioning and force-directed annealing) work well on small planar graphs and that the circuit modules within each round form disjoint planar subgraphs, we develop a stitching scheme, as depicted in Fig. 5.11, that respects the hierarchical structure and internal symmetry of the multilevel block protocol while simultaneously optimizing for the previously discussed congestion heuristics.

As shown in Fig. 5.11, we perform optimizations iterativaly on the interaction graph.

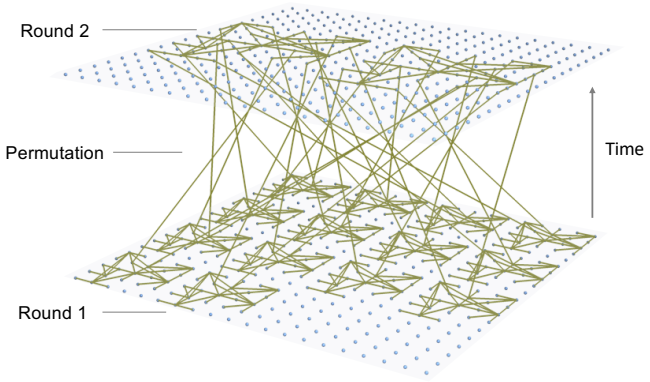
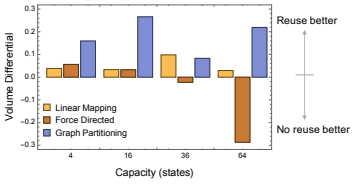


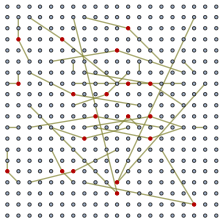
Figure 5.11: Embedding for a capacity $K = 4$, level $L = 2$ factory. The stitching procedure optimizes for each round to execute at nearly critical path length in latency, and optimizes for inter-round permutation step with force-directed optimizations.

Procedure	K	Linear	FD	GP
Latency	4	1.066	1.046	0.931
	16	1.203	1.203	0.913
	36	1.132	1.277	1.145
	64	1.216	1.611	0.978
Area	4	0.203	0.903	0.903
	16	0.804	0.804	0.804
	36	0.801	0.801	0.801
	64	0.798	0.799	0.799
Volume	4	0.964	0.944	0.841
	16	0.967	0.967	0.734
	36	0.906	1.023	0.917
	64	0.971	1.287	0.781

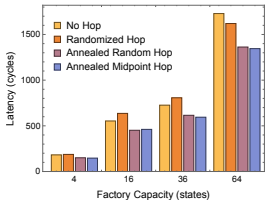
(a) Resource ratios comparing qubit reuse to non-reuse protocols



(b) Volume differentials between qubit reuse and non-reuse protocols



(c) Red dots show optimized intermediate destinations.



(d) Permutation latencies by no midpoint, Valiant, or annealed midpoints.

Figure 5.12: (a)-(b): Sensitivity of achievable quantum volumes by different optimization procedures. Shown is the percentage difference of the protocol with or without reusing qubits. Notably, reuse policy is a better for both the linear mapping and graph partitioning techniques, while no-reuse offers more flexibility for force-directed procedure to optimize. (c)-(d): Circuit latency specifically for the inter-round permutation step. Latency is reduced by 1.3x with Valiant-style intermediate destinations for each interaction, and using force-directed annealing to optimize their locations.

In each iteration, our procedure is decomposed into two phases: (1) *inter-round* optimization that embeds and concatenates each module in the current round, and (2) *intra-round* optimization that stitches permutation edges and arranges modules in the next round.

5.4.1 Intra-Round Graph Concatenation

Starting with the first round of a multilevel factory, we use single-level optimization techniques (such as force-directed annealing or graph partitioning) to nearly optimally embed the *individual* planar modules, each of which is then *concatenated* together, to form a full mapping of the first round of the factory circuitry. The concatenation scheme works well due to the fact that modules in a round do not interact with each other under block code protocol. Notice that putting barrier between rounds enables us to isolate and individually optimize for each round, as discussed in Section 5.3.1. Because the modules in each round of the factory are identical in schedule to those in all other rounds, the optimized graph partitioning embedding does not need to change for each round.

5.4.2 Inter-Round Permutation Optimization

The recursive block code structure requires that the output from lower levels of the factory be permuted and sent to new locations for the subsequent rounds. This can create highly-congested "permutation steps" in the circuit, where even though each round is scheduled and mapped nearly optimally, the cost to permute the outputs of one round to the inputs of the next round are quite high. We therefore present the following sequence of procedures that target the inter-round communication/permutation overhead.

Qubit Reuse and Module Arrangement

Notice that the permutation edges in between two rounds are due to communications between the *output* qubits from the previous round and the *input* qubits in the next round. Given an optimal layout of the modules/blocks from the previous round, we know where the output states are located. Since all qubits except for the outputs are measured, error-checked, and then reinitialized by the time the next round starts, we have the freedom of choosing which regions of qubits to be reused for the next round, as long as for each module

the following constraints are satisfied: (1) do not overlay a module on top of output qubits that are not supposed to be permuted to this particular module (see details about port assignment in 5.4.2), and (2) allocate enough qubits required by the code distance as discussed in 5.2.3. Figures 5.12a and 5.12b show that reusing qubits benefits the linear and graph partitioned mapping techniques, while force-directed annealing prefers the flexibility added by not reusing qubits.

Port Reassignment

To avoid having correlated error in the inputs to the next round, each module in the next round must gather input states from different modules in the previous round, as shown in 5.2.3. Suppose one module from the next round wants a magic state from a previous-round module, when there are multiple outputs produced in that module, it does not matter which one you choose. Therefore, it leaves the optimization procedure to decide which output port to use, so as to minimize congestions in the permutation step.

Intermediate Hop Routing

Lastly, we employ a variation of the force-directed annealing algorithm from Section 5.3.1. Specifically, we introduce *intermediate destinations* between each output state from a prior round and the input state to the next round, as depicted in Fig. 5.12. While Valiant routing with randomized intermediate destinations does not increase performance very significantly, we are able to use force-directed annealing based upon edge distance centroids, edge repulsion, and edge rotations in order to move the intermediate destinations into preferable locations.

This synthesized procedure is able to leverage the scheduling techniques of barrier insertion, combined with nearly optimal planar graph embedding performed by recursive graph partitioning, and force-directed annealing to obtain a significant resource reduction over any

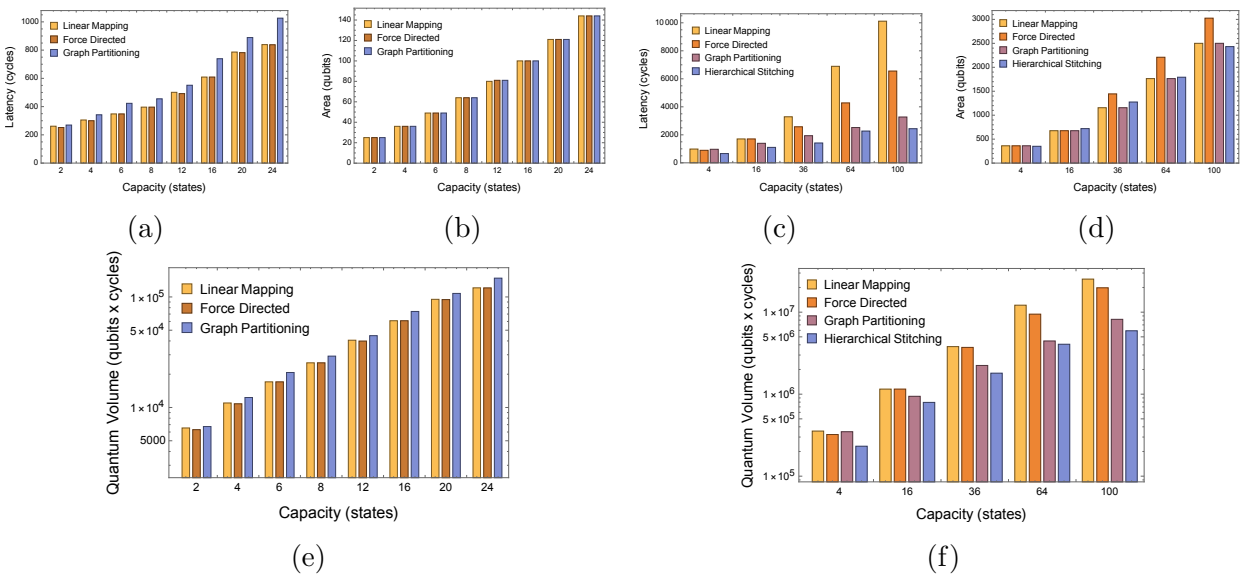


Figure 5.13: One and two level factory resource requirements. Presented are single level factory latencies 5.13a, areas 5.13b and achieved quantum volumes 5.13e, in addition to-two level latencies 5.13c, areas 5.13d and volumes 5.13f. All three optimizations are effective for reducing the overhead of single level factories. For two level factories, each procedure trades off space and time separately, resulting in the lowest achievable volume by that procedure. Hierarchical stitching is able to reduce overheads by 5.64x.

other optimization procedures.

5.5 Performance Results

5.5.1 Evaluation Methodology: Simulation Environment

To perform evaluation of our methods, we implemented each configuration of full Bravyi-Haah distillation protocol in the Scaffold programming language [J+12], and compiled this to gate-level instructions. These instructions are fed into a cycle-precise network simulator [JAGH+17] that accurately executes the scheduling and routing of braids on a 2-dimensional surface-code qubit mesh. We extended both of these tools to support a multi-target CNOT gate. The simulator first schedules braids in parallel where the program-dependency graph allows, and if braids intersect on the machine, the simulator inserts a stall to allow one

			Level 1		
Procedure	$K = 2$	4	8	10	24
Random	1.11×10^4	1.82×10^4	5.43×10^4	6.40×10^4	2.70×10^5
Line(NR)	6.53×10^3	1.10×10^4	2.53×10^4	2.94×10^4	1.29×10^5
Line(R)	6.53×10^3	1.10×10^4	2.53×10^4	2.94×10^4	1.29×10^5
FD	6.30×10^3	1.08×10^4	2.53×10^4	2.88×10^4	1.21×10^5
GP	6.73×10^3	1.23×10^4	2.91×10^4	3.33×10^4	1.48×10^5
HS	–	–	–	–	–
Critical	6.28×10^3	1.07×10^4	2.27×10^4	3.03×10^4	1.12×10^5
			Level 2		
Procedure	$K = 4$	16	36	64	100
Random	–	–	–	–	–
Line(NR)	3.68×10^5	1.19×10^6	4.19×10^6	1.25×10^7	3.34×10^7
Line(R)	3.55×10^5	1.15×10^6	3.80×10^6	1.22×10^7	2.53×10^7
FD	3.22×10^5	1.15×10^6	3.72×10^6	9.45×10^6	1.98×10^7
GP	3.48×10^5	9.41×10^5	2.24×10^6	4.45×10^6	8.17×10^6
HS	2.32×10^5	7.93×10^5	1.80×10^6	4.06×10^6	5.93×10^6
Critical	1.82×10^5	4.48×10^5	8.85×10^5	1.53×10^6	2.43×10^6

Table 5.1: Quantum volumes required by factory designs optimized by: randomization (Random), linear mapping (Line) with and without qubit reuse (R, NR), force-directed (FD), graph partitioning (GP), and hierarchical stitching (HS).

braid to complete before the other. To perform scheduling, the simulator treats any data hazard (i.e. the presence of the same qubit in consecutive instructions) as a true dependency. This eliminates gate-level optimizations from being automatically performed, but it simultaneously allows for the introduction of "barrier" type gates. These are implemented by inserting a single gate involving all qubits of the machine (specifically a multi-target CNOT operation controlled on an extra qubit set to the zero state, and targeting all other qubits of the machine). Gate-level optimizations involving the commutativity relations of specific operations are performed by hand, independent of scheduling.

5.5.2 Single-Level Factory Evaluation

The realistic circuit latency as executed in simulation, required circuit area, and corresponding quantum volume for single level magic state distillation factories are shown in figures

5.13a, 5.13b, 5.13e. We notice first that the linear mapping procedure [FDJ13a] performs well, even as the capacity of the factory increases. In Fig. 5.10a, we see that the linear mapping technique actually is able to approach the theoretical minimum required latency for each of these circuits. These mappings were specifically designed to optimize for these single level factories, which justifes these scaling properties.

Our proposed force-directed mapping approach described in section 5.3.3 is able to improve slightly from the linear mapping technique in most cases. This is due to the correlation between the metrics that the approach optimizes, and the realized circuit latency, as depicted in Fig. 5.9.

Graph partitioning techniques described in 5.3.3 underperform the linear mapping and force directed procedures, although they are still competent with respect to the theoretical minimum resource requirements. Because of the simplicity of the circuit and the targeted optimizations that were perform specifically for these small circuits, the advantage from the global nature of the graph partitioning method is significantly diminished.

It is important to note that the best performing approach for each of these single level factories is able to very closely approximate the theoretical minimum latency and quantum volume required by these circuits. This is leveraged by our iterative procedure and used to ultimately achieve the most efficient circuit expression.

5.5.3 Multi-Level Factory Evaluation

Effects of Qubit Reuse Protocols

As anticipated, the by electing to reuse qubits for later rounds in the distillation circuits, the overall circuit consumes less area at the cost of higher latency. This results in an achieved volume that tends to favor qubit reuse for both the linear mapping and graph partitioning optimization methods.

Notice how the force directed procedure actually achieves a lower volume when qubits are

not explicitly reused. This is due to two factors: first, the average degree of the interaction graph has increased due to the introduction of false dependencies. This restricts the optimization procedure from being able to minimize the heuristics cleanly, as each qubit is more tightly connected to others, reducing the degrees of freedom in the graph. Second, there is more area in the graph, which widens the search space available for the procedure. With more possible configurations, the algorithm is more likely to find more optimized mappings.

Optimization Procedure Comparison

Shown in Fig. 5.13 are the minimal attainable volumes achieved by each optimization procedure. While the linear mapping and force-directed procedures were able to nearly optimally map single level factories, the performance deteriorates significantly when moving to multi-level factories. In these factories, Hierarchical Stitching is able to outperform the other optimization strategies, as it synthesizes the best performing components of each.

We also considered both qubit reuse and non-reuse policies. The optimal combinations vary slightly for each procedure: the linear mapping and graph partitioning strategies always perform best with qubit reuse, while the force directed procedure performs best with qubit reuse for capacity 4 and 16 two level factories, and without qubit reuse for capacity 36 and beyond. This is due to the additional degrees of freedom that avoiding qubit reuse injects, as discussed above. The final results plots show these configurations.

In moving to multi-level factory circuits, even though there is significant modularity and symmetry in the factory, the introduction of the output state permutation from one level to the input states of the next introduces severe latency overheads. Without taking this into consideration, the linear mapping procedures suffer from large latency expansions in attempting to execute multi-level circuits, with the effect compounding as the size (output capacity) of the factory increases. Fig. 5.13f shows that the force-directed approach is able to improve to a maximum reduction of ~ 1.27x from these linear mappings, but is constrained

by how poorly these mappings originally perform.

The graph partitioning technique is able to simultaneously optimize for the entirety of the multi-level circuit, including the inter-round communication steps. With all of this information, the technique is able to minimize interaction graph edge crossings and edge lengths, which results in a more efficient expression of the circuits overall for larger two level circuits. Smaller two level circuits are still dominated by the intra-round execution overheads, which are able to be effectively minimized by linear mapping and force directed techniques. Once multi-level factories become large enough (occurring in Fig. 5.13f at capacity 16), the inter-round effects begin to dominate. This is the point when graph partitioning techniques are able to outperform other methods.

The proposed hierarchical stitching technique is able to leverage the strengths of the force directed and graph partitioning methods to more effectively reduce resource consumption by mapping each round to near optimality and utilizing the same force-directed technique combined with the introduction of intermediate destinations to mitigate the overheads incurred by inter-round communication. Within all explored multi-level factory circuits, these optimizations further reduced resource consumption. In the largest case, a capacity 100 two level factory shows a 5.64x reduction in overall consumed quantum volume when moving from the linear mapping approach without reusing qubits, to hierarchical stitching.

5.6 Related Work

Other work has focused primarily in the direction of optimizing the efficiency of the protocol by which magic states can be distilled. The original proposal [BK05a] considered a procedure by which 15 raw input states would be consumed to produce a single higher fidelity output state. Later works [BH12, Jon13, HHPW17] each explore different realizations of procedures that distill high fidelity magic states, with each procedure optimizing for asymptotic output rate and increasing this rate from the original proposal. These approaches tend to omit

overheads related to actual circuit implementations.

There have been several prior works [FDJ13a, OC17] that have attempted to reduce the circuit depth of an explicit implementation of the Bravyi-Haah distillation circuit, as well as perform a resource estimate by considering the rates at which these factories fail.

Additionally, several efforts have been made to build compilers and tools to be more precise about resource estimation quantification in topological quantum error corrected systems [PDNP12, PPND15, PDF16]. These techniques have resulted in tools that are used to compile and schedule arbitrary quantum circuits to topological assembly, and topological braid compaction techniques are used to reduce circuit depth expansions.

Systems level analysis has been performed by two related projects [IWPK08, TMN+17], in which the former optimizes the structure of early distillation protocols, and the latter proposes a micro-architectural accelerator to handle large amounts of error correction instructions that exist in fault tolerant machines.

Surface code braid scheduling costs were analyzed in [JAGH+17] using an end-to-end toolflow. The work focused on the resource impact of the choice of different implementation styles of surface code logical qubits. That work provides a toolchain upon which we have built in order to optimize scheduling and mapping procedures, as well as perform circuit simulations.

Our work introduces the complexity of braid scheduling into the analysis of the structure of the leading state distillation procedures in an attempt to concretize the procedures into real space and time resource costs. The new annealing heuristics (e.g. dipole-moments) developed specifically for this purpose also generalize well to any circuit executing on a fault tolerant machine that uses braiding to perform two-qubit gates.

5.7 Chapter Summary

Current error detection and correction codes still have prohibitive resource overheads, among which magic-state distillation is the most expensive component. Known optimized scheduling and mapping techniques for state distillation circuits tend to work well for small, single level factories, but quickly incur large overheads for larger factories. We have proposed a technique that synthesizes mapping and scheduling optimizations to take advantage of the unique efficiencies of each, which allows for a significant 5.64x reduction in the realistic space time volume required to implement multi-level magic state distillation factories. Global optimizations like graph partitioning and force-directed annealing work well, but leveraging structure of the block code circuitry combined with the specific strengths of both graph partitioning and force-directed annealing allows for the most improvement, resulting in large factors of resource reduction overall.

CHAPTER 6

THESIS CONCLUSION AND OPEN DISCUSSION

6.1 Concluding Remarks: Towards Practical QC

Current phase of quantum computer (QC) development is commonly referred as the Noisy Intermediate-Scale Quantum (NISQ) era [Pre18]. These quantum computers are able to perform on the order of hundreds of quantum operations (gates) using tens to hundreds of quantum bits (qubits). While modest in scale, these NISQ machines are large and reliable enough to perform some computational tasks. Looking beyond the NISQ era, quantum computers will ultimately arrive at the Fault-Tolerant (FT) era [BDSW96, Got10], where quantum error correction is implemented to ensure operation fidelity is met for arbitrarily large computations.

Quantum technology is transforming a wide range of computing applications. From theoretical computer science applications, to numerical analysis, to simulations of nature, and recently to ML tasks. It has been shown either theoretically or empirically that, by utilizing QM properties such as superposition and entanglement, QC can perform computational tasks that are intractable on conventional digital computers. But why is right now the best time to study quantum computer architecture? QC is getting more and more accessible and powerful, but a systems perspective is critical to guide the development of scalable hardware and software.

Many companies and research labs have deployed quantum testbeds, some with public cloud access. This ability to play around with real hardware has accelerated the development of QC algorithms, software, and devices. We have now reached a point where QC can perform some non-trivial computational tasks that classical computers struggle with, a notion called "quantum advantage." But there are still some outstanding challenges before we reach practical-scale QC.

The first challenge is system size. Current quantum computers have a limited number of qubits, from tens of qubits to hundreds of qubits. The small system size makes it difficult to execute large quantum programs simply because they won't fit.

The second challenge is that quantum systems are noisy. The error rate of a quantum computer can be very high. For example, superconducting devices have an average gate error rate of 10^{-3} to 10^{-4}, so an error would likely occur for every few thousand operations. Qubits have a limited lifetime, too, typically just enough time to perform tens of thousands of operations. Other technologies have similar error rates. And, these errors together will limit the success rate of a quantum program.

The third challenge is the software control complexity. Programming quantum circuits today is like programming assembly language in the 1960s, where we write down and optimize a sequence of gates. The good news is that unlike the 1960s, now we can use the best classical resource, HPC, to simulate, compile and optimize our quantum programs. But the bad news is that even with the help of supercomputers, we cannot faithfully describe highly complex entangled quantum systems. So, long compilation/optimization time could become a bottleneck. We need a very efficient software systems stack to overcome that.

The job of a software systems stack is to bridge the gap between the requirements of the quantum applications and the realities of the quantum hardware. System size and error tolerance have been improving remarkably over the years. They have now moved past the classical simulation capability, but there is still a significant gap until we can run useful applications. One promising approach is to build a reliable and scalable quantum systems stack via software-hardware co-design.

Why is this co-design methodology promising for quantum computing? Traditionally, when we design a software tool to map applications to hardware, we consider the application as a black box (e.g., a quantum algorithm is just a sequence of quantum gates), and the hardware as a black box (with a certain number of qubits). However, if the requirement of

the quantum algorithm exceeds the resource given by the device, then we will not be able to successfully run this application. Typically resulting in a low success rate and high resource cost.

Now, if we open up the boxes, we will see what is inside of an application and how we can change an application to adapt to the hardware. For example, we can synthesize quantum circuits differently or adjust control flow/parallelism differently according to hardware. We can also see what can be changed on the hardware side to serve the target application better, such as qubit selection or measurement reduction based on the application. The software system stack in between is responsible for performing this cross-layer optimization in an automated and systematic fashion. The key here is picking the right amount of information from each side and optimize together. In this book [DC20], we talked about this methodology in more detail.

6.2 Open Discussion

Following the structure of the work presented in the thesis, we again discuss the exciting future directions in three categories:

6.2.1 Systematic Noise Mitigation

It is still an open issue on how to perform calibration for a quantum chip with a large number of qubits. Fabrication variation exists, so pulse optimizations need to be tailored for each qubit. How to apply such calibrations, in complementary to our frequency tuning algorithm, remains an open problem. Arguably, not all qubits need tunability; hence one can imagine building a hybrid architecture of tunable transmon qubits and fixed frequency qubits, balancing the pros and cons of the two. Another angle to see the long-term impact of our work is the open problems it raised.

- *Quantum Intermediate Representation (IR).* What should the quantum IR be? In this

work, we redefined quantum instructions to include both quantum gates and frequency tuning, and achieves surprising noise mitigation benefits.

- *Device Engineering.* What is the appropriate level of tunability? What is the long-term scalable device topology design? We are the first to adapt chip interconnect designs for quantum devices, as a first step towards better understanding of good topology designs.

- *Online Global Calibration.* A new systems model of real-time execution and calibration is proposed. This is in fact a generic framework; a natural, exciting direction is to apply this methodology beyond crosstalk mitigation, and even beyond superconducting transmon technologies.

6.2.2 Efficient Quantum Memory Management

Our work is not only a timely demonstration of the importance of quantum memory management, but also a proposal for a full-stack optimization framework. We found that having such full-stack optimization changes perspective. Specifically, garbage collection is beneficial when it is aware of program structures (e.g. parallelism) and device characteristics (e.g. noise). But this is only one aspect of quantum memory management. An exciting direction is to integrate techniques such as dirty qubit borrowing, qutrits, and noise-aware mapping into this framework, and optimize them together.

Furthermore, this work bridges NISQ and FT systems, by showing that garbage collection techniques not only are useful in near term but also carry over to future large-scale error-corrected machines. It is worthwhile to explore techniques that can guide a path from NISQ to FT systems.

6.2.3 Low-Overhead Quantum Error Correction

The theory of quantum error correction provides robustness guarantees to the computation. But the overall resource cost for realizing error correction is still prohibitively high. Lowering such resource cost will hasten the breakthrough from the NISQ to the FT era of quantum computing.

- *Drawing Insights from Noise Mitigation Techniques.* Traditional error correction techniques usually cast a wide net in detecting and correcting general errors. Adapting them to the special error characteristics in applications and devices will be particularly useful.

- *System-Level Performance.* We are studying the effect of higher-level factory optimizations on application performance. This includes analysis of resource distribution, comparison of factory system layout topologies, as well as architectures with prepared state buffers. The interaction with the Hierarchical Stitching procedure is currently being analyzed.

- *Novel Topological Operations.* Another exciting direction is the invention of new topological operations for communication, such as teleportation, lattice surgery, braiding, etc. It is worth to explore the different optimizations under these communication models, which will likely change the resource trade-off.

While the examples in the thesis laid some groundwork for architecting quantum computer systems in the presence of noise, they perhaps open more questions than they answer. The thesis emphasizes on one message: systems software must be specialized and adapt to the applications and devices. How much specialization is enough? How to make them future-proof? What kind of insights we learned today about noisy intermediate-scale systems can persist to future fault-tolerant quantum systems? I envision future work will address these questions as quantum computer architecture enters an exciting age of innovation.

REFERENCES

[AAA+19] Héctor Abraham, Ismail Yunus Akhalwaya, Gadi Aleksandrowicz, Thomas Alexander, Gadi Alexandrowics, Eli Arbel, Abraham Asfaw, Carlos Azaustre, AzizNgoueya, Panagiotis Barkoutsos, George Barron, Luciano Bello, Yael Ben-Haim, Daniel Bevenius, Lev S. Bishop, Samuel Bosch, Sergey Bravyi, David Bucher, Fran Cabrera, Padraic Calpin, Lauren Capelluto, Jorge Carballo, Ginés Carrascal, Adrian Chen, Chun-Fu Chen, Richard Chen, Jerry M. Chow, Christian Claus, Christian Clauss, Abigail J. Cross, Andrew W. Cross, Simon Cross, Juan Cruz-Benito, Chris Culver, Antonio D. Córcoles-Gonzales, Sean Dague, Tareq El Dandachi, Matthieu Dartiailh, DavideFrr, Abdón Rodríguez Davila, Delton Ding, Jun Doi, Eric Drechsler, Drew, Eugene Dumitrescu, Karel Dumon, Ivan Duran, Kareem EL-Safty, Eric Eastman, Pieter Eendebak, Daniel Egger, Mark Everitt, Paco Martín Fernández, Axel Hernández Ferrera, Albert Frisch, Andreas Fuhrer, MELVIN GEORGE, Julien Gacon, Gadi, Borja Godoy Gago, Jay M. Gambetta, Adhisha Gammanpila, Luis Garcia, Shelly Garion, Juan Gomez-Mosquera, Salvador de la Puente González, Ian Gould, Donny Greenberg, Dmitry Grinko, Wen Guan, John A. Gunnels, Isabel Haide, Ikko Hamamura, Vojtech Havlicek, Joe Hellmers, Łukasz Herok, Stefan Hillmich, Hiroshi Horii, Connor Howington, Shaohan Hu, Wei Hu, Haruki Imai, Takashi Imamichi, Kazuaki Ishizaki, Raban Iten, Toshinari Itoko, Ali Javadi-Abhari, Jessica, Kiran Johns, Tal Kachmann, Naoki Kanazawa, Kang-Bae, Anton Karazeev, Paul Kassebaum, Spencer King, Knabberjoe, Arseny Kovyrshin, Vivek Krishnan, Kevin Krsulich, Gawel Kus, Ryan LaRose, Raphaël Lambert, Joe Latone, Scott Lawrence, Dennis Liu, Peng Liu, Yunho Maeng, Aleksei Malyshev, Jakub Marecek, Manoel Marques, Dolph Mathews, Atsushi Matsuo, Douglas T. McClure, Cameron McGarry, David McKay, Dan McPherson, Srujan Meesala, Martin Mevissen, Antonio Mezzacapo, Rohit Midha, Zlatko Minev, Abby Mitchell, Nikolaj Moll, Michael Duane Mooring, Renier Morales, Niall Moran, Prakash Murali, Jan Müggenburg, David Nadlinger, Giacomo Nannicini, Paul Nation, Yehuda Naveh, Patrick Neuweiler, Pradeep Niroula, Hassi Norlen, Lee James O'Riordan, Oluwatobi Ogunbayo, Pauline Ollitrault, Steven Oud, Dan Padilha, Hanhee Paik, Simone Perriello, Anna Phan, Marco Pistoia, Alejandro Pozas-iKerstjens, Viktor Prutyanov, Daniel Puzzuoli, Jesús Pérez, Quintiii, Rudy Raymond, Rafael Martín-Cuevas Redondo, Max Reuter, Julia Rice, Diego M. Rodríguez, Max Rossmannek, Mingi Ryu, Tharrmashastha SAPV, SamFerracin, Martin Sandberg, Ninad Sathaye, Bruno Schmitt, Chris Schnabel, Zachary Schoenfeld, Travis L. Scholten, Eddie Schoute, Joachim Schwarm, Ismael Faro Sertage, Kanav Setia, Nathan Shammah, Yunong Shi, Adenilton Silva, Andrea Simonetto, Nick Singstock, Yukio Siraichi, Iskandar Sitdikov, Seyon Sivarajah, Magnus Berg Sletfjerding, John A. Smolin, Mathias Soeken, Igor Olegovich Sokolov, SooluThomas, Dominik Steenken, Matt Stypulkoski, Jack Suen, Hitomi Takahashi, Ivano Tavernelli, Charles Taylor,

Pete Taylour, Soolu Thomas, Mathieu Tillet, Maddy Tod, Enrique de la Torre, Kenso Trabing, Matthew Treinish, TrishaPe, Wes Turner, Yotam Vaknin, Carmen Recio Valcarce, Francois Varchon, Almudena Carrera Vazquez, Desiree Vogt-Lee, Christophe Vuillot, James Weaver, Rafal Wieczorek, Jonathan A. Wildstrom, Robert Wille, Erick Winston, Jack J. Woehr, Stefan Woerner, Ryan Woo, Christopher J. Wood, Ryan Wood, Stephen Wood, James Wootton, Daniyar Yeralin, Richard Young, Jessie Yu, Christopher Zachow, Laura Zdanski, Christa Zoufal, Zoufalc, azulehner, bcamorrison, brandhsn, chlorophyll zz, dan1pal, dime10, drholmie, elfrocampeador, faisaldebouni, fanizzamarco, gruu, kanejess, klinvill, kurarrr, lerongil, ma5x, merav aharoni, ordmoj, sethmerkel, strickroman, sumitpuri, tigerjack, toural, vvilpas, welien, willhbang, yang.luh, yelojakit, and yotamvakninibm. Qiskit: An open-source framework for quantum computing, 2019.

[AAB+19] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando GSL Brandao, David A Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[Aar03] Scott Aaronson. Quantum lower bound for recursive fourier sampling. *Quantum Information & Computation*, 3(2):165–174, 2003.

[Aar05] Scott Aaronson. Guest column: Np-complete problems and physical reality. *ACM Sigact News*, 36(1):30–52, 2005.

[AASD16] Nabila Abdessaied, Matthew Amy, Mathias Soeken, and Rolf Drechsler. Technology mapping of reversible circuits to clifford+ t quantum circuits. In *Multiple-Valued Logic (ISMVL), 2016 IEEE 46th International Symposium on*, pages 150–155. IEEE, 2016.

[ABO97] D. Aharonov and M. Ben-Or. Fault-tolerant quantum computation with constant error. In *STOC*, STOC '97, pages 176–188. ACM, 1997.

[AC06] Panos Aliferis and Andrew Cross. Subsystem fault tolerance with the Bacon-Shor code. *arXiv preprint quant-ph/0610063*, 2006.

[ACR+07] Andris Ambainis, Andrew M. Childs, Ben W. Reichardt, Robert Spalek, and Shengyu Zhang. Any AND-OR formula of size N can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '07. IEEE Computer Society, 2007.

[AHB+13] D. T. C. Allcock, T. P. Harty, C. J. Ballance, B. C. Keitch, N. M. Linke, D. N. Stacey, and D. M. Lucas. A microfabricated ion trap with integrated microwave circuitry. *Applied Physics Letters*, 102(4):–, 2013.

[AJLA95] Vicki H Allan, Reese B Jones, Randall M Lee, and Stephen J Allan. Software pipelining. *ACM Computing Surveys (CSUR)*, 27(3):367–432, 1995.

[Alb83] David Z Albert. On quantum-mechanical automata. *Physics Letters A*, 98(5-6):249–252, 1983.

[AMM14] Matthew Amy, Dmitri Maslov, and Michele Mosca. Polynomial-time t-depth optimization of clifford+ t circuits via matroid partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(10):1476–1489, 2014.

[AMMR13] Matthew Amy, Dmitri Maslov, Michele Mosca, and Martin Roetteler. A meet-in-the-middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013.

[ARS17] Matthew Amy, Martin Roetteler, and Krysta M Svore. Verified compilation of space-efficient reversible circuits. In *International Conference on Computer Aided Verification*, pages 3–21. Springer, 2017.

[ASA+09] PB Antohi, D Schuster, GM Akselrod, J Labaziewicz, Y Ge, Z Lin, WS Bakr, and IL Chuang. Cryogenic ion trapping systems with surface-electrode traps. *Review of Scientific Instruments*, 80(1):013103, 2009.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[Bac06] Dave Bacon. Operator quantum error-correcting subsystems for self-correcting quantum memories. *Physical Review A*, 73(1):012340, 2006.

[BAN+19] K Bertels, I Ashraf, R Nane, X Fu, L Riesebos, S Varsamopoulos, A Mouedenne, H Van Someren, A Sarkar, and N Khammassi. Quantum computer architecture: Towards full-stack quantum accelerators. *arXiv preprint arXiv:1903.09575*, 2019.

[Bar82] Earl R Barnes. An algorithm for partitioning the nodes of a graph. *SIAM Journal on Algebraic Discrete Methods*, 3(4):541–550, 1982.

[BBC+95] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457, 1995.

[BBC+17] Lev S Bishop, Sergey Bravyi, Andrew Cross, Jay M Gambetta, and John Smolin. Quantum volume. *Quantum Volume. Technical Report*, 2017.

[BCH+18] Markus Brink, Jerry M Chow, Jared Hertzberg, Easwar Magesan, and Sami Rosenblatt. Device challenges for near term superconducting quantum processors: frequency collisions. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 6–1. IEEE, 2018.

[BCKH13] CDB Bentley, ARR Carvalho, David Kielpinski, and JJ Hope. Fast gates for ion traps by splitting laser pulses. *New Journal of Physics*, 15(4):043006, 2013.

[BCMS19] Colin D Bruzewicz, John Chiaverini, Robert McConnell, and Jeremy M Sage. Trapped-ion quantum computing: Progress and challenges. *Applied Physics Reviews*, 6(2):021314, 2019.

[BCT92] Preston Briggs, Keith D Cooper, and Linda Torczon. Rematerialization. *ACM SIGPLAN Notices*, 27(7):311–321, 1992.

[BCT94] Preston Briggs, Keith D Cooper, and Linda Torczon. Improvements to graph coloring register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):428–455, 1994.

[BD88] M. E. Benitez and J. W. Davidson. A portable global optimizer and linker. *SIGPLAN Not.*, 23(7):329–338, June 1988.

[BDGR06] Florent Bouchez, Alain Darte, Christophe Guillon, and Fabrice Rastello. Register allocation: What does the np-completeness proof of chaitin et al. really prove? or revisiting register allocation: Why and how. In *International Workshop on Languages and Compilers for Parallel Computing*, pages 283–298. Springer, 2006.

[BDSW96] Charles H Bennett, David P DiVincenzo, John A Smolin, and William K Wootters. Mixed-state entanglement and quantum error correction. *Physical Review A*, 54(5):3824, 1996.

[Bea] S. Bourdeauducq et al. Advanced Real-Time Infrastructure for Quantum physics, ARTIQ 1.0. zenodo. `https://github.com/m-labs/artiq`. Accessed: 2020-01-05.

[Bel64] John S Bell. On the einstein podolsky rosen paradox, 1964.

[Ben73a] C. H. Bennett. Logical reversibility of computation. *IBM J. Res. Dev.*, 17(6):525–532, November 1973.

[Ben73b] Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532, 1973.

[Ben80] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22:563–591, 05 1980.

[Ben89] Charles Bennett. Time/space trade-offs for reversible computation. *SIAM Journal on Computing*, 18(4):766–776, 1989.

[Ber08] Daniel J Bernstein. The salsa20 family of stream ciphers. In *New stream cipher designs*, pages 84–97. Springer, 2008.

[BG16] Sergey Bravyi and David Gosset. Improved classical simulation of quantum circuits dominated by clifford gates. *Physical review letters*, 116(25):250501, 2016.

[BGL+12] R Bowler, J Gaebler, Yiheng Lin, Ting Rei Tan, David Hanneke, John D Jost, JP Home, D Leibfried, and David J Wineland. Coherent diabatic ion transport and separation in a multizone trap array. *Physical review letters*, 109(8):080502, 2012.

[BGLL08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[BH12] Sergey Bravyi and Jeongwan Haah. Magic-state distillation with low overhead. *Phys. Rev. A*, 86:052329, Nov 2012.

[BHLL14] CJ Ballance, TP Harty, NM Linke, and DM Lucas. High-fidelity two-qubit quantum logic gates using trapped calcium-43 ions. *arXiv preprint arXiv:1406.5473*, 2014.

[BK98] Sergey B Bravyi and A Yu Kitaev. Quantum codes on a lattice with boundary. *arXiv preprint quant-ph/9811052*, 1998.

[BK05a] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71(2):022316, 2005.

[BK05b] Sergey Bravyi and Alexei Kitaev. Universal quantum computation with ideal clifford gates and noisy ancillas. *Physical Review A*, 71(2), 2005.

[BKM+13a] R Barends, J Kelly, A Megrant, D Sank, E Jeffrey, Yu Chen, Y Yin, B Chiaro, J Mutus, C Neill, et al. Coherent josephson qubit suitable for scalable quantum integrated circuits. *Physical review letters*, 111(8):080502, 2013.

[BKM+13b] Rami Barends, Julian Kelly, Anthony Megrant, Daniel Sank, Evan Jeffrey, Yu Chen, Yi Yin, Ben Chiaro, Josh Mutus, Charles Neill, et al. Coherent josephson qubit suitable for scalable quantum integrated circuits. *Physical review letters*, 111(8):080502, 2013.

[BKM+14a] R Barends, J Kelly, A Megrant, A Veitia, D Sank, E Jeffrey, TC White, J Mutus, AG Fowler, B Campbell, Y Chen, Z Chen, B Chiaro, A Dunsworth,

C Neill, P. J. J O'Malley, P Roushan, A Vainsencher, J Wenner, A. N Korotkov, A. N Cleland, and John M Martinis. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

[BKM+14b] Rami Barends, Julian Kelly, Anthony Megrant, Andrzej Veitia, Daniel Sank, Evan Jeffrey, Ted C White, Josh Mutus, Austin G Fowler, Brooks Campbell, et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

[BKRB08] Jan Benhelm, Gerhard Kirchmair, Christian F Roos, and Rainer Blatt. Towards fault-tolerant quantum computing with trapped ions. *Nature Physics*, 4(6):463–466, 2008.

[BKSO05] Steven Balensiefer, Lucas Kregor-Stickles, and Mark Oskin. An evaluation framework and instruction set architecture for ion-trap based quantum micro-architectures. In *ACM SIGARCH Computer Architecture News*, volume 33, pages 186–196. IEEE Computer Society, 2005.

[Blo46] Felix Bloch. Nuclear induction. *Physical review*, 70(7-8):460, 1946.

[BMDM04] BB Blinov, DL Moehring, L-M Duan, and Chris Monroe. Observation of entanglement between a single trapped atom and a single photon. *Nature*, 428(6979):153–157, 2004.

[BO01] CD Batista and Gerardo Ortiz. Generalized jordan-wigner transformations. *Physical review letters*, 86(6):1082, 2001.

[BOW19] Costin Bădescu, Ryan O'Donnell, and John Wright. Quantum state certification. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 503–514, 2019.

[BPF15] Nikolaj Bjørner, Anh-Dung Phan, and Lars Fleckenstein. νz-an optimizing smt solver. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 194–199. Springer, 2015.

[BQP+19] R Barends, CM Quintana, AG Petukhov, Yu Chen, D Kafri, K Kechedzhi, R Collins, O Naaman, S Boixo, F Arute, et al. Diabatic gates for frequency-tunable superconducting qubits. *Physical Review Letters*, 123(21):210501, 2019.

[BR12] Rainer Blatt and CF Roos. Quantum simulations with trapped ions. *Nature Physics*, 8(4):277–284, 2012.

[BSL+16a] R Barends, A Shabani, L Lamata, J Kelly, A Mezzacapo, U Las Heras, R Babbush, AG Fowler, B Campbell, Yu Chen, Z Chen, B Chiaro, A Dunsworth, E Jeffrey, A Lucero, A Megrant, JY Mutus, M Neeley, C Neill, P. J. J

O'Malley, C Quintana, P Roushan, D Sank, A Vainsencher, and J Wenner. Digitized adiabatic quantum computing with a superconducting circuit. *Nature*, 534(7606):222–226, 2016.

[BSL+16b] Rami Barends, Alireza Shabani, Lucas Lamata, Julian Kelly, Antonio Mezzacapo, Urtzi Las Heras, Ryan Babbush, Austin G Fowler, Brooks Campbell, Yu Chen, et al. Digitized adiabatic quantum computing with a superconducting circuit. *Nature*, 534(7606):222–226, 2016.

[BTV01] Harry Buhrman, John Tromp, and Paul Vitányi. Time and space bounds for reversible simulation. In *International Colloquium on Automata, Languages, and Programming*, pages 1017–1027. Springer, 2001.

[BV97] Ethan Bernstein and Umesh Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997.

[BWC+11] K. R. Brown, A. C. Wilson, Y. Colombe, C. Ospelkaus, A. M. Meier, E. Knill, D. Leibfried, and D. J. Wineland. Single-qubit-gate error below $\mathbf{10^{-4}}$ in a trapped ion. *Phys. Rev. A*, 84:030303, Sep 2011.

[C] Coherence times are obtained from daily calibration data from ibm q experience.

[C+07] Eric Chi et al. Tailoring quantum architectures to implementation style: a quantum computer for mobile and persistent qubits. In *ISCA*, ISCA '07. ACM, 2007.

[CBS+18] Andrew W Cross, Lev S Bishop, Sarah Sheldon, Paul D Nation, and Jay M Gambetta. Validating quantum computers using randomized model circuits. *arXiv preprint arXiv:1811.12926*, 2018.

[CBSG17] Andrew W Cross, Lev S Bishop, John A Smolin, and Jay M Gambetta. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

[CCD+03] Andrew M. Childs, Richard Cleve, Enrico Deotto, Edward Farhi, Sam Gutmann, and Daniel A. Spielman. Exponential algorithmic speedup by a quantum walk. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '03. ACM, 2003.

[CDD+18] Weilong Cui, Yongshan Ding, Deeksha Dangwal, Adam Holmes, Joseph McMahan, Ali Javadi-Abhari, Georgios Tzimpragos, Frederic Chong, and Timothy Sherwood. Charm: A language for closed-form high-level architecture modeling. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 152–165. IEEE, 2018.

[CDKM04] Steven A Cuccaro, Thomas G Draper, Samuel A Kutin, and David Petrie Moulton. A new quantum ripple-carry addition circuit. *arXiv preprint quant-ph/0410184*, 2004.

[CDL+00] Stefania Cavallar, Bruce Dodson, ArjenK. Lenstra, Walter Lioen, PeterL. Montgomery, Brian Murphy, Herman te Riele, Karen Aardal, Jeff Gilchrist, GÈrard Guillerm, Paul Leyland, J??el Marchand, FranÁois Morain, Alec Muffett, ChrisandCraig Putnam, and Paul Zimmermann. Factorization of a 512–bit RSA modulus. In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2000.

[CDQ20] Jinglei Cheng, Haoqing Deng, and Xuehai Qian. Accqoc: Accelerating quantum optimal control based pulse generation. *arXiv preprint arXiv:2003.00376*, 2020.

[CDR+18] SA Caldwell, N Didier, CA Ryan, EA Sete, A Hudson, P Karalekas, R Manenti, MP da Silva, R Sinclair, E Acala, et al. Parametrically activated entangling gates using transmon qubits. *Physical Review Applied*, 10(3):034050, 2018.

[CDT07] Andrew W Cross, David P DiVincenzo, and Barbara M Terhal. A comparative code study for quantum fault-tolerance. *arXiv preprint arXiv:0711.1556*, 2007.

[CFM17] Frederic T Chong, Diana Franklin, and Margaret Martonosi. Programming languages and compiler design for realistic quantum hardware. *Nature*, 549(7671):180, 2017.

[CG72] Jr. Coffman, E.G. and R.L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1(3):200–213, 1972.

[CH17] Earl T Campbell and Mark Howard. Unified framework for magic state distillation and multiqubit gate synthesis with reduced resource cost. *Physical Review A*, 95(2):022316, 2017.

[Cha82] Gregory J Chaitin. Register allocation & spilling via graph coloring. *ACM Sigplan Notices*, 17(6):98–101, 1982.

[Chu] I Chuang. Quantum architectures: qasm2circ.

[CLH+13] Diana P. L. Aude Craik, N. M. Linke, T. P. Harty, C. J. Ballance, D. M. Lucas, A. M. Steane, and D. T. C. Allcock. Microwave control electrodes for scalable, parallel, single-qubit operations in a surface-electrode ion trap, August 2013.

[CLNV15] Siu Man Chan, Massimo Lauria, Jakob Nordstrom, and Marc Vinyals. Hardness of approximation in pspace and separation results for pebble games. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 466–485. IEEE, 2015.

[CLS+04] J Chiaverini, D Leibfried, T Schaetz, MD Barrett, RB Blakestad, J Britton, WM Itano, JD Jost, E Knill, C Langer, R Ozeri, and D. J. Wineland. Realization of quantum error correction. *Nature*, 432(7017):602–605, 2004.

[CLS+17] DPL Aude Craik, NM Linke, MA Sepiol, TP Harty, JF Goodwin, CJ Ballance, DN Stacey, AM Steane, DM Lucas, and DTC Allcock. High-fidelity spatial and polarization addressing of ca+ 43 qubits using near-field microwave control. *Physical Review A*, 95(2):022337, 2017.

[CMS11] Julia Chuzhoy, Yury Makarychev, and Anastasios Sidiropoulos. On graph crossing number and edge planarization. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, pages 1050–1069. SIAM, 2011.

[CMS+15] Antonio D Córcoles, Easwar Magesan, Srikanth J Srinivasan, Andrew W Cross, Matthias Steffen, Jay M Gambetta, and Jerry M Chow. Demonstration of a quantum error detection code using a square lattice of four superconducting qubits. *Nature communications*, 6(1):1–10, 2015.

[CNR+14] Yu Chen, C Neill, P Roushan, N Leung, M Fang, R Barends, J Kelly, B Campbell, Z Chen, B Chiaro, et al. Qubit architecture with high coherence and fast tunable coupling. *Physical review letters*, 113(22):220502, 2014.

[COI+03] Dean Copsey, Mark Oskin, Francois Impens, Tzvetan Metodiev, Andrew Cross, Frederic T Chong, Isaac L Chuang, and John Kubiatowicz. Toward a scalable, silicon-based quantum computing architecture. *Selected Topics in Quantum Electronics, IEEE Journal of*, 9(6):1552–1569, 2003.

[CS96] A Robert Calderbank and Peter W Shor. Good quantum error-correcting codes exist. *Physical Review A*, 54(2):1098, 1996.

[CZ95] J. I. Cirac and P. Zoller. Quantum computations with cold trapped ions. *Phys. Rev. Letters*, 74:4091–4094, May 1995.

[DA05] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

[Dal91] William J Dally. Express cubes: Improving the performance ofk-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 40(9):1016–1023, 1991.

[DBL05] *38th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-38 2005), 12-16 November 2005, Barcelona, Spain*. IEEE Computer Society, 2005.

[DBL06] *33rd International Symposium on Computer Architecture (ISCA 2006), June 17-21, 2006, Boston, MA, USA*. IEEE Computer Society, 2006.

[DBL08] *35th International Symposium on Computer Architecture (ISCA 2008), June 21-25, 2008, Beijing, China*. IEEE, 2008.

[DC20] Yongshan Ding and Frederic T Chong. *Quantum Computer Systems: Research for Noisy Intermediate-Scale Quantum Computers*, volume 15. Morgan & Claypool Publishers, 2020.

[DCG+09] Leonardo DiCarlo, Jerry M Chow, Jay M Gambetta, Lev S Bishop, Blake R Johnson, DI Schuster, J Majer, Alexandre Blais, Luigi Frunzio, SM Girvin, et al. Demonstration of two-qubit algorithms with a superconducting quantum processor. *Nature*, 460(7252):240–244, 2009.

[DCS12] Guillaume Duclos-Cianci and Krysta M Svore. A state distillation protocol to implement arbitrary single-qubit rotations. *arXiv preprint arXiv:1210.1980*, 2012.

[Deu85] David Deutsch. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 400(1818):97–117, 1985.

[Deu89] David Elieser Deutsch. Quantum computational networks. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 425(1868):73–90, 1989.

[Dev16] Simon J Devitt. Programming quantum computers using 3-d puzzles, coffee cups, and doughnuts. *XRDS: Crossroads, The ACM Magazine for Students*, 23(1):45–50, 2016.

[DGL+20] Yongshan Ding, Pranav Gokhale, Sophia Fuhui Lin, Richard Rines, Thomas Propson, and Frederic T Chong. Systematic crosstalk mitigation for superconducting qubits via frequency-aware compilation. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 201–214. IEEE, 2020.

[DH73] William E Donath and Alan J Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

[DHJA+18] Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 828–840. IEEE, 2018.

[DiV00] David P. DiVincenzo. The physical implementation of quantum computation. *Fortschritte der Physik*, 48(9-11):771–783, 2000.

[DiV01] D. P. DiVincenzo. Dogma and heresy in quantum computing. *Quantum Info. Comput.*, 1(4):1–6, December 2001.

[DJ96] Jack W Davidson and Sanjay Jinturkar. Aggressive loop unrolling in a retargetable, optimizing compiler. In *Compiler Construction*, pages 59–73. Springer, 1996.

[DKLP02] Eric Dennis, Alexei Kitaev, Andrew Landahl, and John Preskill. Topological quantum memory. *Journal of Mathematical Physics*, 43(9):4452–4505, 2002.

[DLF+16] Shantanu Debnath, Norbert M Linke, Caroline Figgatt, Kevin A Landsman, Kevin Wright, and Christopher Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63, 2016.

[DMB08] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340. Springer, 2008.

[DN05a] Christopher M Dawson and Michael A Nielsen. The solovay-kitaev algorithm. *arXiv preprint quant-ph/0505030*, 2005.

[DN05b] Christopher M Dawson and Michael A Nielsen. The solovay-kitaev algorithm. *arXiv preprint quant-ph/0505030*, 2005.

[DS13] Michel H Devoret and Robert J Schoelkopf. Superconducting circuits for quantum information: an outlook. *Science*, 339(6124):1169–1174, 2013.

[DSMN13] Simon J Devitt, Ashley M Stephens, William J Munro, and Kae Nemoto. Requirements for fault-tolerant factoring on an atom-optics quantum computer. *Nature communications*, 4, 2013.

[DWH+20] Yongshan Ding, Xin-Chuan Wu, Adam Holmes, Ash Wiseth, Diana Franklin, Margaret Martonosi, and Frederic T Chong. Square: strategic quantum ancilla reuse for modular quantum programs via cost-effective uncomputation. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 570–583. IEEE, 2020.

[EJ96] Artur Ekert and Richard Jozsa. Quantum computation and shor's factoring algorithm. *Rev. Mod. Phys.*, 68:733–753, Jul 1996.

[EK09] Bryan Eastin and Emanuel Knill. Restrictions on transversal encoded quantum gate sets. *Physical Review Letters*, 102(11), 2009.

[ESG+07] Andreas Ermedahl, Christer Sandberg, Jan Gustafsson, Stefan Bygde, and Björn Lisper. Loop bound analysis based on a combination of program slicing, abstract interpretation, and invariant analysis. In *WCET*, 2007.

[FD12] Austin G Fowler and Simon J Devitt. A bridge to lower overhead quantum computation. *arXiv preprint arXiv:1209.0510*, 2012.

[FDJ13a] Austin G Fowler, Simon J Devitt, and Cody Jones. Surface code implementation of block code state distillation. *Scientific reports*, 3, 2013.

[FDJ13b] Austin G Fowler, Simon J Devitt, and Cody Jones. Surface code implementation of block code state distillation. *Scientific Reports*, 3:1939, jun 2013.

[Fey18] Richard P Feynman. Simulating physics with computers. In *Feynman and computation*, pages 133–153. CRC Press, 2018.

[FGG14] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm. *arXiv preprint arXiv:1411.4028*, 2014.

[Fie73] Miroslav Fiedler. Algebraic connectivity of graphs. *Czechoslovak mathematical journal*, 23(2):298–305, 1973.

[Fis81] J. Fisher. Trace scheduling: A technique for global microcode compaction. *Computers, IEEE Transactions on*, C-30(7):478–490, 1981.

[Fis84] Daniel S Fisher. Random walks in random environments. *Physical Review A*, 30(2):960, 1984.

[FKJ99] Michael Patrick Frank and Thomas F Knight Jr. *Reversibility for efficient computing*. PhD thesis, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, 1999.

[FMMC12] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[Fow12a] Austin Fowler. Time-optimal quantum computation. *arXiv preprint quant-ph/1210.4626*, 2012.

[Fow12b] Austin G. Fowler. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3), 2012.

[FR91] Thomas MJ Fruchterman and Edward M Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991.

[FRR+19] Xiang Fu, Leon Riesebos, MA Rol, Jeroen van Straten, J van Someren, Nader Khammassi, Imran Ashraf, RFL Vermeulen, V Newsum, KKL Loh, et al. eqasm: An executable quantum instruction set architecture. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 224–237. IEEE, 2019.

[FWH12] Austin G Fowler, Adam C Whiteside, and Lloyd CL Hollenberg. Towards practical classical processing for the surface code. *Physical review letters*, 108(18):180501, 2012.

[GAD+19] Pranav Gokhale, Olivia Angiuli, Yongshan Ding, Kaiwen Gui, Teague Tomesh, Martin Suchara, Margaret Martonosi, and Frederic T Chong. Minimizing state preparations in variational quantum eigensolver by partitioning into commuting families. *arXiv preprint arXiv:1907.13623*, 2019.

[GAD+20] Pranav Gokhale, Olivia Angiuli, Yongshan Ding, Kaiwen Gui, Teague Tomesh, Martin Suchara, Margaret Martonosi, and Frederic T Chong. O (N**3) measurement cost for variational quantum eigensolver on molecular hamiltonians. *IEEE Transactions on Quantum Engineering*, 1:1–24, 2020.

[Gay06] Simon J Gay. Quantum programming languages: survey and bibliography. *Mathematical Structures in Computer Science*, 16(4):581–600, 2006.

[GC99a] Daniel Gottesman and Isaac L Chuang. Demonstrating the viability of universal quantum computation using teleportation and single-qubit operations. *Nature*, 402(6760):390–393, 1999.

[GC99b] Daniel Gottesman and Isaac L Chuang. Quantum teleportation is a universal computational primitive. *arXiv preprint quant-ph/9908010*, 1999.

[GCS15] Jay M Gambetta, Jerry M Chow, and Matthias Steffen. Building logical qubits in a superconducting quantum computing system. *arXiv preprint arXiv:1510.04375*, 2015.

[GDP+19] Pranav Gokhale, Yongshan Ding, Thomas Propson, Christopher Winkler, Nelson Leung, Yunong Shi, David I Schuster, Henry Hoffmann, and Frederic T Chong. Partial compilation of variational algorithms for noisy intermediate-scale quantum machines. In *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 266–278, 2019.

[GESL06] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper. Automatic derivation of loop bounds and infeasible paths for wcet analysis using abstract execution. In *Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International*, pages 57–66, 2006.

[GF05] Harold N. Gabow and Ronald Fagin, editors. *Proceedings of the 37th Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*. ACM, 2005.

[GH14] James R Goodman and W-C Hsu. Code scheduling and register allocation in large basic blocks. In *ACM International Conference on Supercomputing 25th Anniversary Volume*, pages 88–98. ACM, 2014.

[GJ83] Michael R Garey and David S Johnson. Crossing number is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983.

[GLR+13a] Alexander S Green, Peter LeFanu Lumsdaine, Neil J Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pages 333–342, 2013.

[GLR+13b] Alexander S. Green, Peter LeFanu Lumsdaine, Neil J. Ross, Peter Selinger, and Benoît Valiron. Quipper: a scalable quantum programming language. *SIGPLAN Not.*, 48(6):333–342, June 2013.

[GMC+09] Mark Gebhart, Bertrand A. Maher, Katherine E. Coons, Jeff Diamond, Paul Gratz, Mario Marino, Nitya Ranganathan, Behnam Robatmili, Aaron Smith, James Burrill, Stephen W. Keckler, Doug Burger, and Kathryn S. McKinley. An evaluation of the TRIPS computer system. In *Proceedings of the Fourteenth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, March 2009.

[GN02] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[Goo] A Preview of Bristlecone, Google's New Quantum Processor. `https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html`. Accessed: 2019-03-29.

[Got97] Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv preprint quant-ph/9705052*, 1997.

[Got98a] Daniel Gottesman. The heisenberg representation of quantum computers. *arXiv preprint quant-ph/9807006*, 1998.

[Got98b] Daniel Gottesman". Theory of fault-tolerant quantum computation. *Physical Review A*, 57(1):127–137, 1998.

[Got98c] Daniel Gottesman. Theory of fault-tolerant quantum computation. *Physical Review A*, 57(1):127, 1998.

[Got02] Daniel Gottesman. An introduction to quantum error correction. In *Proceedings of Symposia in Applied Mathematics*, volume 58, pages 221–236, 2002.

[Got10] Daniel Gottesman. An introduction to quantum error correction and fault-tolerant quantum computation. In *Quantum information science and its contributions to mathematics, Proceedings of Symposia in Applied Mathematics*, volume 68, pages 13–58, 2010.

[GP17] G. Giacomo Guerreschi and J. Park. Gate scheduling for quantum algorithms. *ArXiv e-prints*, July 2017.

[GPK+21] András Gyenis, Agustin Di Paolo, Jens Koch, Alexandre Blais, Andrew A. Houck, and David I. Schuster. Moving beyond the transmon: Noise-protected superconducting quantum circuits, 2021.

[Gro96a] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[Gro96b] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, 1996.

[GRZC03] Juan José García-Ripoll, Peter Zoller, and J Ignacio Cirac. Speed optimized two-qubit gates with laser coherent control techniques for ion trap quantum computing. *Physical Review Letters*, 91(15):157901, 2003.

[GWDD09] Daniel Große, Robert Wille, Gerhard W Dueck, and Rolf Drechsler. Exact multiple-control toffoli network synthesis with sat techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(5):703–715, 2009.

[HAB+14] TP Harty, DTC Allcock, CJ Ballance, L Guidoni, HA Janacek, NM Linke, DN Stacey, and DM Lucas. High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit. *Physical review letters*, 113(22):220501, 2014.

[Hal05] Sean Hallgren. Fast quantum algorithms for computing the unit group and class group of a number field. In Gabow and Fagin [GF05].

[Hal07] Sean Hallgren. Polynomial-time quantum algorithms for pell's equation and the principal ideal problem. *J. ACM*, 54(1):4:1–4:19, March 2007.

[HC18] Luke E Heyfron and Earl T Campbell. An efficient quantum compiler that reduces t count. *Quantum Science and Technology*, 4(1):015004, 2018.

[HCW+12] DA Hite, Y Colombe, AC Wilson, KR Brown, U Warring, R Jördens, JD Jost, KS McKay, DP Pappas, D Leibfried, et al. 100-fold reduction of electric-field noise in an ion trap cleaned with in situ argon-ion-beam bombardment. *Physical review letters*, 109(10):103001, 2012.

[HDJA+19] Adam Holmes, Yongshan Ding, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T Chong. Resource optimized quantum architectures for surface code implementations of magic-state distillation. *Microprocessors and Microsystems*, 67:56–70, 2019.

[HFDVM12a] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[HFDVM12b] Clare Horsman, Austin G Fowler, Simon Devitt, and Rodney Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[HHJ+10] D. Hanneke, J. P. Home, J. D. Jost, J. M. Amini, D. Leibfried, and D. J. Wineland. Realization of a programmable two-qubit quantum processor. *Nature Physics*, 6, 2010.

[HHL09a] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

[HHL09b] Aram W Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Physical review letters*, 103(15):150502, 2009.

[HHL+17] MD Hutchings, Jared B Hertzberg, Yebin Liu, Nicholas T Bronn, George A Keefe, Markus Brink, Jerry M Chow, and BLT Plourde. Tunable superconducting qubits with flux-independent coherence. *Physical Review Applied*, 8(4):044003, 2017.

[HHPW17] Jeongwan Haah, Matthew B Hastings, D Poulin, and D Wecker. Magic state distillation with low space overhead and optimal asymptotic input count. *arXiv preprint arXiv:1703.07847*, 2017.

[HJG+18] Adam Holmes, Sonika Johri, Gian Giacomo Guerreschi, James S Clarke, and AY Matsuura. Impact of qubit connectivity on quantum algorithm performance. *arXiv preprint arXiv:1811.02125*, 2018.

[HJK+96] Richard J Hughes, Daniel FV James, Emanuel H Knill, Raymond Laflamme, and Albert G Petschek. Decoherence bounds on quantum computation with trapped ions. *Physical review letters*, 77(15):3240, 1996.

[HJP+20] Adam Holmes, Mohammad Reza Jokar, Ghasem Pasandi, Yongshan Ding, Massoud Pedram, and Frederic T Chong. Nisq+: Boosting quantum computing power by approximating quantum error correction. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 556–569. IEEE, 2020.

[HMF+09] Ferdinand Helmer, Matteo Mariantoni, Austin G Fowler, Jan von Delft, Enrique Solano, and Florian Marquardt. Cavity grid for scalable quantum computation with superconducting circuits. *EPL (Europhysics Letters)*, 85(5):50007, 2009.

[HN60] Frank Harary and Robert Z Norman. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 9(2):161–168, 1960.

[HOS+06] WK Hensinger, S Olmschenk, D Stick, D Hucul, M Yeo, M Acton, L Deslauriers, C Monroe, and J Rabchuk. T-junction ion trap array for two-dimensional ion shuttling, storage, and manipulation. *Applied Physics Letters*, 88(3):034101, 2006.

[HPJ+15a] Jeff Heckey, Shruti Patil, Ali JavadiAbhari, Adam Holmes, Daniel Kudrow, Kenneth R Brown, Diana Franklin, Frederic T Chong, and Margaret Martonosi. Compiler management of communication and parallelism for quantum computation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 445–456, 2015.

[HPJ+15b] Jeff Heckey, Shruti Patil, Ali JavadiAbhari, Adam Holmes, Daniel Kudrow, Kenneth R. Brown, Diana Franklin, Frederic T. Chong, and Margaret Martonosi. Compiler management of communication and parallelism for quantum computation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, pages 445–456, New York, NY, USA, 2015. ACM.

[HSA+16] TP Harty, MA Sepiol, DTC Allcock, CJ Ballance, JE Tarlton, and DM Lucas. High-fidelity trapped-ion quantum logic using near-field microwaves. *arXiv preprint arXiv:1606.08409*, 2016.

[HSSC08] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[Hu05] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Mathematica Journal*, 10(1):37–71, 2005.

[HWO98] Lance Hammond, Mark Willey, and Kunle Olukotun. Data speculation support for a chip multiprocessor. *ACM SIGOPS Operating Systems Review*, 32(5):58–69, 1998.

[IBM] Quantum Takes Flight: Moving from Laboratory Demonstrations to Building Systems. `https://www.ibm.com/blogs/research/2020/01/quantum-volume-32/`. Accessed: 2020-04-05.

[Ion] IonQ harnesses single-atom qubits to build the world's most powerful quantum computer. `https://ionq.co/news/december-11-2018`. Accessed: 2019-03-29.

[Isa10] Nemanja Isailovic. *An investigation into the realities of a quantum datapath*. PhD thesis, University of California, Berkeley, 2010.

[IWPK08] Nemanja Isailovic, Mark Whitney, Yatish Patel, and John Kubiatowicz. Running a quantum circuit at the speed of data. *ACM SIGARCH Computer Architecture News*, 36(3):177–188, 2008.

[J+12] Ali JavadiAbhari et al. Scaffold: Quantum programming language. Technical report, Princeton University, 2012.

[JAGH+17] Ali Javadi-Abhari, Pranav Gokhale, Adam Holmes, Diana Franklin, Kenneth R Brown, Margaret Martonosi, and Frederic T Chong. Optimized surface code communication in superconducting quantum computers. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 692–705. ACM, 2017.

[JAPK+14] Ali Javadi-Abhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T. Chong, and Margaret Martonosi. Scaffcc: A framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, CF '14, pages 1:1–1:10, New York, NY, USA, 2014. ACM.

[JBT+09] M. Johanning, A. Braun, N. Timoney, V. Elman, W. Neuhauser, and Chr. Wunderlich. Individual addressing of trapped ions and coupling of motional and spin states using RF radiation. *Phys. Rev. Lett.*, 102:073004, Feb 2009.

[Jon13] Cody Jones. Multilevel distillation of magic states for quantum computing. *Physical Review A*, 87(4):042305, 2013.

[JOP+01] Eric Jones, Travis Oliphant, Pearu Peterson, et al. Scipy: Open source scientific tools for python. 2001.

[JPK+14] Ali JavadiAbhari, Shruti Patil, Daniel Kudrow, Jeff Heckey, Alexey Lvov, Frederic T Chong, and Margaret Martonosi. Scaffcc: a framework for compilation and analysis of quantum computing programs. In *Proceedings of the 11th ACM Conference on Computing Frontiers*, pages 1–10, 2014.

[JVMF+12a] N Cody Jones, Rodney Van Meter, Austin G Fowler, Peter L McMahon, Jungsang Kim, Thaddeus D Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Physical Review X*, 2(3):031007, 2012.

[JVMF+12b] N Cody Jones, Rodney Van Meter, Austin G Fowler, Peter L McMahon, Jungsang Kim, Thaddeus D Ladd, and Yoshihisa Yamamoto. Layered architecture for quantum computing. *Physical Review X*, 2(3):031007, 2012.

[K+05] J Kim et al. System design for large-scale ion trap quantum information processor. *Quantum Information & Computation*, 5(7):515–537, 2005.

[KB09] Stephen W. Keckler and Luiz André Barroso, editors. *36th International Symposium on Computer Architecture (ISCA 2009), June 20-24, 2009, Austin, TX, USA*. ACM, 2009.

[KBD+13] Daniel Kudrow, Kenneth Bier, Zhaoxia Deng, Diana Franklin, Yu Tomita, Kenneth R Brown, and Frederic T Chong. Quantum rotations: a case study

in static and dynamic machine-code generation for quantum computers. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, pages 166–176, 2013.

[KBF+15] Julian Kelly, Rami Barends, Austin G Fowler, Anthony Megrant, Evan Jeffrey, Theodore C White, Daniel Sank, Josh Y Mutus, Brooks Campbell, Yu Chen, et al. State preservation by repetitive error detection in a superconducting quantum circuit. *Nature*, 519(7541):66–69, 2015.

[Kit97] Aleksei Yur'evich Kitaev. Quantum computations: algorithms and error correction. *Uspekhi Matematicheskikh Nauk*, 52(6):53–112, 1997.

[KK95] George Karypis and Vipin Kumar. Metis–unstructured graph partitioning and sparse matrix ordering system, version 2.0. 1995.

[KK99] George Karypis and Vipin Kumar. A fast and highly quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1999.

[KK00] George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3):285–300, 2000.

[KKMN20] Paul V Klimov, Julian Kelly, John M Martinis, and Hartmut Neven. The snake optimizer for learning quantum processor control parameters. *arXiv preprint arXiv:2006.04594*, 2020.

[KKY+19] Philip Krantz, Morten Kjaergaard, Fei Yan, Terry P Orlando, Simon Gustavsson, and William D Oliver. A quantum engineer's guide to superconducting qubits. *Applied Physics Reviews*, 6(2):021318, 2019.

[KL70] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307, 1970.

[KMM] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. SQCT: Single qubit circuit toolkit.

[KMM12] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single qubit unitaries generated by clifford and t gates. *arXiv preprint arXiv:1206.5236*, 2012.

[KMM16] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Practical approximation of single-qubit unitaries by single-qubit quantum clifford and t circuits. *IEEE Transactions on Computers*, 65(1):161–172, 2016.

[KMN+02] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892, 2002.

[KMW02] David Kielpinski, Chris Monroe, and David J Wineland. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417(6890):709–711, 2002.

[Kni95] Emanuel Knill. An analysis of bennett's pebble game. *arXiv preprint math/9508218*, 1995.

[Kni05] Emanuel Knill. Quantum computing with realistically noisy devices. *Nature*, 434(7029):39–44, 2005.

[KSB+20] Morten Kjaergaard, Mollie E Schwartz, Jochen Braumüller, Philip Krantz, Joel I-J Wang, Simon Gustavsson, and William D Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11:369–395, 2020.

[KSG+20] M Kjaergaard, ME Schwartz, A Greene, GO Samach, A Bengtsson, M O'Keeffe, CM McNally, J Braumüller, DK Kim, P Krantz, et al. A quantum instruction set implemented on a superconducting quantum processor. *arXiv preprint arXiv:2001.08838*, 2020.

[KSO08] Lucas Kreger-Stickles and Mark Oskin. Microcoded architectures for ion-tap quantum computers. In *ISCA* [DBL08], pages 165–176.

[KSS18] Balagopal Komarath, Jayalal Sarma, and Saurabh Sawlani. Pebbling meets coloring: Reversible pebble game on trees. *Journal of Computer and System Sciences*, 91:33–41, 2018.

[KTG+07] Jens Koch, M Yu Terri, Jay Gambetta, Andrew A Houck, DI Schuster, J Majer, Alexandre Blais, Michel H Devoret, Steven M Girvin, and Robert J Schoelkopf. Charge-insensitive qubit design derived from the cooper pair box. *Physical Review A*, 76(4):042319, 2007.

[Kub15] Aleksander Kubica". Universal transversal gates with color codes: A simplified approach. *Physical Review A*, 91(3), 2015.

[L+03] Dietrich Leibfried et al. Experimental demonstration of a robust, high-fidelity geometric two ion-qubit phase gate. *Nature*, 422(6930):412–415, 2003.

[LA04] C. Lattner and V. Adve. LLVM: a compilation framework for lifelong program analysis and transformation. In *CGO*, pages 75–86, 2004.

[Lar18] F Lardinois. Rigetti announces its hybrid quantum computing platform and a 1 m prize. *TechCrunch*, 2018.

[LBR17] AP Lund, Michael J Bremner, and TC Ralph. Quantum sampling problems, bosonsampling and quantum supremacy. *npj Quantum Information*, 3(1):1–8, 2017.

[LCFM09] Paul Lokuciejewski, Daniel Cordes, Heiko Falk, and Peter Marwedel. A fast and precise static loop analysis based on abstract interpretation, program slicing and polytope models. In *Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization*, CGO '09, pages 136–146, Washington, DC, USA, 2009. IEEE Computer Society.

[LdST+13] Andrei Lapets, Marcus P da Silva, Mike Thome, Aaron Adler, Jacob Beal, and Martin Rötteler. Quafl: a typed dsl for quantum programming. In *Proceedings of the 1st annual workshop on Functional programming concepts in domain-specific languages*, pages 19–26. ACM, 2013.

[LDX19] Gushu Li, Yufei Ding, and Yuan Xie. Tackling the qubit mapping problem for nisq-era quantum devices. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1001–1014, 2019.

[LDX20] Gushu Li, Yufei Ding, and Yuan Xie. Towards efficient superconducting quantum processor architecture design. In *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1031–1045, 2020.

[LJL+10] Thaddeus D Ladd, Fedor Jelezko, Raymond Laflamme, Yasunobu Nakamura, Christopher Monroe, and Jeremy L O'Brien. Quantum computers. *Nature*, 464(7285):45–53, 2010.

[Llo96] Seth Lloyd. Universal quantum simulators. *Science*, pages 1073–1078, 1996.

[LMPZ96] Raymond LaFlamme, Cesar Miquel, Juan Pablo Paz, and Wojciech Hubert Zurek. Perfect quantum error correcting code. *Physical Review Letters*, 77(1):198, 1996.

[LO18] Daniel Litinski and Felix von Oppen. Lattice surgery with a twist: Simplifying clifford gates of surface codes. *Quantum*, 2, 2018.

[LPV81] Gavriela Freund Lev, Nicholas Pippenger, and Leslie G Valiant. A fast parallel algorithm for routing in permutation networks. *IEEE transactions on Computers*, 100(2):93–100, 1981.

[LR13] Andrei Lapets and Martin Roetteler. Abstract resource cost derivation for logical quantum circuit descriptions. In *Proceedings of the 1st annual workshop on Functional programming concepts in domain-specific languages*, FPCDSL '13, pages 35–42, New York, NY, USA, 2013. ACM.

[LS77] Shui Lam and Ravi Sethi. Worst-case analysis of two scheduling algorithms. *SIAM Journal of Computing*, 6:518–536, 1977.

[LW99] Daniel A Lidar and Haobin Wang. Calculating the thermal rate constant with exponential speedup on a quantum computer. *Physical Review E*, 59(2):2429, 1999.

[LW18] Seth Lloyd and Christian Weedbrook. Quantum generative adversarial learning. *Physical review letters*, 121(4):040502, 2018.

[LWF+15] B Lekitsch, S Weidt, AG Fowler, K Mølmer, SJ Devitt, Ch Wunderlich, and WK Hensinger. Blueprint for a microwave ion trap quantum computer. *arXiv preprint arXiv:1508.00420*, 2015.

[LY12] Chun-Cheng Lin and Hsu-Chun Yen. A new force-directed graph drawing method based on edge–edge repulsion. *Journal of Visual Languages & Computing*, 23(1):29–42, 2012.

[M+95] Chris Monroe et al. Demonstration of a fundamental quantum logic gate. *Physical Review Letters*, 75(25):4714, 1995.

[M+05] Tzvetan S. Metodi et al. A quantum logic array microarchitecture: Scalable quantum data movement and computation. In *MICRO* [DBL05], pages 305–318.

[MAB+18] David C McKay, Thomas Alexander, Luciano Bello, Michael J Biercuk, Lev Bishop, Jiayin Chen, Jerry M Chow, Antonio D Córcoles, Daniel Egger, Stefan Filipp, et al. Qiskit backend specifications for openqasm and openpulse experiments. *arXiv preprint arXiv:1809.03452*, 2018.

[Mas16] Dmitri Maslov. Advantages of using relative-phase toffoli gates with an application to multiple control toffoli optimization. *Physical Review A*, 93(2):022311, 2016.

[MBJA+19a] Prakash Murali, Jonathan M Baker, Ali Javadi-Abhari, Frederic T Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1015–1029, 2019.

[MBJA+19b] Prakash Murali, Jonathan M. Baker, Ali Javadi-Abhari, Frederic T. Chong, and Margaret Martonosi. Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '19, 2019.

[MC06] Tzvetan S Metodi and Frederic T Chong. Quantum computing for computer architects. *Synthesis Lectures in Computer Architecture*, 1(1):1–154, 2006.

[MDM05] Dmitri Maslov, Gerhard W Dueck, and D Michael Miller. Toffoli network synthesis with templates. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(6):807–817, 2005.

[MEK12] Adam M Meier, Bryan Eastin, and Emanuel Knill. Magic-state distillation with the four-qubit code. *arXiv preprint arXiv:1204.4221*, 2012.

[MG14] John M Martinis and Michael R Geller. Fast adiabatic qubit gates using only σ z control. *Physical Review A*, 90(2):022307, 2014.

[MK13] Christopher Monroe and Jungsang Kim. Scaling the ion trap quantum processor. *Science*, 339(6124):1164–1169, 2013.

[MMD03] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. A transformation based algorithm for reversible logic synthesis. In *Design Automation Conference, 2003. Proceedings*, pages 318–323. IEEE, 2003.

[MMMJA20] Prakash Murali, David C McKay, Margaret Martonosi, and Ali Javadi-Abhari. Software mitigation of crosstalk on noisy intermediate-scale quantum computers. *arXiv preprint arXiv:2001.02826*, 2020.

[MNAU02] John M Martinis, S Nam, J Aumentado, and C Urbina. Rabi oscillations in a large josephson-junction qubit. *Physical Review Letters*, 89(11):117901, 2002.

[MNJ+14] J Mizrahi, B Neyenhuis, KG Johnson, WC Campbell, C Senko, D Hayes, and C Monroe. Quantum control of qubits and atomic motion using ultrafast laser pulses. *Applied Physics B*, 114(1-2):45–61, 2014.

[MOL+99] JE Mooij, TP Orlando, L Levitov, Lin Tian, Caspar H Van der Wal, and Seth Lloyd. Josephson persistent-current qubit. *Science*, 285(5430):1036–1039, 1999.

[Mon16] Ashley Montanaro. Quantum algorithms: an overview. *npj Quantum Information*, 2:npjqi201523, 2016.

[Mos09] Michele Mosca. Quantum algorithms. In Robert A. Meyers, editor, *Encyclopedia of Complexity and Systems Science*, pages 7088–7118. Springer New York, 2009.

[MR19] Margaret Martonosi and Martin Roetteler. Next steps in quantum computing: computer science's role. *arXiv preprint arXiv:1903.10541*, 2019.

[MRR+12] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. Large Scale Modular Quantum Computer Architecture with Atomic Memory and Photonic Interconnects. *arXiv:quant-ph/1208.0391*, 2012.

[MS12] Igor L Markov and Mehdi Saeedi. Constant-optimized quantum circuits for modular multiplication and exponentiation. *arXiv preprint arXiv:1202.6614*, 2012.

[MSB+11] Thomas Monz, Philipp Schindler, Julio T Barreiro, Michael Chwalla, Daniel Nigg, William A Coish, Maximilian Harlander, Wolfgang Hänsel, Markus Hennrich, and Rainer Blatt. 14-qubit entanglement: Creation and coherence. *Physical Review Letters*, 106(13):130506, 2011.

[MSK+00] C. Monroe, C. Sackett, D. Kielpinski, B. King, C. Langer, V. Meyer, C. Myatt, M. Rowe, Q. Turchette, W. Itano, and D. Wineland. Scalable entanglement of trapped ions. *Workshop on Trapped Ion Quantum Computing (National Institute of Standards and Technology, Boulder, Colorado)*, 2000.

[MSR+19] Giulia Meuli, Mathias Soeken, Martin Roetteler, Nikolaj Bjorner, and Giovanni De Micheli. Reversible pebbling game for quantum memory management. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 288–291. IEEE, 2019.

[MSS05] Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. In *Proceedings of the Sixteenth annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 1109–1117, 2005.

[MSS+19] David C McKay, Sarah Sheldon, John A Smolin, Jerry M Chow, and Jay M Gambetta. Three-qubit randomized benchmarking. *Physical review letters*, 122(20):200502, 2019.

[MTC+06] Tzvetan Metodi, Darshan Thaker, Andrew Cross, Frederic T. Chong, and Isaac L. Chuang. Scheduling physical operations in a quantum information processor. *Proceedings for the SPIE Defense & Security symposium, Orlando, FL, April*, 2006.

[MW01] Florian Mintert and Christof Wunderlich. Ion-trap quantum logic using long-wavelength radiation. *Physical Review Letters*, 87(25):257904, 2001.

[MZHH19] Pranav Mundada, Gengyan Zhang, Thomas Hazard, and Andrew Houck. Suppression of qubit crosstalk in a tunable coupling superconducting circuit. *Physical Review Applied*, 12(5):054023, 2019.

[NC02] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[NC10] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[NFB13] Ali Abu Nada, Ben Fortescue, and Mark Byrd. Relative performance of ancilla verification and decoding in the [[7,1,3]] Steane code. *arXiv preprint quant-ph/1303.4026*, 2013.

[NPT99] Yu Nakamura, Yu A Pashkin, and JS Tsai. Coherent control of macroscopic quantum states in a single-cooper-pair box. *Nature*, 398(6730):786–788, 1999.

[NRK+18] Charles Neill, Pedran Roushan, K Kechedzhi, Sergio Boixo, Sergei V Isakov, V Smelyanskiy, A Megrant, B Chiaro, A Dunsworth, K Arya, et al. A blueprint for demonstrating quantum supremacy with superconducting qubits. *Science*, 360(6385):195–199, 2018.

[NRS+18] Yunseong Nam, Neil J Ross, Yuan Su, Andrew M Childs, and Dmitri Maslov. Automated optimization of large quantum circuits with continuous parameters. *npj Quantum Information*, 4(1):23, 2018.

[NSZ17] Eesa Nikahd, Mehdi Sedighi, and Morteza Saheb Zamani. Nonuniform code concatenation for universal fault-tolerant quantum computing. *Physical Review A*, 96(3):032337, 2017.

[OC17] Joe O'Gorman and Earl T. Campbell. Quantum computation with realistic magic-state factories. *Phys. Rev. A*, 95:032338, Mar 2017.

[OCC02] M. Oskin, F.T. Chong, and I.L. Chuang. A practical architecture for reliable quantum computers. *Computer*, 35(1):79–87, 2002.

[OKB+15] PJJ O'Malley, J Kelly, R Barends, B Campbell, Y Chen, Z Chen, B Chiaro, A Dunsworth, AG Fowler, I-C Hoi, et al. Qubit metrology of ultralow phase noise using randomized benchmarking. *Physical Review Applied*, 3(4):044009, 2015.

[OLA+08] Christian Ospelkaus, Christopher E Langer, Jason M Amini, Kenton R Brown, Dietrich Leibfried, and David J Wineland. Trapped-ion quantum logic gates based on oscillating magnetic fields. *Physical review letters*, 101(9):090502, 2008.

[Öme98] Bernhard Ömer. A procedural formalism for quantum computing. 1998.

[oST12] National Institute of Standards and Technology. *FIPS PUB 180-4: Secure Hash Standard (SHS)*. U.S. Department of Commerce, 2012.

[OWC+11] C. Ospelkaus, U. Warring, Y. Colombe, K. R. Brown, J. M. Amini, D. Leibfried, and D. J. Wineland. Microwave quantum logic gates for trapped ions. *Nature*, 476:181–184, 2011.

[P+13] Anargyros Papageorgiou et al. A fast algorithm for approximating the ground state energy on a quantum computer. *Mathematics of Computation*, 82(284):2293–2304, 2013.

[PDF16] Alexandru Paler, Simon J Devitt, and Austin G Fowler. Synthesis of arbitrary quantum circuits to topological assembly. *Scientific reports*, 6:30600, 2016.

[PDNP12] Alexandru Paler, Simon Devitt, Kae Nemoto, and Ilia Polian. Synthesis of topological quantum circuits. In *Proceedings of the 2012 IEEE/ACM International Symposium on Nanoscale Architectures*, pages 181–187. ACM, 2012.

[PF13] Adam Paetznick and Austin G Fowler. Quantum circuit optimization by topological compaction in the surface code. *arXiv preprint arXiv:1304.2807*, 2013.

[PFW19] Alexandru Paler, Austin G Fowler, and Robert Wille. Faster manipulation of large quantum circuits using wire label reference diagrams. *Microprocessors and Microsystems*, 66:55–66, 2019.

[Pin93] Shlomit S. Pinter. Register allocation with instruction scheduling. In *Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation*, PLDI '93, pages 248–257, New York, NY, USA, 1993. ACM.

[PPND15] Alexandru Paler, Ilia Polian, Kae Nemoto, and Simon J Devitt. A compiler for fault-tolerant high level quantum circuits. *arXiv preprint arXiv:1509.02004*, 2015.

[PPND17] Alexandru Paler, Ilia Polian, Kae Nemoto, and Simon J Devitt. Fault-tolerant, high-level quantum circuits: form, compilation and description. *Quantum Science and Technology*, 2(2):025003, 2017.

[PR96] François Pellegrini and Jean Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *International Conference on High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[PR11] Adam Paetznick and Ben W Reichardt. Fault-tolerant ancilla preparation and noise threshold lower bounds for the 23-qubit golay code. *arXiv preprint arXiv:1106.2190*, 2011.

[Pre98] John Preskill. Fault-tolerant quantum computation. *Introduction to quantum computation and information*, 213, 1998.

[Pre18] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[PRS15] Alex Parent, Martin Roetteler, and Krysta M Svore. Reversible circuit compilation with space constraints. *arXiv preprint arXiv:1510.00377*, 2015.

[PS99] Massimiliano Poletto and Vivek Sarkar. Linear scan register allocation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 21(5):895–913, 1999.

[PS16] M. Pedram and A. Shafaei. Layout optimization for quantum circuits with linear nearest neighbor architectures. *IEEE Circuits and Systems Magazine*, 16(2):62–74, Secondquarter 2016.

[PSB+11] Hanhee Paik, DI Schuster, Lev S Bishop, G Kirchmair, G Catelani, AP Sears, BR Johnson, MJ Reagor, L Frunzio, LI Glazman, Steven M Girvin, Michel H Devoret, and Robert J Schoelkopf. Observation of high coherence in josephson junction qubits measured in a three-dimensional circuit qed architecture. *Physical Review Letters*, 107(24):240501, 2011.

[PVDC+20] Bishnu Patra, Jeroen PG Van Dijk, Andrea Corna, Xiao Xue, Nodar Samkharadze, Amir Sammak, Giordano Scappucci, Menno Veldhorst, Lieven MK Vandersypen, Masoud Babaie, et al. A scalable cryo-cmos 2-to-20ghz digitally intensive controller for 4× 32 frequency multiplexed spin qubits/transmons in 22nm finfet technology for quantum computers. In *2020 IEEE International Solid-State Circuits Conference, ISSCC 2020*, pages 304–306. Institute of Electrical and Electronics Engineers (IEEE), 2020.

[PWD16] Alexandru Paler, Robert Wille, and Simon J Devitt. Wire recycling for quantum circuit optimization. *Physical Review A*, 94(4):042337, 2016.

[Rau07] Robert Raussendorf. Fault-tolerant quantum computation with high threshold in two dimensions. *Physical Review Letters*, 98(19), 2007.

[RBM+19] MA Rol, F Battistel, FK Malinowski, CC Bultink, BM Tarasinski, R Vollmer, N Haider, N Muthusubramanian, A Bruno, BM Terhal, et al. Fast, high-fidelity conditional-phase gate exploiting leakage interference in weakly anharmonic superconducting qubits. *Physical review letters*, 123(12):120502, 2019.

[RDN+12] Matthew D Reed, Leonardo DiCarlo, Simon E Nigg, Luyan Sun, Luigi Frunzio, Steven M Girvin, and Robert J Schoelkopf. Realization of three-qubit quantum error correction with superconducting circuits. *Nature*, 482(7385):382–385, 2012.

[RHR+04] Mark Riebe, H Häffner, CF Roos, W Hänsel, J Benhelm, GPT Lancaster, TW Körber, C Becher, F Schmidt-Kaler, and DFV James. Deterministic quantum teleportation with atoms. *Nature*, 429(6993):734–737, 2004.

[Roo08] Christian F Roos. Ion trap quantum gates with amplitude-modulated laser beams. *New Journal of Physics*, 10(1):013002, 2008.

[ROT+18] Matthew Reagor, Christopher B Osborn, Nikolas Tezak, Alexa Staley, Guenevere Prawiroatmodjo, Michael Scheer, Nasser Alidoust, Eyob A Sete, Nicolas Didier, Marcus P da Silva, et al. Demonstration of universal parametric entangling gates on a multi-qubit lattice. *Science advances*, 4(2):eaao3603, 2018.

[RS14] Neil J Ross and Peter Selinger. Optimal ancilla-free clifford+ t approximation of z-rotations. *arXiv preprint arXiv:1403.2975*, 2014.

[S+13] C. M. Shappert et al. Spatially uniform single-qubit gate operations with near-field microwaves and composite pulse compensation. *New Journal of Physics*, 15(083053), 2013.

[SAC+06] K.M. Svore, A.V. Aho, A.W. Cross, I. Chuang, and I.L. Markov. A layered software architecture for quantum computing design tools. *Computer*, 39(1):74–83, 2006.

[Sch90] Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148. Society for Industrial and Applied Mathematics, 1990.

[SDT06] Krysta M Svore, David P DiVincenzo, and Barbara M Terhal. Noise threshold for a fault-tolerant two-dimensional lattice architecture. *arXiv preprint quant-ph/0604090*, 2006.

[Sel13] Peter Selinger. Quantum circuits of t-depth one. *Physical Review A*, 87(4), 2013.

[SGM+20] Yunong Shi, Pranav Gokhale, Prakash Murali, Jonathan M Baker, Casey Duckering, Yongshan Ding, Natalie C Brown, Christopher Chamberland, Ali Javadi-Abhari, Andrew W Cross, et al. Resource-efficient quantum computing by breaking abstractions. *Proceedings of the IEEE*, 108(8):1353–1370, 2020.

[Sho94a] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[Sho94b] Peter W Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 124–134. IEEE, 1994.

[Sho95] Peter W Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

[Sho99] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[SHR18] Mathias Soeken, Thomas Haener, and Martin Roetteler. Programming quantum computers using design automation. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 137–146. IEEE, 2018.

[SHT18] Damian S Steiger, Thomas Häner, and Matthias Troyer. Projectq: an open source software framework for quantum computing. *Quantum*, 2:49, 2018.

[Sim97] Daniel R Simon. On the power of quantum computation. *SIAM journal on computing*, 26(5):1474–1483, 1997.

[Sim16] Tom Simonite. Google's quantum dream machine. *TECHNOLOGY REVIEW*, 119(2):17–19, 2016.

[SJM+14] Daniel Sank, Evan Jeffrey, JY Mutus, TC White, J Kelly, R Barends, Y Chen, Z Chen, B Chiaro, A Dunsworth, A Megrant, P. J. J O'Malley, C Neill, P Roushan, A Vainsencher, J Wenner, A. N Cleland, and John M Martinis. Fast scalable state measurement with superconducting qubits. *arXiv preprint arXiv:1401.0257*, 2014.

[SK+03] Ferdinand Schmidt-Kaler et al. Realization of the Cirac–Zoller controlled-NOT quantum gate. *Nature*, 422(6930):408–411, 2003.

[SKF+13] M. Suchara, J. Kubiatowicz, A. Faruque, F. T. Chong, C. Y. Lai, and G. Paz. Qure: The quantum resource estimator toolbox. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 419–426, Oct 2013.

[SLG+19] Yunong Shi, Nelson Leung, Pranav Gokhale, Zane Rossi, David I Schuster, Henry Hoffmann, and Frederic T Chong. Optimized compilation of aggregated instructions for realistic quantum computers. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 1031–1044, 2019.

[SM13] Mehdi Saeedi and Igor L Markov. Synthesis and optimization of reversible circuits - a survey. *ACM Computing Surveys (CSUR)*, 45(2):21, 2013.

[SMCG16] Sarah Sheldon, Easwar Magesan, Jerry M Chow, and Jay M Gambetta. Procedure for systematically tuning up cross-talk in the cross-resonance gate. *Physical Review A*, 93(6):060302, 2016.

[SRWDM17] Mathias Soeken, Martin Roetteler, Nathan Wiebe, and Giovanni De Micheli. Hierarchical reversible logic synthesis using luts. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 78. ACM, 2017.

[SSA13] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. In *Proceedings of the eighteenth international conference on Architectural support for programming languages and operating systems*, ASPLOS '13, pages 305–316, New York, NY, USA, 2013. ACM.

[SSM+07] Steven Swanson, Andrew Schwerin, Martha Mercaldi, Andrew Petersen, Andrew Putnam, Ken Michelson, Mark Oskin, and Susan J. Eggers. The WaveScalar architecture. *ACM Trans. Comput. Syst.*, 25(2):4:1–4:54, May 2007.

[SSP14] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Qubit placement to minimize communication overhead in 2d quantum architectures. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 495–500. IEEE, 2014.

[SSSV14] Seung Woo Shin, Graeme Smith, John A Smolin, and Umesh Vazirani. How" quantum" is the d-wave machine? *arXiv preprint arXiv:1401.7087*, 2014.

[Ste96a] A. Steane. Error correcting codes in quantum theory. *Physical Review Letters*, 77(5):793–797, 1996.

[Ste96b] Andrew M Steane. Simple quantum error-correcting codes. *Physical Review A*, 54(6):4741, 1996.

[Ste97a] A. M. Steane. Active stabilization, quantum computation, and quantum state synthesis. *Phys. Rev. Lett.*, 78:2252–2255, Mar 1997.

[Ste97b] Andrew Steane. Space, time, parallelism and noise requirements for reliable quantum computing. *arXiv preprint quant-ph/9708021*, 1997.

[Ste04] Andrew M Steane. How to build a 300 bit, 1 giga-operation quantum computer. *arXiv preprint quant-ph/0412165*, 2004.

[SV06] Ethan Schuchman and T. N. Vijaykumar. A program transformation and architecture support for quantum uncomputation. *SIGARCH Comput. Archit. News*, 34(5):252–263, October 2006.

[SVB+14] G Shu, G Vittorini, A Buikema, CS Nichols, C Volin, D Stick, and Kenneth R Brown. Heating rates and ion-motion control in a y-junction surface-electrode trap. *Physical Review A*, 89(6):062308, 2014.

[SWD10] Mathias Soeken, Robert Wille, and Rolf Drechsler. Hierarchical synthesis of reversible circuits using positive and negative davio decomposition. In *Design and Test Workshop (IDT), 2010 5th International*, pages 143–148. IEEE, 2010.

[T+11] Ole Tange et al. GNU Parallel-the Command-Line Power Tool. *The USENIX Magazine*, 36(1):42–47, 2011.

[Ter15] Barbara M Terhal. Quantum error correction for quantum memories. *Reviews of Modern Physics*, 87(2):307, 2015.

[TLM+04] Michael Bedford Taylor, Walter Lee, Jason Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matt Frank, Saman Amarasinghe, and Anant Agarwal. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *Proceedings of the*

31st Annual International Symposium on Computer Architecture, ISCA '04, pages 2–. IEEE Computer Society, 2004.

[TMC+06] Darshan D Thaker, Tzvetan S Metodi, Andrew W Cross, Isaac L Chuang, and Frederic T Chong. Quantum memory hierarchies: Efficient designs to match available parallelism in quantum computing. In *33rd International Symposium on Computer Architecture (ISCA'06)*, pages 378–390. IEEE, 2006.

[TMN+17] Swamit S Tannu, Zachary A Myers, Prashant J Nair, Douglas M Carmean, and Moinuddin K Qureshi. Taming the instruction bandwidth of quantum computers via hardware-managed error correction. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 679–691. ACM, 2017.

[TQ18] Swamit S Tannu and Moinuddin K Qureshi. A case for variability-aware policies for nisq-era quantum computers. *arXiv preprint arXiv:1805.10224*, 2018.

[Tro59] Hale F Trotter. On the product of semi-groups of operators. *Proceedings of the American Mathematical Society*, 10(4):545–551, 1959.

[TS14] Yu Tomita and Krysta M Svore. Low-distance surface codes under realistic quantum noise. *Physical Review A*, 90(6):062320, 2014.

[VPK+17] R Versluis, S Poletto, N Khammassi, B Tarasinski, N Haider, DJ Michalak, A Bruno, K Bertels, and L DiCarlo. Scalable quantum circuit and control for a superconducting surface code. *Physical Review Applied*, 8(3):034021, 2017.

[W+09] Mark G. Whitney et al. A fault tolerant, area efficient architecture for shor's factoring algorithm. In *ISCA*, 2009.

[W+11] Nathan Wiebe et al. Simulating quantum dynamics on a quantum computer. *Journal of Physics A: Mathematical and Theoretical*, 44(44):445308, 2011.

[W+14] Nathan Wiebe et al. Quantum nearest-neighbor algorithms for machine learning. *arXiv preprint arXiv:1401.2142*, 2014.

[WAF+13] Kenneth Wright, Jason M Amini, Daniel L Faircloth, Curtis Volin, S Charles Doret, Harley Hayden, CS Pai, David W Landgren, Douglas Denison, Tyler Killian, Richart Slusher, and Alexa Harter. Reliable transport through a microfabricated x-junction surface-electrode ion trap. *New Journal of Physics*, 15(3):033004, 2013.

[WBAG10] James D. Whitfield, Jacob Biamonte, and Alan Aspuru-Guzik. Simulation of electronic structure Hamiltonians using quantum computers. *Molecular Physics*, 109(5):735, 2010.

[WBAG11] James D Whitfield, Jacob Biamonte, and Alán Aspuru-Guzik. Simulation of electronic structure hamiltonians using quantum computers. *Molecular Physics*, 109(5):735–750, 2011.

[WBC+14] Dave Wecker, Bela Bauer, Bryan K Clark, Matthew B Hastings, and Matthias Troyer. Gate-count estimates for performing quantum chemistry on small quantum computers. *Physical Review A*, 90(2):022305, 2014.

[WBD+19] K Wright, KM Beck, S Debnath, JM Amini, Y Nam, N Grzesiak, J-S Chen, NC Pisenti, M Chmielewski, C Collins, et al. Benchmarking an 11-qubit quantum computer. *Nature Communications*, 10(1):1–6, 2019.

[WBS14] Jonathan Welch, Alex Bocharov, and Krysta M Svore. Efficient approximation of diagonal unitaries over the clifford+ t basis. *arXiv preprint arXiv:1412.5608*, 2014.

[WDD+] Xin-Chuan Wu, Dripto M Debroy, Yongshan Ding, Jonathan M Baker, Yuri Alexeev, Kenneth R Brown, and Frederic T Chong. Tilt: Achieving higher fidelity on a trapped-ion linear-tape quantum computing architecture. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 153–166. IEEE.

[Wei13] Yaakov S Weinstein. How often must quantum error correction be implemented? *arXiv preprint arXiv:1305.2763*, 2013.

[WMI+97] David J Wineland, C Monroe, WM Itano, D Leibfried, BE King, and DM Meekhof. Experimental issues in coherent quantum-state manipulation of trapped atomic ions. *arXiv preprint quant-ph/9710025*, 1997.

[WMI+98] DJ Wineland, C Monroe, WM Itano, BE King, D Leibfried, DM Meekhof, C Myatt, and C Wood. Experimental primer on the trapped ion quantum computer. *Spectroscopy*, 7:8, 1998.

[WOC+13] U. Warring, C. Ospelkaus, Y. Colombe, K. R. Brown, J. M. Amini, M. Carsjens, D. Leibfried, and D. J. Wineland. Techniques for microwave near-field quantum control of trapped ions. *Phys. Rev. A*, 87:013437, Jan 2013.

[WP67] Dominic JA Welsh and Martin B Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1):85–86, 1967.

[WR14] Nathan Wiebe and Martin Roetteler. Quantum arithmetic and numerical analysis using repeat-until-success circuits. *arXiv preprint arXiv:1406.2040*, 2014.

[WS14a] Dave Wecker and Krysta M Svore. Liqui—¿: A software design architecture and domain-specific language for quantum computing. *arXiv preprint arXiv:1402.4467*, 2014.

[WS14b] Dave Wecker and Krysta M Svore. Liqui—¿;: A software design architecture and domain-specific language for quantum computing. *arXiv preprint arXiv:1402.4467*, 2014.

[WSB+04] Andreas Wallraff, David I Schuster, Alexandre Blais, L Frunzio, R-S Huang, J Majer, S Kumar, Steven M Girvin, and Robert J Schoelkopf. Strong coupling of a single photon to a superconducting qubit using circuit quantum electrodynamics. *Nature*, 431(7005):162–167, 2004.

[WZR+12] Andreas Walther, Frank Ziesel, Thomas Ruster, Sam T Dawkins, Konstantin Ott, Max Hettrich, Kilian Singer, Ferdinand Schmidt-Kaler, and Ulrich Poschinger. Controlling fast transport of cold trapped ions. *Physical review letters*, 109(8):080501, 2012.

[XLL+15] Xiaolong Xie, Yun Liang, Xiuhong Li, Yudong Wu, Guangyu Sun, Tao Wang, and Dongrui Fan. Enabling coordinated register allocation and thread-level parallelism optimization for gpus. In *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, pages 395–406, New York, NY, USA, 2015. ACM.

[YG92] Tao Yang and A Gerasoulis. PYRROS: static task scheduling and code generation for message passing multiprocessors. In *Proceedings of the 6th ACM International Conference on Supercomputing*, pages 428–437, 1992.

[YG93] Tao Yang and Apostolos Gerasoulis. List scheduling with and without communication delays. *Parallel Comput.*, 19(12):1321–1344, December 1993.

[ZBL20] Alexander Zlokapa, Sergio Boixo, and Daniel Lidar. Boundaries of quantum supremacy via random circuit sampling. *arXiv preprint arXiv:2005.02464*, 2020.

[ZCC11] B. Zeng, A. Cross, and I. L. Chuang. Transversality versus universality for additive quantum codes. *IEEE Transactions on Information Theory*, 57(9):6272–6284, Sept 2011.

[ZMH18] Gengyan Zhang, Pranav S Mundada, and Andrew A Houck. Suppression of qubit crosstalk in a tunable coupling superconducting circuit. *arXiv preprint arXiv:1810.04182*, 2018.