THE UNIVERSITY OF CHICAGO

NEW ABSTRACTIONS FOR QUANTUM COMPUTING

A DISSERTATION PROPOSAL
FOR CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
CASEY DUCKERING

CHICAGO, ILLINOIS
DECEMBER 2022

TABLE OF CONTENTS

ABSTRACT

The field of quantum computing is at an exciting time where we are building devices, running programs, and finding out what works best. As qubit technology grows and matures, we need to be ready to design and program larger quantum computer systems. An important aspect of systems design is layered abstractions to reduce complexity and guide intuition. Classical computer systems have built up many abstractions over its history including the layers of the hardware stack and programming abstractions like loops. Researchers initially ported these abstractions with little modification when designing quantum computer systems and only in recent years have some of those abstractions been broken in the name of optimization and efficiency. We argue that new or quantum-tailored abstractions are needed to get the most benefit out of quantum computer systems. We keep the benefits gained through breaking old abstraction by finding abstractions aligned with quantum physics and the technology. This dissertation is supported by three examples of abstractions that could become a core part of how we design and program quantum computers: third-level logical state as scratch space, memory as a third spacial dimension for quantum data, and architecture-tuned hierarchical programs.

CHAPTER 1
INTRODUCTION

Moore's Law and the expectation that computers double in speed every 18 months is at an end, so hard problems in chemistry, physics simulation, and combinatorial optimization cannot be solved by waiting for a faster computer. Since the end of Moore's Law, researchers have been developing special-purpose accelerators to squeeze better performance out of each transistor. However once fully realized, quantum computers can solve specific classes of problems in simulation and cryptography exponentially faster.

Quantum computers work by harnessing quantum physics instead of classical Newtonian physics. Because quantum physics is a superset of classical physics, we often treat quantum computers as classical computers with the additional features of *superposition*, *entanglement*, and *interference*. This view is apparent in Shor's algorithm which creates a quantum superposition, followed by classical arithmetic, and finishes with quantum phase estimation.

Seeing quantum programming through a classical lens can be limiting and sometimes harmful. It is common for programmers who are new to quantum to invent a "quantum algorithm" that is simply a randomized classical algorithm run on a quantum computer. More subtly, concepts such as binary representation of data, random access memory, and hierarchical modularity of programs when used in the design of quantum computers limit the performance due to mismatches with the underlying technology. Even classical concepts of causality and movement of data can be limiting; quantum teleportation moves quantum data long distances by pre-transferring another resource *before* the data exists.

When we design quantum architectures and compilers, the abstractions we use are key to a good design. The abstraction of two-level *bits* is very beneficial to for classical computer reliability but is yet to be decided for quantum. Early classical computers used base-10 addresses and arithmetic until early computer architects settled on binary as the most efficient and reliable design. This history informs the general assumption that binary (base-2) is best for quantum computers but that is not necessarily the case. We discuss this further in Chapter 2.

Because quantum computing is a rapidly developing field with many competing technologies there is no clear "best" for any use case. Each quantum technology has capabilities and constraints that inform a variety of architectures that show how to turn a qubit technology into a practical quantum computer. The principles of abstraction and modularity we use to build any complex system still apply when we design a quantum computer architecture, instruction set, compiler, and programming language but we must tailor the abstractions to best fit the physics and the technology or we will limit future efficiency.

This dissertation presents three cases of new or old abstractions that we have tailored for quantum computing. We discuss the methodologies to select these abstractions and how we use them with a particular class of quantum architectures. We show that good abstractions can allow more space efficient algorithms or more effective compilers.

This dissertation comprises three core papers introducing three abstractions covered in the following chapters. Several additional papers are included that show further benefits and refinement to the abstractions. We start in Chapter 2 by introducing three-level quantum *trits* and d-level quantum *dits*: Asymptotic Improvements to Quantum Circuits via Qutrits **?**. These abstractions replace and augment the use of binary qubits with three-level *qutrits* or d-level *qudits* but require us to completely rethink how algorithms and compilers allocate and use scratch space. Most quantum technologies can reliably support three or more quantum states with minor changes to the control signal design and no change to the hardware design. Supported technologies include superconducting transmon, ion trap, and neutral atom, but notably not some types of photonic qubits.

Chapter 3 considers abstractions that spatially separate quantum data storage or memory from computation on that data: Virtualized Logical Qubits: A 2.5D Architecture for Error-Corrected Quantum Computing **?**. Classical computers contain high speed buses that can transfer data between memory (RAM) and computation (CPU) but this extreme separation of memory from compute does not make sense either for current small (NISQ) or for future (fault-tolerant) quantum computers. The typical abstraction for both kinds is a monolithic 2D array of qubits because NISQ computers cannot sacrifice the data-parallelism and fault-tolerance requires constant error correction to prevent errors. Compiler design is simple in this monolithic model because there is no heterogeneity; just place related data nearby in the plane. But we compare an alternative to the monolithic model. We redesign the surface code to use small amounts of distributed memory and find that it improves the space efficiency of fault-tolerant algorithms.

Classical programmers have used a hierarchy of function calls and modules in the design of a program to great effect. Hierarchy gives structure to what would otherwise be a very long list of assembly instructions. Compilers use this structure to guide optimizations and to avoid duplicate work. However, quantum programmers are now trending toward highly hand-optimized programs with no hierarchy for small to mid-size programs. Chapter 4 introduces Orchestrated Trios: Compiling for Efficient Communication in Quantum Programs with 3-Qubit Gates to show that hierarchy can help guide quantum compiler heuristics even for these mid-size programs.

Picking the right abstractions are crucial for quantum programming, compiling, and execution. Chapter 5 concludes with a discussion and other places where we still need better abstractions.

CHAPTER 2

MULTI-VALUE QUDITS

2.1 Introduction

TODO Revise content here.

Recent advances in both hardware and software for quantum computation have demonstrated significant progress towards practical outcomes. In the coming years, we expect quantum computing will have important applications in fields ranging from machine learning and optimization Biamonte et al. [2016] to drug discovery Kassal et al. [2010]. While early research efforts focused on longer-term systems employing full error correction to execute large instances of algorithms like Shor factoring Shor [1995] and Grover search Grover [1996], recent work has focused on NISQ (Noisy Intermediate Scale Quantum) computation Preskill [2018]. The NISQ regime considers near-term machines with just tens to hundreds of quantum bits (qubits) and moderate errors.

Given the severe constraints on quantum resources, it is critical to fully optimize the compilation of a quantum algorithm in order to have successful computation. Prior architectural research has explored techniques such as mapping, scheduling, and parallelism Ding et al. [2018], Javadi-Abhari et al. [2017], Guerreschi and Park [2017] to extend the amount of useful computation possible. In this work, we consider another technique: quantum trits (qutrits).

While quantum computation is typically expressed as a two-level binary abstraction of qubits, the underlying physics of quantum systems are not intrinsically binary. Whereas classical computers operate in binary states at the physical level (e.g. clipping above and below a threshold voltage), quantum computers have natural access to an infinite spectrum of discrete energy levels. In fact, hardware must actively suppress higher level states in order to achieve the two-level qubit approximation. Hence, using three-level qutrits is simply a choice of including an additional discrete energy level, albeit at the cost of more opportunities for error.

Prior work on qutrits (or more generally, d-level *qudits*) identified only constant factor gains from extending beyond qubits. In general, this prior work Pavlidis and Floratos [2017] has emphasized the information compression advantages of qutrits. For example, N qubits can be expressed as $\frac{N}{\log_2(3)}$ qutrits, which leads to $\log_2(3) \approx 1.6$-constant factor improvements in runtimes.

Our approach utilizes qutrits in a novel fashion, essentially using the third state as temporary storage, but at the cost of higher per-operation error rates. Under this treatment, the runtime (i.e. circuit depth or critical path) is *asymptotically* faster, and the reliability of computations is also improved. Moreover, our approach only applies qutrit operations in an intermediary stage: the input and output are still qubits, which is important for initialization and measurement on real devices Randall et al. [2015, 2018].

The net result of our work is to extend the frontier of what quantum computers can compute. In particular, the frontier is defined by the zone in which every machine qubit is a data qubit, for example a 100-qubit algorithm running on a 100-qubit machine. This is indicated by the yellow region in Figure **??**. In this frontier zone, we do not have room for non-data workspace qubits known as ancilla. The lack of ancilla in the frontier zone is a costly constraint that generally leads to inefficient circuits. For this reason, typical circuits instead operate below the frontier zone, with many machine qubits used as ancilla. Our work demonstrates that ancilla can be substituted with qutrits, enabling us to operate efficiently within the ancilla-free frontier zone.

We highlight the three primary contributions of our work:

1. A circuit construction based on qutrits that leads to asymptotically faster circuits ($633N \rightarrow 38 \log_2 N$) than equivalent qubit-only constructions. We also reduce total gate counts from $397N$ to $6N$.

2. An open-source simulator, based on Google's Cirq Cir [2018], which supports realistic noise simulation for qutrit (and qudit) circuits.

3. Simulation results, under realistic noise models, which demonstrate our circuit construction outperforms equivalent qubit circuits in terms of error. For our benchmarked circuits, our reliability advantage ranges from 2x for trapped ion noise models up to more than 10,000x for superconducting noise models. For completeness, we also benchmark our circuit against a qubit-only construction augmented by an ancilla and find our construction is still more reliable.

The rest of this paper is organized as follows: Section **??** presents relevant background about quantum computation and Section **??** outlines related prior work that we benchmark our work against. Section **??** demonstrates our key circuit construction, and Section **??** surveys applications of this construction toward important quantum algorithms. Section **??** introduces our open-source qudit circuit simulator. Section **??** explains our noise modeling methodology (with full details in Appendix **??**), and Section **??** presents simulation results for these noise models. Finally, we discuss our results at a higher level in Section **??**.

CHAPTER 3
2.5D MEMORY

3.1 Introduction

TODO Revise content here.

Quantum devices have improved significantly in the last several years both in terms of physical error rates and number of usable qubits. For example, IBM and others have made accessible via the cloud several devices with 5 to 53 qubits with moderate error rates **?**. Concurrently, great progress has been made at the software level such as improved compilation procedures reducing required overhead for program execution. These efforts are directed at enabling NISQ (Noisy Intermediate-Scale Quantum) **?** algorithms to demonstrate the power of quantum computing. Machines in this era are expected to run some important programs and have recently been used to by Google to demonstrate “quantum supremacy” **?**.

Despite this, these machines will be too small for error correction and unable to run large-scale programs due to unreliable qubits. The ultimate goal is to construct fault-tolerant machines capable of executing thousands of gates and in the long-term to execute large-scale algorithms such as Shor’s **?** and Grover’s **?** with speedups over classical algorithms. There are a number of promising error correction schemes which have been proposed such as the color code **?** or the surface code **???**. The surface code is a particularly appealing candidate because of its low overhead, high error threshold, and its reliance on few nearest-neighbor interactions in a 2D array of qubits, a common feature of superconducting transmon qubit hardware. In fact, Google’s next milestone is to demonstrate error corrected qubits **??**.

Current architectures for both NISQ and fault-tolerant quantum computers make no distinction between the memory and processing of quantum information (represented in qubits). While currently viable, as larger devices are built, the engineering challenges of scaling up to hundreds of qubits becomes readily apparent. For transmon technology used by

Google, IBM, and Rigetti, some of these issues include fabrication consistency and crosstalk during parallel operations. Every qubit needs dedicated control wires and signal generators which fill the refrigerator the device runs in. To scale to the millions of qubits needed for useful fault-tolerant machines **?**, we need to a memory-based architecture to decouple qubit-count from transmon-count.

In this work, we use a recently realized qubit memory technology which stores qubits in a superconducting cavity **?**. This technology, while new, is expected to become competitive with existing transmon devices. Stored in cavity, qubits have a significantly longer lifetime (coherence time) but must be loaded into a transmon for computation. Although the basic concept of a compute qubit and associated memory has been demonstrated experimentally, the contribution of our work is to design and evaluate a system-level organization of these components within the context of a novel surface code embedding and fault-tolerant quantum operations. We provide a proof of concept in the form of a practical use case motivating more complex experimental demonstrations of larger systems using this technology.

Our proposed 2.5D memory-based design is a typical 2D grid of transmons with memory added as shown in Figure **??**. This can be compared with the traditional 2D error correction implementation in Figure **??**, where the checkerboards represent error-corrected logical qubits. The logical qubits in this system are stored at unique virtual addresses in memory cavities when not in use. They are loaded to a physical address in the transmons and made accessible for computation on request and are periodically loaded to correct errors, similar to DRAM refresh. This design allows for more efficient operations such as the transversal CNOT between logical qubits sharing the same physical address i.e. co-located in the same cavities. This is not possible on the surface code in 2D which requires methods such as braiding or lattice surgery for a CNOT operation.

We introduce two embeddings of the 2D surface code to this new architecture that spread logical qubits across many cavities. Despite serialization due to memory access, we are able to store and error-correct stacks of these logical qubits. Furthermore, we show surface code operations via lattice surgery can be used unchanged in this new architecture while also enabling a more efficient CNOT operation. Similarly, we are able to use standard and architecture-specific magic-state distillation protocols **?** in order to ensure universal computation. Magic-state distillation is a critical component of error-corrected algorithms so any improvement will directly speed up algorithms including Shor's and Grover's.

We discuss several important features of any proposed error correction code, such as the threshold error rate (below which the code is able to correct more errors than its execution causes), the code distance, and the number of physical qubits to encode a logical qubit. In many codes, the number of physical qubits can be quite large. We develop an embedding from the standard representation to this new architecture which reduces the required number of physical transmon qubits by a factor of approximately k, the number of resonant modes per cavity. We also develop a Compact variant saving an additional 2x. This is significant because we can obtain a code distance $\sqrt{2k}$ times greater or use hardware with only $\frac{1}{2k}$ the required physical transmons for a given algorithm. In the near-to-intermediate term, when qubits are a highly constrained resource this will accelerate a path towards fault-tolerant computation. In fact, the smallest instance of Compact requires only 11 transmons and 9 cavities for k logical qubits.

We evaluate variants of our architecture by comparing against the surface code on a larger 2D device. Specifically, we determine the error correction threshold rates via simulation for each and find they are all close to the baseline threshold. This shows the additional error sources do not significantly impact the performance. We explore the sensitivity of the threshold to many different sources of error, some of which are unique to the memory used in this architecture. We end by evaluating magic-state distillation protocols which have a large impact on overall algorithm performance and find a 1.22x speedup normalized by the number of transmon qubits.

In summary, we make the following contributions:

- We introduce a 2.5D architecture where qubit-local memory is used for random access to error-corrected, logical qubits stored across different memories. This allows a simple virtual and physical address scheme. Error correction is performed continuously by loading each from memory.

- We give two efficient adaptations of the surface code in this architecture, Natural and Compact. Unlike a naive embedding, both support fast transversal CNOTs in addition to lattice surgery operations with improved connectivity between logical qubits.

- We develop an error correction implementation optimized for Compact and designed to maximise parallelism and minimize the spread of errors.

- Via simulation, we determine the surface code adapted to our 2.5D architecture is still an effective error correction code while greatly reducing hardware requirements.

CHAPTER 4

ORCHESTRATED TRIOS

4.1 Introduction

TODO Revise content here.

In recent years, quantum hardware has improved dramatically in terms of number of accessible quantum bits (qubits), device error rates, and qubit lifetimes. However, we are still years away from obtaining fully error corrected devices which are required to run important algorithms like Grover's **?** and Shor's **?**. In the current Noisy-Intermediate Scale Quantum (NISQ) Preskill [2018] era, where despite recent substantial improvements, error rates on current devices are still prohibitive, requiring programs to be highly optimized to have a good chance at succeeding.

Quantum program compilation involves many passes of transformations and optimizations similar in many ways to classical compilers. Some optimizations occur at the abstract circuit level, independent of the underlying hardware, such as gate cancellation **?**. One of the first steps usually taken is to convert an input program into a gate set (ISA) supported by the target hardware. For example, on IBM devices, gates are typically rewritten using only gates in the set $\{u1, u2, u3, cx\}$ **?** (single-qubit gates and the common CNOT gate described later). One critical limitation of many current available architectures is the inability to execute more complex multi-qubit operations, like the Toffoli, directly; instead, these gates must be decomposed into the supported one- and two-qubit gates. Furthermore, many current superconducting architectures only support two qubit operations on adjacent hardware qubits wired together with a coupler. This requires the insertion of additional operations called SWAPs to move the data onto adjacent (and connected) qubits.

The process of transforming an optimized and decomposed program to the desired target is typically broken down into three distinct steps: decomposing the program into basic gates, mapping the logical qubits of a program to hardware qubits and routing interacting qubits so that they are adjacent on hardware when they interact, and scheduling operations in order to minimize total program run time (depth) or to minimize errors due to crosstalk **?**. Each of these steps is critical to the success of the input program. A well-mapped and well-routed program will reduce the total number of communication operations added and subsequently reduce the compiled program's depth, both of which will increase the chance of success. Conventionally, these three steps occurs sequentially. By doing so, current strategies are unable to account for structure in the input program, resulting in inefficient routing of qubits. An optimal compiler could find the best routing despite the lack of structure but at the cost of much slower compilation. Consider the SWAP paths inserted by IBM's Qiskit compiler for a single Toffoli compiled to IBM's Johannesburg device in Figure **??**a. This baseline strategy adds a large number of unnecessary SWAPs as it individually routes each CNOT composing the Toffoli, dramatically reducing the probability of successful execution.

Our approach, Orchestrated Trios (Trios) decomposes and routes qubits in multiple stages, as seen in Figure **??**b. For example, first decompose an input program to one- two-, *and* three-qubit gates (e.g. do not decompose Toffoli gates) and route as before except for three-qubits, route all three to a common location with minimal SWAPs. This new program can then undergo a second round of decomposition to produce a circuit containing only hardware permitted one- and two-qubit gates. The second round may use the now known mapping (locations of data qubits on the device) to generate fine-tuned decompositions for the architecture.

This layered approach has a major advantage over current routing techniques: we are better able to capture program structure by inspecting intermediate complex operations for routing. This better informs how qubits should be moved around the device during program execution. In Figure **??**, the Trios strategy reduces the total number of SWAPs added to 21: fewer than half compared to Qiskit. This was an extreme example we selected to present the issue, not an average case.

We specifically propose a two-pass approach to circuit decomposition. We will focus on superconducting hardware systems like IBM's cloud accessible devices, but our strategy can easily be adapted to other systems. An overview of our compilation structure is found in Figure **??**b. This strategy has a substantial benefit on the overall success rate of programs. We demonstrate these improvements by executing Toffoli gates on a real IBM quantum computer and estimating success probability of a suite of benchmarks via simulation.

Our contributions are as follows:

- A new compiler structure, Trios, with two passes for decomposition with a modified routing pass in between which greatly improves qubit routing.
- A simple method for architecture-tuned Toffoli decompositions during the second decompose pass that allows for a new kind of location-aware optimization.
- On Toffoli-only experiments, Trios reduces the total number of gates by 35% geomean (geometric mean) resulting in 23% geomean increase in success rate when run on real IBM hardware as compared to Qiskit.
- On near-term algorithms shown in Figure **??** (4 to 20 qubit benchmarks), Trios reduces total gate count by 37% geomean resulting in 344% geomean increase in (or 4.44x) simulated success rate on IBM Johannesburg with noise rates of near-future hardware as compared to programs compiled without Trios. A sensitivity analysis over four architecture types shows the benefit range from 133% to 3020% increase in success rate.

CHAPTER 5
CONCLUSION

From the three cases we present and evaluate, we find that the right abstraction enables system design, compiler design, and program design to align. Treating extra logical levels as scratch space enables programs with simplified communication patterns. Qubit-local memory enables a third dimension, improving spatially-local computation. And finally, re-introducing the hierarchical program abstraction improves compiler heuristics and enables additional cross-pass optimizations.

These abstractions and others will hopefully be integrated into the quantum computer scientist's hardware stack and toolchain. As the stack evolves, quantum computing will grow more into its own niche as a fundamentally new computing paradigm and be more than simply a classical computer with superposition.

CHAPTER 6

TIMELINE

For completion of this dissertation, all research has been completed and all that remains is to finish writing this draft. The remaining parts to complete are the body content of Chapters 2, 3, and 4.

- Complete draft for feedback (finish content in the three main chapters) - Mid July
- Finish writing - August 5th
- Defense - Mid August (will schedule soon)
- Incorporate any committee feedback - September 2nd

REFERENCES

Cirq: A python framework for creating, editing, and invoking noisy intermediate scale quantum (nisq) circuits. https://github.com/quantumlib/cirq, 2018.

Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. 2016. doi: 10.1038/nature23474.

Yongshan Ding, Adam Holmes, Ali Javadi-Abhari, Diana Franklin, Margaret Martonosi, and Frederic T. Chong. Magic-state functional units: Mapping and scheduling multi-level distillation circuits for fault-tolerant quantum architectures, 2018.

Lov K. Grover. A fast quantum mechanical algorithm for database search, 1996.

Gian Giacomo Guerreschi and Jongsoo Park. Two-step approach to scheduling quantum circuits. 2017. doi: 10.1088/2058-9565/aacf0b.

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. 2017. doi: 10.1145/3123939.3123949.

Ivan Kassal, James D. Whitfield, Alejandro Perdomo-Ortiz, Man-Hong Yung, and Alán Aspuru-Guzik. Simulating chemistry using quantum computers. 2010. doi: 10.1146/annurev-physchem-032210-103512.

Archimedes Pavlidis and Emmanuel Floratos. Arithmetic circuits for multilevel qudits based on quantum fourier transform, 2017.

John Preskill. Quantum computing in the nisq era and beyond. 2018. doi: 10.22331/q-2018-08-06-79.

J. Randall, S. Weidt, E. D. Standing, K. Lake, S. C. Webster, D. F. Murgia, T. Navickas, K. Roth, and W. K. Hensinger. Efficient preparation and detection of microwave dressed-state qubits and qutrits with trapped ions. *Phys. Rev. A*, 91:012322, 01 2015. doi: 10.1103/PhysRevA.91.012322. URL https://link.aps.org/doi/10.1103/PhysRevA.91.012322.

J. Randall, A. M. Lawrence, S. C. Webster, S. Weidt, N. V. Vitanov, and W. K. Hensinger. Generation of high-fidelity quantum control methods for multilevel systems. *Phys. Rev. A*, 98:043414, 10 2018. doi: 10.1103/PhysRevA.98.043414. URL https://link.aps.org/doi/10.1103/PhysRevA.98.043414.

Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. 1995. doi: 10.1137/S0097539795293172.