Methods for parallel quantum circuit synthesis, fault-tolerant quantum RAM, and quantum state tomography

by

Olivia Di Matteo

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Physics - Quantum Information

Waterloo, Ontario, Canada, 2019

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Seth Lloyd Professor, Massachusetts Institute of Technology
Supervisor:	Michele Mosca Professor, University of Waterloo
Internal Member:	Roger Melko Professor, University of Waterloo
Internal-External Member:	Richard Cleve Professor, University of Waterloo
Other Member:	Kevin Resch Professor, University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The content of Chapter 3 and Appendix A was previously published as [1]:

O. Di Matteo, M. Mosca (2016) *Parallelizing quantum circuit synthesis.* Quantum Science and Technology **1** (1)

The early stages were included in ODM's MSc thesis [2]:

O. Di Matteo. *Parallelizing quantum circuit synthesis.* Master's thesis, University of Waterloo, Waterloo ON, April 2015.

Author contributions

The project was proposed by Michele Mosca.

ODM developed the parallel circuit synthesis framework.

ODM derived the analytical expressions for the algorithm scaling.

ODM wrote the software; high-performance computing technical support was provided on the Blue Gene/Q by Barbara Collignon.

ODM wrote the paper with input from Michele Mosca and Barbara Collignon.

The content of Section 3.7 was previously published as [3]:

O. Di Matteo, D. Z. Djokovic, I. Kotsireas (2015) *Symmetric Hadamard matrices of order 116 and 172 exist.* Special Matrices, **3** (1).

Author contributions

The project was proposed by Ilias Kotsireas and Dragomir Djokovic.

Ilias Kotsireas and Dragomir Djokovic constructed new Hadamard matrices of orders 92 and 172.

ODM adapted the parallel circuit synthesis framework of [1] to solve a 3-way matching problem that found symmetric Hadamard matrices of order 116.

All authors contributed to the writing of the paper.

A paper based on the content of Chapter 4 is in preparation at the time of writing.

Author Contributions

The project was proposed by Michele Mosca and Vlad Gheorghiu, as a follow up to previous work [4].

ODM and Vlad Gheorghiu designed the circuits with input from Michele Mosca.

ODM performed the analytical resource estimates.

ODM wrote the accompanying software for the fault-tolerant resource estimates with input from Vlad Gheorghiu.

ODM wrote the first drafts of the paper; all authors are contributing to the final version.

The content of Chapter 5, Appendix B and Appendix C was previously published as [5]:

O. Di Matteo, L. L. Sánchez-Soto, G. Leuchs, M. Grassl (2017) *Coarse-graining the phase space of N qubits*. Phys. Rev. A **95** 022340

Author contributions

The project was proposed by Markus Grassl and Luis Sánchez-Soto.

The key analytical expressions were derived by ODM and Markus Grassl.

ODM performed the calculations and made the figures for the provided examples.

ODM wrote the accompanying software.

Luis Sánchez-Soto and ODM wrote the paper with input from Markus Grassl.

Abstract

The pace of innovation in quantum information science has recently exploded due to the hope that a quantum computer will be able to solve a multitude of problems that are intractable using classical hardware. Current quantum devices are in what has been termed the "noisy intermediate-scale quantum", or NISQ stage. Quantum hardware available today with 50-100 physical qubits may be among the first to demonstrate a quantum advantage. However, there are many challenges to overcome, such as dealing with noise, lowering error rates, improving coherence times, and scalability.

We are at a time in the field where minimization of resources is critical so that we can run our algorithms sooner rather than later. Running quantum algorithms "at scale" incurs a massive amount of resources, from the number of qubits required to the circuit depth. A large amount of this is due to the need to implement operations *fault-tolerantly* using error-correcting codes.

For one, to run an algorithm we must be able to efficiently read in and output data. Fault-tolerantly implementing quantum memories may become an input bottleneck for quantum algorithms, including many which would otherwise yield massive improvements in algorithm complexity. We will also need efficient methods for tomography to characterize and verify our processes and outputs. Researchers will require tools to automate the design of large quantum algorithms, to compile, optimize, and verify their circuits, and to do so in a way that minimizes operations that are expensive in a fault-tolerant setting. Finally, we will also need overarching frameworks to characterize the resource requirements themselves. Such tools must be easily adaptable to new developments in the field, and allow users to explore tradeoffs between their parameters of interest.

This thesis contains three contributions to this effort: improving circuit synthesis using large-scale parallelization; designing circuits for quantum random-access memories and analyzing various time/space tradeoffs; using the mathematical structure of discrete phase space to select subsets of tomographic measurements. For each topic the theoretical work is supplemented by a software package intended to allow others researchers to easily verify, use, and expand upon the techniques herein.

Acknowledgments

I would like to thank my supervisor, Michele Mosca, for his guidance throughout my grad school career. I would also like to thank the members of my committee, Richard Cleve, Seth Lloyd, Roger Melko, and Kevin Resch for taking the time to read and discuss my work.

Thanks to all my wonderful co-authors for their contributions to the work mentioned within: Matthew Amy, Hubert de Guise, Dragomir Djokovic, Vlad Gheorghiu, Markus Grassl, Ilias Kotsireas, Gerd Leuchs, Michele Mosca, Alex Parent, Luis Sánchez-Soto, John Schanck.

I thank Matthew Amy, Vadym Kliuchnikov, Mathias Soeken, and Tom Draper for discussions about quantum circuit synthesis and optimization that were helpful for the work of Chapter 3 and Chapter 4.

Thanks to Michele Mosca for connecting myself and Ilias Kotsireas, which led to the work of Section 3.7, and to Gary Graham for suggestions on how to set up SQL databases.

Computing resources throughout were provided by SHARCNET, SOSCIP, and Scinet. I am grateful to Barbara Collignon for providing technical support on the Blue Gene/Q platform for the work in Chapter 3.

Many thanks to Hubert de Guise, Jean-Philippe MacLean, and Michele Mosca for their feedback on the early versions of this thesis.

Thank you to Chin Heng Lee for keeping the group organized and running smoothly, and always being around to help out.

The work in this thesis was completed in many different places. Thanks to Luis, Markus, and Gerd at the Max Planck Institute for the Science of Light for hosting me thrice. A portion of the parallel circuit synthesis work was completed while attending the Quantum Computer Science workshop at the Banff International Research Station (17-22 April 2016).

Funding was provided by NSERC, CIFAR, and SOSCIP. IQC is supported in part by the Government of Canada and the Province of Ontario. SOSCIP is funded by the Federal Economic Development Agency of Southern Ontario, the Province of Ontario, IBM Canada Ltd., Ontario Centres of Excellence, Mitacs and 15 Ontario academic member institutions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The current state of quantum computers

The work presented in this thesis was completed over the course of roughly three years, from 2015 until 2018. During this time, the pace of innovation in the field of quantum information and computation exploded, shifting from a largely academic venture to a full-blown multi-billion dollar industrial one. Major technology companies such as Google, Microsoft, IBM, Intel, Alibaba, Baidu, have all made significant investments in the development of quantum technology. Simultaneously a diverse ecosystem of quantum startups has emerged, ranging from the broad, with full software and hardware stacks, to more specific endeavours focused on cryptography, cloud-based hardware, or algorithms. The number of qubits usable for universal computing has gone from a handful to Google's recent 72, with even larger systems planned such as a 128-bit chip from the startup Rigetti.

Even with such rapid progress, there is still much work to be done. For one, existing machines tend to be noisy. As an example, the IBM Quantum Experience machines are frequently calibrated (sometimes multiple times a day) due to decoherence, and have single-qubit gate errors of 10^{-3} and two-qubit gate errors of 10^{-2} [6]. Such "noisy intermediate-scale quantum" (NISQ) devices, and what we can use them for, are a topic under active exploration [7].

The scalability of fabrication processes is another barrier, with some implementations being far more scalable than others. For example, D-Wave has manufactured specialized chips for quantum annealing with as many as 2000 superconducting qubits. Hardware for universal computing is currently being designed with between 50 and 100 qubits. If they are to run production-level quantum algorithms fault-tolerantly, that is, able to withstand the inevitable errors present in such delicate processes, millions or even billions of qubits are required.

Discussion of increasing machine size begets the idea of quantum *advantage* - how many qubits are required to accomplish a task that no classical computer can accomplish? Early estimates put this threshold at around 50 qubits. However, recent efforts in the area of quantum simulation have shown using various techniques that, given enough classical

processors [1] and memory, it is possibly to simulate roughly 50 qubits [8–11], and so perhaps this threshold must be raised.

The availability of such simulators is a blessing, in that even when we don't have access to a powerful supercomputer, algorithms on 30 or so qubits can be run one's laptop, providing researchers a testbed to ensure that their algorithms work as intended. Given the sometimes unintuitive nature of quantum mechanics, this is an indispensable resource. However, to design by hand an algorithm on 30 qubits down to to the level of individual gates is no easy task. Moreover, surely at some point the size of physical implementations will surpass what can accomplish with simulators, and what will happen then?

Researchers require tools to automate the design of algorithms and circuits. We will need quantum programming languages, in which a user can write code at a high level, coupled with compilers that decompose the algorithms in to the 'assembly language' of a universal quantum gate set. Furthermore we will need tools for verification of these compilers for when simply plugging it into a simulator becomes intractable.

Moving from NISQ-era machines towards fully fault-tolerant computation will entail a massive overhead in the number of qubits required. We will need additional qubits in special circuits to prepare resource states, and additional qubits for embedding into error correcting codes. We will need automated tools to optimize our algorithms to minimize the amount of these additional resources, and a framework in which we can easily vary physical parameters to produce resource estimates.

Finally, we will need efficient tools for the input and output processes of quantum computers. On the input side, storing and loading data requires a quantum memory, which we will soon see is non-trivial to implement. As for the output, the number of measurements required to fully characterize an unknown multi-qubit state grows exponentially in the number of qubits. We must find efficient means of reconstruction that use fewer measurements, but still retain useful information. Ongoing work is being done on all these fronts, and the work included in this thesis is a contribution to these efforts.

1.2 This thesis

We begin in Chapter 3 with quantum circuit synthesis, the decomposition of arbitrary operations into those of a universal gate set, accomplished using advanced parallel computation techniques. We demonstrate in Section 3.7 how these same techniques can be applied to the search for large symmetric Hadamard matrices in dimensions in which their existence was previously unknown. In Chapter 4 we perform fault-tolerant resource estimation of quantum random access memories. We will explore questions such as "how many seconds will it take to query an 8 'gigabyte' quantum RAM?". Finally in Chapter 5 we present a systematic method to choose a subset of the tomographic measurements required to fully reconstruct a quantum state. We show that this method, based on the discrete

[1] In some cases, as many as one can get on the Sunway TaihuLight, which at the time was the world's most powerful supercomputer.

Wigner function and the underlying mathematical structure of the space in which qubits live, produces 'coarse-grained' reconstructions that retain some key features of the original states.

The unifying theme for this collection of papers is that they are all quantum analogs of key processes and devices found in classical computers, as highlighted in Figure 1.1. Quantum circuit synthesis is a large component of the quantum *compiler*. Our resource estimation techniques focus on a quantum *RAM*. Finally, reading in data using quantum RAM and output characterization with quantum tomography can be considered as *I/O* processes.

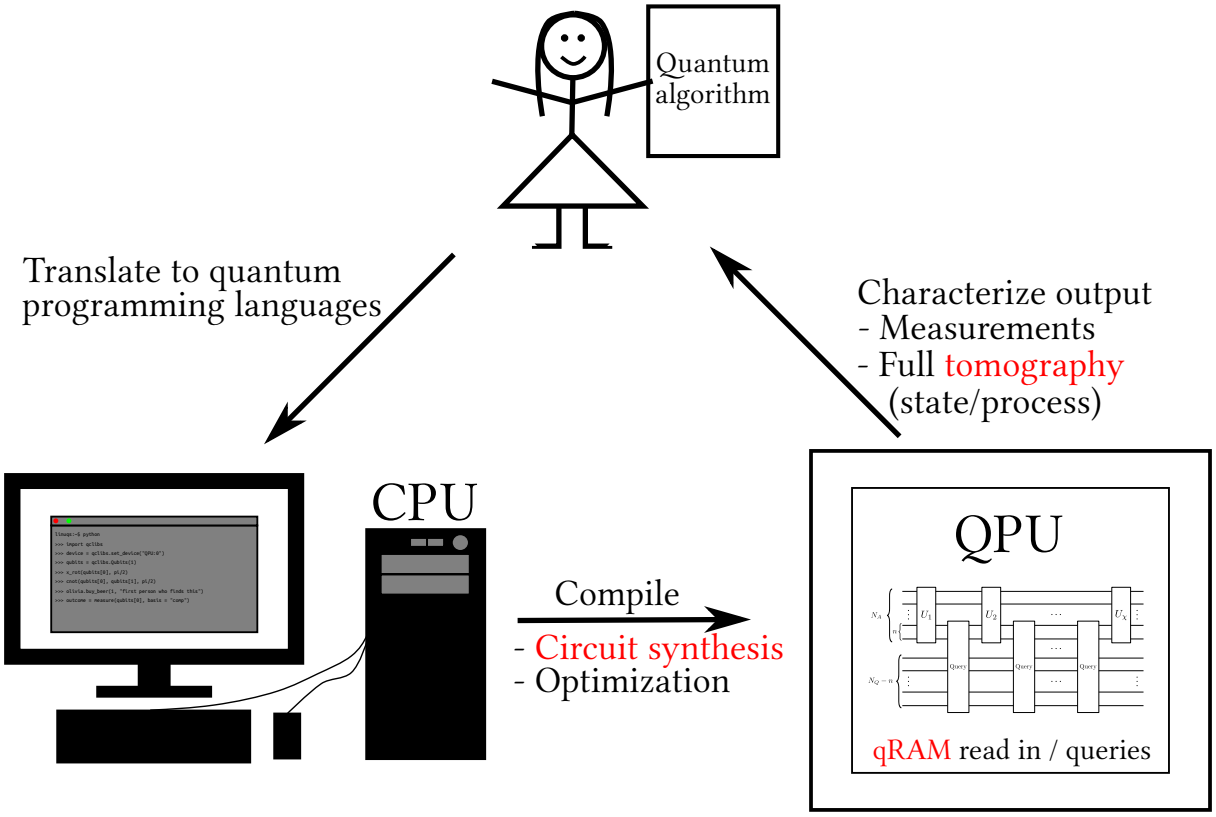


Figure 1.1: What it might look like to run an algorithm on a quantum computer. A user writes up an algorithm in a high-level language on a classical computer. This is then compiled down to the native gate set of the quantum processing unit (QPU) using circuit synthesis and optimization. An algorithm running on the QPU may need to read in and query data in superposition using a quantum random-access memory. After the computation, tomography/measurements are performed, sending classical data back to the user.

These techniques were all designed with the exponential scaling of qubit systems in mind. As physically realizable quantum systems get bigger, we will need to use parallel computing techniques to keep up; we will need tools that tell us whether our algorithm requires millions of qubits, or trillions; and we may need to reduce the number of measurements we take to make an algorithm run time more tractable.

As a final note, every paper included here comes bundled with a software package. Cumulatively they contain over 5000 lines of C++ and Python code, as well as over 3000 lines of supporting documentation. They are available online for anyone to download, and contain usage instructions and examples. They are key components of each work; not only do they provide transparency and an immediate means for other researchers to use and verify results, they also serve as a tangible workbench for the theoretical ideas and circuits in this thesis.

1.3 Other work

In addition to the four papers highlighted in this thesis, I participated in a variety of other collaborations. Two have led to published work (discussed below); two more are on-going, one with a manuscript in preparation.

The resource estimation techniques that will be used in Chapter 4 were originally developed to analyze the application of Grover's algorithm to a pre-image attack on the SHA family of hash functions [12]. We designed a framework and cost model to assess the level of threat posed to SHA by a quantum computer. We constructed reversible circuits for the SHA256 and SHA3-256 functions, synthesized and optimized them over the Clifford+T gate set, and finally embedded them into a surface code to compute the amount of physical qubits and time required. We found, under an optimistic set of assumptions about the error rates and operational speed of a surface code, that performing such an attack on SHA256 (SHA3-256) would require on the order of 10 (100) million physical qubits and take on the order of 10^{32} (10^{29}) years. These functions, widely in use today, are therefore in no immediate danger of being broken by quantum devices.

In [13], we developed a recursive procedure to decompose $SU(n)$ transformations into $SU(2)$ transformations. This was accomplished by factorizing the $SU(n)$ transformations into a single $SU(2)$ transformation sandwiched between two $SU(n-1)$ transformations, which can themselves be decomposed in the same manner. Such decompositions are used, for example, in linear optical networks to turn arbitrary operations on n modes into operations on only 2 modes, which are implemented physically using beam splitters. A particularly nice property of our decomposition is that all the transformations occur on adjacent modes and this greatly simplifies experimental setups. We showed how the same recursive structure can be used to compute the Haar measure of $U(n)$. It thus serves as a convenient tool to parameterize the most probable regions of the space of Haar-random unitary matrices. We provided a Python implementation of the factorization, which can be found at `https://github.com/glassnotes/Caspar`.

Chapter 2

Background

This chapter provides a very brief introduction to quantum computation, and background information on the key topics of the subsequent chapters: quantum circuit synthesis, resource estimation, quantum RAM, and quantum tomography.

2.1 Quantum computation in a nutshell

We begin with the humble *bit*, the fundamental unit of information that is either 0 or 1. The value of a bit is typically based on a physical quantity, such as if some voltage is above or below a threshold value. In quantum computation the bit is augmented to a quantum bit, or *qubit*.

A qubit is a two-level system that can be controlled quantum mechanically. The levels may be the ground and excited states of an atom, or the spin-up and spin-down states of a spin-1/2 particle. Independent of the physical implementation we express these two levels as the quantum states $|0\rangle$ and $|1\rangle$.

Under the hood, the theory of quantum computation is largely based on linear algebra. Quantum states are represented as vectors, and we typically make the correspondence

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{2.1}$$

These states live in a 2-dimensional complex vector space called the *Hilbert space*.

We commonly use $\{|0\rangle, |1\rangle\}$ as the basis vectors of the Hilbert space, though this choice is by no means unique. Linear combinations of these vectors,

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \qquad \alpha, \beta \in \mathbb{C}, \tag{2.2}$$

also live in this space. This ability to prepare states in a *superposition* of basis states is a critical difference between classical and quantum computing, and all successful quantum algorithms are built to take advantage of this principle.

States that can be represented by a single vector are known as *pure* states (as opposed to mixed states, which we will encounter in Section 2.4). Equation (2.2) is the most general expression for a single-qubit pure state.

Measurement outcomes of qubits are probabilistic. The complex numbers α and β are called probability amplitudes, and contain information about the likelihood of the qubit being in either $|0\rangle$ or $|1\rangle$ after a measurement in that basis:

$$\begin{aligned} \text{Prob}(|0\rangle) &= |\langle 0|\psi\rangle|^2 = |\alpha|^2 \quad (2.3) \\ \text{Prob}(|1\rangle) &= |\langle 1|\psi\rangle|^2 = |\beta|^2 \quad (2.4) \end{aligned}$$

After a measurement the system remains in whatever state was obtained as the outcome. Typically we impose the additional constraint that our states are normalized such that the probabilities satisfy $|\alpha|^2 + |\beta|^2 = 1$.

Multi-qubit states are formed by composing the underlying Hilbert spaces with the tensor product: $\mathcal{H}_{12} = \mathcal{H}_1 \otimes \mathcal{H}_2$. One can create multi-qubit states by simply tensoring together the vectors of single-qubit ones. For example,

$$|01\rangle = |0\rangle \otimes |1\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}. \quad (2.5)$$

However, there also exist multi-qubit states that cannot be created by tensoring two vectors:

$$|\Phi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}. \quad (2.6)$$

The measurement outcomes of this state are correlated in the classical sense - if we measure the first qubit in the computational basis and find it in $|0\rangle$, we know immediately that the second qubit will also be found in state $|0\rangle$. However as this state cannot be factored as a tensor product state, it is impossible to describe the qubits individually. To describe the state of the system we must describe it as a *whole*, and so we call such states *entangled*. Entangled qubits are used as a resource in numerous quantum information protocols, such as quantum teleportation wherein a qubit state can be transferred between two parties if they each hold one of the qubits of an entangled pair $|\Phi\rangle$.

Armed with the knowledge of superposition and entanglement, we move forward and discuss how qubit systems evolve. Suppose we have a qubit whose state, for simplicity, we assume to be pure. This state is time-dependent, $|\psi(t)\rangle$, and as with other (closed) quantum systems its evolution is governed by Schrödinger's equation:

$$i\hbar \frac{d|\psi(t)\rangle}{dt} = \hat{H}|\psi(t)\rangle. \quad (2.7)$$

$\hat{H}$ is a *Hamiltonian* containing information about the energy of the system; $\hat{H}$ is Hermitian and its real eigenvalues are interpreted as the possible outcomes of measuring the energy. We can solve Schrödinger's equation to obtain

$$|\psi(t')\rangle = e^{-i\hat{H}(t'-t)}|\psi(t)\rangle. \tag{2.8}$$

When $\hat{H}$ is Hermitian, $U = e^{-i\hat{H}(t'-t)}$ is *unitary* ($UU^\dagger = \mathbb{1}$), and so qubits evolve under unitary transformations. Such evolution is reversible, and preserves the length of vectors and the angles between them. Normalized quantum states evolve into other normalized quantum states, so that the measurement probabilities after an operation still sensibly add to 1. Multi-qubit operations can be created under composition by the tensor product, or they may be entangling gates, i.e. gates that send some non-entangled multi-qubit state to an entangled one. Finally, unitary operations are linear:

$$U\left(\alpha|0\rangle + \beta|1\rangle\right) = \alpha U|0\rangle + \beta U|1\rangle, \tag{2.9}$$

and so they act distributively on all the constituent states in a superposition.

2.2 Quantum circuit synthesis

We will focus for a while on the unitary operations that closed off the previous section. A quantum computer must be able to physically realize such unitary operations on its qubits. However, there are many different physical implementations of quantum computers. Unitary operations are applied to superconducting flux qubits using electromagnetic pulses, whereas photonic qubits are mathematically rotated in Hilbert space using physical objects such as beamsplitters. Thus it is natural that each system may have a specific set of unitaries that it is able to implement well (i.e. with high fidelity), but others perhaps not so well. It is necessary, then, to develop methods to decompose arbitrary operations, or sequences of operations comprising a *quantum circuit*, into those that can be done well for a specific implementation.

In a sense, this process of *quantum circuit synthesis* is analogous to the task of a compiler. Some high-level operation must be broken down into the set of operations that the machine can understand. Of course, modern compilers have other functions as well, such as lexing and parsing code written by a human, so in this regard, circuit synthesis will be just one facet of a future full-scale quantum compiler.

At a more fundamental level, there exists the idea of a *universal gate set*. This is a finite set of unitary operations, or gates, that can be used to implement any other unitary up to arbitrary precision ε:

$$||U_{true} - U_{synth}|| < \varepsilon, \tag{2.10}$$

where U_{synth} is the version that we have *synthesized* from the gates in the universal set.

For the simplest case of a single qubit, the operations

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}, \quad T = \begin{pmatrix}1 & 0\\ 0 & e^{\frac{i\pi}{4}}\end{pmatrix} \tag{2.11}$$

can be combined to produce any other single-qubit operation within some tolerance ε (and up to a global phase). Depending on the size of ε, this sequence may be prohibitively long, but it is always possible to find one. Mathematically the set $\{H, T\}$ is *dense* in the space of single-qubit unitaries, meaning that for any ε, an arbitrary unitary operation is always within ε of some operation from the gate set.

For multiple qubits, the universal set of a single qubit must be augmented by some entangling gate, as we mentioned in the previous section that multi-qubit states are not necessarily simple tensor products. A common choice is $\{H, T, \text{CNOT}\}$, where

$$\text{CNOT} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}. \tag{2.12}$$

This is known as the *Clifford + T* gateset, as H, $S = T^2$, and CNOT are the generators that define the Clifford group.

Classical algorithms for circuit synthesis have been well-studied over the years, in particular for the case of a single qubit [14–21]. There exists the Solovay-Kitaev algorithm which can synthesize over any fixed, finite gate set using a sequence of $O(\log^{3.97}(1/\varepsilon))$ gates [16]. It was later shown that this can be reduced to $O(\log(1/\varepsilon))$ for the special case of the Clifford+T gate set [14, 22]. Specialized techniques exist also for other universal gate sets, such as the V-basis [17, 19]. The V-basis is given by gates of the form

$$V_k = \frac{1}{\sqrt{5}}\left(\mathbb{1} + 2i\sigma_k\right), \quad \sigma_k \in \{X, Y, Z\}, \tag{2.13}$$

along with their inverses, and synthesis is performed over the V_k and the Pauli operations.

Multi-qubit synthesis is a much harder problem. There exist some classes of algorithms that decompose over CNOT gates and single-qubit rotations using cosine-sine decomposition techniques [23–25], as well as specialized techniques for diagonal operations [26].

Multi-qubit synthesis techniques over Clifford+T depend strongly on number theory to find nearby operations in the gate set which can then be synthesized *exactly* [27–30]. There is a special set of multi-qubit unitaries for which we are guaranteed to be able to synthesize exactly [15, 27]. Their matrix entries must be of the form

$$U_{mn} = \frac{a + b\sqrt{2} + ci + di\sqrt{2}}{2^k}, \quad a, b, c, d \in \mathbb{Z}, k \geq 0, \tag{2.14}$$

i.e. they are elements of the ring $\mathbb{Z}\left[i, \frac{1}{\sqrt{2}}\right]$. Furthermore, in order for synthesis to be performed 'in-place', without using any extra ancillary qubits, the determinant of U must

be in the set of determinants obtainable from combinations of gates in the universal set. These conditions are consequences of the fact that 1) all matrix entries in $\{H, T, \text{CNOT}\}$ are in the ring $\mathbb{Z}\left[i, \frac{1}{\sqrt{2}}\right]$, and thus so are elements in products of these matrices, and 2) the determinant of a matrix product is the product of determinants.

Exact synthesis scales exponentially in the number of qubits, and typically relies on some form of search technique [29, 30]. Of course one could brute force this search, taking every possible combination of gates on every possible combination of qubits, building up a circuit until the desired operation is found. However this scales poorly; thankfully there are more clever ways of searching.

The work presented in Chapter 3 incorporates a few such ways into a new framework for quantum circuit synthesis that leverages parallel computing techniques. It adapts a parallel search algorithm, originally used in cryptanalysis, for circuits by defining mappings that send circuits to and from binary strings. Running on roughly 4,000 processors of a Blue Gene/Q, it was able to synthesize from scratch a circuit with 4 qubits and 7 T gates, surpassing the previous result of 3 qubits and 7 T gates. In Section 2.3 and Subsection 3.4.1 we will see why the number of T gates is an important metric. While a difference of one qubit may seem trivial, one must recall the exponential scaling, and also that there is great potential for more - 4,000 processors is a small amount compared to what is available on many modern supercomputers.

2.3 Quantum RAM and resource estimation

2.3.1 The case for quantum RAM

Random-access memory (RAM) is essential to classical computers. Information about the current state of a calculation or application is stored as bits in an array of transistors and capacitors. Each bit is addressable and its value can be either queried or overwritten. RAM is alive, in the sense that it is being constantly powered and refreshed to counteract the leakage of the capacitors.

Similar to RAM is read-only memory (ROM). Here the data is written once and can be queried, but (in its simplest form) it cannot be overwritten. ROM retains its state even when there is no source of power. While RAM is perhaps now the more plentiful resource, ROM still has its place in the storage of fixed data, such as in firmware for single-purpose appliances, or storage of look-up tables for cryptographic algorithms and mathematical functions.

In the future quantum computers will require a quantum memory, though the context in which we use it may differ from that of a classical computer. We focus here on quantum memories that will function as oracles to query stored data, rather than to store the activate state of a computation. Such memories may hold classical information (which we will denote qROM, or qRAM), or quantum states (QROM or QRAM).

In Chapter 4 we will design and analyze families of circuits where the information stored is classical. An isolated circuit of this form may be considered a qROM, as we will

design them by hardcoding in the addresses in which 1s are stored. However there may also be algorithms in which the memory contents will change; then we must be constantly updating the circuitry, more in the spirit of a qRAM.

The key difference between classical and quantum memories is that quantum memories will not be limited to querying one location at a time, but instead will query in superposition. We must design circuits to perform the mapping

$$\sum_j \alpha_j |j\rangle|0\rangle \rightarrow \sum_j \alpha_j |j\rangle|b_j\rangle \tag{2.15}$$

where $\sum_j \alpha_j |j\rangle$ represents a superpositions of addresses, and the state $|b_j\rangle$ (where $b_j \in \{0,1\}$) represents the contents of the address $|j\rangle$.

Architectures for qRAM began to emerge roughly a decade ago with the bucket brigade model of [31,32]. In this model, 2^n bits of classical data (that we would like to quantumly query) are stored in the 2^n leaves of a binary tree. The nodes of the tree relate to the address, with the j-th address bit corresponding to the j-th layer of the tree, as will be described below.

At each node is a three-level state (a qutrit), with levels denoted by $|wait\rangle, |left\rangle, |right\rangle$. All the qutrits begin in $|wait\rangle$. Address qubits are then sent through one by one and the j-th bit follows a path to the j-th layer. When a qubit reaches a qutrit in its destination layer, it changes the state of that qutrit. An address bit in 0 initiates a unitary operation that sends $|wait\rangle \rightarrow |left\rangle$, while a 1 sends $|wait\rangle \rightarrow |right\rangle$. To reach their destination layers, the qubit simply follows the path directed by the qutrits. After all the address qubits are sent, a quantum 'bus' traverses the path, couples to the desired memory cell to gather the data, and is reflected back the way it came. All the nodes are then re-initialized to the $|wait\rangle$ state.

When the number of queries to the qRAM is small, the overall computation will be resilient to a sufficiently small error per query. Even though there are an exponential number of nodes $(2^n - 1)$, we are performing a number of operations that is polynomial in n and so they can have error rates on the order of $O(1/n)$ [32].

The situation is different, however, in cases where (for example) exponentially many qRAM queries are necessary. Under assumptions made in [4], we would need error rates that are exponentially small. This may necessitate the use of full fault-tolerant error correction, which incurs an enormous resource overhead.

The scale of this overhead is what we address in the work within. It is currently a very pertinent question, as numerous quantum algorithms [33–40] rely on the availability of large amounts of classical or quantum data that can be queried by a quantum computer. In many cases, the algorithms see a theoretical quantum speedup, but only under the assumption that a qRAM can be queried efficiently.

We will perform resource estimates on generic (unoptimized) versions of qRAM circuits of our own design, as well as for bucket brigade style circuits as presented in [4]. We will analyze the cost when we make tradeoffs between the number of qubits vs. circuit depth, and see some special cases of address structures where we can significantly reduce the

number of resources. These preliminary estimates paint a somewhat bleak picture of the feasibility of a fault-tolerant error-corrected qRAM - when all the fault-tolerant machinery is included, we obtain query times on the order of milliseconds but require millions, or even quadrillions of qubits, depending on the size and sparsity of the database. Compared to classical RAM in which the query time is on the order of 150 ns, querying a qRAM may prove to be a bottleneck in the quantum algorithms that rely on it.

2.3.2 Resource estimation pipeline

Though the work of Chapter 4 pertains specifically to a qRAM, the idea of resource estimation in quantum computing is a more general one. As the size of available quantum computing devices increases, it will become even more important. Some quantum algorithms, in particular Shor's algorithm, have serious implications for modern-day cryptography. It is thus necessary to calculate how large of a quantum computer we will need to run it, and how much time its execution will take.

The resource estimation techniques used in Chapter 4 were originally developed for use in [12], wherein the resources required to perform a pre-image search on the cryptographic hash functions SHA-256 and SHA3-256 were computed. In this work, we constructed a pipeline for resource estimation that divides the process into a number of layers, beginning at the very broad algorithmic level, down to the level of individual qubits.

While we leave most of the details to [12], we present here a brief description of the key components of this pipeline:

a) Algorithms and the classical query model,

b) Circuits and the logical layer,

c) Error correction and the fault-tolerant layer,

d) The physical layer.

This pipeline is versatile, and modular in that it can be followed to arbitrary depth. In some cases we may only be interested in parameters of the circuits used at the logical layer; in others we may want a very fine-grained analysis in which we perform a massive amount of circuit optimization and count the individual qubits needed to embed it into an error-correcting code. Afterwards we discuss the final metric, the overall cost, a numerical measure of the time and space requirements of an algorithm.

2.3.2.a Algorithms and the classical query model

This stage is the most high-level, and is essentially a measure of 'how many times do we need to run or query a black box'? In some cases, we may be interested in only a single iteration of an algorithm or circuit. In others, we may need to perform them many times.

For example, consider Grover's algorithm for quantum search. A search through a space of size 2^N can be performed using $2^{N/2}$ executions of something called the Grover iterate. In the case of [12], the Grover iterate consisted of two instances of SHA, followed by a smaller subroutine called the Grover diffusion operator.

However, simply stopping at "we can do it with $2^{N/2}$ queries" ignores a significant portion of the work, namely all the subroutines that are involved in each query. The next stages of the pipeline go deeper into the inner workings of this black box.

2.3.2.b Circuits and the logical layer

At the logical layer, we analyze the particular quantum circuit that performs the query. Here we compute parameters such as the number of qubits required, the overall depth of the circuit, and the number of each gate required. Typically we will also perform circuit synthesis down to our desired gate set, and then circuit optimization in order to eliminate any redundant sequences of gates. We may also make tradeoffs at this stage: we can add additional qubits in order to parallelize certain portions of the circuit and reduce the depth (and as a consequence, time).

Often these processes are done with respect to the Clifford+T gate set. For this set, it is critical to minimize parameters relating to the T gate: the T-count, and the T-depth, i.e. the number of layers of depth in which T gates are performed. The reasons for this will be discussed in the next section.

2.3.2.c Error correction and the fault-tolerant layer

In Section 1.1, it was mentioned that quantum computers today suffer from a great deal of noise. This noise can compromise the implementation of many algorithms as they cause the decoherence of states in superposition. As we scale up and begin to implement more sophisticated algorithms, it will be necessary to ensure that our machines are *fault-tolerant*, that is, able to withstand and correct any errors or failures that may occur. The search for fault-tolerant quantum error correcting codes is an active area of research.

At the fault-tolerant layer, we choose an error-correction scheme and compute its resources based on some physical assumptions. For example, we may assume that errors occur at a fixed rate in our system, and tailor the code parameters to take this into account. A particular parameter of importance is the *distance* of a code, as this parameter is directly related to the number of errors a code can correct. A code with distance d can correct $\lfloor (d-1)/2 \rfloor$ errors.

In order to implement error correction, we must embed the logical circuit into some correction scheme. A simple example of this process is shown in Figure 2.1. We encode our original qubits into logical ones, using some number of additional physical qubits, in such a way that the collective logical qubit will be immune to any errors. We then perform operations on these logical qubits, before decoding them back.

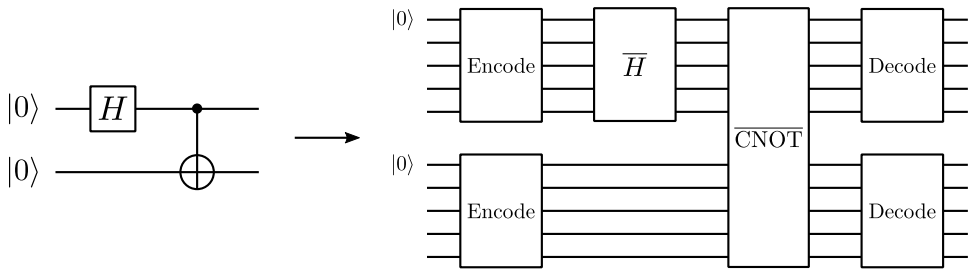


Figure 2.1: A simple example of how error correction is implemented. Left: original circuit on two qubits. Right: the two qubits are first encoded into logical states using extra physical qubits; logical equivalents of the operations are then applied, before decoding back to the original two qubits.

The error correcting code we choose for our resource estimation is the *surface code*. We will not describe the code in detail, as this is outside the scope of this thesis; a good introduction to the topic can be found in [41]. In a surface code, physical qubits are laid out in a two-dimensional lattice. Some qubits are designated *data* qubits, and are surrounded by what are called *measure* qubits. These measure qubits serve to *stabilize* the data qubits; after performing a logical operation on the system, the surrounding measure qubits will perform a stabilizer measurement of the data qubit. This is called a *surface code cycle*, and is performed simultaneously across the whole lattice. Measurement outcomes are kept track of classically and corrections are made when needed.

In the surface code framework T gates are considered expensive to implement. For every T gate performed, one needs to prepare an additional state called a magic state:

$$|M\rangle = |0\rangle + e^{\frac{i\pi}{4}}|1\rangle. \tag{2.16}$$

The production of these magic states occurs independently from the rest of the operations, and is performed in separate *factories* or *distilleries*. Each factory is embedded in its own surface code. The number of factories depends on both the T-count T_c and the T-depth T_d. In our case we use an estimate we denote as the T-width T_c/T_d. This is a rough measure of how many T gates are being done per layer of T-depth, thus determining the number of simultaneously running factories we need.

Distillation happens in a number of layers. This is determined by the initial error rate of the input magic states, and the desired output error rate. As errors are additive, the desired rate is taken to be $1/T_c$. Magic states are distilled through successive layers of surface codes, each having different distances, until states with the desired error rate are produced.

It is for these reasons that circuit synthesis and optimization techniques have been designed with the minimization of the T-count and T-depth in mind - there is a massive overhead due the number of qubits and the number of cycles to run the distilleries in a fault-tolerant scheme.

The distances of the regular circuit as well as the distilleries are the important quantities obtained at the fault-tolerant layer, and are then piped down to the physical layer.

2.3.2.d The physical layer

Here we compute the number of physical qubits required, as well as the number of surface code cycles and the overall run time.

The number of qubits is directly related to the surface code distance d. For every logical qubit, we require $2.5 \times 1.25 \times d^2$ physical qubits [41]. We must calculate this not only for the original circuit, but also for the distilleries. In most cases there are multiple layers of distillation required, and so we can re-use the qubits from the most populous layers in the subsequent ones.

The number of cycles required is different for each gate. For distillation, the number of cycles required is 10 times the sum of the distances over all the layers [41]. Using the T-depth, number of cycles, and a cycle time, we can calculate an overall time required to perform all the distillation.

We can compute similar quantities for the circuit itself. Gates all take differing amount of cycles to run: for instance, H gates require d cycles, while CNOT gates require 2 cycles [41]. One can thus produce estimates of the number of cycles required at each layer of depth, and calculate a total time. However in both the resources estimates performed in [12] and Chapter 4, the time required for distillation dwarfs the time required to run the circuit itself, once again demonstrating the need to optimize with respect to the T gates.

2.3.2.e Overall cost

Incorporation of the ideas of all the previous sections leads us to a single number. We define the *cost* as

$$\text{Cost} = \log_2 \left(\#\text{ of logical qubits} \times \#\text{ of surface code cycles}\right). \tag{2.17}$$

This clearly represents a product of space (number of qubits) and time (number of cycles), where lower costs are more desirable. The intention of the $\log_2$ was to make a clearer comparison with standard cost metrics used in other cryptanalysis techniques. These are typically given as some number 2^k invocations of the cryptographic function, where k relates to a security parameter such as key size, or hash length.

2.4 Quantum tomography

In Section 2.1, we saw quantum states represented as vectors. However, this is not the most general representation of a quantum state, as there are some states, called *mixed states* that cannot be represented by only a single vector. Instead, a mixed state is represented by a *density matrix*, typically labelled ρ. A valid ρ for a d-dimensional quantum state is a $d \times d$ Hermitian matrix that is positive semidefinite and has trace 1.

For a vector, or *pure* state $|\psi\rangle$, $\rho = |\psi\rangle\langle\psi|$. However ρ can also be expressed as a probabilistic mixture of pure states that is *not* simply an outer product of a ket and its

bra. For example,

$$\rho = \frac{1}{2}|0\rangle\langle 0| + \frac{1}{2}|1\rangle\langle 1|. \tag{2.18}$$

This state describes a system that, when measured in the computational basis, will be found half the time in state $|0\rangle$, and half the time in state $|1\rangle$. While these outcomes look identical to those of the superposition $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, the mixed state is fundamentally different in how the measurement outcomes respond to a change of basis. For example, if we measure instead in the basis $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle), |-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$, one sees (after re-expression in the new basis) that the mixed state is equally likely to be found in $|+\rangle$ as it is in $|-\rangle$, whereas the superposition is now firmly found in $|+\rangle$.

In the density matrix formalism, operation by a unitary matrix is achieved through *conjugation*, i.e. $U[\rho] = U\rho U^\dagger$. It is straightforward to see why this must be the case by noting that U is a linear operation, and that the density matrix of some pure state $|\psi\rangle$ under operation by U will be $(U|\psi\rangle) \cdot \left(\langle\psi|U^\dagger\right)$.

Suppose you are given an unknown mixed state and told to determine its density matrix by making as few quantum measurements as possible. Two obvious questions follow: how few measurements do you need, and what are they?

The answer to the first problem can be derived by a simple parameter counting argument. A $d \times d$ Hermitian matrix contains d^2 complex numbers, or $2d^2$ real numbers. The diagonal entries must be real, and so we eliminate d potential parameters. Another consequence of the matrix being Hermitian is that we only need to determine all the upper triangular entries, or all the lower triangular entries. This reduces the number of parameters by $d(d-1)$. Finally, since the trace of the matrix is 1, we can determine the final diagonal element as 1 minus the rest of them. This means that we must find $2d^2 - d - d(d-1) - 1 = d^2 - 1$ parameters. So, we should be able to determine our state by taking $d^2 - 1$ measurements.

The particular choice of measurements is a less trivial problem, but thankfully a very well-studied one. It turns out that if we take our measurements using a special set of bases, it is possible to fully reconstruct the state using $d^2 - 1$ measurements. Such bases are called *mutually unbiased*; their defining feature is that for any vector $|a\rangle$ in basis A, and any vector $|b\rangle$ in basis B,

$$|\langle a|b\rangle|^2 = \frac{1}{d}. \tag{2.19}$$

Each basis provides $d-1$ independent measurement outcomes, and we must find d^2-1 parameters. This suggests that we need $(d^2-1)/(d-1) = d+1$ such bases, all pairwise mutually unbiased. It is well-known that such a *complete set* of $d+1$ mutually unbiased bases (MUBs) exists when d is a prime or power-of-prime number. Their existence in composite dimensions is also a well-studied problem, but unfortunately a concrete proof of (non-)existence has yet to be found, even in the smallest composite dimension 6 (a valiant effort has been made over the past 15 years and suggests that no more than a triple of such bases exist there [42–50]).

MUBs, when they exist, can be constructed in a number of ways. Early construction methods created vectors entry-by-entry using finite field arithmetic [51,52]. Later methods

made use of the generalized Pauli operators. There are $d^2 - 1$ non-identity Pauli operators in dimension d [53]. They can be partitioned into $d+1$ disjoint sets of $d-1$ commuting operators. Commuting operators share eigenvectors, and the eigenvectors of each set become the mutually unbiased bases. These operators can also be considered as observables, a set of $d^2 - 1$ quantities that one measures in an experiments and can then use to reconstruct the state.

The critical problem that arises here is one of scaling. In multi-particle systems such as quantum computers, where $d = 2^n$, n being the number of qubits, we now need $2^n + 1$ bases and $2^{2n} - 1$ measurements.

It is thus crucial that we devise new tomographic techniques to mitigate this problem. Recently developed methods involve adaptive measurement schemes [54] or Bayesian inference [55, 56]. Others are based on improving maximum likelihood estimation [57], and providing a means of reconstructing a positive semidefinite matrix from only a subset of the mutually unbiased bases [58].

The work of Chapter 5 proposes a new, systematic method to select a subset of the d^2-1 generalized Pauli observables. Intuitively, it is based on the discrete Wigner function, a graphical representation of quantum states. The discrete Wigner functions are 'smoothed out' and re-expressed in an effectively smaller dimension, thus reducing the number of observables. This bears some similarity to the idea of the renormalization group in condensed matter physics, though under the hood we will see a lot of beautiful math involving finite fields and some combinatorial design theory.

Chapter 3

Parallelizing quantum circuit synthesis

The contents of this chapter from Section 3.1-Section 3.6 were published on 12 October 2016 in [1]. The associated codebase can be found at `https://qsoft.iqc.uwaterloo.ca/#software`. Additional details about our decision to use cryptographic hash function was added to the end of Section 3.2.

The contents of Section 3.7 were published on 8 October 2015 in [3].

This work provides a framework and implementation for parallel quantum circuit synthesis based on pseudorandom/deterministic walk methods. It has demonstrated the ability to synthesize, with optimal T-count, 1) known circuits faster than contemporary algorithms, and 2) larger circuits.

While the algorithm retains an exponential dependence on the number of qubits and circuit depth, it inherits the scaling of the underlying parallel claw finding algorithm. A significant majority of the algorithm takes place in parallel, and the runtime is inversely proportional to the number of processors.

The exponential scaling may prohibit synthesis of entire algorithms on more than a few tens of qubits, even on large-scale classical computing systems. However, these methods can greatly expand what can be achieved with first-generation quantum devices with tens of qubits. They can be used to optimize frequently repeated subcircuits, or combined with optimization heuristics such as resynthesis [59–61] to substantially reduce the cost of implementing much larger circuits. In particular when the limiting cost of the algorithm is T-count, reduction by even a handful of T gates has the potential for substantial savings when considered over the course of a full-scale algorithm.

The implementation was also successfully extended to the problem of finding symmetric Hadamard matrices of order 116, simultaneously answering the question of their existence in this dimension.

3.1 Introduction

Quantum computers, like their classical counterparts, will require a compiler which can translate from a human-readable input or programming language into operations which can be executed directly on quantum hardware. Circuit synthesis is an integral part of the compilation process. Given an arbitrary quantum circuit C and a universal gate set $\mathcal{G}$, one seeks to find a decomposition

$$U_k U_{k-1} \cdots U_2 U_1 = C, \quad U_i \in \mathcal{G}, \tag{3.1}$$

where k represents the depth of the circuit. A myriad of algorithms currently exist to find such a decomposition [14–17, 19–22, 26–30, 62, 63]. They are generally divided into two classes, those which synthesize approximately (i.e. $||U_k \cdots U_1 - C|| < \epsilon$) and others which synthesize exactly. Some procedures work for a single qubit, whereas others have been generalized to multiple qubits. Most of these algorithms were designed to work over the Clifford+T universal gate set, though other gate sets such as the V-basis have also been studied [17, 19].

Many of the algorithms which perform exact synthesis fall victim to the fact that the time and space used depend exponentially on both the number of qubits and the depth of the circuit in question. Even on a reasonably fast machine, synthesis of circuits with more than a handful of qubits and layers of depth becomes intractable.

In this work, we propose a method of circuit synthesis based on a heuristic search technique commonly used in cryptanalysis: collision finding based on deterministic, or pseudorandom walks. These are walks through a search space such that once a starting point is chosen, the path is completely determined. More generally, we show how we can use deterministic walks to traverse the space of possible circuits of a given depth and find solutions to the synthesis problem. A key ingredient in our method is a mapping from the unitary operators constructed from the gate set $\mathcal{G}$ to binary strings of a constant length, and a suitable mapping back to the set of unitary operators. When such mappings are defined, we can synthesize circuits over any universal gate set, on any number of qubits, by applying any existing walk method which can search the space.

The structure of this article is as follows. We begin in Section 3.2 with a discussion of deterministic walks, and how we can map quantum circuit synthesis to these types of problems. The subsequent sections pertain to our choice of implementation of one such method, namely parallel circuit synthesis. In Section 3.3 we briefly lay out the procedure for parallel synthesis and provide a runtime complexity estimate, detailing the important parameters which affect the scaling of our algorithm. Section 3.4 pertains to our software implementation, pQCS, which performs optimal T-count synthesis using parallel search. Section 3.5 contains the numerical results of large-scale experiments run on a Blue Gene/Q supercomputer. Here we showcase the significant advantages afforded to us by parallelization. We conclude in Section 3.6 and suggest avenues of future research on this topic.

3.2 Walking through circuits

Consider a hash function $h : \mathcal{D} \to \mathcal{R}$, typically considered to operate over binary strings. If h is a good hash function, then for an arbitrary input $x \in \mathcal{D}$, the value $h(x) = y \in \mathcal{R}$ will be in practice indistinguishable from a random output. Suppose there exists another function $r : \mathcal{R} \to \mathcal{D}$, unrelated to h, which maps elements of its range back to the domain (such a function is commonly termed a *reduction function*). Repeatedly applying $r \circ h$ to an input will produce a trail of points scattered throughout $\mathcal{D}$. However, once the initial input is chosen, the progression of the trail is completely determined, hence we use the term *deterministic* rather than random walk even though the path of the walk appears random due to the natures of h and r.

Such determinism has led to a set of algorithms with a variety of applications. One well-known variation is rainbow tables [64], which are used for finding pre-images of hash functions (conventionally with the intention of cracking passwords). Collision finding in one hash function, or claw finding between two functions has also been accomplished in parallel using deterministic walks [65], and was used to find collisions in double DES [66].

Deterministic walks are advantageous due to their low storage requirement: one need only store the starting point of a walk, its ending point, and the number of intermediate steps, whereas conventional search techniques would store the value of every point computed throughout.

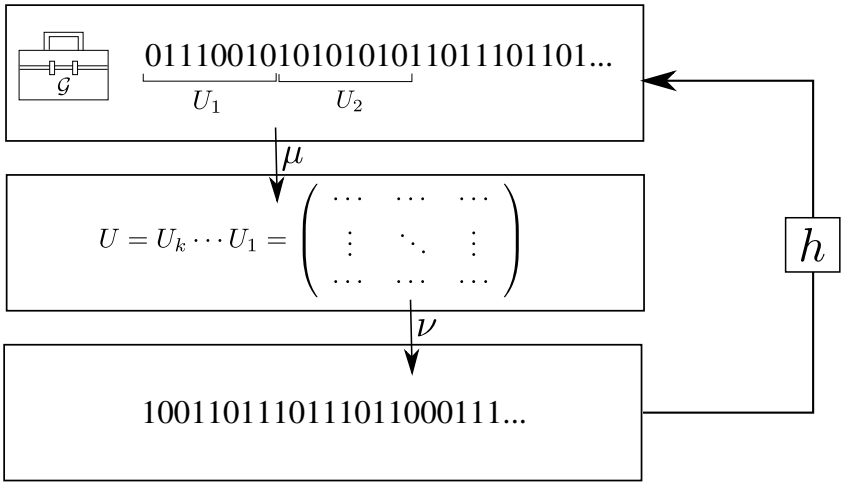


Figure 3.1: A schematic diagram showing the process of walking over circuits. Binary strings are mapped to products of unitary matrices over the gate set $\mathcal{G}$ via some correspondence μ. The product of these matrices is then mapped via ν back to a binary string, which is then passed through a hash function h. Repeated application of $h \circ \nu \circ \mu$ allows us to traverse the set of possible circuits in a pseudorandom fashion.

With this in mind, we show how one can map the problem of circuit synthesis to a problem that can be solved using an algorithm based on deterministic walks. We have, as per Equation (3.1), a product constructed from the universal gate set $\mathcal{G}$. It is possible to specify a unique way of encoding the information about $\{U_1, \ldots, U_k\}$ into binary strings

$\{\mathbf{b}_1, \ldots, \mathbf{b}_k\}$ of equal length ℓ (where we assume ℓ is sufficiently long as to encompass all the information described in what follows). Suppose $\mathcal{G}$ contains a number of single- and two-qubit gates. If we enumerate all the gates in $\mathcal{G}$, then for each U_i we might use a few bits to identify all the constituent gates, and maybe a few more to specify if we should use their Hermitian conjugates. We will also need to indicate on which qubit(s) they act. Furthermore, there must be some space to indicate controls and targets where appropriate. Given any gate set, we can find a way of doing this such that every possible U_i can be represented by a unique string $\mathbf{b}_i$. Then, the concatenation $(\mathbf{b}_k|\cdots|\mathbf{b}_1)$ will be a unique string of length $k\ell$ representing the product of unitaries $U_k \cdots U_1$.

We can perform a deterministic walk over unitary matrices as follows; this process is displayed graphically in Figure 3.1. Let us define a function μ which maps a binary string of length $k\ell$ to a unitary matrix over a specified gate set $\mathcal{G}$. Then define a mapping ν from the unitaries over $\mathcal{G}$ back to binary strings $\{0,1\}^*$. Finally, choose a good hash function h from $\{0,1\}^*$ to strings of length $k\ell$ (this may be a simple hash function, or a combination of hash and reduction-type functions). Repeatedly applying $h \circ \nu \circ \mu$ to a randomly chosen binary string of length $k\ell$ will allow us to traverse products of unitaries in a pseudorandom fashion; we can then use this to search the space of possible solutions to Equation (3.1).

In theory, one could choose other methods to walk through the unitaries. The main requirement is that the function must be deterministic, yet provide enough randomness to not significantly bias the unitaries it outputs so that the space can be searched (approximately) uniformly. Thus, we chose to represent the unitaries as bit strings and used cryptographic hash functions. Cryptographic hash functions have been optimized for speed, and intensely scrutinized for their ability to produce outputs that are hard to distinguish from true random outputs

3.3 Parallel circuit synthesis

Once we have mappings as proposed in Section 3.2, we can reformulate the circuit synthesis problem as a problem which can be solved using search algorithms based on deterministic walks. We specifically implemented one which performs parallel claw finding. Let $h_1 : \mathcal{D}_1 \to \mathcal{R}$ and $h_2 : \mathcal{D}_2 \to \mathcal{R}$ be two hash functions. A *claw* between h_1 and h_2 is a pair of inputs $x_1 \in \mathcal{D}_1, x_2 \in \mathcal{D}_2$ such that

$$h_1(x_1) = h_2(x_2). \tag{3.2}$$

This is, in a sense, a collision search between two functions.

Our interest in claw finding stems from recent work on circuit synthesis using a meet-in-the-middle (MITM) approach [29]. The motivation for that work is as follows. One can of course find a decomposition of Equation (3.1) by brute force, computing all possible combinations starting from depth 1 up until a solution is found. Let ξ represent the number of unitaries having depth 1. Typically ξ will depend exponentially on the number of qubits, n. Then, the runtime for brute force synthesis of a circuit with depth k takes time $\mathcal{O}(\xi^k)$.

A MITM approach achieves a roughly square-root speedup over this, accomplished by dividing the synthesis equation in half:

$$U_{\lceil \frac{k}{2} \rceil} \cdots U_1 = U^\dagger_{\lceil \frac{k}{2} \rceil + 1} \cdots U_k^\dagger C, \quad U_i \in \mathcal{G}. \tag{3.3}$$

Databases of unitaries having the form of each side of Equation (3.3) are sequentially constructed (starting from depth 1), stored in binary trees, and then searched through until a suitable decomposition is found. This reduces the size of the search space by a square root factor, yielding runtime $\mathcal{O}\left(\xi^{\lceil \frac{k}{2} \rceil} \log\left(\xi^{\lceil \frac{k}{2} \rceil}\right)\right)$, where the log factor is picked up due to the binary search.

To parallelize circuit synthesis, we build on the principles of the MITM algorithm. Rather than searching through static binary trees, we search the space in parallel, adapting a search technique originally developed for cryptanalysis [65]. Though our runtime will retain the exponential dependence on n and $\lceil k/2 \rceil$, it scales inversely with the number of processors, allowing us to tackle larger problems which were infeasible using previous methods, as well as speed up the synthesis of some known circuits. We provide a brief description of the algorithm here as it pertains specifically to circuit synthesis. For a more detailed description, the reader is referred to [65] or [2].

Recall Equation (3.3), and for simplicity, let us define

$$V := U_{\lceil \frac{k}{2} \rceil} \cdots U_1, \tag{3.4}$$

$$W := U^\dagger_{\lceil \frac{k}{2} \rceil + 1} \cdots U_k^\dagger C, \tag{3.5}$$

as representing the left and right sides of this equation. Define a suitable mapping between unitary matrices and binary strings of length $k\ell$ as in Section 3.2. Then let $\mathcal{V}'$ represent the set of binary strings that are of the form V, and likewise $\mathcal{W}$ those of the form W. When k is odd, $\mathcal{V}'$ and $\mathcal{W}$ may differ in size by a factor of ξ. In this case, we partition $\mathcal{V}'$ into equal sized chunks $\mathcal{V}'_0, \ldots, \mathcal{V}'_{\xi-1}$, and consider $\mathcal{V} = \mathcal{V}'_i$ independently (a search can then be executed with each $\mathcal{V}'_i$ sequentially or in parallel, adding another layer of parallelism to the implementation). When k is even, we simply let $\mathcal{V} = \mathcal{V}'$.

Let $\mathcal{N} = \{0,1\}^{k\ell}$. Define functions $z_1 : \mathcal{N} \to \mathcal{N}$ and $z_2 : \mathcal{N} \to \mathcal{N}$. One way these functions might be implemented is by converting the input string into a sequence of unitary matrices (in $\mathcal{V}$ for z_1 and $\mathcal{W}$ for z_2), computing their product, deriving a new binary string with the information about each of the matrix elements, and then running that string through a known hash function so that the outputs of both functions are in the same space and in practice appear to be random.

Let us define a 'super' function $f : \mathcal{N} \times \{1,2\} \to \mathcal{N} \times \{1,2\}$ such that one application of f is a single step in the deterministic walk, i.e. $f(x,b) = z_b(x)$. Finding a claw between z_1 and z_2 is now equivalent to finding a collision in f with distinct values for b, i.e. we must find two inputs x_1 and x_2 such that

$$f(x_1, 1) = f(x_2, 2). \tag{3.6}$$

Consider m processors all having access to a shared memory. We will denote some fraction θ of points in $\mathcal{N}$ as marked, or distinguished. Every processor chooses a random starting pair (n_0, b_0) in $\mathcal{N} \times \{1, 2\}$. Repeatedly applying f produces a *trail* through the space of possible circuits, which roughly half the time will produce a part of Equation (3.3) which is an element of $\mathcal{V}$, and the other half of the time will produce an element of $\mathcal{W}$. The trail continues until the next input, say x_d, is a distinguished point. The trail is then terminated.

The collection of found distinguished points is stored in the shared memory. Distinguished points are stored as a triple consisting of the first pair (n_0, b_0), the last pair (n_d, b_d), and the value d, which is the number of steps taken to reach the distinguished point. When a processor finishes its trail, it will attempt to add its distinguished point to the shared memory. If it sees that a trail ending at the given point is not present in this shared memory, it will insert it and then begin a new trail. However, if it sees that there is already a triple in storage which ended at the same distinguished point but had a *different* starting point, it means that somewhere along the way these two trails must have merged. The processor then takes the starting points of these two trails, and traces back through them to locate the merge point.

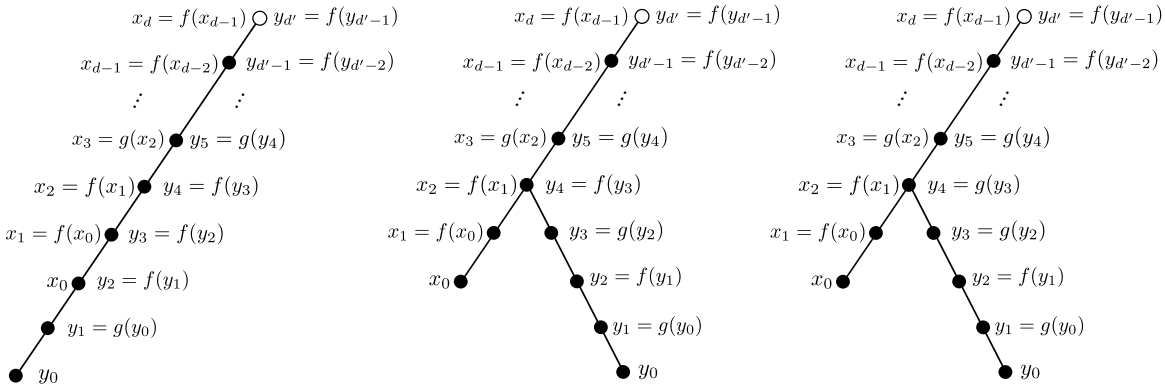


Figure 3.2: Possible ways two trails can merge. Let f and g be two functions between which we want to find a claw. (Left) One trail starts before the other. (Centre) The two trails merge after performing the same function, i.e. a collision $f(x_1) = f(y_3)$. (Right) The two trails merge after performing a different function, i.e. a claw $f(x_1) = g(y_3)$.

There are a number of possibilities here, as depicted in Figure 3.2. First, it could be that one trail started "before" the other, i.e. the merge point was at the beginning of the shorter trail. Another possibility is that when the trails merged, both had just performed z_1, or both had just performed z_2. Even if the inputs were different, this case does not provide us with a solution to the problem at hand. The final case is that immediately before they merged, one trail performed z_1 and one performed z_2; it is only in this final case that we have found a solution. With the information about the inputs in the step just before the collision, we can extract the unitary matrices from the binary string, and have fully synthesized our circuit.

The runtime complexity of this algorithm can be estimated by applying the parameters

of our problem directly to that in [65]. The size of the spaces $\mathcal{V}'$ and $\mathcal{W}$ are

$$N_{\mathcal{V}'} = \xi^{\lceil \frac{k}{2} \rceil}, \quad N_{\mathcal{W}} = \xi^{\lfloor \frac{k}{2} \rfloor}. \tag{3.7}$$

Our algorithm then scales as

$$T_{QCS} \propto \xi^{\lceil \frac{k}{2} \rceil + \frac{1}{2} \lfloor \frac{k}{2} \rfloor} \frac{1}{\sqrt{w}} \frac{1}{m} \tau, \tag{3.8}$$

where w is the number of distinguished points that can be held in memory. The parameter τ is the execution time for a single iteration of z_1 or z_2, the bulk of which will likely be spent performing matrix multiplication. Let us assume in the worst case that we are taking the product of $\lceil \frac{k}{2} \rceil$ $2^n \times 2^n$ unitaries using a multiplication algorithm which scales as $(2^n)^\alpha$, where α is some constant, typically $2 \leq \alpha \leq 3$. Thus, we obtain our final estimate

$$T_{QCS} \propto 2^{\alpha n} \xi^{(\lceil \frac{k}{2} \rceil + \frac{1}{2} \lfloor \frac{k}{2} \rfloor)} \frac{1}{\sqrt{w}} \frac{1}{m} \left\lceil \frac{k}{2} \right\rceil. \tag{3.9}$$

As previously mentioned, this time is still exponential in the number of qubits as well as the depth of the circuit. We also note that it is often the case that matrix multiplication can be parallelized, or that some specific properties of the implementation at hand (such as sparsity) can be leveraged so as to improve the scaling. What is key here is that the runtime benefits from being inversely proportional to the number of processors and available memory.

3.4 Implementation details

3.4.1 Optimal T-count synthesis

The synthesis algorithm we chose to apply our approach to is the optimal T-count algorithm presented in [30]. Such an algorithm is relevant as in many state-of-the-art methods for fault-tolerant quantum computation, T gates are considered to be expensive to implement due to the need to distill magic states (see, for example, [41]).

Let $\mathcal{P}_n$ represent the n-qubit Pauli group. We reshuffle and rewrite the decomposition of a circuit C as

$$e^{i\phi} R(P_t) \cdots R(P_1) D = C, \tag{3.10}$$

where t is the T-count, D is a Clifford, $P_i \in \mathcal{P}_n$, and

$$R(P_i) = \frac{1}{2}\left(1 + e^{\frac{i\pi}{4}}\right) I_{2^n} + \frac{1}{2}\left(1 - e^{\frac{i\pi}{4}}\right) P_i. \tag{3.11}$$

It thus suffices to find a set of t Paulis and a Clifford which will satisfy Equation (3.10) up to a global phase. The dependence on the global phase can also be removed by using the

channel representation of every matrix in the above equation:

$$\widehat{R(P_t)} \cdots \widehat{R(P_1)} \widehat{D} = \widehat{C}, \tag{3.12}$$

where the channel representation of some matrix U is the matrix with coefficients

$$\widehat{U}_{ij} = \frac{1}{2^n} \mathrm{Tr}\left(P_i U P_j U^\dagger\right), \quad P_i, P_j \in \mathcal{P}_n. \tag{3.13}$$

The channel representation of an n-qubit unitary has dimension $4^n \times 4^n$, with each row and column being indexed by a Pauli operator.

Using the optimal T-count algorithm has afforded us with a number of advantages. First of all, the T-count formulation allows us to represent each unitary matrix in the sequence as a list of n-qubit Paulis. With binary symplectic representation we can then represent each Pauli directly as a binary string, which leads to a very simple mapping with which we can perform our deterministic walks. Another strong point of the algorithm is that the channel representations of $R(P)$ for $P \in \mathcal{P}_n$ are sparse matrices. Thus, we were able to implement a sparse matrix multiplication algorithm which allows us to very quickly compute most matrix products, despite the channel representations having dimension $4^n \times 4^n$.

We can apply Equation (3.8) and Equation (3.9) to the optimal T-count synthesis to obtain a runtime estimate. Each $R(P)$ contributes a single T gate to the circuit, and can be considered as a single layer of depth in this implementation. Thus, we have that $\xi = 4^n - 1$, as all Paulis save for the identity are valid choices. Our estimate for the runtime is thus

$$T_{QCS-T} \propto 2^{n\left(2\alpha+2\left\lceil \frac{t}{2} \right\rceil + \left\lfloor \frac{t}{2} \right\rfloor\right)} \frac{1}{\sqrt{w}} \frac{1}{m} \left\lceil \frac{t}{2} \right\rceil. \tag{3.14}$$

3.4.2 Computer specifications

We implemented the optimal T-count version of the parallel algorithm in C++11. It is called pQCS (**p**arallel **q**uantum **c**ircuit **s**ynthesis), and is available for download and research use at `https://qsoft.iqc.uwaterloo.ca/#software`. Parallelization was accomplished using the Boost.MPI compiled library [67]. A scaled down version of pQCS which uses only OpenMP for parallelization (and can be run on a standard multi-core personal computer) is also available in the above package.

pQCS was extensively tested on two large-scale machines. The OpenMP-only version was tested on SHARCNET's Orca using a single node with up to 16 processors at 2.2GHz speed. The MPI version was tested on Scinet's Blue Gene/Q (BG/Q) supercomputer, which has 65536 processors at 1.6GHz speed. The largest test we have run to date involved a total of 8192 cores. All results below are from trials on the BG/Q. A flowchart and description of the distribution of work in the MPI version is presented in Figure 3.3.

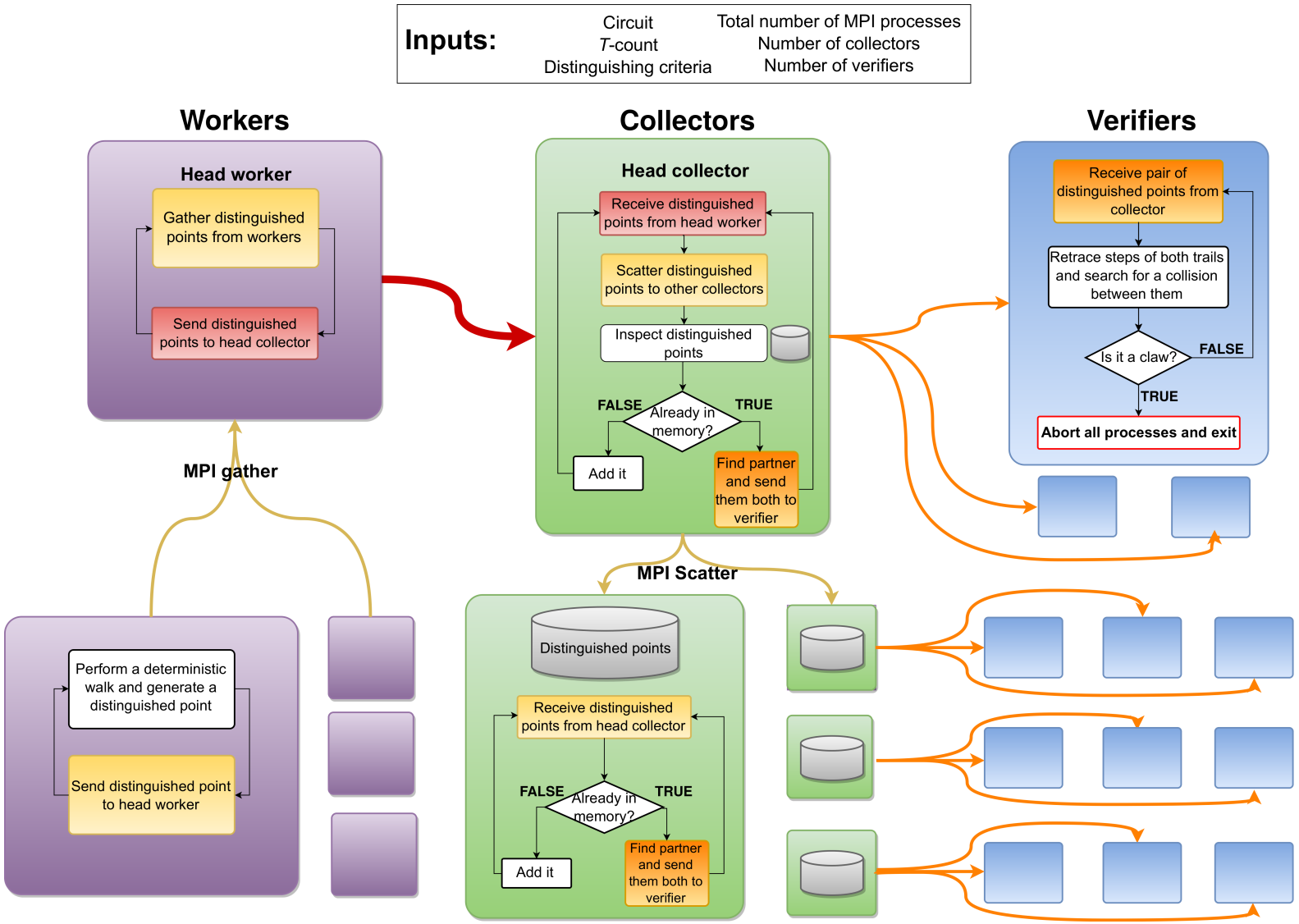


Figure 3.3: A flowchart of work distribution for the version of pQCS run on the Blue Gene/Q. Worker processors perform walks and generate distinguished points. These are funneled through a head worker to a head collector processor, which then distributes the points amongst all the collectors for processing and storage. Each collector has access to a number of verifiers. Collectors which find pairs of walks ending at the same distinguished point distribute the pairs to their verifiers to check for a claw.

3.5 Results

3.5.1 Determining effective simulation parameters

pQCS has a number of tunable parameters. In what follows we will synthesize a known circuit, the Toffoli gate, and explore the scaling of our algorithm.

In the original description of the parallel collision finding algorithm [65], each processor was responsible for performing not only the search for a distinguished point, but also storing it and subsequently checking the validity of any possible solutions; it is from this setup that the heuristic runtimes are derived. In pQCS, however, processors are divided into three categories (as per Figure 3.3) which communicate via MPI. Worker processors perform deterministic walks and generate distinguished points. Distinguished points are collected and stored in-core on collector processes. Each collector has access to a number of verifier processors, to which pairs of walks are sent for verification when the possibility of a claw occurs. The parameters m and w may not necessarily depend then on the total number of processors, but rather only on one or more of each class. For example, w will depend solely on the number of collectors, whereas we expect m to be a function of the number of workers, assuming a sufficient number of collectors and verifiers are in place.

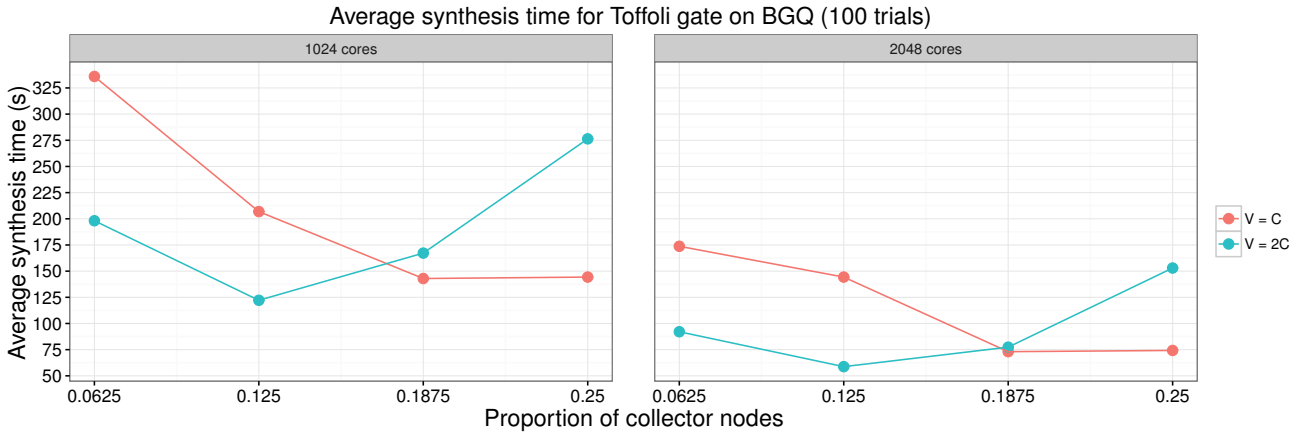


Figure 3.4: Variation of the number of collectors and verifiers when synthesizing the Toffoli gate. The legend $V = C$ indicates equal amounts of collectors and verifiers, whereas $V = 2C$ indicates two verifiers per collector. We find that the optimal number of collectors seems to be at about 1/8 the total number of processors, and the number of verifiers to be twice that, at 1/4 the total number.

First, we focus on how many collectors and verifiers we should use. We chose two values for the total number of cores, 1024 and 2048. We then varied the fraction of nodes designated as collectors in increments of 1/16, from 1/16 to 1/4 the total (values outside this range clearly yielded inferior results). For each fraction of collectors, we either used the same, or double the number of verifiers. The results of these trial runs are shown in Figure 3.4. In all these trials we let 1/4 of the points in the space be designated as distinguished (later we will fine-tune this parameter as well). Each point is the average of

100 independent trials. We find that for both total quantities of processors, the optimal number of collectors is 1/8 the total number, and for verifiers 1/4 the total. When more than 3/8 of the total processes are being used on storage and verification, there are not enough workers to perform the deterministic walks. On the other hand, when there are too many workers, each collector must store and process a larger collection of distinguished points each time. Furthermore, more time will be spent by the workers gathering and sending the increased quantity of distinguished points.

With this knowledge, we then tested the Toffoli with varying number of cores. Again, we let 1/4 of the points be distinguished and take the average of 100 independent trials. The results are shown in Figure 3.5. We see clearly here the expected inverse dependence on the number of processors as predicted by Equation (3.8). We do note that there is significant deviation from the expected trend when we reach 8192 cores. We suspect that for a problem of this size, the parallel overhead and communication costs outweigh the potential benefits of using this many cores.

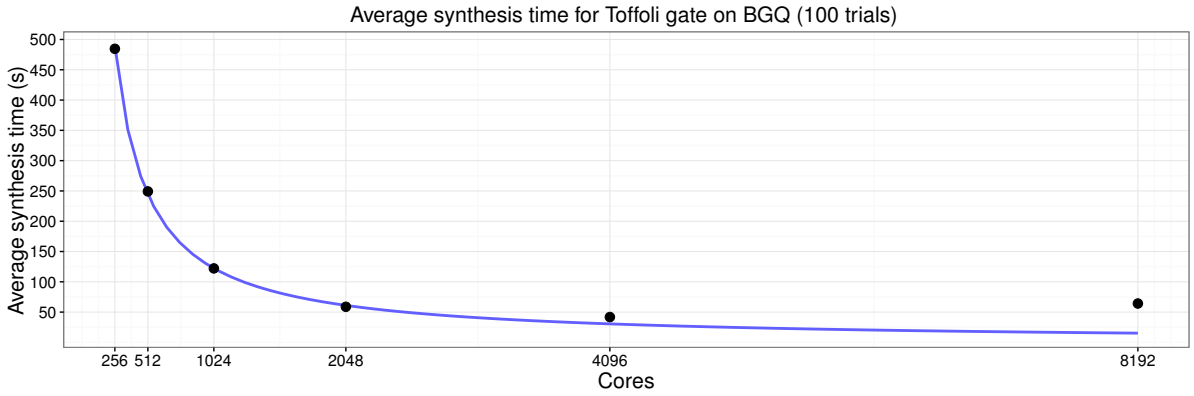


Figure 3.5: Varying the total number of cores when synthesizing the Toffoli gate. We used 512 collectors and 1024 verifiers with 1/4 of the points distinguished. Data follows the inverse trend line quite closely until around the 4096 core mark. Here, it is likely that the overhead and communication costs are too large for a problem of this size.

1

Finally, we investigate how the runtime varies with the fraction of distinguished points, θ. In the case of the Toffoli, the amount of available memory using the above number of processors on the BG/Q is significantly greater than that required to store even the entire space. Variation of this parameter is thus somewhat contrived for such a (relatively) small problem. In this case we would expect an inverse dependence on θ (see the Appendix for more details). We ran 100 trials on 4096 processors (512 collectors and 1024 verifiers) using fractions of distinguished points $\{1/2,\ 1/4,\ 1/8,\ 1/16,\ 1/32\}$. The results are displayed in Figure 3.6, where we see the expected dependence. We also report here our best synthesis times for the Toffoli gate, clocking in at roughly 26s on average. To fully explore the effects of this parameter (and more importantly the dependence on the available memory w), we will need to use a much larger circuit.

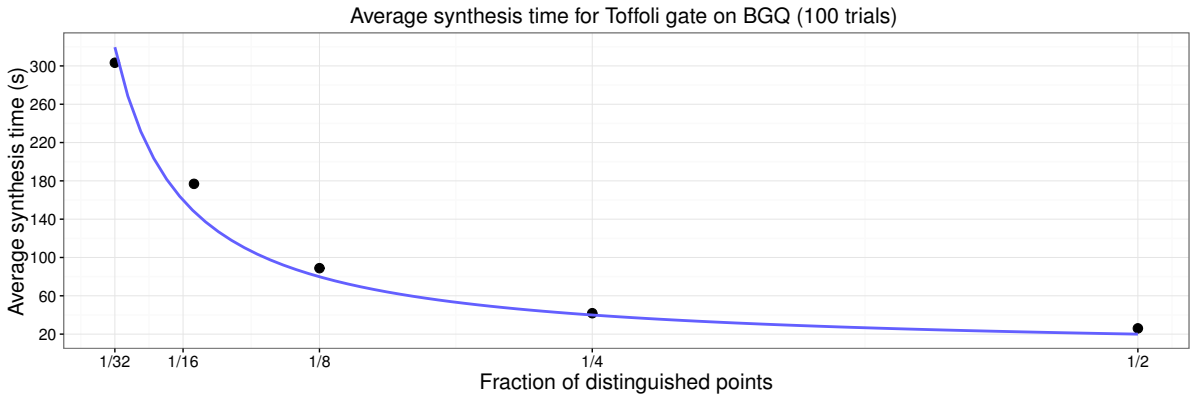


Figure 3.6: Varying the fraction of distinguished points while synthesizing the Toffoli gate. As the size of the search space is much less than the available memory, we see roughly the expected inverse dependence on the fraction of distinguished points.

3.5.2 Benchmarking known circuits

Some of the largest circuits which were directly synthesizable by both the original MITM algorithm and optimal T-count algorithm were those with T-count 7 on 3 qubits [29, 30]. There are a number of such circuits, shown in Figure 3.7. Using our knowledge from optimization of parameters in the previous section (4096 cores, 1/2 points distinguished, 512 collectors and 1024 verifiers), we obtain the synthesis times reported in Table 3.1. We note that at roughly 25s, these times are a marked improvement over those reported in [2], which were greater than 4 minutes. This highlights the advantage of using many processors, and is a promising sign that we will be able to synthesize circuits which are much larger in a reasonable amount of time.

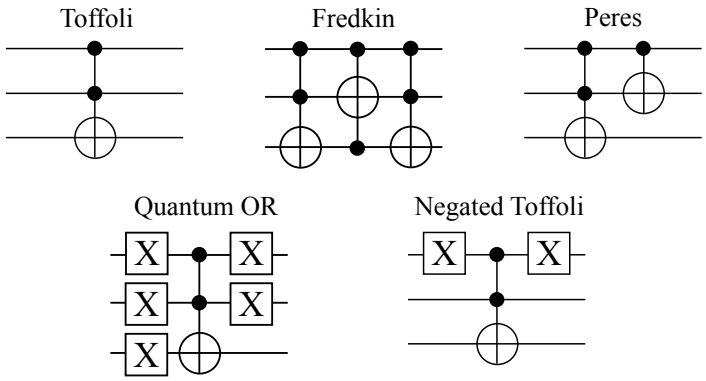


Figure 3.7: Circuit diagrams for the five 3-qubit circuits with T-count 7 which we synthesized.

Circuit	Average time (s)	Std. dev. (s)
Toffoli	25.9870	11.0733
Fredkin	25.0031	9.4869
Peres	25.4931	11.1753
Quantum OR	24.1854	9.1417
Negated Toffoli	26.9162	11.1561

Table 3.1: Synthesis of a known set of 3-qubit circuits all having optimal T-count 7. All results come from 100 independent trials using 4096 cores (512 collectors, 1024 verifiers), and 1/2 of points distinguished as per the results of Subsection 3.5.1.

3.5.3 Pushing the boundaries

The largest circuit synthesized to date using pQCS is the 4-qubit 1-bit full adder, shown in Figure 3.8. A synthesized version of this adder appeared in [29] with T-count 8, where it was accomplished using peephole optimization techniques. It was suspected that it has T-count 7 [68], which we confirm.

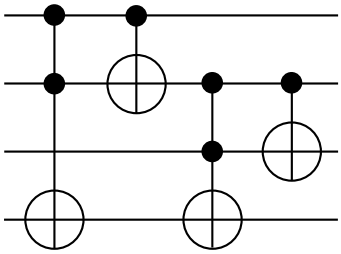


Figure 3.8: The 4-qubit adder. We find directly that it has T-count 7 and T-depth 3, and that these results are optimal.

The first successful synthesis of the adder took 12.5 hours using 4096 cores (512 collectors, 1024 verifiers) and 1/2 points distinguished. We note that a circuit as large as the adder would likely benefit from a larger number of processors, and so more testing is in progress. A full version of the circuit is shown in Figure 3.9. The initial output of pQCS is a sequence of Paulis and a unitary corresponding to a Clifford gate as per Equation (3.10). The Pauli portion of the circuit $(R(P_7)\cdots R(P_1))$ was generated using the algorithm given in the appendix of [30], and the Clifford component was generated using the algorithm in [69]. The resultant sequence of gates was then optimized for T-depth using T-par [**?**]. Interestingly, this new synthesis of the adder led to the observation that this adder requires identical resources as the Toffoli gate, i.e. T-count 7, T-depth 3, and to the question of whether this is a coincidence. In fact, it was subsequently pointed out to us that this adder is affine equivalent to the Toffoli (i.e. unitarily equivalent up to application of CNOTs) [70].

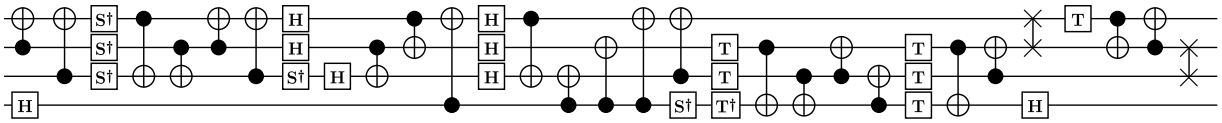


Figure 3.9: A decomposition of the 4-qubit adder over Clifford+T, optimized for T-depth. The X gates indicate swaps.

3.6 Concluding remarks

We have presented a framework for quantum circuit synthesis based on deterministic walks, as well as an algorithm and software for parallel quantum circuit synthesis. We have observed a clear advantage over existing techniques using a relatively modest number of processors, and were able to directly synthesize a 4-qubit circuit which would have been intractable using previous methods.

Ongoing and future work on pQCS includes improvements to the application structure and parallelization routines, extensions for synthesis in general over a specified gate set, and the implementation of approximate circuit synthesis. Furthermore, we seek to push the application to its limits in order to fully characterize the scaling, in particular with respect to the available memory once the circuit search spaces become sufficiently large.

3.7 Special application: the search for new symmetric Hadamard matrices

Recall that the parallel search algorithms, originally designed for cryptanalysis, can be adapted to a wide range of problems provided they can be represented as walks over binary strings. In this section we showcase one such problem, which was tackled by repurposing the pQCS software engine: searching for new symmetric Hadamard matrices.

As quantum information scientists we are very familiar with the Hadamard matrix of order 2:

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{3.15}$$

In fact, Hadamard matrices can be defined more generally, as any matrix with elements ± 1 in which all the rows and columns are orthogonal. *Hadamard designs*, their construction and existence, have been widely studied in combinatorial design theory and have applications in the construction of error-correcting codes [71].

Hadamard matrices exist in all orders that are multiples of 4, $n = 4v$, as well as $n = 1$ and $n = 2$ [71]. However, it is unknown whether *symmetric* Hadamards exist in all $n = 4v$. A portion of the work presented below shows how we used parallel collision finding techniques to find 15 new symmetric Hadamard matrices of order 116, answering the open question of their existence in this dimension. When [3] was published, order 116 ($v = 29$) was the smallest unknown case. Since then, symmetric Hadamards of order 156

($v = 39$) have been found [**?**]; at the time of writing of this thesis, the smallest unsolved case is that of order 188 ($v = 47$).

3.7.1 Abstract

We construct new symmetric Hadamard matrices of orders $92, 116$, and 172. While the existence of those of order 92 was known since 1978, the orders 116 and 172 are new. Our construction is based on a recent new combinatorial array discovered by N. A. Balonin and J. Seberry. For order 116 we used an adaptation of an algorithm for parallel collision search. The adaptation pertains to the modification of some aspects of the algorithm to make it suitable to solve a 3-way matching problem. We also point out that a new infinite series of symmetric Hadamard matrices arises by plugging into the GP array the matrices constructed by Xia, Xia, Seberry, and Wu in 2005.

3.7.2 Introduction

A Hadamard matrix of order n is an $n \times n$ matrix $H = (h_{ij})$ with elements ± 1 such that $HH^T = H^T H = nI_n$, where I_n is the $n \times n$ identity matrix and T stands for transposition. If in addition H is symmetric, i.e. $h_{ij} = h_{ji}$, then it is called a symmetric Hadamard matrix. A list of currently open cases for symmetric Hadamard matrices appears in [71], page 277, Table 1.52, and states that the only 12 odd values of $v < 100$ for which a symmetric Hadamard matrix of order $4v$ is not known to exist are:

$$23, 29, 39, 43, 47, 59, 65, 67, 73, 81, 89, 93.$$

However, symmetric conference matrices of order 46 were constructed by R. Mathon [72] in 1978, and it is well known that this implies the existence of symmetric Hadamard matrices of order $2 \cdot 46 = 92$. By modifying Mathon's construction, Balonin and Seberry [73] have constructed two more symmetric conference matrices of order 46 inequivalent to those of Mathon.

In this paper we construct symmetric Hadamard matrices of orders $92, 116, 172$. All of them are constructed by using the GP array of Balonin and Seberry [74]:

$$GP = \begin{pmatrix} A & BR & CR & DR \\ CR & D^T R & -A & -B^T R \\ BR & -A & -D^T R & C^T R \\ DR & -C^T R & B^T R & -A \end{pmatrix}, \tag{3.16}$$

where R is the back-diagonal matrix obtained from the identity matrix by reversing the order of rows. In order to obtain a symmetric Hadamard matrix of order $4v$ from this array we need four circulant $\{\pm 1\}$-matrices (also known as *binary matrices*) A, B, C, D of order v such that

(i) $AA^T + BB^T + CC^T + DD^T = 4vI_v$;

(ii) $A^T = A$ and $B = C$.

Such quadruples $[A, B, C, D]$ can be constructed from suitable difference families, also known as *supplementary difference sets (SDS)*, in the cyclic group $\mathbf{Z}_v$ consisting of four blocks. The authors of [74] refer to these quadruples as *propus matrices* and to the corresponding symmetric Hadamard matrix as a *propus-Hadamard matrix*.

In the case $v = 23$, our construction is quite different from those in [72] and [73]. In the case $v = 29$ we use a method for parallel collision search. We had to adapt this method in order to be able to apply it to the problem of searching for suitable quadruples $[A, B, C, D]$. In the case $v = 43$ we in fact construct D-optimal matrices (see e.g. [75]) of order 86 from two binary circulants A, D with $A^T = A$. The blocks B and $C = B$ are provided by the Paley difference set in $\mathbf{Z}_{43}$.

The smallest order for which the existence question for symmetric Hadamard matrices is still undecided is now $4 \cdot 39 = 156$.

Moreover we give a new infinite series of symmetric Hadamard matrices derived from the series of Hadamard matrices constructed in [76, Theorem 3].

3.7.3 Some infinite series of symmetric Hadamard matrices

We summarize some results pertaining to the existence of infinite series of symmetric Hadamard matrices. The summary is far from being exhaustive.

The following result is proved in [77, Corollary 4.6.5].

Theorem 1. *If $q \equiv 3 \pmod 4$ is a prime power and $q + 2$ is a prime power, then there exists a symmetric conference matrix of order $q^2(q + 2) + 1$ and a symmetric Hadamard matrix of order $2q^2(q + 2) + 2$.*

Note that for $q = 3$ one obtains a symmetric Hadamard matrix of order 92.

The following result is mentioned in [71, Theorem 1.48, p. 277].

Theorem 2. *If $n + 1$ and $n - 1$ are both odd prime powers, then there exists a symmetric regular Hadamard matrix of order n^2.*

A list of 11 classes of orders of symmetric Hadamard matrices appears in [78], Appendix D. Some of these classes are infinite. For example class SHIII is an infinite class, as a consequence of Dirichlet's theorem on the existence of primes in arithmetic progressions.

Another infinite series has been discovered recently in [74, Lemma 1]:

Theorem 3. *Let $q \equiv 1 \pmod 4$ be a prime power. Then propus matrices exist for orders $n = \frac{q+1}{2}$ which give propus-Hadamard matrices of order $2(q + 1)$.*

This series is derived from Turyn's infinite series of Williamson matrices [79] and can be plugged into the GP array. Another construction of symmetric Hadamard matrices of the same order has been known for long time, see [78, Lemma 5.2, p. 339].

One of us (D.D.) has subsequently observed that the same construction is applicable to the infinite series of Hadamard matrices of Goethals–Seidel type constructed by Xia, Xia, Seberry, and Wu [76, Theorem 3]:

Theorem 4. *Let $q = 4n-1$ be a prime power $\equiv 3 \pmod 8$. Then there exists an Hadamard matrix of order $4n$ of Goethals–Seidel type in which*

$$(I - A)^T = -I + A, \quad B^T = B \quad \textit{and} \quad C = D.$$

In fact their matrix is not just a Hadamard matrix but also a skew Hadamard matrix.

Instead of plugging the quadruple $[A, B, C, D]$ into the Goethals–Seidel array, we can plug the permuted quadruple $[B, C, D = C, A]$ into the GP array to obtain a propus-Hadamard matrix. Thus we have the following theorem:

Theorem 5. *Let $q = 4n-1$ be a prime power $\equiv 3 \pmod 8$. Then there exists a symmetric Hadamard matrix of order $4n$ of GP-type, i.e., obtained by using the GP array.*

For $n = 11, 17, 33, 35, 53, 71, 77, 83, 123, 125$ these symmetric Hadamard matrices are displayed on Balonin's webpages http://mathscinet.ru/catalogue/propus/dragomir/.

3.7.4 Overview of the algorithm for order 116

We want to construct a SDS $[A, D, B, C]$ in $\mathbf{Z}_{29}$ with parameters $(29; 13, 13, 11, 11; 19)$ such that the subset A is symmetric and $B = C$. This is necessary in order to use the GP array. Thus A and D have cardinality 13, and B cardinality 11. (The other option, with A and D of cardinality 11 and B of cardinality 13, was treated separately in the same manner.) We generate three files, one for each of A, D, B. The A-file contains the symmetric subsets of cardinality 13, the D-file arbitrary subsets of cardinality 13, and the B-file arbitrary subsets of cardinality 11. (Each subset is recorded on a separate line.) We do not collect all such subsets, but only those that pass the power spectral density (PSD) test. This test is an important tool as it cuts down the size of the file considerably. For a description of the PSD test see [80, section 4]. For each of these three files and subset recorded there, we compute and record in a new file the periodic autocorrelation function (PAF) of the corresponding binary sequence of length 29. The subsets $A, D, B, C = B$ will form a SDS if and only if the sum of their PAFs takes the value $\lambda = 19$ at all shifts different from 0. Thus the search for SDSs boils down to selecting one line in each of the A, D, B PAF-files such that the element-wise sum of the first, second and twice the third line is equal to 19.

To find such a triple of lines, we adapted the meet-in-the-middle parallel collision finding technique of [65]. Let us represent a line in the PAF file of A as the sequence $(a_1 \;\; a_2 \cdots a_n)$, and similarly for the PAF files of D and B. To find a triple of lines such that

$$a_i + d_i + 2b_i = \lambda, \quad \forall \;\; i \in \{1, \ldots, n\}, \tag{3.17}$$

we define two functions, f_{ad} and f_b:

$$f_{ad}(i,j) := \text{Element-wise sum of line } i \text{ from } A \text{ and line } j \text{ from } D \tag{3.18}$$
$$f_b(k) := \text{Element-wise difference of } \lambda \text{ and twice line } k \text{ from } B \tag{3.19}$$

With these definitions, any case where $f_{ad}(i,j) = f_b(k)$ constitutes a solution, and thus the existence of a symmetric Hadamard.

To execute the search, we have a large number of processors perform random walks through the space of combinations of lines. Walks start at random positions, and deterministically decide at each step whether to execute f_{ad} or f_b, and which lines of the file to read. To determine which function to perform in the next step, we concatenate the values in the result of f_{ad} or f_b, and run that through a SHA-1 hash function. An indicator for the next function to perform, and corresponding line indices, are then derived from this hash value. A walk terminates when the resultant hash value reaches some pre-defined condition (usually a certain number of 0s at the beginning of the hash string, the choice of which depends on the time-memory tradeoff between computation and storage time). Starting and ending points of all completed walks are stored in a set shared by all processors. This means that, with high probability, if two concatenated sums from both f_{ad} and f_b are the same, then the hash value will be the same, and the two walks will 'merge' and arrive at the same point in the collective set of stored walks. When such an instance occurs, we have found our solution.

We implemented this algorithm in C++11, using Boost.MPI. The specific implementation was adapted from [2]. All the data from the files was stored in a SQLite database. As the initial files are rather large, we perform a preprocessing step before doing the matching. All three initial PAF files are divided into subfiles based on the first number in each line. Then, only combinations of three subfiles such that the first numbers of each line sum to λ are actually run through the program.

As the algorithm is random and parallel, it is difficult to benchmark its runtime. We ran it on SHARCNET's Orca cluster, a machine whose nodes have processors with speeds of either 2.2GHz or 2.7GHz, and a minimum of 32GB RAM. We used 16 MPI processes: 14 of them continuously executed random walks, one held the shared set of completed walks, and the last was responsible for receiving pairs of walks which ended at the same point, and extracting possible solutions. Searches were done for a fixed amount of time, usually 24h. Of the 15 matches found for $v = 29$, the shortest time taken was 238 seconds; the longest took over nine hours. All but five of the matches were found in less than three hours.

3.7.5 Results

In this section we present the construction of symmetric Hadamard matrices of orders $4 \cdot 23 = 92$, $4 \cdot 29 = 116$ and $4 \cdot 43 = 172$. The order 92 is not new. The orders $116, 172$ are new.

The solutions are listed in the form of SDSs with four base blocks. From them one can construct the corresponding binary sequences and also the circulant matrices. To be specific, we label the positions of a binary sequence of length v with $0, 1, \ldots, v-1$ in that order. To a given subset $X \subseteq \{0, 1, \ldots, v-1\}$ we associate the binary sequence whose -1 entries occur exactly at the positions labeled by the elements of X. Further, to such a binary sequence we associate the circulant matrix of order v whose first row is that sequence. These circulant matrices can be plugged in to the GP array, in a suitable order, to obtain the symmetric Hadamard matrix. We say that a block of a SDS is *symmetric* if the corresponding binary circulant matrix is symmetric.

3.7.5.a Four non-equivalent solutions for $v = 23$

All four SDSs have parameters $(23; 10, 10, 9, 8; 14)$ and are written as $[B, C = B, A, D]$ with A symmetric. The quadruple of corresponding circulant matrices $[A, B, C, D]$ should be plugged in to the GP array. A graphical representation of the first solution listed below is displayed in Figure 3.10.

```
[[[0,1,2,3,5,7,9,12,17,18],[0,1,2,3,5,7,9,12,17,18],
     [0,2,3,6,10,13,17,20,21],[0,1,2,4,5,10,13,14]],
    [[0,1,2,3,5,7,10,11,13,19],[0,1,2,3,5,7,10,11,13,19],
     [0,5,7,8,11,12,15,16,18],[0,1,2,7,10,11,14,16]],
    [[0,1,2,3,6,8,9,10,14,19],[0,1,2,3,6,8,9,10,14,19],
     [0,1,3,8,11,12,15,20,22],[0,1,3,5,7,10,13,16]],
    [[0,1,2,5,6,8,10,13,14,16],[0,1,2,5,6,8,10,13,14,16],
     [0,1,3,4,10,13,19,20,22],[0,1,2,5,7,12,16,18]]]:
```

3.7.5.b Fifteen non-equivalent solutions for $v = 29$

All 15 SDSs have parameters $(29; 13, 13, 11, 11; 19)$. The first six are written as $[A, D, B, C = B]$ and the remaining nine as $[B, C = B, A, D]$, with A symmetric in all cases. The quadruple of corresponding circulant matrices $[A, B, C, D]$ should be plugged in to the GP array. The first solution from those listed below is displayed in Figure 3.11.

```
[[[0,4,5,6,7,9,13,16,20,22,23,24,25],
     [0,1,2,3,5,8,10,13,14,15,18,22,25],
     [0,1,2,5,6,8,11,12,14,16,22],
     [0,1,2,5,6,8,11,12,14,16,22]],
    [[0,1,5,8,12,13,14,15,16,17,21,24,28],
     [0,1,4,5,6,7,10,11,14,16,17,19,24],
     [0,1,2,4,7,8,10,12,15,19,21],
     [0,1,2,4,7,8,10,12,15,19,21]],
    [[0,2,6,7,8,10,11,18,19,21,22,23,27],
```

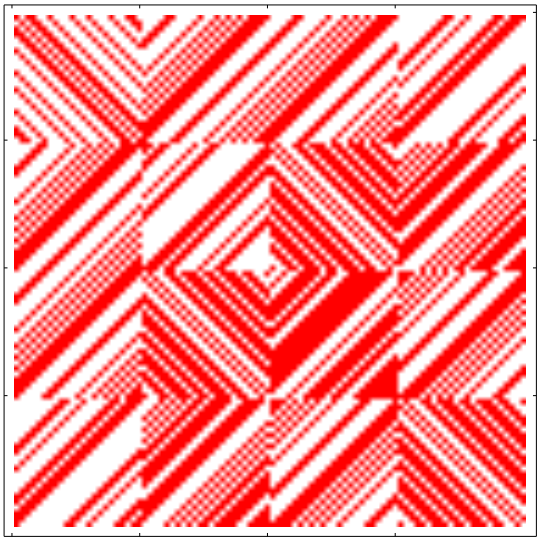


Figure 3.10: A visualization of the first solution of order $4 \cdot 23 = 92$. White indicates a value of +1, and red -1. We can see clearly the symmetry of the matrix, as well as the distinct blocks of the GP array.

```
 [0,1,3,4,5,7,10,13,14,15,17,20,24],
 [0,1,2,5,6,7,9,12,14,20,23],
 [0,1,2,5,6,7,9,12,14,20,23]],
[[0,2,3,4,6,11,13,16,18,23,25,26,27],
 [0,1,2,5,6,8,10,11,14,17,18,20,22],
 [0,1,2,3,6,7,11,13,14,17,22],
 [0,1,2,3,6,7,11,13,14,17,22]],
[[0,6,8,9,10,12,13,16,17,19,20,21,23],
 [0,1,3,5,6,7,10,12,13,16,18,21,22],
 [0,1,2,3,7,9,12,14,17,18,22],
 [0,1,2,3,7,9,12,14,17,18,22]],
[[0,1,5,8,12,13,14,15,16,17,21,24,28],
 [0,1,2,3,6,8,9,12,13,15,19,20,23],
 [0,1,2,5,7,9,11,12,15,17,20],
 [0,1,2,5,7,9,11,12,15,17,20]],
[[0,1,2,3,6,8,9,11,12,16,18,20,25],
 [0,1,2,3,6,8,9,11,12,16,18,20,25],
 [0,5,6,8,9,13,16,20,21,23,24],
 [0,1,2,3,4,7,8,11,13,17,19]],
[[0,1,2,3,4,6,9,10,11,14,17,19,23],
 [0,1,2,3,4,6,9,10,11,14,17,19,23],
 [0,2,7,11,12,14,15,17,18,22,27],
 [0,1,2,3,6,9,13,14,18,20,24]],
[[0,1,2,3,5,6,9,11,12,15,17,22,24],
 [0,1,2,3,5,6,9,11,12,15,17,22,24],
```

```
 [0,2,6,7,8,11,18,21,22,23,27],
 [0,1,2,3,5,6,10,13,14,17,21]],
[[0,1,2,4,5,6,9,10,13,15,16,17,23],
 [0,1,2,4,5,6,9,10,13,15,16,17,23],
 [0,1,2,7,10,12,17,19,22,27,28],
 [0,1,3,5,7,10,13,16,19,21,25]],
[[0,1,3,5,7,8,10,12,13,14,18,21,22],
 [0,1,3,5,7,8,10,12,13,14,18,21,22],
 [0,3,6,11,12,13,16,17,18,23,26],
 [0,1,2,3,4,6,10,12,16,17,20]],
[[0,1,2,3,5,7,8,10,14,16,19,20,24],
 [0,1,2,3,5,7,8,10,14,16,19,20,24],
 [0,1,2,4,10,11,18,19,25,27,28],
 [0,1,4,5,7,9,12,13,16,20,23]],
[[0,1,2,4,6,7,8,10,11,14,17,19,22],
 [0,1,2,4,6,7,8,10,11,14,17,19,22],
 [0,1,2,8,10,14,15,19,21,27,28],
 [0,1,2,5,7,10,14,15,18,19,24]],
[[0,1,3,4,6,7,8,11,13,15,17,22,23],
 [0,1,3,4,6,7,8,11,13,15,17,22,23],
 [0,1,9,11,12,14,15,17,18,20,28],
 [0,1,2,5,6,10,12,17,18,21,26]],
[[0,1,2,3,5,7,9,11,12,14,15,20,24],
 [0,1,2,3,5,7,9,11,12,14,15,20,24],
 [0,5,6,8,9,13,16,20,21,23,24],
 [0,1,4,5,7,8,10,16,17,18,23]]:
```

3.7.5.c One solution for $v = 43$

In this case we start by constructing D-optimal matrices of order 86 by using the well known two-circulant construction. The parameter set of the relevant SDS, $[D, A]$, is $(43; 21, 15; 15)$. The important feature of this SDS is that one of the sets has to be symmetric. Our example is the following:

```
[[0,1,3,4,5,8,12,13,14,18,19,20,21,23,26,27,29,30,32,34,36],
 [0,1,2,4,8,11,16,21,22,27,32,35,39,41,42]].
```

The second block, A, is symmetric. When combined with the Paley difference set B in $\mathbf{Z}_{43}$, we can plug $[A, B, C = B, D]$ into the GP array Equation (3.16) to obtain a symmetric Hadamard matrix of order $4 \cdot 43 = 172$. This matrix is visualized in Figure 3.12.

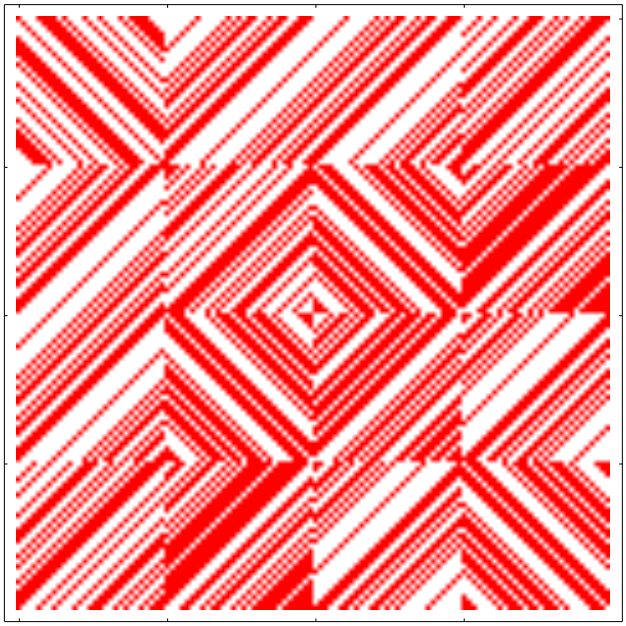


Figure 3.11: A visualization of the first solution of order $4 \cdot 29 = 116$.

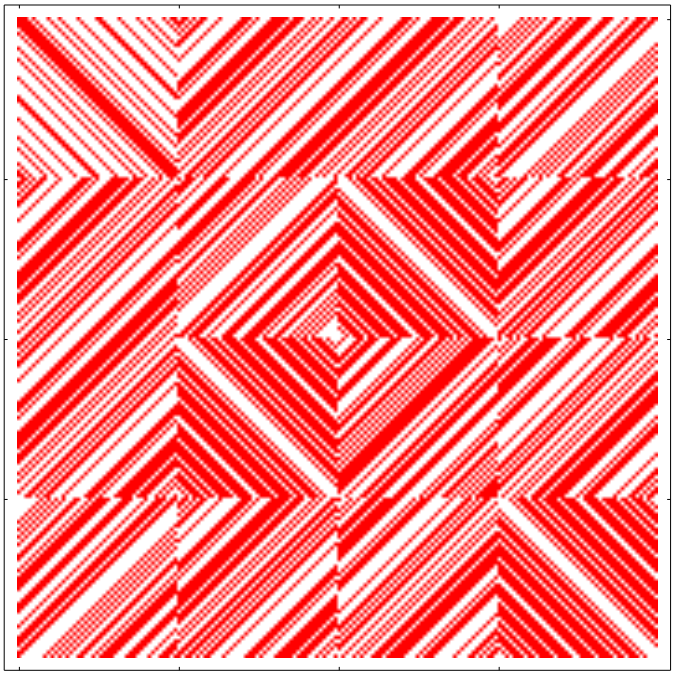


Figure 3.12: A visualization of the single solution found of order $4 \cdot 43 = 172$.

3.7.6 Acknowlegdements

The authors wish to acknowledge generous support by NSERC. This work was made possible by the facilities of the Shared Hierarchical Academic Research Computing Network (SHARCNET) and Compute/Calcul Canada. We thank N. A. Balonin and J. Seberry for

sending us copies of their papers [73,74] and Michele Mosca for useful discussions. O.D.M. would like to thank Gary Graham for some suggestions regarding the adaptation of the parallel collision finding implementation.

Chapter 4

Building a fault-tolerant quantum RAM

A paper and code repository based on the contents of this chapter are in preparation at the time of submission.

Understanding the construction of a fault-tolerant qRAM is important for the development of quantum algorithms. At the time of writing, there is an abundance of algorithms being developed that hinge on being able to efficiently prepare input states or query a qRAM. However when fault-tolerance is required, as appears to be the case for any algorithm that requires a superpolynomial amount of queries [4], we will incur a large overhead in physical resources that may limit the performance of promising new techniques.

The goal of this work is to develop an accessible framework for resource estimation of quantum RAM, and to analyze the space-time tradeoffs for a number of circuit families that can query classical bits in superposition. People designing or implementing algorithms that require a qRAM can choose the tradeoff that best suits their needs and constraints. We explore the extremes of circuit depth and width, and show circuits designed to interpolate between the two. While the large amount of resources they require is hardly surprising, this is a first effort in quantifying such circuits, and it is based on 'worst case' scenarios; we discuss how optimization and tailoring circuit design to specific problems can help mitigate some of the overhead.

Another contribution is the associated software, which provides a highly extensible framework for resource estimation of *any* quantum circuit. The release contains the circuit families explored herein, as well as routines for surface code resource counts in which it is easy to change physical parameters, such as gate error rates. This enables researchers to perform resource estimation over a broad range of parameters and values to determine both what is required for their specific algorithm, as well as how large of a problem they can run given the hardware they have.

4.1 Introduction

Random-access memory (RAM) is an essential component of classical computing architectures. Many quantum algorithms require an analogous system, a so-called quantum RAM, or qRAM, where the input is a quantum state, the routing components are inherently quantum, and the information stored can be either classical, i.e. $|0\rangle$ or $|1\rangle$ but not a superposition of both, or quantum, i.e. any arbitrary superposition of $|0\rangle$ and $|1\rangle$. In the present paper we consider qRAM that stores only classical information. Generically, such a memory allows for the querying of a superposition of addresses

$$\sum_j \alpha_j |j\rangle|0\rangle \xrightarrow{qRAM} \sum_j \alpha_j |j\rangle|b_j\rangle, \tag{4.1}$$

where $\sum_j \alpha_j |j\rangle$ is a superposition of queried addresses and $|b_j\rangle$ represents the content of the j-th memory location. A memory that stores classical information but allows queries in superposition is required for quantum algorithms such as Grover's search on a classical database [33], collision finding [34], element distinctness [35], dihedral hidden subgroup problem [36] and various practical applications mentioned in [81]. In fact, such a quantum memory plays the role of the oracle and is ideal in implementing any oracle-based quantum algorithm, in which the oracle is used to query classical data in superposition.

Several authors described algorithms that require only a polynomial amount of resources such as computational qubits or depth, and some larger number of 'quantumly accessible' classical bits [82]. Such quantumly accessible classical bits are less costly for classical simulations of quantum computers, since a quantum algorithm with n qubits and m qRAM bits can be simulated with $O(2^n + m)$ classical bits, instead of $O(2^{n+m})$.

There are also numerous algorithms, for example in quantum machine learning and Hamiltonian simulation, that are shown to demonstrate a speedup but must assume a qRAM can be queried efficiently (for example, [37–39]). For many algorithms this dependence stems from using the HHL algorithm for solving linear systems as a subroutine [40], which itself is successful in practice only if we can query efficiently, though we note that the particular operation performed by the qRAM differs slightly than the one we analyze here [1] .

One such implementation, the bucket brigade method [31], may allow algorithms that make only a few queries to a qRAM to avoid the usual overhead associated with fault-tolerant implementation of a binary-tree type look-up circuit. For such few-query algorithms this may bring substantial savings. However for many-query algorithms, it does not appear that one can bypass fault-tolerant error correction [4]. It remains unclear whether there is in fact much of a savings over general purpose quantum memory when implementing a fault-tolerant qRAM for a quantum computation.

Our work here seeks to address the question of the cost of fault-tolerant qRAM in a quantum computation. This cost is comprised of a number of different factors. The

[1] In contrast to Equation (4.1), the data to be read in is a vector of complex numbers $\mathbf{b} = (b_1, b_2, ..., b_n)$ which become the amplitudes in a superposition, e.g. $\mathbf{b} \to \sum_i b_i |i\rangle$.

primary one we consider here is the execution of a quantum circuit which performs the query, completely embedded in a fault-tolerant error correction scheme. One must also take into account external factors that may stem from the specific physical implementation and/or algorithm which is querying the qRAM. In principle, we can boot up our qRAM qubits only when we need to make a query, and turn them off after the fact. However, a combination of the overhead cost of initializing the qubits, cost of resetting them to $|0\rangle$, as well as any idle time between queries may warrant a more active approach to error correction so that after initialization the qRAM remains perpetually on (in a sense this would be similar to how conventional RAMs refresh themselves).

Roughly speaking, there are two natural ways to implement a quantum query to a classical memory. At one extreme, classical information $b_1, \ldots, b_N$ could be laid out in static physical hardware which is quantumly queried. This can be accomplished using, for example, Controlled-NOT (CNOT) gates on some target register conditioned on the values of the b_j, or by some binary tree circuit or a bucket brigade-style circuit, which has depth logarithmic in the size of the database. Such approaches require a number of query qubits that is proportional to the size of the database. By query qubits, we are referring to the qubits used in essentially all such memory schemes in order to connect the computational qubits that store the index that needs to be queried, and the (qu)bits that store the classical information being queried. These query qubits (or qudits) are only used ephemerally in order to perform the query, and do not store any information before or after each query.

At the other extreme, instead of storing the classical information in a static physical memory, one can simply implement a sequence of mixed-polarity multi-controlled CNOTs, conditioned on the control bits representing the memory address of a 1. This implies that the classical database is implicitly stored in the logical circuit, and this circuit will have depth proportional to the number of 1s in the database. In this model, reading from the qRAM consists of simply running the circuit, while writing involves the addition or removal of a multi-controlled CNOT for the desired address. Of course, if there are more 1s than 0s, it suffices to condition instead on the control bits representing the memory addresses of the 0s, and then finish with a single NOT gate.) Such circuits can in general be optimized using any number of known optimization techniques for Boolean circuits.

We present variants of this latter approach that only require resources roughly proportional to the number of 1s in the database. Suppose there are m 1s in the database. We can perform a sequence of m multiple controlled gates, where each address $x_1 x_2 \ldots x_n$ in which there is a 1 controls the output on some target bit. We can also parallelize this process: in $O(\log m)$ depth we can compute m copies of the desired index. Then in parallel, using the j-th copy of the desired index we compute 1 if the index equals the index of the jth 1 in the database. Finally, using a binary tree type circuit we can in $O(\log m)$ depth compute whether a 1 was computed on any of the m copies. We then uncompute all the intermediate computations.

There are also many natural ways to interpolate between the two approaches, for example using the same fan-out like operation to make 2^k copies of the first k index bits, and then use 2^k parallel logical circuits to explore the remaining $n - k$ index bits.

In this paper, we outline these various questions and approaches, and consider their costs and trade-offs. Such an analysis is important for optimizing the physical resources needed to implement a quantum algorithm in practice, using the best-known methods. We begin by describing our general cost model and how we presume an algorithm will query the qRAM. We then introduce a number of circuit families: circuits for the bucket brigade qRAM model, the basic highly sequential or parallel circuits mentioned above, as well as some interpolations between the two. We compute concrete parameters such as real-time cost, number of qubits, etc. of our circuits embedded within a fault tolerant implementation using a defect-based surface code [41]. We also discuss a special case of the address structure, which offers a massive savings in the aforementioned resources. We conclude with some final thoughts and present avenues for future research.

4.2 Modeling the cost of a qRAM

We consider a qRAM which stores quantumly accessible classical bits. Locations in memory are addressed by n-bit strings $x_1x_2\cdots x_n$, and are queried by inputting the associated state $|x_1x_2\cdots x_n\rangle$ to a circuit. We assume that the memory addresses in which 1s are stored are all known, so that the qRAM can be represented using only a logical circuit and not by physical qubits prepared in $|0\rangle$ or $|1\rangle$. For each address $x_1x_2\cdots x_n$, the qRAM should implement

$$|x_1x_2\cdots x_n\rangle|0\rangle \to |x_1x_2\cdots x_n\rangle|b_{x_1x_2\cdots x_n}\rangle \tag{4.2}$$

where $b_{x_1x_2\cdots x_n}$ is the stored value at the specified address. This form ensures the qRAM can be queried in superposition.

Consider a quantum algorithm with a structure similar to that in Figure 4.1. Queries to a qRAM are interspersed between some number of arbitrary unitary operations that comprise the main portion of the algorithm. We suppose that the entire circuit is embedded in a surface code in order to make it fault-tolerant. Then, we can use the same cost metric as [12], wherein

$$\text{Cost} = \log_2 (\text{Logical qubits} \times \text{Surface code cycles.}) \tag{4.3}$$

In essence this cost represents a tradeoff of space vs. time.

The cost is calculated using a framework [12] that starts from the high-level algorithmic description. The algorithm begets a quantum circuit, which is then synthesized and optimized over an elementary gate set. The optimized circuit is then embedded into a surface code, in which the number of logical qubits and operations determine parameters such as code distance and resources required for magic state distillation. With these, we can compute physical layer parameters such as the number of physical qubits and surface code cycles.

In the present work we will take a more general approach to calculating cost. We will not be dealing with any specific algorithms or circuits, and so we will not perform any circuit optimization, but rather we compute costs and resources in terms of parameters like n, and the number of 1s stored in the memory.

We will also use of a 'rough' estimate of cost before performing the surface code analysis:

$$\text{Rough cost} = \log_2 (\text{Logical qubits} \times T\text{-depth}) . \tag{4.4}$$

One often finds that magic state distillation is the most expensive part of implementation, and so the algorithm will be time-limited to how quickly we can produce the required magic states. We will assume in our analysis that we always have as many distillation factories as is required to implement a single layer of T-depth. The number of T gates in each layer, the T-width, is estimated as $T_w = T_c/T_d$ where T_c is the total T-count of the algorithm. Thus the T-depth is an appropriate stand-in for time, and as we will see, gives a good description of the overall scaling of cost.

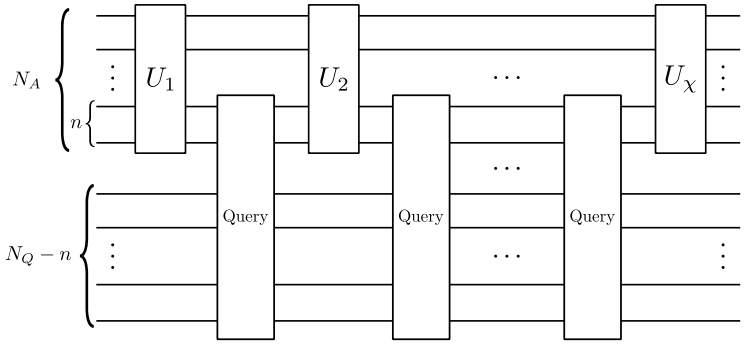


Figure 4.1: An example of how an algorithm might query a qRAM. The algorithm itself runs on N_A logical qubits, n of which will be used as an input address to query a qRAM, which itself requires N_Q logical qubits. We suppose that the algorithm queries the qRAM periodically after performing each of some number χ of unitary operations.

4.3 Should I try turning it off and on again?

One question posed in the Introduction was whether or not we should keep the qRAM "on" between queries. By this, we mean performing active error correction on the idle qubits in the qRAM while the algorithmic components run. We will consider this question in terms of the difference in cost of both approaches.

For simplicity, let us assume that the algorithm performs χ instances of the same operation U on N_A qubits, and $\chi-1$ queries to the qRAM in between these operations, using N_Q qubits, as is depicted in Figure 4.1. As we alternate between queries and operations, the logical qubits in the algorithm (minus the n used as an address for the query) will be in an idle state while we query the qRAM; similarly, if we keep the qRAM on, the query qubits will be idle while the algorithmic portions run.

Let c_A be the number of surface code cycles taken by one instance of U. We assume that c_i cycles are required for the initialization of a logical qubit, using for example the procedures outlined in [41]. We assume as well that it takes c_t cycles to reset and dispose of the qubits once the query is complete. In a sense, these operations are analogous to C memory management functions `calloc()` and `free()` [83].

It is straightforward, from Figure 4.1, to see that we should take the following approach:

$$c_A > c_i + c_t \rightarrow \text{turn the qRAM qubits off during the execution of } U, \quad (4.5)$$

$$c_A < c_i + c_t \rightarrow \text{always keep qubits on.} \quad (4.6)$$

Note, however, that since cost involves multiplication by the number of logical qubits, performing any sort of cost analysis on a qRAM is naturally most important when $N_Q >> N_A$. The choice of query circuit, various options for which will be discussed in the ensuing sections, may thus depend on a tradeoff between available resources and the relative size of the algorithm circuit versus that of the qRAM.

One can also imagine situations in more complex algorithms where the operations U_i are not identical. Here one might design a smart compiler that will choose to periodically turn off the qRAM during comparatively long algorithm operations, but leave it on for shorter ones. This requires prior knowledge, or a way to determine during execution whether the query qubits have been properly returned to their initial state and are not entangled with any of the algorithm (address) qubits. A smart compiler may also take into consideration the relative error rates between, say, re-initializing a qubit in $|0\rangle$ versus keeping its existing state error-corrected. This alone may warrant always turning off the qRAM, if $|0\rangle$ can be initialized quickly and with very high accuracy.

Finally, we note that if an algorithm requires writing regular updates to the classical database (such as in [82]), then one would need to update the query circuit during the course of the algorithm. Thus any latency between changing the database based on a measurement during the execution of the quantum circuit and updating the circuit in the software must be considered. Latency would also exist for any other mechanism for storing the database, and the precise cost for each approach would need to be considered in each case.

4.4 Bucket brigade circuits

The first family of circuits we will analyze are a family introduced in [4] that function as a bucket brigade qRAM [31,32]. A reproduction of the circuit is shown in Figure 4.2. These circuits assume that the contents of the memory are stored statically in the lower register of qubits in Figure 4.2, which is in contrast to the circuits we will see in later sections.

Bucket brigade circuits are constructed using only CNOTs and Toffolis, and we will further decompose the Toffolis over the Clifford+T gate set. We will perform a cost estimate for two types of bucket brigade circuit, those constructed precisely as in Figure 4.2, as well as an improved version where we parallelize the execution of the Toffolis in each layer.

For memories with n-bit addresses, a bucket brigade circuit requires $2 \cdot 2^n - 2$ Toffoli gates. We can choose from a number of different implementations of a Toffoli [84, 85], which will affect the overall resource counts:

- No ancillae, T-depth 3,

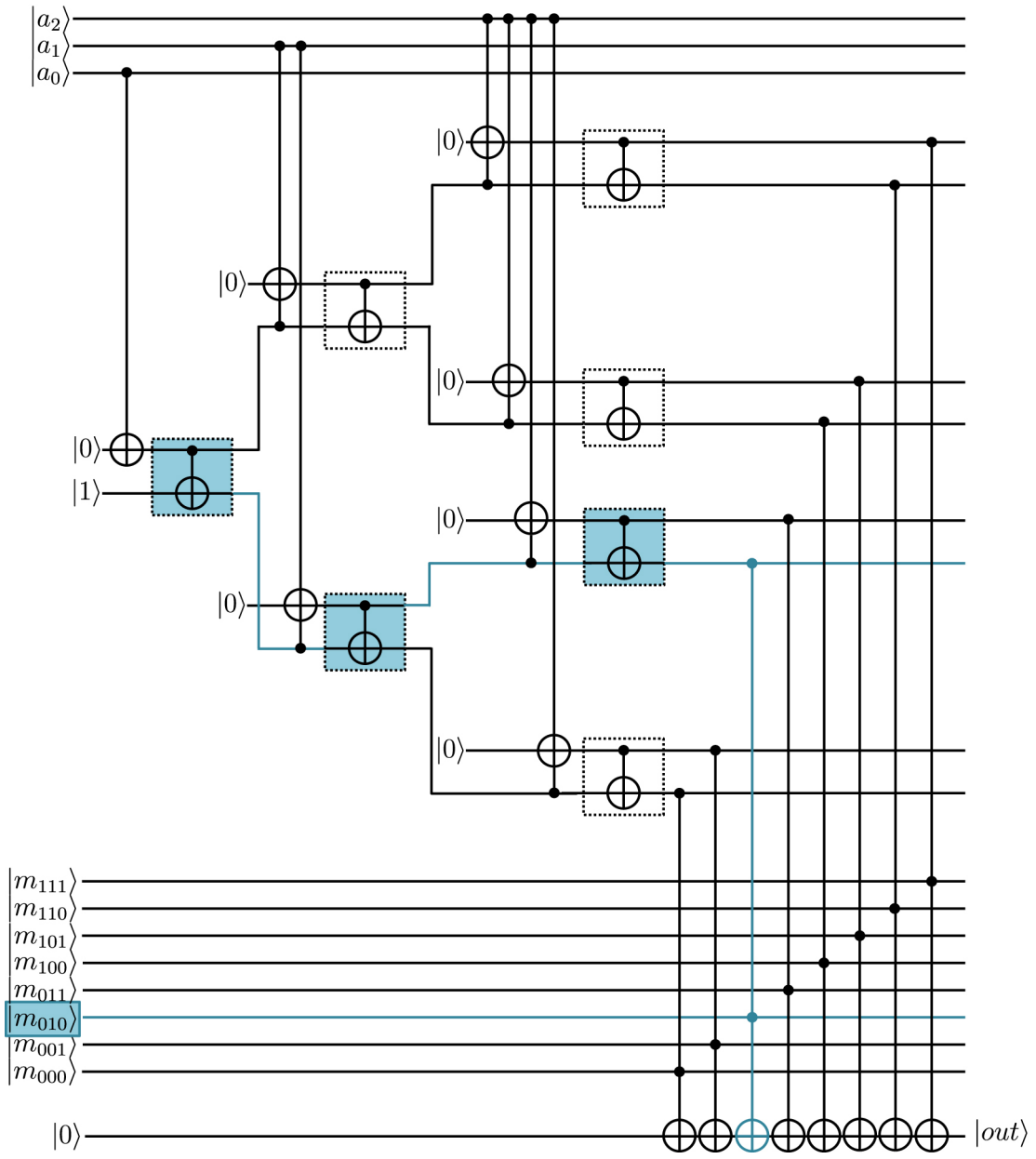


Figure 4.2: One method of constructing a bucket brigade style qRAM circuit. Original image taken from [4]. The circuit is implemented using only CNOTs and Toffolis, which we decompose over Clifford+T. The circuit is independent of the contents of the memory, which is initialized separately in the lower register. A full qRAM query consists of running this circuit, copying the output to an additional qubit, and then re-running it backwards.

- One ancilla, T-depth 2,
- Four ancillae, T-depth 1.

Note that in all cases the ancillae can be reused whenever Toffolis are applied sequentially. We focus on the third option, which while it uses the most ancillae, a cursory analysis using Equation (4.4) found this to always produce the lowest cost due to the savings in T-depth.

As the memory contents are stored statically, we know the full form of the circuit and so we can perform some optimization. In particular, in the final cascade of Toffolis sharing a target, we can reduce the number of CNOTs and Hadamards by removing 'interior' Hadamards and CNOTs at the beginning and end of each Toffoli that use only the target qubit and the ancillae (see Toffoli implementation in Fig. 6 of [85]).

For a fully reversible qRAM query, we must run the circuit in Figure 4.2, copy the output to an additional qubit, and then run it again backwards. This yields resource counts:

$$\begin{aligned} N_Q &= n + 2^{n+1} + 6 \quad \text{(logical qubits)}, && (4.7)\\ D &= 56 \cdot 2^n + 2n - 53 \quad \text{(depth)}, && (4.8)\\ T_c &= 28(2^n - 1) \quad (T\text{-count}), && (4.9)\\ T_d &= 4(2^n - 1) \quad (T\text{-depth}), && (4.10)\\ H_c &= 4(2^n - 1) \quad \text{(Hadamard count)}, && (4.11)\\ \text{CNOT}_c &= 62 \cdot 2^n - 59 \quad \text{(CNOT count)}. && (4.12) \end{aligned}$$

We can also parallelize this circuit by a) copying down the address qubits to ancillae so that the Toffolis in the first $n-1$ layers can be performed in parallel, and b) adding an extra output register of 2^n qubits to collect the results of the final set Toffolis, followed by a sequence of 2^n CNOTs. While this still yields an exponential depth due to this final set of CNOTs, it will yield a significant savings in T-depth, which is the more important parameter later when we perform the surface code analysis.

We require enough ancillae to perform the largest amount of simultaneous Toffolis, which will be $4 \cdot 2^n$ using the T-depth 1 implementation. However, we can no longer use the same depth reduction trick for the last layer of Toffolis as they now all have different target qubits. We obtain:

$$\begin{aligned} N_Q &= 8 \cdot 2^n + 1 && (4.13)\\ D &= 2 \cdot 2^n + n^2 + 35n + 3 && (4.14)\\ T_c &= 28(2^n - 1), && (4.15)\\ T_d &= 2n, && (4.16)\\ H_c &= 8(2^n - 1), && (4.17)\\ \text{CNOT}_c &= 78 \cdot 2^n - 2n - 73. && (4.18) \end{aligned}$$

We note here that the rough cost scaling of $N_Q \times T_d$ in terms of n is greatly improved in the parallelized circuit. Moving forward we will consider only the parallel versions of the circuit; we will return to this point and perform a more thorough comparison in Subsection 4.5.3.

4.5 Basic query circuits

We now construct another two families of qRAM circuits that perform the operation described by Equation (4.2). These circuits have opposite properties when it comes to logical qubits and circuit depth. We will see in the end that they have comparable overall cost, but vastly differing use of resources down at the physical level.

For these circuits, we suppose for simplicity that the memory contains 2^q 1s, the locations of which are known, with the rest being 0. Note that this is in contrast to the bucket brigade circuits where the contents, while unknown, were provided to us statically stored in hardware. We note that such a memory is capable of holding arbitrary k-bit values by simply storing the elements bitwise in the memory. Furthermore, if the number of 1s is ever greater than the number of 0s, we can equally well build our circuits by inputting the locations of the 0s.

We consider a small running example for the purpose of creating the circuit diagrams. Suppose we have $n = 3$ and $q = 2$, i.e. 4 of 8 memory locations store a 1. We arbitrarily set those locations to be 000, 001, 011, 111.

4.5.1 Large depth, small width circuit

We can easily create a circuit which will output a 1 for the valid addresses by implementing a sequence of 2^q n-bit mixed-polarity multiple control Toffolis (MPMCTs). The associated circuit for the running example is shown in Figure 4.3. Each MPMCT is tied to one of the addresses, and sets a target bit to 1 only if its associated address is fed in. For purposes of the resource estimation here, we will assume the worst case: we don't know the exact sequence of MPMCTs, only that we have 2^q such operations. As such, *we will not perform any extensive circuit optimization in our analysis.*

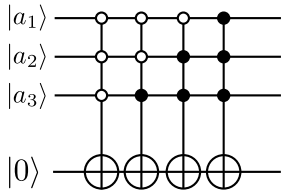


Figure 4.3: A qRAM circuit with few qubits but large depth. The addresses of all 2^q locations known to contain a 1 are hard-coded into the circuit as mixed-polarity multiple control Toffoli gates.

In general, when such a sequence of MPMCTs *is* known, we may be able to greatly simplify the circuit. For example, since a sequence of MPMCTs represents a Boolean function which is a sum of product terms, we can find its ESOP expression using a tool such as EXORCISM-4 [86]. This can offer great savings - for example, the circuit in our example can be reduced to 2 Toffolis just by factoring some terms in the Boolean expression. We can also use the method of [87] which first computes the ESOP, and then breaks the expression down into common cofactors of the expression terms; cofactors are then reversibly synthesized individually before being used in their constituent terms.

Instead we begin our analysis by performing quantum circuit synthesis to decompose the circuit down into the 1- and 2-qubit operations of the Clifford+T gate set. Again, when the precise sequence of MPMCTs is known, further optimization can be done to reduce parameters such as the T-count and T-depth (see, for example, the methods in [85]).

The decomposition we choose for the MPMCTs is that of [88,89], which is an optimization of an older algorithm from [90] that performs an n-controlled NOT using $n-2$ ancillae by turning it into a cascade of $4(n-2)$ Toffoli gates. We take advantage of an additional optimization which, at the cost of one more ancilla qubit, can further parallelize some of the T gates [84]. While it may be possible to implement such gates with fewer ancilla qubits, recall that we are interested in plugging this circuit into the surface code, wherein minimization of T-count and T-depth plays a critical role in reducing the overall cost incurred by magic state distillation. Finally, we note that this MPMCT implementation is valid only for $n \geq 4$.

The resources required for an n-controlled MPMCT gate are as follows:

$$\begin{aligned} D &= 28n - 60, && (4.19)\\ T_c &= 12n - 20, && (4.20)\\ T_d &= 4(n-2), && (4.21)\\ H_c &= 4n - 6, && (4.22)\\ \mathrm{CNOT}_c &= 24n - 40. && (4.23) \end{aligned}$$

We see immediately that in theory, the circuit in Figure 4.3 requires only $n+1$ qubits, plus $n-1$ ancilla qubits. As the ancillae are returned to their initial state after each MPMCT, we can reuse them for all 2^q gates. Thus, the total number of qubits is

$$N_Q = 2n. \tag{4.24}$$

We note that the X gates to change polarity of the controls can be applied in the first layer of depth of each MPMCT; in addition, the final polarity change can be performed in the last layer of the last MPMCT. In all cases the X gates do not contribute to depth.

Thus we obtain total logical resource counts,

$$\begin{aligned} D &= 2^q(28n-60), && (4.25)\\ T_c &= 2^q(12n-20), && (4.26)\\ T_d &= 2^{q+2}(n-2), && (4.27)\\ H_c &= 2^q(4n-6), && (4.28)\\ \mathrm{CNOT}_c &= 2^q(24n-40). && (4.29) \end{aligned}$$

Finally, in order to make the circuit fully reversible, we must perform the circuit of Figure 4.3, copy down the result of the last qubit to a new qubit using a CNOT, and then undo the computation of the full circuit. The ultimate logical resource counts come to:

$$\begin{aligned} N_Q &= 2n+1, && (4.30)\\ D &= 2^{q+1}(28n-60)+1, && (4.31)\\ T_c &= 2^{q+1}(12n-20), && (4.32)\\ T_d &= 2^{q+3}(n-2), && (4.33)\\ H_c &= 2^{q+1}(4n-6), && (4.34)\\ \mathrm{CNOT}_c &= 2^{q+1}(24n-40)+1. && (4.35) \end{aligned}$$

4.5.2 Small depth, large width circuit

Whereas the circuit of the previous section performed all the MPMCTs sequentially, here we present an implementation which can parallelize their execution. We provide an example of such a circuit in Figure 4.4.

We begin with 2^q registers of qubits, one for each address containing a 1. The address is input to the first register of qubits and then copied down to the others using a logarithmic number of CNOTs (this step could in theory be skipped if we could prepare 2^q identical copies of the address).

Each register performs an MPMCT which will trigger one of the qubits in an additional register if the input address matches. The qubits in the additional register are then summed together using a sequence of CNOTs, the result of which is passed to an output bit.

The number of qubits, including all required ancillae, is

$$\begin{aligned} N_Q &= n\cdot 2^q + 2^q(n-1) + 2^q + 1\\ &= n2^{q+1}+1. && (4.36) \end{aligned}$$

To compute the depth of this circuit, we must take into account the sequences of CNOTs. These are implemented in logarithmic depth; the initial copying is performed in

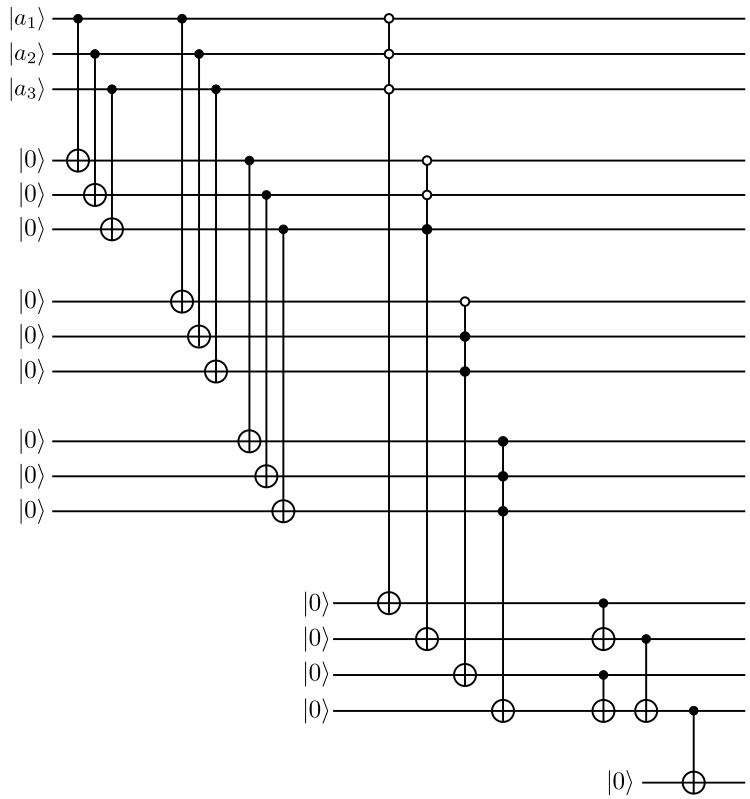


Figure 4.4: A circuit with small depth but a large number of qubits. The implementation of the MPMCTs is performed in parallel, each requiring $n-1$ ancillae, which significantly increases the number of qubits required.

depth q, the final sum in depth q, plus 1 for the terminal CNOT. Thus the depth is

$$\begin{aligned} D &= q + (28n - 60) + q + 1 \\ &= 2q + (28n - 60) + 1, \end{aligned} \tag{4.37}$$

which scales linearly in both n and q.

The T-count will be the same as for the circuit in the previous section, however the T-depth will now be $T_d = 4(n-2)$ as all the MPMCTs are performed in parallel.

In addition, we will need the Clifford counts. The number of Hadamards is unchanged. The number of CNOTs in the initial copying layer plus the terminal parity layer is

$$\begin{aligned} \mathrm{CNOT}_{c-init} &= n \sum_{i=0}^{q-1} 2^i + \sum_{i=0}^{q-1} 2^i + 1 \\ &= (n+1)(2^q - 1) + 1 \end{aligned} \tag{4.38}$$

Together with the $24n - 40$ CNOTs in the MPMCTs we obtain

$$\begin{aligned} \mathrm{CNOT}_c &= (n+1)(2^q - 1) + 2^q(24n - 40) + 1 \\ &= 2^q(25n - 39) - n - 1 + 1. \end{aligned} \tag{4.39}$$

Again, to make everything fully reversible, we must perform everything that takes place before the final CNOT of Figure 4.4 again. This yields total logical resources

$$N_Q = n2^{q+1} + 1, \tag{4.40}$$
$$D = 2(2q + 28n - 60) + 1, \tag{4.41}$$
$$T_c = 2^{q+1}(12n - 20), \tag{4.42}$$
$$T_d = 8(n - 2), \tag{4.43}$$
$$H_c = 2^{q+1}(4n - 6), \tag{4.44}$$
$$\mathrm{CNOT}_c = 2^{q+1}(25n - 39) - 2n - 1. \tag{4.45}$$

4.5.3 Preliminary cost estimate

Recall that our definition of cost is a measure of space vs time. We can analyze the product $N_Q \times T_d$ to get a rough first estimate of how the cost will depend on n and q. In Figure 4.5 we plot the overall costs (including constant prefactors) for differing values of n, and q for the circuits in which it is relevant. We summarize our observations in Table 4.1 to make it easier to see the tradeoff between the number of qubits and the depth for each circuit.

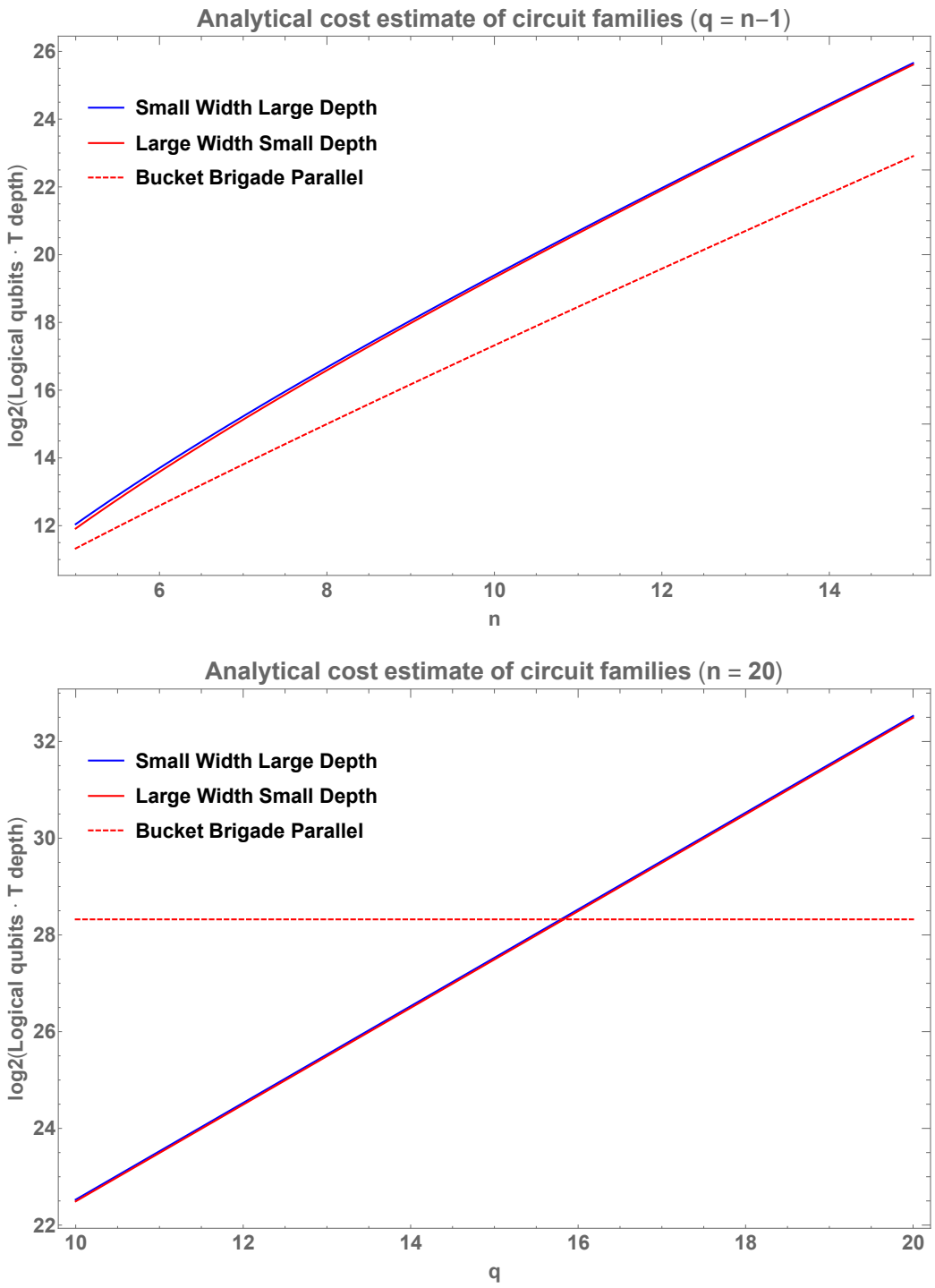


Figure 4.5: Analytical cost estimates $N_Q \times T_d$ for large depth/width and parallel bucket brigade circuits. (Top) cost for $q = n - 1$, a half-full memory. Costs for large depth/width circuits are comparable, showing a clear tradeoff between space and time. However, the parallelized bucket brigade algorithm has lower cost overall. (Bottom) Dependence of cost on the fullness of the memory (number of 1s). For sparser memories it may be cheaper, up to a point, to use a large depth/width circuit over the parallel bucket brigade circuit.

Circuit	Large depth	Large width	Bucket brigade parallel
N_Q	$2n+1$	$n2^{q+1}+1$	$8 \cdot 2^n + 1$
T_d	$2^{q+3}(n-2)$	$8(n-2)$	$2n$
Cost	$O(n^2 \cdot 2^q)$	$O(n^2 \cdot 2^q)$	$O(n \cdot 2^n)$

Table 4.1: Cost scaling for bucket brigade and large depth/width circuits.

The top panel of Figure 4.5 shows the situation of a half-full memory in which $q = n-1$. This is the worst case, because as mentioned previously, if the memory is more than half-full with 1s, we can switch the polarities of the gates to pick out the locations that are 0 instead. For a half-full memory, the parallel bucket brigade circuit is the best choice, as it always has lower cost.

For memories that are emptier, there is a cross-over point before which it is in fact better to use either the large depth or large width circuit as opposed to the bucket brigade circuit (assuming, of course, that one has knowledge or control over the location of the 1s in the memory). We can solve for this transition point by equating the cost functions of the two circuits from 4.1 (we choose the large width one for example). Then we find that $q_{max} \approx n - \log_2(n-1)$ and so past this memory fullness it is preferable to use the parallel bucket brigade circuit.

4.5.4 Surface code analysis

We now embed these circuits into a surface code using the same procedure as in [12]. We will make available our Python code for performing this task at `https://github.com/glassnotes/FT_QRAM_Circuits`, which contains resource estimation methods for the surface code as well as separate classes for each circuit considered here and in future sections. The (very optimistic) surface code parameters are input injection error probability $p_{in} = 10^{-4}$, gate error probability of $p_g = 10^{-5}$, and a cycle time of $t_c = 200$ns.

Figure 4.6 plots the numerical equivalent of Figure 4.5. While the relative relationships remain the same, the overall cost is significantly higher due to the large amount of logical qubits needed in the distillation factories.

Figure 4.7 is perhaps the more interesting plot, as it shows the explicit tradeoffs between the number of physical qubits and the 'real' query time. Even though we observed on Figure 4.5 that the costs of the large depth/width circuits are comparable, Figure 4.7 shows us exactly how large the space vs time tradeoff is.

We present numerical data for the largest and smallest values of n we chose, $n = 15$ and $n = 36$, in Table 4.2. The $n = 36$ case corresponds to 8 'GB' of classical data in the memory, whereas $n = 15$ corresponds to 4 'KB'. These particular choices are somewhat meaningful: 4KB was the amount of RAM that the Apple I computer shipped with back in 1976, while 8GB is a fairly standard amount of RAM on a laptop at the time of writing.

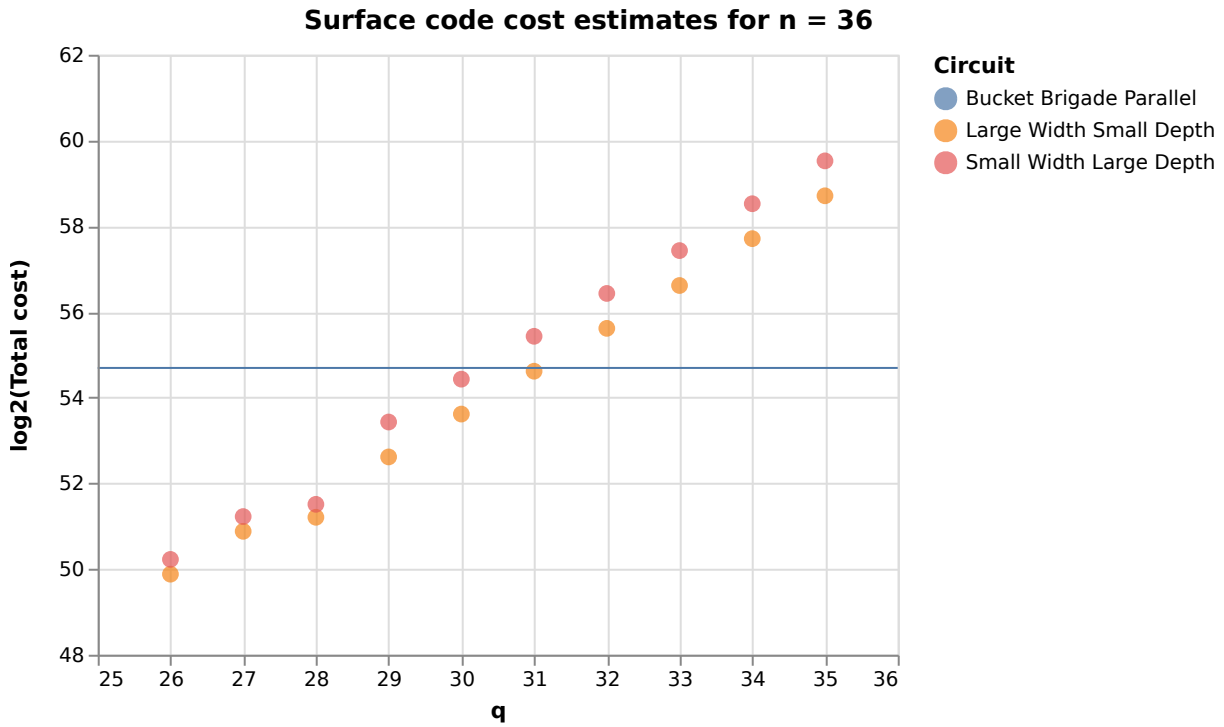


Figure 4.6: Cost vs. memory fullness q for basic circuits with $n = 36$ when embedded in the surface code. The horizontal line represents the parallel bucket brigade circuit which has only a fixed value of n. This matches closely with the analytical predictions in Figure 4.5, and we see that the cross-over point for memory fullness is around $q = 30$. The bumps in the graph correspond to increases in surface code distances of the magic state distilleries.

Circuit	n	q	Total time (s)	Physical qubits
Bucket brigade parallel	15	-	$3.60 \cdot 10^{-4}$	$8.49 \cdot 10^{7}$
Large width small depth	15	14	$1.25 \cdot 10^{-3}$	$1.47 \cdot 10^{8}$
Small width large depth	15	14	15.73	$1.07 \cdot 10^{4}$
Bucket brigade parallel	36	-	$2.16 \cdot 10^{-3}$	$4.16 \cdot 10^{14}$
Large width small depth	36	35	$8.70 \cdot 10^{-3}$	$1.97 \cdot 10^{15}$
Small width large depth	36	35	$1.51 \cdot 10^{8}$	$7.65 \cdot 10^{4}$

Table 4.2: Time and physical qubits required for fault-tolerant qRAM queries. The sizes $n = 15$ and $n = 36$ are analogous to 4KB and 8GB memory sizes respectively.

Our analysis shows that quantumly querying the 4KB qRAM can be done with nearly 100 million qubits in roughly 0.4ms (with parallel bucket brigade), or with 10000 qubits in roughly 16s (small width large depth), however the latter can only be used in cases where we know where the 1s in our memory are. As a reference point, modern-day RAMs have query times on the order of 150ns. To query this fast would first of all require significant advances in operational speed (recall our estimate of surface code cycle time

was an ambitious 200ns), as well as an astronomical amount of qubits.

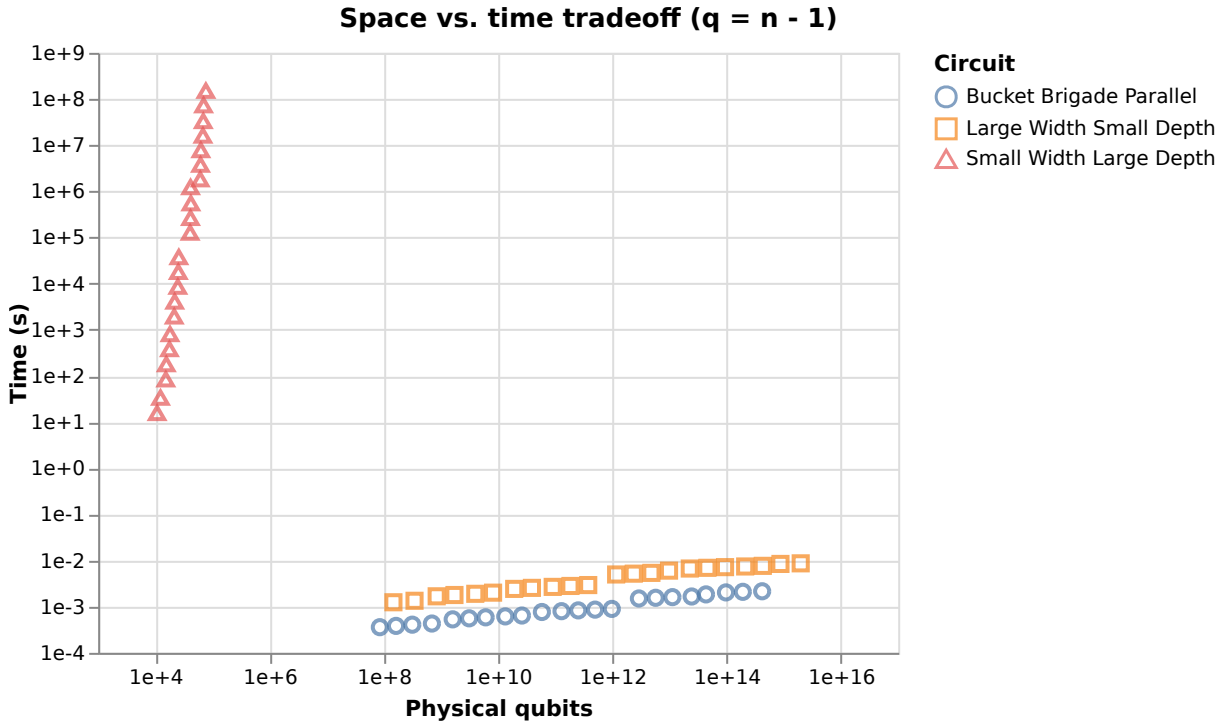


Figure 4.7: Space (physical qubits) vs. time tradeoff for basic circuits. Each point corresponds to a different memory size n from 15-36. Memory fullness q is set to $n-1$ for each n for the large width/depth circuits.

4.6 Hybrid query circuits

We now investigate a compromise between the two circuits in Section 4.5 by creating a sort of hybrid of the two extremes. Our motivation is to explore a wide range of options for the tradeoff between memory and depth to enable an algorithm designer to choose a qRAM implementation based on available resources.

We will need a larger running example: suppose our addresses are now 5 bits, and that addresses 00000, 01001, 10010, 11011, 00100, 01101, 10110, 11111 all contain the value 1 (2^q full addresses, $q = 3$).

4.6.1 Circuit design

The idea behind the hybrid circuit is, rather than checking the validity of all n bits of the address, check only the first k, and then use those outputs as controls for checking the rest of the bits. For brevity of analysis we will show here only the case where $k < q$. Full details for the case $k \geq q$ can be found in the accompanying code. We also must have $4 \leq k \leq n-3$, as recall the MPMCT implementations must have at least 4 control bits.

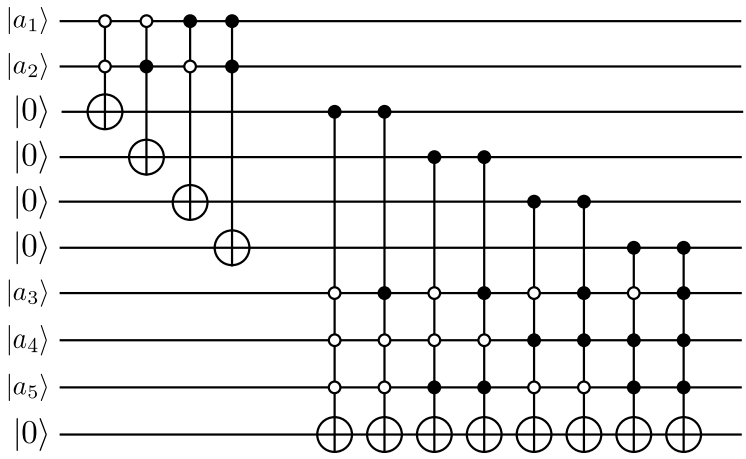


Figure 4.8: A basic hybrid circuit. The initial set of controlled gates recognizes the first k bits of an address; addresses that pass this condition go on to control readout of the remaining $n-k$ bits (here $n=5, q=3, k=2$).

The circuit for our running example is shown in Figure 4.8. In the worst case, there could be 2^k unique bit strings on the first k bits. Thus our top 'tier' consists of at most 2^k k-controlled MPMCTs, while the bottom tier will always consist of 2^q $n-k+1$-controlled MPMCTs. Again, in specific cases we could compute the ESOP of the Boolean expressions to simplify the products of MPMCTs, but here we assume them to be unknown.

We use again the decomposition of the MPMCTs in the previous section. The number of ancillae now depends on the size of $k-1$ vs. $n-k$ - we will need enough ancillae to implement the larger of the two gates. As the ancillae are returned to their initial state after use, we can use the same ancillae for all the gates in the sequence. In addition, we need extra qubits to store the intermediate results from the first tier; this is at most 2^k, one for each MPMCT. Finally, we will need an additional qubit for when we copy down the result before running everything backwards. Then,

$$N_Q = n + 2^k + 1 + \max(k-1, n-k) + 1. \tag{4.46}$$

We can use the results of the previous sections to estimate the other quantities as well,

$$\begin{aligned}
D &= 2(2^k(28k-60) + 2^q(28(n-k+1)-60)) + 1 && (4.47)\\
T_c &= 2(2^k(12k-20) + 2^q(12(n-k+1)-20)) && (4.48)\\
T_d &= 2(2^k \cdot 4(k-2) + 2^q \cdot 4(n-k+1-2)) && (4.49)\\
H_c &= 2(2^k(4k-6) + 2^q(4(n-k+1)-6)) && (4.50)\\
\mathrm{CNOT}_c &= 2(2^k(24k-40) + 2^q(24(n-k+1)-40)) + 1 && (4.51)
\end{aligned}$$

We can, in addition, parallelize the hybrid circuit in the same way that we parallelized the original deep circuit. There are three ways to do this: parallelize only the first tier (as shown in Figure 4.9), parallelize only the second tier, or parallelize both tiers. Parallelizing both tiers is clearly the best choice, as not fully parallelizing will incur additional cost (more

qubits) without seeing the full benefit in terms of time saved. We include these other approaches only as an intermediate step and plot them to show how they are sub-optimal. Resource counts for the aforementioned parallelizations can be found in the companion code https://github.com/glassnotes/FT_QRAM_Circuits.

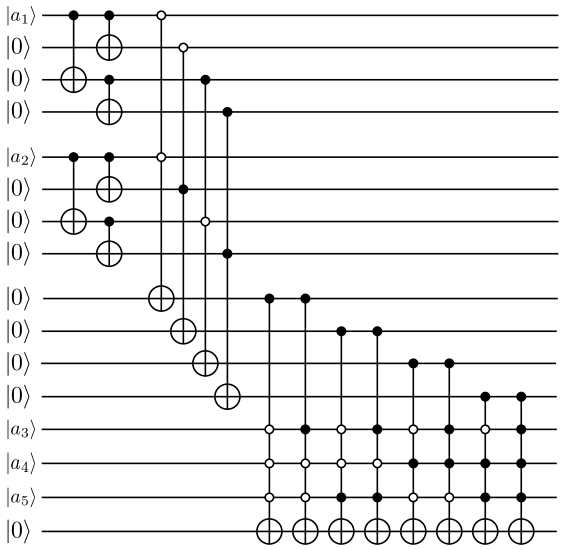


Figure 4.9: A hybrid circuit with its first tier parallelized. Parallelizing only one tier of a hybrid circuit leads to an increase in cost in the worst case, as a larger number of qubits are required for parallelization, while the other tier still runs in 'series' which may take a significant amount of time.

4.6.2 Surface code analysis

We now study the tradeoffs between n, q, and our new parameter k. We will again perform this in two ways, using $N_Q \times T_d$, as well as a full surface code analysis. The former is plotted in Figure 4.10, while the surface code results are shown in Figure 4.11 for a fixed n and two different values of q.

Unlike in the previous section where the dependence on n and q could be expressed simply as $O(n^2 2^q)$, the hybrid circuits carry a very complex, intertwined connection between n, q, and k. In particular, while we only ever see up to a quadratic dependence on n, we see many terms with exponential dependence on k and q, such as $nk2^k$, $n2^{k+q}$, etc. For the most part, Figure 4.10 shows a clear cut exponential dependence on k when n and q are fixed, deviating only at the extremal choices of k.

We observe that despite our best intentions, when no circuit optimization is performed, the cost of the basic hybrid circuit is actually worse - despite having the same number of MPMCTs but with fewer controls, the exponential increase in the number of logical qubits required (2^k of them) yields a much higher cost. Similarly, parallelizing only a single tier of the circuit was detrimental to the overall cost. As anticipated, the time saved in parallelizing only one part did not compensate for the substantial increase in the number of qubits required to do so.

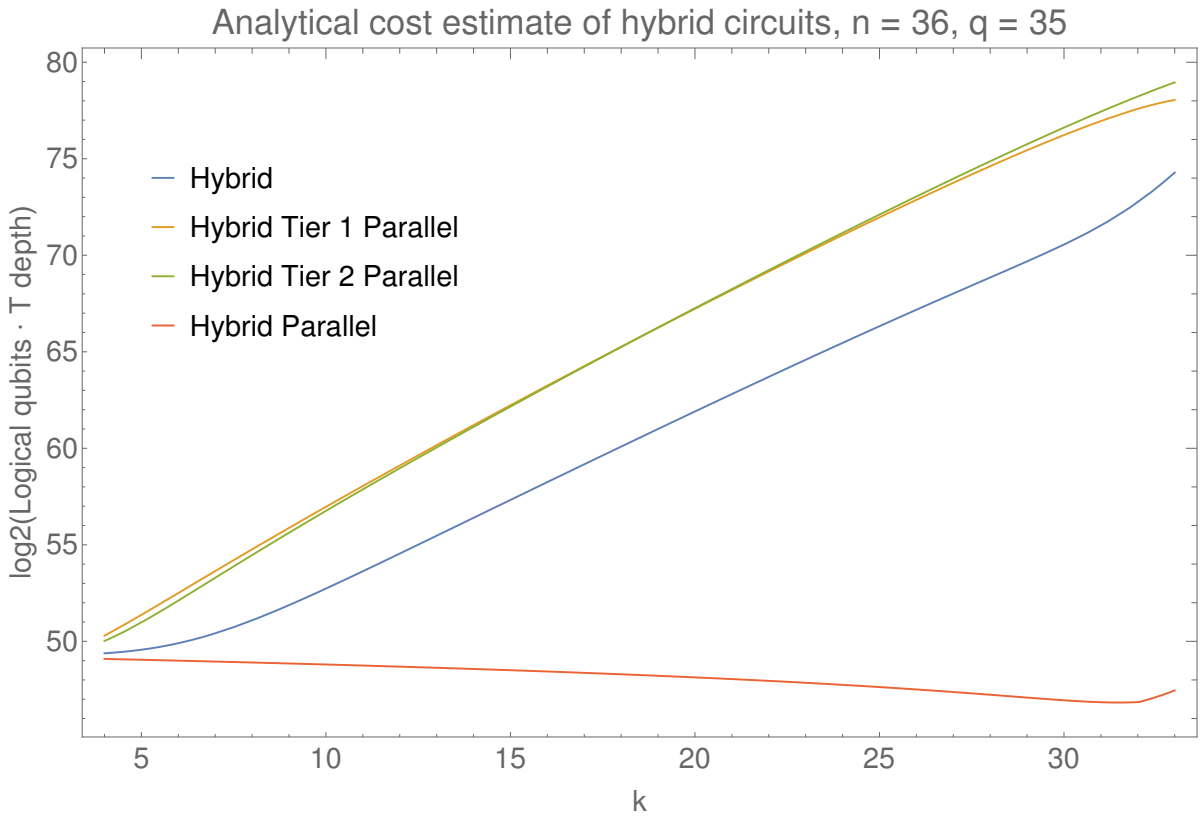


Figure 4.10: Analytical dependence of cost vs the hybrid splitting parameter k for a fixed memory size $n = 36$ and fullness $q = 35$. The partially parallelized versions do very poorly because parallelization of one half incurs a large overhead in the number of qubits, while running the other half takes a significant amount of time. Fully parallelizing the circuit yields lower costs overall, and we also observe an optimal value of k for a given n, q.

Where we do observe an improvement is with the fully parallelized hybrid circuit. In fact the cost of this circuit actually *decreases* when k is increased, and reaches a minimum which we can solve for analytically. For purposes of example, setting $n = 36, q = 35$, we differentiate the product $N_Q \times T_d$. There are two possibilities due to the presence of the max function in N_Q. Using the value $(n-k)2^q$ provides us with a first expression to solve:

$$2^k(1 + k \ln 2) = 2^{q+1}. \tag{4.52}$$

When $q = 35$, $k \approx 31.4$. Using $(k-1)2^k$ instead yields the second expression

$$2^k(2 - \ln 2 + 2k \ln 2) = 2^q, \tag{4.53}$$

which has solution $k \approx 29.6$. This corroborates the minimum seen in Figure 4.10 which occurs somewhere around $k = 31$. Furthermore, this analysis could be used by an algorithm designer to choose an optimal value of k based only on the fullness of the memory.

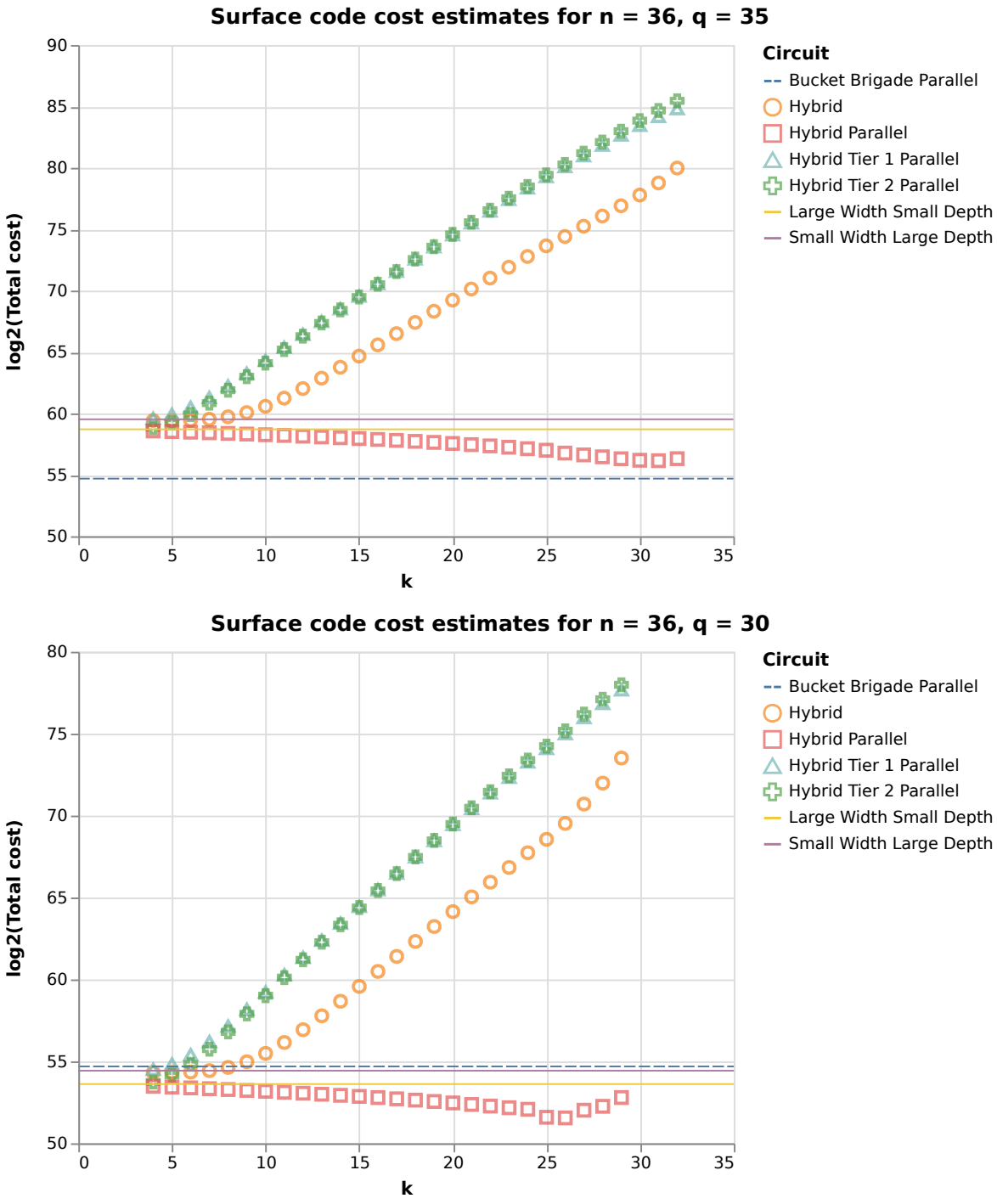


Figure 4.11: Surface code costs for hybrid circuits with $n = 36$, and $q = 35$, $q = 30$.

Figure 4.12 shows again the tradeoff between physical qubits and time for the hybrid circuits. For our 8GB case, to obtain millisecond-order query times we must still use on the order of 10^{15} physical qubits in the fully parallel version, while a million physical qubits in the basic version yields query times on the order of 3 years.

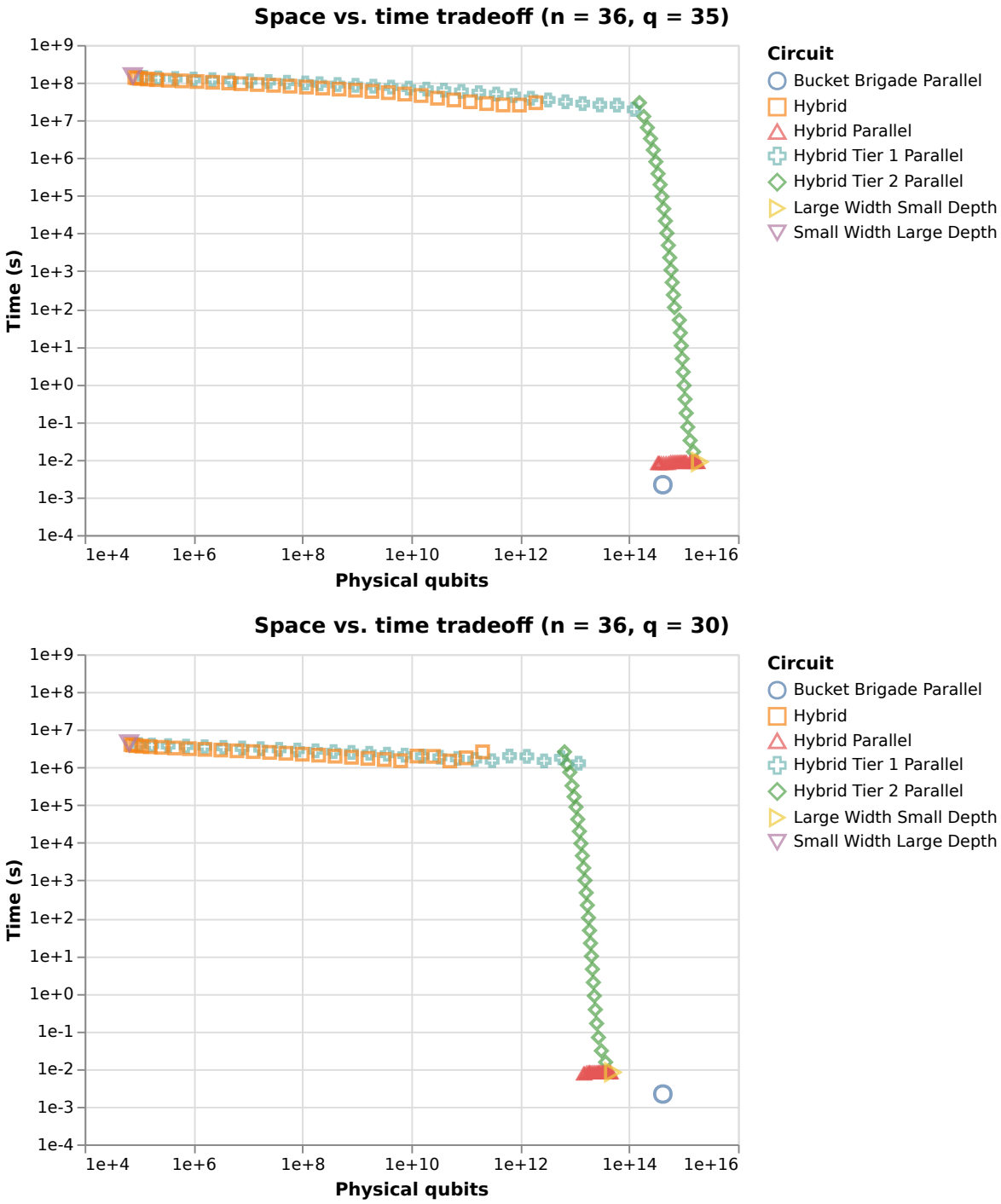


Figure 4.12: Time vs. physical qubits for the hybrid circuits for $n = 36$, and $q = 35, q = 30$. Distinct points represent different values of k from 4 to 32. Resource requirements for the bucket brigade circuit is fixed as there is no notion of q or k.

4.7 Special case: Cartesian product address structure

The resource estimates in the previous sections were all performed in the worst case, where we assume no knowledge of the structure of the address space. Armed with some information about the structure we can presumably take advantage of it and design special-purpose families of qRAM circuits. In this section we provide one such example of an address structure that yields a significant reduction in the resource count.

Consider the circuit shown in Figure 4.13. This circuit picks out 1s in the address space given by the Cartesian product $\{00, 01\} \times \{000, 100, 111, 101\}$, i.e. 8 total addresses, but uses only 6 MPMCTs. We will term such a circuit a 'double qRAM', as it looks like two circuits in the style of Figure 4.3 stacked on top of each other. The two parts of the address are checked for validity simultaneously, and the output is conditioned on the success of both parts.

We can parameterize such address spaces more generally. Suppose our memory contains 2^q locations with the value 1, and we are using a hybrid circuit with hybrid parameter k. Choose b_1, b_2 such that $b_1 + b_2 = q$. Store the values in the address space given by 2^{b_1} k-bit addresses on the first k bits, and 2^{b_2} $(n-k)$-bit addresses on the last $n-k$ bits.

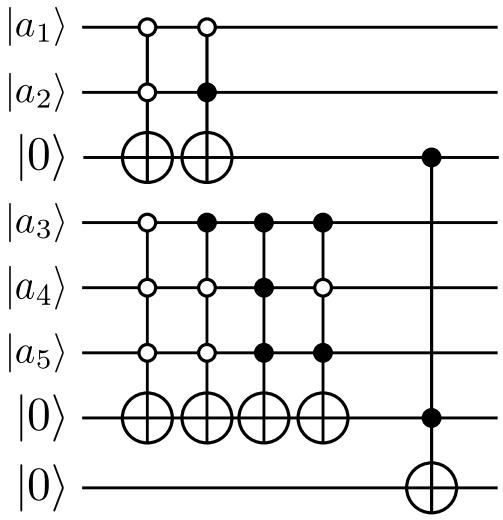


Figure 4.13: A double qRAM using a Cartesian product address space. Here $n = 5, q = 3, k = 2, b_1 = 1, b_2 = 2$. Such a circuit uses only 6 MPMCTs to pick out a total of 8 different addresses, as opposed to the 12 required to pick out 8 addresses in Figure 4.8.

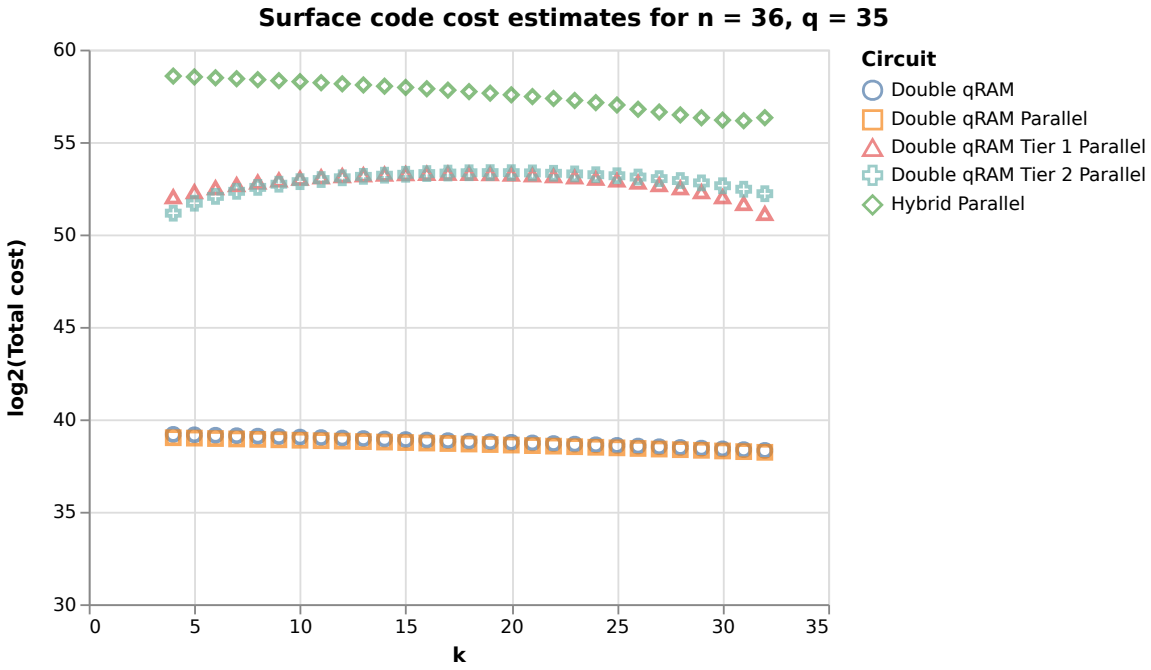


Figure 4.14: Cost vs. k for double qRAM circuits, highlighting the stark decrease in cost from the parallel hybrid model, when the same-sized address space is partitioned as a Cartesian product of two smaller spaces, i.e. a double qRAM. We chose the size of the first partition as $b_1 = 17$ bits. We note that the double qRAM and its fully parallelized version have very similar cost, not unlike the original wide vs. deep circuits. We also see that the cost of the partially parallelized versions peaks around $k = 18$, which makes sense due to the symmetry of the circuit when $b_1 \sim q/2$.

Resource counts can again be analyzed as in previous sections. Here, however, we no longer have at most 2^k k-controlled MPMCTs and 2^q $(n-k+1)$-controlled MPMCTs, rather we have by design 2^{b_1} k-controlled and $2^{b_2}(n-k)$-controlled MPMCTs. Furthermore, we no longer require 2^k qubits to hold the results of the first tier as in the hybrid models; we need only a single qubit, because all possible combinations of top-tier address and bottom-tier address yield valid addresses.

To determine a suitable choice of b_1 and b_2, we plotted the cost as a function of b_1 for a fixed n, q and varying k. We found choosing b_1, b_2 as roughly equal to $q/2$ to be the best choice, which is sensible as it is the most balanced choice, and it will not be the case that one tier takes significantly longer than the other. In this case, k need not be larger than $n/2$ due to the symmetry of the circuit structure. Figure 4.14 shows a comparison of cost vs. k for the double qRAM and the parallel hybrid circuit. Even for the double qRAM the tiered parallel versions have worse cost than their normal or fully parallel versions. However, we see that all types of double qRAM circuits have cost significantly less than any previous circuits.

Figure 4.15 shows the space vs. time tradeoffs in comparison to the parallel hybrid circuit. We see that the double qRAM circuits have overall fewer qubits and lower time

as compared to the analogous regular hybrid circuit. We see that we can query the 8GB qRAM in the millisecond range with just shy of 10^{10} qubits, or in the tens of minutes range with around 10000. We show for comparison the 4KB memory in Figure 4.16. Using the double qRAM we can always achieve query time of less than 1s, while the number of physical qubits ranges from nearly 10000 to nearly one million.

While such an address structure yields promising results, the pertinent question now becomes whether or not we can make good use of this structure in practice. We note that in theory it should be possible to map any address space into that of a Cartesian product, but performing such a mapping may destroy other structural properties of the data (for example the deliberate grouping of adjacent addresses). Furthermore, such a mapping is unlikely to be efficient, and this may negate any advantages gained from the new structure.

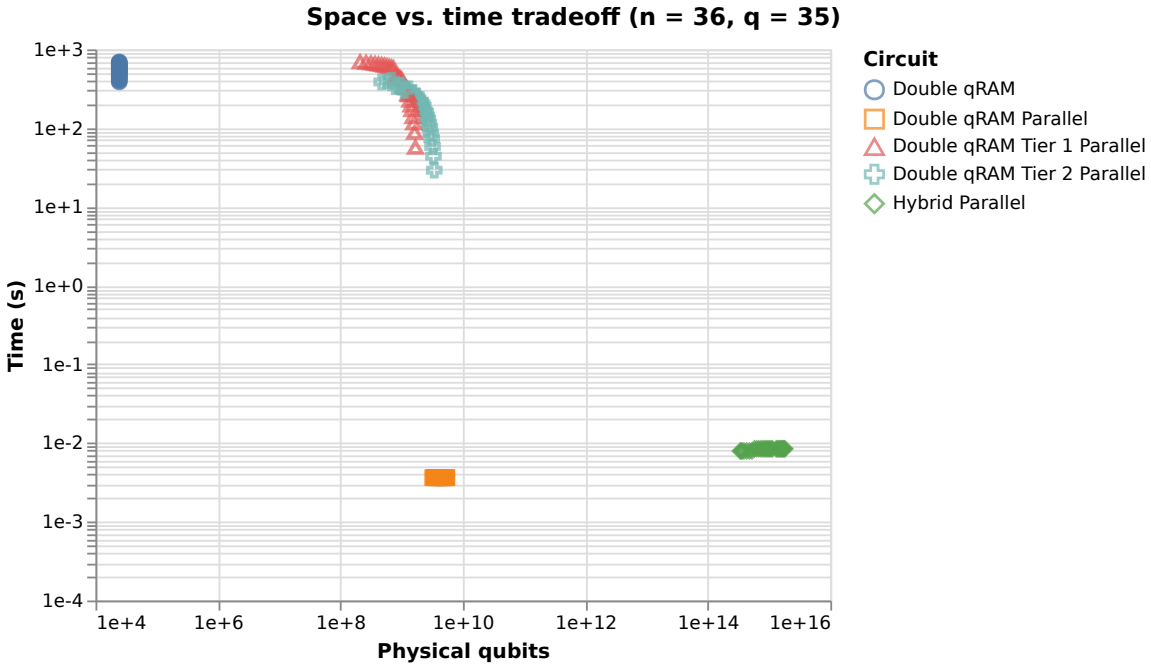


Figure 4.15: Comparison of time vs. physical qubits for $n = 36, q = 35$ for a double qRAM ($b_1 = 17$) and the parallel hybrid circuit. Each point corresponds to a different value of k. If the address structure affords it, the double qRAMs offer a better time/qubit tradeoff than all other circuits. In the parallelized versions, most of the cost savings comes from the smaller number of logical qubits required.

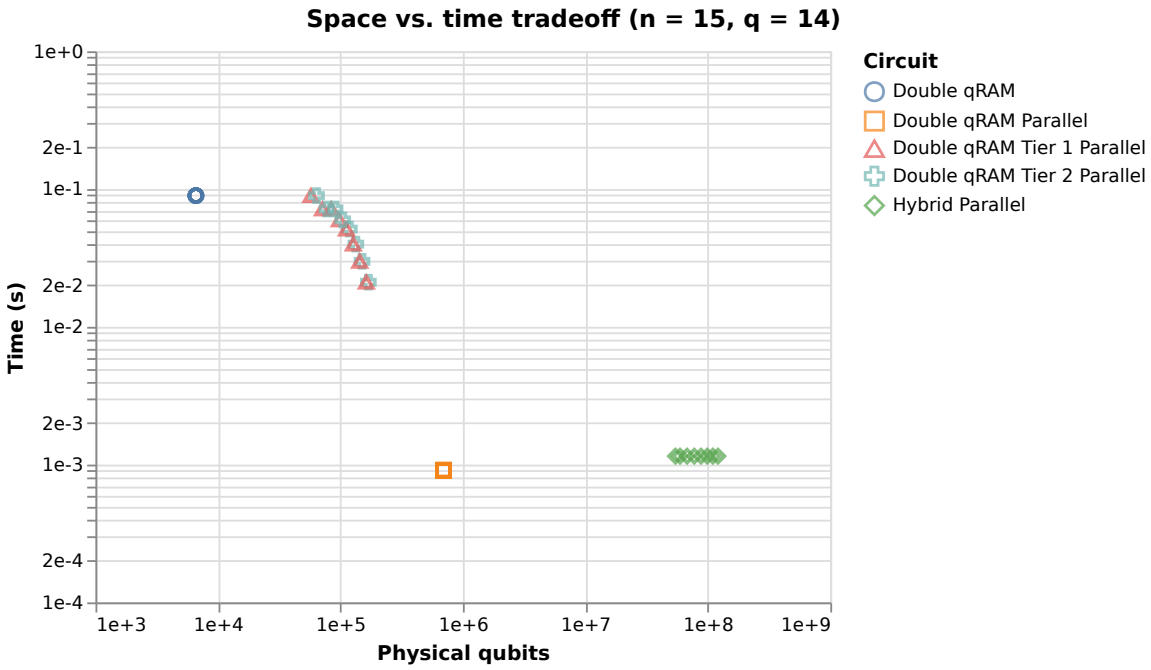


Figure 4.16: Comparison of time vs. physical qubits for $n = 15, q = 14$ for a double qRAM ($b_1 = 7$) and the parallel hybrid circuit. Each point corresponds to a different value of k. Overall resource requirements are significantly reduced as compared to Figure 4.15 however even for the double qRAM the time and number of qubits are beyond the capabilitites of current implementations.

4.8 Conclusion

We have presented a number of different circuit families which perform the task of a qRAM. It is important to note that our resource estimates were based on the worst-case situations of each. One should always, of course, do what's best for the problem at hand. For a specific algorithm, application of circuit synthesis and optimization techniques may yield lower cost for, e.g. one of the partially-parallelized hybrid circuits rather than the fully-parallelized versions.

Regardless, we can still draw some interesting conclusions from our analysis. First, unsurprisingly, to implement a fault-tolerant qRAM with as much memory as a current-generation laptop will remain an unfeasible task for the foreseeable future. Fast fault-tolerant query times ($\sim$ms) require quadrillions of qubits, and millions of qubits yield query times that are large and impractical. While circuit optimization may alleviate this to some degree, we are skeptical that it will have an impact that shaves off so many orders of magnitude.

Next is that we can take advantage of special address structures to reduce the amount of resources. We showed one such case with Cartesian product structure. Such structure is unlikely to occur in general, but when it does we can perform substantial optimization.

Finding such structures could be an interesting area for future work.

One significant opportunity for improvement is in the implementation of the surface code. Lattice surgery has recently shown to yield a decrease in resource estimates, in some cases lowering the number of physical qubits by a factor of 4 to 5 [91–93]. While a factor of 5 may not have much of an effect on circuits requiring quadrillions of qubits, it is promising for smaller qRAMs requiring on the order of 10000 qubits. Further improvements in fault-tolerant methods as well as advances in experimental techniques for reducing physical error rates may make small qRAMs feasible in the nearer term.

Chapter 5

Systematic selection of measurements for incomplete tomography

The associated codebase can be found at https://www.github.com/glassnotes/Balthasar.

Coarse graining techniques are used in many areas of physics. A classical example is that of the universality hypothesis and the behaviour of Ising models at a critical point (the ideas that eventually led to the renormalization group) [94]. We coarse grain by subdividing a lattice of Ising spins into chunks, and assign an effective spin up (or down) depending on which direction is more prevalent in each chunk. This gives us a 'zoomed-out' picture of the lattice, but in an effectively smaller space which, at criticality, retains the same properties as the original lattice.

In general, coarse graining is a tool used to reduce the complexity of a system. Recent work develops a more general framework for coarse graining state space in a quantum setting, as there is no obvious direct translation from other classical methods [95]. The work in this chapter focuses on the particular case of quantum state tomography of multi-qubit systems, and designs a coarse graining technique to select a subset of tomographic measurements based on the natural symmetries present in the state space.

Given an unknown multi-qubit system, it is unclear, at first, how one should partition a state space or qubits in such a way that 'averaging' over them in some sense will give us meaningful tomographic results. How should we choose such partitions? Should we partition qubits into groups and measure only certain groups fully, or perform measurements across the different groups? Should we partition in terms of qubits, or some other properties of the space? Furthermore, how many partitions should we choose?

This differs from other ideas using coarse graining in tomography that have been developed at the numerical level, such as 'binning' the outcomes of a continuous variable to

perform reconstruction more efficiently with a discretized version, and to perform tomography in the presence of noisy measurements [96–98].

In this work we coarse-grain discrete Wigner functions. Constructed by associating lines in phase space to quantum states, the values of a discrete Wigner function can be summed along a given line to find the probability of the system being found in that state. A coarse-grained Wigner function can give us a broader idea of what the state might be, such as the probability of finding it over a collection of states as opposed to individual ones.

Based on partitioning finite fields into cosets, this coarse graining can be accomplished in different ways for the same system by varying the cosets. The procedure naturally produces a subset of the observables required for full quantum state tomography. Different cosets produce observables with different structure - some subsets may be equivalent under unitary transformations to fully measuring only a part of the system, whereas others produce measurements that will probe only partially but spread out over the whole system. Therefore prior knowledge about the entanglement structure of a state might even suggest in advance a specific choice of coset.

Such a procedure may be used as a 'first pass' in cases when the dimension of an unknown system is large, and it is intractable to take the full set of measurements required. In particular, for an N qubit system where 2^N is a perfect square, we can coarse grain to sub-select $2^{\frac{N}{2}} + 1$ bases to measure in, as opposed to the $2^N + 1$ that are required for full tomography. This makes it efficient to try out different coset choices as well. One might then develop adaptive techniques to decide on a further subset of measurements to refine the reconstruction, taking advantage of the prior information provided by coarse graining.

Note: the published version contains an error in Equation (5.19) and Equation (C.4); the outer sum over λ should instead be a sum over α.

5.1 Abstract

We develop a systematic coarse-graining procedure for systems of N qubits. We exploit the underlying geometrical structures of the associated discrete phase space to produce a coarse-grained version with reduced effective size. Our coarse-grained spaces inherit key properties of the original ones. In particular, our procedure naturally yields a subset of the original measurement operators, which can be used to construct a coarse discrete Wigner function. These operators also constitute a systematic choice of incomplete measurements for the tomographer wishing to probe an intractably large system.

5.2 Introduction

Recently, the understanding of many-body quantum systems has dramatically progressed. Nowadays we are achieving an amazing degree of control over larger and larger systems [99,

100]. Therefore, verification during each stage of experimental procedures is of utmost importance; quantum tomography is the appropriate tool for that purpose.

The goal of quantum tomography is to reconstruct the state of a system by performing multiple measurements on identically prepared copies of the system. Once the experimental data are extracted, a numerical procedure determines which density matrix fits best the measurements. This estimation can be performed using different approaches, such as maximum likelihood estimation [101], or Bayesian methods [55, 56, 102, 103]. However, tomography becomes harder as we explore more intricate systems. If we look at the simple, yet illustrative case of N qubits, which will serve as the consistent thread in this paper, one has to make at least $2^N + 1$ measurements in different bases before one can claim to know everything about an *a priori* unknown system. With such an exponential scaling in the number of qubits, it is clear that current methods rapidly become intractable for present state-of-the-art experiments.

As a result, more sophisticated tomographical techniques are called for. New protocols try to simplify the process by making an educated guess about the nature of the state. Among other assumptions, this includes rank deficiency [104–108], extra symmetries [109–111], or Gaussianity [112]. While all these approaches are extremely efficient, their pitfall is that when the starting guess is inaccurate, they produce significant systematic errors.

Here, we pursue a different approach, inspired by a notion from statistical mechanics: coarse graining [113]. This operation transforms a probability density in phase space into a "coarse-grained" density that is a piecewise constant function, a result of density averaging in cells. This is the chief idea behind the renormalization group [114], which allows a systematic investigation of the changes of a physical system as viewed at different scales.

In our case, we consider a system of qubits and look at the associated phase space, which turns out to be a discrete grid of $2^N \times 2^N$ points. We assign to each suitably defined line in phase space a specific rank-one projection operator representing a pure quantum state. For each point of the grid, a suitable quasi-probability as the Wigner function can be directly computed from the measurement of the states associated with the lines passing through that point. We coarse grain by combining groups of these lines into thick lines, which we will show to be lines in the phase space of an effectively smaller system. Our coarse-grained phase spaces are endowed with many nice properties.

Most notably, our procedure systematically and naturally reveals a subset of measurements which one could use to perform incomplete tomography. In addition, using the coarse-grained points and lines, we show that one can define a discrete Wigner function in largely the same way as it is defined in the original space. When plotted, the coarse functions resemble smoothed out versions of the originals, preserving many of their prominent visual features.

5.3 Phase space of N qubits

A qubit is a two-dimensional quantum system, with Hilbert space isomorphic to $\mathbb{C}^2$. It is customary to choose two normalized orthogonal states, say $\{|0\rangle, |1\rangle\}$, as a computational basis. The unitary matrices

$$\sigma_z = |0\rangle\langle 0| - |1\rangle\langle 1|\,, \qquad \sigma_x = |0\rangle\langle 1| + |1\rangle\langle 0|\,, \tag{5.1}$$

generate the Pauli group $\mathcal{P}_1$, which consists of all the Pauli matrices plus the identity, with multiplicative factors $\pm 1, \pm i$ [115].

For N qubits, the corresponding Hilbert space is the tensor product $\mathbb{C}^2 \otimes \cdots \otimes \mathbb{C}^2 = \mathbb{C}^{2^N}$. A compact way of labeling both states and elements of the corresponding Pauli group $\mathcal{P}_N$ is by using the finite field $\mathbb{F}_{2^N}$. In Appendix B we briefly summarize the basic notions of finite fields needed to proceed.

Let $|\nu\rangle$, $\nu \in \mathbb{F}_{2^N}$, be an orthonormal basis in the Hilbert space $\mathbb{C}^{2^N}$ (henceforth, field elements will be denoted by Greek letters). The elements of the basis can be labeled by powers of a primitive element σ (i.e., a root of an irreducible primitive polynomial): $\{|0\rangle, |\sigma\rangle, \ldots, |\sigma^{2^N-1} = 1\rangle\}$. Now the equivalent version of Equation (5.1) is [116–118]

$$Z_\alpha = \sum_\nu \chi(\alpha\nu)\, |\nu\rangle\langle\nu|\,, \qquad X_\beta = \sum_\nu |\nu + \beta\rangle\langle\nu|\,, \tag{5.2}$$

so that

$$Z_\alpha X_\beta = \chi(\alpha\beta)\, X_\beta Z_\alpha\,, \tag{5.3}$$

which is the discrete counterpart of the Weyl-Heisenberg algebra for continuous variables [119]. Here, the additive character χ is defined as $\chi(\alpha) = \exp[i\pi\,\mathrm{tr}(\alpha)]$ and the trace of a field element (we distinguish it from the trace of an operator by the lower case "tr") is defined in Appendix B. Moreover, Z_α and X_β are related through the finite Fourier transform [53]

$$\mathcal{F} = \frac{1}{\sqrt{2^N}} \sum_{\nu,\nu'} \chi(\nu\,\nu')\, |\nu\rangle\langle\nu'|\,, \tag{5.4}$$

so that $X_\alpha = \mathcal{F}\, Z_\alpha\, \mathcal{F}^\dagger$.

The operators (5.2) generate the Pauli group $\mathcal{P}_N$ of N qubits and, with a suitable choice of basis, they can be factorized into a tensor product of single-qubit Pauli operators. To this end, it is convenient to consider $\mathbb{F}_{2^N}$ as an N-dimensional linear space over $\mathbb{Z}_2$. It is spanned by an abstract basis $\{\theta_1, \ldots, \theta_N\}$, so that given a field element α the expansion

$$\alpha = \sum_{i=1}^N a_i\, \theta_i\,, \qquad a_i \in \mathbb{Z}_2\,, \tag{5.5}$$

allows us the identification $\alpha \Leftrightarrow (a_1, \ldots, a_N)$. The basis $\{\theta_i\}$ can be chosen to be orthonormal with respect to the trace operation; i.e., $\mathrm{tr}(\theta_i\,\theta_j) = \delta_{ij}$. This is a self-dual basis, which always exist for the case of qubits. In this way, we associate each qubit with a particular

element of the self-dual basis: $\text{qubit}_i \Leftrightarrow \theta_i$. Using this basis, we have the factorization

$$Z_\alpha = \sigma_z^{a_1} \otimes \cdots \otimes \sigma_z^{a_N}, \qquad X_\beta = \sigma_x^{b_1} \otimes \cdots \otimes \sigma_x^{b_N}, \tag{5.6}$$

where $a_i = \text{tr}(\alpha\theta_i)$ and $b_i = \text{tr}(\beta\theta_i)$ are the corresponding expansion coefficients for α and β in the self-dual basis.

We next recall [120, 121] that the grid defining the phase space for N qubits can be appropriately labeled by the discrete points (α, β), which are precisely the indices of the operators Z_α and X_β: α is the "horizontal" axis and β the "vertical" one. In this grid we can introduce the set of displacements

$$D(\alpha, \beta) = \Phi(\alpha, \beta)\, Z_\alpha X_\beta\,, \tag{5.7}$$

where $\Phi(\alpha, \beta)$ is a phase required to avoid plugging extra factors when acting with D. A sensible choice for the case of qubits is $\Phi^2(\alpha, \beta) = \chi(\alpha\beta)$, which ensures the Hermiticity of the displacement operators. In addition, we impose $\Phi(\alpha, 0) = 1$ and $\Phi(0, \beta) = 1$, which means that the displacements along the "position" axis α and the "momentum" axis β are not associated with any phase. These displacement operators shift phase space points, so the action of $D(\alpha', \beta')$ maps $(\alpha, \beta) \mapsto (\alpha + \alpha', \beta + \beta')$, justifying their designation. Note that we still have to fix the sign of the phase $\Phi(\alpha, \beta)$. We choose the phase as

$$\Phi(\alpha, \beta) = i^{\text{tr}(\alpha\beta)}(-1)^{f(\alpha\beta)}\,, \tag{5.8}$$

where $f(x) = \sum_{0 \le j < i \le m-1} x^{2^i + 2^j}$, which ensures that the operators defined in Equation (5.17) below are rank-one projections.

On the phase space grid one can introduce a variety of geometrical structures with much the same properties as in the continuous case [122–124]. The simplest are the straight lines passing through the origin (also called rays), with equations

$$\alpha = 0, \qquad \text{or} \qquad \beta = \lambda\alpha\,. \tag{5.9}$$

The rays have a very remarkable property: the monomials $D(\alpha, \beta)$ belonging to the same ray commute, and thus, have a common system of eigenvectors $\{|\psi_{\nu,\lambda}\rangle\}$,

$$D(\alpha, \lambda\alpha)|\psi_{\nu,\lambda}\rangle = \exp(i\xi_{\nu,\lambda})|\psi_{\nu,\lambda}\rangle, \tag{5.10}$$

where λ is fixed and $\exp(i\xi_{\nu,\lambda})$ is the corresponding eigenvalue, so $|\psi_{\nu,0}\rangle = |\nu\rangle$ are eigenstates of Z_α (displacement operators labeled by the ray $\beta = 0$, which we take as the horizontal axis). The projection operators associated with the lines of equal slope are the projections onto these eigenvactors. Indeed, we have that

$$|\langle\psi_{\nu,\lambda}|\psi_{\nu',\lambda'}\rangle|^2 = \delta_{\lambda,\lambda'}\delta_{\nu,\nu'} + \frac{1}{2^N}(1 - \delta_{\lambda,\lambda'}), \tag{5.11}$$

and, in consequence, they are mutually unbiased bases (MUBs) [52].

Now suppose for each ray we disregard the origin $(0, 0)$, whose monomial is the identity

operator. This leaves us with $2^N - 1$ commuting operators. If we then consider the whole bundle of $2^N + 1$ rays (which are obtained by varying the "slope" λ over all of $\mathbb{F}_{2^N}$), we can construct a complete set of MUB operators arranged in a $(2^N - 1) \times (2^N + 1)$ table [125].

To round up the scenario, we need to represent states in phase space. The discrete Wigner function [126] is the appropriate tool. It can be considered as an invertible mapping

$$W_\varrho(\alpha, \beta) = \frac{1}{2^N} \operatorname{Tr}[\varrho\, \Delta(\alpha, \beta)]\,, \tag{5.12}$$

so that

$$\varrho = \sum_{\alpha,\beta} \Delta(\alpha, \beta)\, W_\varrho(\alpha, \beta)\,. \tag{5.13}$$

The operational kernel is defined as

$$\Delta(\alpha, \beta) = \frac{1}{2^N} \sum_{\alpha',\beta'} \chi(\alpha\alpha' - \beta\beta')\, D(\alpha', \beta')\,, \tag{5.14}$$

which, in view of Equation (5.4), can be interpreted as a double Fourier transform of $D(\alpha, \beta)$. One can check that this kernel has all the good properties [127]: it is Hermitian, normalized and covariant under the Pauli group. As a result, for each point on the grid, the corresponding value of the Wigner function can be computed from the probabilities of measuring the pure states associated with the lines passing through that point.

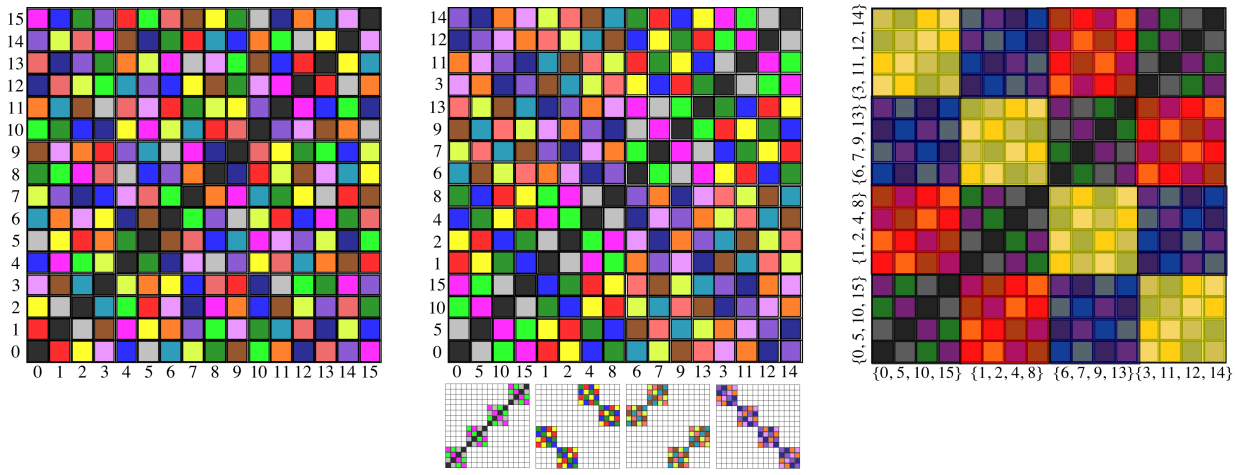


Figure 5.1: Graphical sketch of coarse graining. Here we consider dimension 16, and its diagonal ray, $\beta = \alpha$. The first panel plots all the lines of the form $\beta = \alpha + \gamma$, parametrized by the shift γ. Points on the same line have the same colour. Axis labels correspond to powers of the primitive element of $\mathbb{F}_{16}$, with the convention that σ^0 is denoted by 0 and $\sigma^{15} = 1$. The middle panel shows the original grid with the axis labels permuted such that coset elements are grouped together. We can see that this leads to distinct 4×4 blocks containing points of exactly four different colours. These are shown expanded out in the small, lower four grids. One notices that these "coarse" blocks form the diagonal ray and all its translates in dimension 4, which we show superimposed in the last panel.

5.4 Coarse graining

As heralded in the Introduction, our goal is to tailor a procedure that allows us to coarse grain the phase space of a multiqubit system; i.e., to break it down into simpler subcomponents.

To this end, we consider the number N of qubits to be composite, i.e. $N = mn$. Let $\{\mu_0, \ldots, \mu_{n-1}\}$ be a basis of $\mathbb{F}_{2^{mn}}$ with respect to $\mathbb{F}_{2^m}$. We define

$$\mathfrak{C}_0 = \left\{ \sum_{j=1}^{n-1} \tau_j \mu_j \;\middle|\; \tau_j \in \mathbb{F}_{2^m} \right\}, \tag{5.15}$$

i.e., the subspace made of linear combinations of basis elements $\mu_1, \ldots, \mu_{n-1}$ with coefficients in the base field $\mathbb{F}_{2^m}$. We can use this set $\mathfrak{C}_0$, which we henceforth refer to as the initial coset, to decompose the field $\mathbb{F}_{2^{mn}}$ into cosets:

$$\mathfrak{C}_\tau = \tau\mu_0 + \mathfrak{C}_0, \qquad \tau \in \mathbb{F}_{2^m}. \tag{5.16}$$

The coarse-grained space will be labeled according to these cosets.

We can imagine the process of coarse graining as partitioning the grid $\mathbb{F}_{2^{mn}} \times \mathbb{F}_{2^{mn}}$ in such a way that we superimpose a grid of size $2^m \times 2^m$ on top, with each superimposed point indexed by cosets rather than field elements in the original grid. Each point in the coarse grid then contains a sub-grid the same size as $\mathbb{F}_{2^m}^{n-1} \times \mathbb{F}_{2^m}^{n-1}$. To provide some intuition for this, we show a visual example of this process in action in Figure 5.1.

Our procedure for coarse-graining the grid arises naturally from consideration of the line structure of phase space. We will use the *thin* lines in $\mathbb{F}_{2^{mn}}$ to create *thick* lines in the coarse phase space, by grouping together lines having the same slope, and with intercepts in the same coset. We write thin lines in the big field $\mathbb{F}_{2^{mn}}$ as $|\ell_\gamma^{(\lambda)}\rangle$, where λ is the slope, and γ is the intercept. A large, coarse-grained line is denoted as $|L_{\mathfrak{C}_\tau}^{(\lambda)}\rangle$, where now the intercept is a whole coset.

To each line in the fine-grained phase space we can assign a projector $|\ell_\gamma^{(\lambda)}\rangle\langle\ell_\gamma^{(\lambda)}|$, constructed as a linear combination of the displacement operators. We choose as our convention for the rays ($\gamma = 0$) the all-positive sum

$$|\ell_0^{(\lambda)}\rangle\langle\ell_0^{(\lambda)}| = \frac{1}{2^{mn}} \sum_\alpha D(\alpha, \lambda\alpha). \tag{5.17}$$

These lines are eigenstates with eigenvalue $+1$ for all displacement operators in the sum. Projectors with nonzero intercepts are obtained by conjugating that of the ray with an appropriate displacement operator.

The coarse lines are produced by grouping together lines with intercepts in the same coset:

$$|L_{\mathfrak{C}_\tau}^{(\lambda)}\rangle\langle L_{\mathfrak{C}_\tau}^{(\lambda)}| = \sum_{\gamma \in \mathfrak{C}_\tau} |\ell_\gamma^{(\lambda)}\rangle\langle\ell_\gamma^{(\lambda)}|. \tag{5.18}$$

The possible choices of slope for these lines will be limited to elements of the subfield $\mathbb{F}_{2^m}$, as these have natural analogues between the two fields.

As discussed in more detail in Appendix C, the coarse rays of Equation (5.18) can be simplified and rewritten as the sum of displacement operators

$$|L_{\mathfrak{C}_0}^{(\lambda)}\rangle\langle L_{\mathfrak{C}_0}^{(\lambda)}| = \frac{1}{2^{mn}} \sum_{\lambda} \left[\sum_{\gamma \in \mathfrak{C}_0} \chi(\gamma\alpha) \right] D(\alpha, \lambda\alpha)\,. \tag{5.19}$$

One can check here that the inner sum over the elements of $\mathfrak{C}_0$ will cause some of the displacement operators to vanish. The sum in brackets in Equation (5.19) is either zero or a positive constant. Hence, the projection associated to the thick lines are a sum over a subset of the displacement operators associated with the thin lines. This leads us to the key idea of our work: rather than measuring all the displacement operators, we measure only those which are present in the rays of the coarse-grained space.

We note here that the choice of $\mathfrak{C}_0$ is not unique, and will ultimately determine the resultant set of displacement operators. For example, a special case occurs when the dimension of the system is square. In this case, we can consider the relationship between the fields as a quadratic field extension, i.e. when $n = 2$. In this case we can partition $\mathbb{F}_{2^{2m}}$ into $\mathbb{F}_{2^m} \times \mathbb{F}_{2^m}$. We can then choose the initial coset as the copy of the subfield $\mathbb{F}_{2^m} \subset \mathbb{F}_{2^{2m}}$:

$$\mathfrak{C}_0 = \{\sigma^{i(2^m+1)},\ \ i = 0, \ldots, 2^m - 1\}\,, \tag{5.20}$$

where σ is a primitive element of $\mathbb{F}_{2^{mn}}$ and we use the notation σ^0 for 0. The subsequent cosets are obtained additively from this subfield using the representatives $\tau_i = \sigma^{2^m(i-1)+i}$.

Finally, the coarse-grained phase space inherits a coarse-grained Wigner function. A coarse kernel can be constructed by grouping together kernel operators from the same coset, i.e.

$$\mathfrak{D}(\mathfrak{C}_\tau, \mathfrak{C}_\xi) = \sum_{\alpha \in \mathfrak{C}_\tau} \sum_{\beta \in \mathfrak{C}_\xi} \Delta(\alpha, \beta)\,. \tag{5.21}$$

Desired properties of a Wigner function all follow from the original kernel. As was the case with the displacement operators, differing choices of the subset $\mathfrak{C}_0$ will lead to differing Wigner functions.

5.5 Examples

We illustrate the previous ideas with some relevant examples. We have written a Python software package capable of generating all the following results, which we make available online [128]

The first nontrivial instance we can have is the case of two qubits, so dimension 4. Using the irreducible primitive polynomial $x^2 + x + 1 = 0$, we have that $\mathbb{F}_4 = \{0, 1, \sigma, \sigma^2 = \sigma + 1\}$. The self-dual basis is $\{\sigma, \sigma + 1\}$, and we use it to produce the displacement operators.

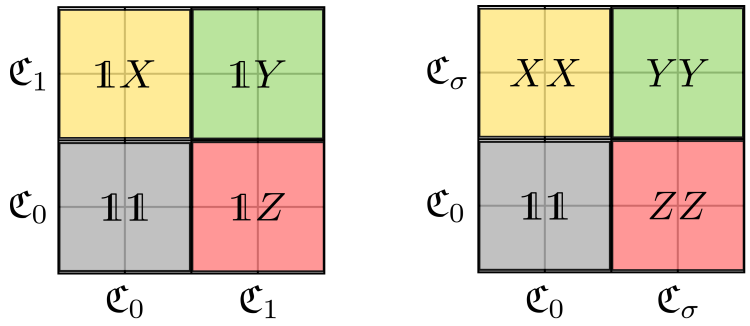


Figure 5.2: Resultant operators from coarse-graining a dimension 4 system down to dimension 2. Colours are indicative of particular coarse rays. The left image coarse grains by taking $\mathfrak{C}_0 = \{0, \sigma\}$, whereas the lower image uses the subfield $\mathfrak{C}_0 = \{0, 1\}$.

Another basis for $\mathbb{F}_4/\mathbb{F}_2$ is $\{1, \sigma\}$. Taking all scalar multiples of $\mu_1 = \sigma$ from the prime field gives us $\mathfrak{C}_0 = \{0, \sigma\}$. We then obtain $\mathfrak{C}_1 = 1 + \mathfrak{C}_0 = \{1, \sigma^2\}$. For each ray, we can list the operators which survive in the inner sum over $\mathfrak{C}_0$ in Equation (5.19). Moreover, we can label the points of the coarse-grained grids by those displacement operators. Disregarding the identity operator, the resulting set $\{\mathbb{1}X, \mathbb{1}Z, \mathbb{1}Y\}$ constitutes the appropriate measurements to be performed to determine which coarse-grained line they are in. They are essentially Pauli measurements on one of the two qubits in the system.

Alternatively, the dimension is a square, so we can choose as our initial coset the subfield $\mathbb{F}_2$: $\mathfrak{C}_0 = \{0, 1\}$. This yields the second coset $\mathfrak{C}_\sigma = \{\sigma, \sigma^2\}$. We once again compute the surviving operators using Equation (5.19). The final result now is $\{XX, YY, ZZ\}$. Here, we see that we are making a measurement with the same Pauli operator on both qubits, thereby capturing the full correlations between the two qubits. Figure 5.2 shows both partitioning methods side by side.

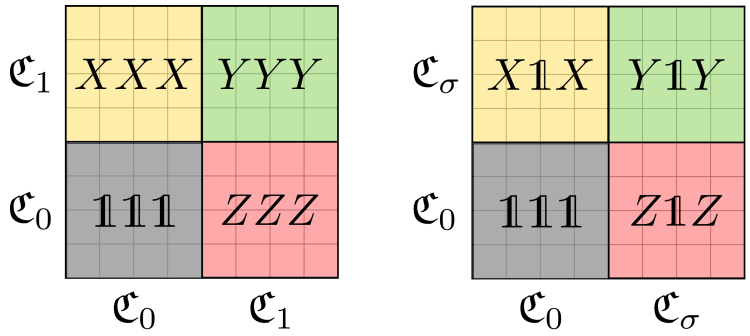


Figure 5.3: Resultant operators from coarse-graining a dimension 8 system down to dimension 2. (Left panel) Coarse graining using the basis $\{1, \sigma, \sigma^2\}$. The resultant measurements are unitarily equivalent to a case where two of the qubits remain untouched. (Right panel) Resultant operators when the coarse-graining uses the initial basis $\{\sigma, \sigma^4, \sigma^5\}$. Here we obtain the interesting result that all resultant operators commute.

Our next example is the case of dimension 8. We choose σ a root of the irreducible primitive polynomial $x^3 + x + 1 = 0$, and obtain a self-dual basis $\{\sigma^3, \sigma^5, \sigma^6\}$. An obvious

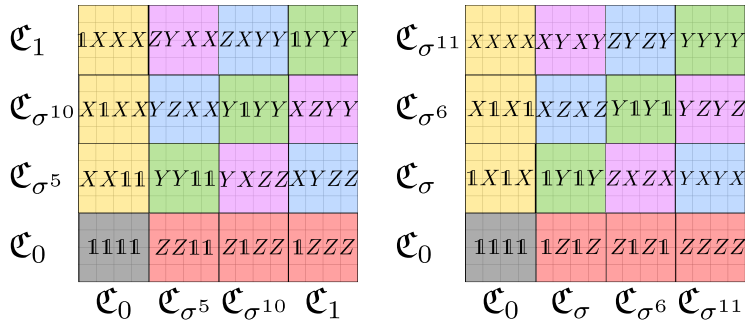


Figure 5.4: Resultant operators from coarse-graining a dimension 16 system down to dimension 4. The left panel contains the surviving operators from the general basis method, the right panel from choosing the subfield as $\mathfrak{C}_0$. The cosets are listed in Equation (5.25) and Equation (5.26) respectively. In the case of the left panel, these operators are unitarily equivalent to a set where two qubits are untouched and the 2-qubit MUB operators are applied to the rest. The right panel has no such transformation.

choice for a basis of $\mathbb{F}_8/\mathbb{F}_2$ is a polynomial basis $\{1, \sigma, \sigma^2\}$. To construct $\mathfrak{C}_0$, we must take all possible linear combinations of σ and σ^2 with coefficients in $\mathbb{F}_2$. This produces

$$\mathfrak{C}_0 = \{0, \sigma, \sigma^2, \sigma^4\}. \tag{5.22}$$

We obtain the second coset by adding the remaining subfield element 1 to $\mathfrak{C}_0$:

$$\mathfrak{C}_1 = \{1, \sigma^3, \sigma^5, \sigma^6\}. \tag{5.23}$$

The traces of all elements in $\mathfrak{C}_0$ are 0, and the traces for all elements in $\mathfrak{C}_1$ are 1. The surviving four operators are shown in Figure 5.3.

Using a Clifford transformation, we can "trace out" two of the qubits. The sequence of CNOT gates: $\mathrm{CNOT}_{12} - \mathrm{CNOT}_{13} - \mathrm{CNOT}_{21} - \mathrm{CNOT}_{31}$ transforms the set into $\{X\mathbb{1}\mathbb{1}, Z\mathbb{1}\mathbb{1}, Y\mathbb{1}\mathbb{1}\}$, so we see that this partitioning is, after a global change of basis, equivalent to measuring each Pauli on only a single qubit.

If we choose instead the basis $\{\sigma, \sigma^4, \sigma^5\}$ to build our cosets, we get a more interesting result:

$$\mathfrak{C}_0 = \{0, 1, \sigma^4, \sigma^5\}, \qquad \mathfrak{C}_\sigma = \{\sigma, \sigma^2, \sigma^3, \sigma^6\}. \tag{5.24}$$

The operators that survive have the form $Z_\alpha X_\beta$, $\alpha, \beta \in \{0, \sigma^4\}$, yielding the operators in Figure 5.3, which all commute. In this case, we are already ignoring one of the three qubits. However, it is not possible to find a Clifford which will trace out a remaining one as was the case with the polynomial basis case. So, in a sense, using this partitioning we are ignoring fewer qubits than before.

Dimension 16 is perhaps the first really interesting case. First of all, we can consider it in two ways: $m = 1, n = 4$, or $m = 2, n = 2$. Essentially, to do the partitioning, we can look at $\mathbb{F}_{16}$ as a quartic extension over $\mathbb{F}_2$, or a quadratic extension over $\mathbb{F}_4$. We consider the quadratic case, so we can coarse grain in two ways. We work with $\mathbb{F}_{16}$ as constructed by the irreducible primitive polynomial $x^4 + x + 1$ over $\mathbb{F}_2$, and $x^2 + x + \sigma'$ over $\mathbb{F}_4$ where we

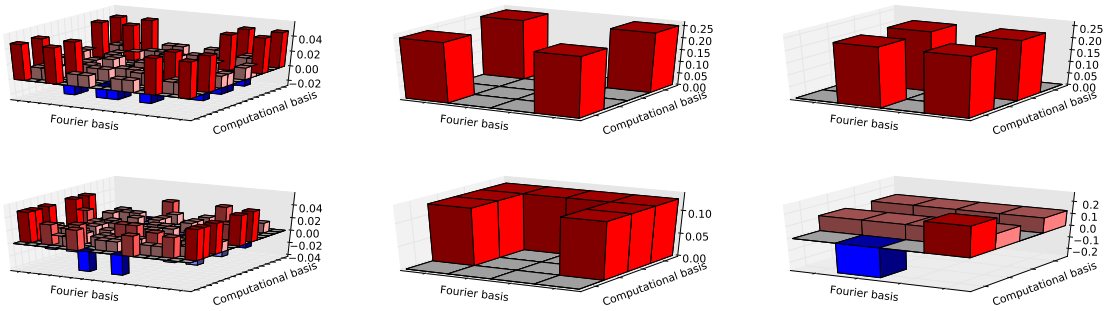


Figure 5.5: (Top) Coarse-grained Wigner function for the state $\frac{1}{2}(|00\rangle + |11\rangle) \otimes (|00\rangle + |11\rangle)$. (Left) The original Wigner function in dimension 16. The x-axis represents the computational basis, in the standard ordering $|0000\rangle, |0001\rangle, |0010\rangle$, etc. The Fourier basis, as defined via Equation (5.4), is on the y-axis and is similarly ordered. (Centre) Coarse graining over $\mathbb{F}_4$ with the polynomial basis $\{1, \sigma\}$. Here the axes are not labeled by single states, but rather by a set of states associated with each coset. (Right) Coarse graining with the subfield as the initial coset. (Bottom) The same coarse graining procedure as above, but applied to the state $\frac{1}{2}(|0001\rangle + |0010\rangle + |0100\rangle + |1000\rangle)$.

denote a primitive element of $\mathbb{F}_4$ as σ'. We know from Equation (5.20) that $\sigma' = \sigma^5$, where σ is the primitive element in $\mathbb{F}_{16}$. Then $\mathbb{F}_4$ in $\mathbb{F}_{16}$ can be written as $\{0, \sigma^5, \sigma^{10}, \sigma^{15} = 1\}$.

For the general case, we choose the basis $\{1, \sigma\}$. Taking all $\mathbb{F}_4$-multiples of σ, we obtain $\mathfrak{C}_0 = \{0, \sigma, \sigma^6, \sigma^{11}\}$. The full set of cosets is:

$$\begin{aligned} \mathfrak{C}_0 &= \{0, \sigma, \sigma^6, \sigma^{11}\}, & \mathfrak{C}_{\sigma^5} &= \{\sigma^5, \sigma^2, \sigma^9, \sigma^3\}, \\ \mathfrak{C}_{\sigma^{10}} &= \{\sigma^{10}, \sigma^8, \sigma^7, \sigma^{14}\}, & \mathfrak{C}_1 &= \{1, \sigma^4, \sigma^{13}, \sigma^{12}\}. \end{aligned} \tag{5.25}$$

Proceeding in the standard way, and taking into account that a self-dual basis is $\{\sigma^3, \sigma^7, \sigma^{12}, \sigma^{13}\}$, we obtain the operators in Figure 5.4. What is (un)interesting about these operators is that we can transform them all into operators which completely ignore two of the qubits. In particular, consider the following sequence of operations: CNOT_{43} – CNOT_{32} – CNOT_{31} – CNOT_{14} – CNOT_{24}. Application of this to the operators of the first panel of Figure 5.4 yields a new set of operators where the last two qubits contain only $\mathbb{1}$, and the first two qubits contain the full set of MUB operators on two qubits.

Alternatively, we can choose our initial coset as the subfield, and the coset representatives as $\tau_i = \sigma^{4(i-1)+i}$. We obtain the cosets

$$\begin{aligned} \mathfrak{C}_0 &= \{0, 1, \sigma^5, \sigma^{10}\}, & \mathfrak{C}_\sigma &= \{\sigma, \sigma^4, \sigma^2, \sigma^8\}, \\ \mathfrak{C}_{\sigma^6} &= \{\sigma^6, \sigma^{13}, \sigma^9, \sigma^7\}, & \mathfrak{C}_{\sigma^{11}} &= \{\sigma^{11}, \sigma^{12}, \sigma^3, \sigma^{14}\}. \end{aligned} \tag{5.26}$$

Using Equation (5.19) we get the table shown in the right panel of Figure 5.4. Unlike in the previous case, there is no transformation which will lead to us 'tracing out' two of the qubits. However, we can bring these operators into a more basic form by applying the sequence CNOT_{13} – CNOT_{24}. The resultant operators have the property that on the first

two qubits, we only have X, and on the last two qubits only Z, so that they all commute.

To conclude, we present some of the coarse-grained Wigner functions we obtain using our method. Those in dimensions 4 and 8 are somewhat trivial, so we focus on dimension 16. Wigner functions for the states $\frac{1}{2}(|00\rangle + |11\rangle)(|00\rangle + |11\rangle)$ and $\frac{1}{2}(|0001\rangle + |0010\rangle + |0100\rangle + |1000\rangle)$ are presented in Figure 5.5.

Recall in Section 5.3 that we could associate the elements of $\mathbb{F}_{2^N}$ with a basis in our Hilbert space. Then, in the coarse Wigner functions, when we group the field elements into cosets, we can consider this also as grouping together the associated basis states. Hence, the probabilities in these Wigner functions become distributed over the cosets which contain the constituent basis states of our target state. As a result, the coarse Wigner functions resemble 'smoother' versions of the original one to varying degrees.

5.6 Conclusions

Compared to the continuous Wigner function, the discrete Wigner function is an adolescent formulation, slowly developing into adult maturity. Discrete phase space imposes several new challenges, which leads to an intricate mapping of the Wigner function.

Our coarse graining procedure shows a way to facilitate our understanding when the number of qubits is high. While it is always possible to ignore part of the system and to determine the full Wigner function of the resulting reduced density matrix, our approach allows more choices regarding which information of the whole system is measured. In another extremal case, the coarse-grained Wigner function is completely determined by a set of commuting operators that can be measured simultaneously.

However, several open questions remain. An obvious next step would be to extend the coarse graining procedure to multi-qudit systems. Furthermore, knowing the coarse-grained function, does there exist another subset of measurements which will allow us to zoom in on specific areas of it and gain more information? A logical first choice would be to extend the set of measurements such that they include all operators that correspond to slopes in the subfield. For example, in the dimension 16 case, we would measure all operators for the rays $\alpha = 0$ and $\beta = \lambda\alpha, \lambda \in \{0, \sigma^5, \sigma^{10}, \sigma^{15}\}$, rather than just three from each. This strategy would allow us to optimize measurements in a very subtle way. Work along these lines is in progress.

5.7 Acknowledgments

O.D.M. is funded by Canada's NSERC. She is also grateful for hospitality at the MPL. IQC is supported in part by the Government of Canada and the Province of Ontario. L.L.S.S. acknowledges financial support from the Spanish MINECO (Grant No. FIS2015-67963-P).

Chapter 6

Conclusions and future directions

We have seen that large-scale parallelization can help us compile larger quantum operations; fault-tolerantly implementing a quantum RAM may be detrimental for many quantum algorithms; we can reconstruct 'blurry' representations of quantum states by choosing only subsets of tomographic measurements. However, none of these techniques alone tells the full story, and it is essential that we continue research along all these fronts.

We need to make better compilers, and we need to make them user-friendly so that anyone wishing to execute an algorithm on a quantum computer can easily transform their desired operations into the native set of gates for that machine.

One particular area of interest is implementation-specific compilers. Quantum devices today have very different connectivity graphs, and not every qubit is coupled to every other. We must take this into consideration in the placement of gates when we do circuit synthesis. The study of this *qubit allocation* problem is only in its very early stages [129–133], and will become very interesting as more complex hardware emerges.

Quantum RAM will soon become very important, as unless we make it efficient *and* fault-tolerant, many promising algorithms will suffer from an input bottleneck. We must explore different qRAM models, as well as different methods for resource estimation. In particular for the surface code, a promising technique called lattice surgery has recently become the gold standard due to it requiring fewer resources than the defect-based techniques used within [91–93].

As for quantum tomography, there have been numerous new developments since the work of Chapter 5 was carried out in mid-2015. Bayesian tomography [55], gateset tomography [134–136] and other machine learning techniques [137–139] are becoming more accessible [140], and are steadily transforming tomography from the mathematical ideas presented here to a more practical setting.

In summary, the speed at which the hardware is being developed is increasing. It is important that we work now to develop components, tools, and techniques for quantum computers so that in the future when we have a full-scale machine we will be ready to make good use of the available resources.

Letters of copyright permission

Chapter 3

The work of Section 3.1-Section 3.6 and Appendix A was published in [1], in the open-access journal Quantum Science and Technology, under the Creative Commons Attribution 3.0 licence. A copy of the licence may be found at https://creativecommons.org/licenses/by/3.0/legalcode. The work has been used in this thesis mostly 'as-is', save the addition of a paragraph at the end of Section 3.2, and the removal of the abstract and acknowledgments.

The work of Section 3.7 was published in [3] in the open-access journal Special Matrices, under the Creative Commons Attribution-NonCommercial-NoDerivs 3.0 License. A copy of the license may be found at https://creativecommons.org/licenses/by-nc-nd/3.0/legalcode. The work has been used in this thesis 'as-is'.

Chapter 5

The work of Chapter 5, Appendix B, and Appendix C was published in [5], https://doi.org/10.1103/PhysRevA.95.022340, by the American Physical Society. A copyright letter granting permission of use and license terms follows on the next two pages.

American Physical Society
Reuse and Permissions License

15-Oct-2018

This license agreement between the American Physical Society ("APS") and Olivia Di Matteo ("You") consists of your license details and the terms and conditions provided by the American Physical Society and SciPris.

Licensed Content Information

License Number:	**RNP/18/OCT/008527**
License date:	15-Oct-2018
DOI:	10.1103/PhysRevA.95.022340
Title:	Coarse graining the phase space of N qubits
Author:	Olivia Di Matteo et al.
Publication:	Physical Review A
Publisher:	American Physical Society
Cost:	USD $ 0.00

Request Details

Does your reuse require significant modifications: No

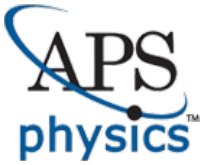
Specify intended distribution locations:	Worldwide
Reuse Category:	Reuse in a thesis/dissertation
Requestor Type:	Author of requested content
Items for Reuse:	Whole Article
Format for Reuse:	Electronic and Print
Total number of print copies:	Up to 1000

Information about New Publication:

University/Publisher:	University of Waterloo
Title of dissertation/thesis:	Methods for parallel quantum circuit synthesis, fault-tolerant quantum RAM, and quantum state tomography
Author(s):	Olivia Di Matteo
Expected completion date:	Dec. 2018

License Requestor Information

Name:	Olivia Di Matteo
Affiliation:	Individual
Email Id:	odimatte@uwaterloo.ca
Country:	Canada



American Physical Society
Reuse and Permissions License

TERMS AND CONDITIONS

The American Physical Society (APS) is pleased to grant the Requestor of this license a non-exclusive, non-transferable permission, limited to Electronic and Print format, provided all criteria outlined below are followed.

1. You must also obtain permission from at least one of the lead authors for each separate work, if you haven't done so already. The author's name and affiliation can be found on the first page of the published Article.

2. For electronic format permissions, Requestor agrees to provide a hyperlink from the reprinted APS material using the source material's DOI on the web page where the work appears. The hyperlink should use the standard DOI resolution URL, http://dx.doi.org/{DOI}. The hyperlink may be embedded in the copyright credit line.

3. For print format permissions, Requestor agrees to print the required copyright credit line on the first page where the material appears: "Reprinted (abstract/excerpt/figure) with permission from [(FULL REFERENCE CITATION) as follows: Author's Names, APS Journal Title, Volume Number, Page Number and Year of Publication.] Copyright (YEAR) by the American Physical Society."

4. Permission granted in this license is for a one-time use and does not include permission for any future editions, updates, databases, formats or other matters. Permission must be sought for any additional use.

5. Use of the material does not and must not imply any endorsement by APS.

6. APS does not imply, purport or intend to grant permission to reuse materials to which it does not hold copyright. It is the requestor's sole responsibility to ensure the licensed material is original to APS and does not contain the copyright of another entity, and that the copyright notice of the figure, photograph, cover or table does not indicate it was reprinted by APS with permission from another source.

7. The permission granted herein is personal to the Requestor for the use specified and is not transferable or assignable without express written permission of APS. This license may not be amended except in writing by APS.

8. You may not alter, edit or modify the material in any manner.

9. You may translate the materials only when translation rights have been granted.

10. APS is not responsible for any errors or omissions due to translation.

11. You may not use the material for promotional, sales, advertising or marketing purposes.

12. The foregoing license shall not take effect unless and until APS or its agent, Aptara, receives payment in full in accordance with Aptara Billing and Payment Terms and Conditions, which are incorporated herein by reference.

13. Should the terms of this license be violated at any time, APS or Aptara may revoke the license with no refund to you and seek relief to the fullest extent of the laws of the USA. Official written notice will be made using the contact information provided with the permission request. Failure to receive such notice will not nullify revocation of the permission.

14. APS reserves all rights not specifically granted herein.

15. This document, including the Aptara Billing and Payment Terms and Conditions, shall be the entire agreement between the parties relating to the subject matter hereof.

Bibliography

[1] Olivia Di Matteo and Michele Mosca. Parallelizing quantum circuit synthesis. *Quantum Science and Technology*, 1(1):015003, 2016.

[2] Olivia Di Matteo. Parallelizing quantum circuit synthesis. Master's thesis, University of Waterloo, Waterloo ON, April 2015.

[3] Olivia Di Matteo, Dragomir Ž. Đoković, and Ilias S. Kotsireas. Symmetric Hadamard matrices of order 116 and 172 exist. *Special Matrices*, 3(1), 2015.

[4] Srinivasan Arunachalam, Vlad Gheorghiu, Tomas Jochym-O'Connor, Michele Mosca, and Priyaa Varshinee Srinivasan. On the robustness of bucket brigade quantum ram. *New Journal of Physics*, 17(12):123010, 2015.

[5] Olivia Di Matteo, Luis L. Sánchez-Soto, Gerd Leuchs, and Markus Grassl. Coarse graining the phase space of n qubits. *Phys. Rev. A*, 95:022340, Feb 2017.

[6] IBM Quantum Experience Device Status page. https://quantumexperience.ng.bluemix.net/qx/devices. Accessed: 2018-10-04.

[7] J. Preskill. Quantum Computing in the NISQ era and beyond. *ArXiv e-prints*, January 2018.

[8] H. De Raedt, F. Jin, D. Willsch, M. Nocon, N. Yoshioka, N. Ito, S. Yuan, and K. Michielsen. Massively parallel quantum computer simulator, eleven years later. *ArXiv e-prints*, May 2018.

[9] R. Li, B. Wu, M. Ying, X. Sun, and G. Yang. Quantum Supremacy Circuit Simulation on Sunway TaihuLight. *ArXiv e-prints*, April 2018.

[10] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff. Breaking the 49-Qubit Barrier in the Simulation of Quantum Circuits. *ArXiv e-prints*, October 2017.

[11] I. L. Markov, A. Fatima, S. V. Isakov, and S. Boixo. Quantum Supremacy Is Both Closer and Farther than It Appears. *ArXiv e-prints*, July 2018.

[12] Matthew Amy, Olivia Di Matteo, Vlad Gheorghiu, Michele Mosca, Alex Parent, and John Schanck. Estimating the cost of generic quantum pre-image attacks on sha-2 and

sha-3. In Roberto Avanzi and Howard Heys, editors, *Selected Areas in Cryptography – SAC 2016*, pages 317–337, Cham, 2017. Springer International Publishing.

[13] Hubert de Guise, Olivia Di Matteo, and Luis L. Sánchez-Soto. Simple factorization of unitary transformations. *Phys. Rev. A*, 97:022328, Feb 2018.

[14] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Asymptotically optimal approximation of single qubit unitaries by Clifford and T circuits using a constant number of ancillary qubits. *Phys. Rev. Lett.*, 110:190502, May 2013.

[15] Vadym Kliuchnikov, Dmitri Maslov, and Michele Mosca. Fast and efficient exact synthesis of single-qubit unitaries generated by Clifford and T gates. *Quantum Info. Comput.*, 13(7-8):607–630, July 2013.

[16] Christopher M. Dawson and Michael A. Nielsen. The Solovay-Kitaev algorithm. *Quantum Info. Comput.*, 6(1):81–95, January 2006.

[17] Neil J. Ross. Optimal ancilla-free Clifford+V approximation of Z-rotations. *Quantum Information and Computation*, 15:932–950, 2015/03/06 2015.

[18] Alex Bocharov and Krysta M. Svore. A Depth-Optimal Canonical Form for Single-qubit Quantum Circuits. *Physical Review Letters*, 109(19), November 2012. arXiv: 1206.3223.

[19] Alex Bocharov, Yuri Gurevich, and Krysta M. Svore. Efficient decomposition of single-qubit gates into V basis circuits. *Phys. Rev. A*, 88:012313, Jul 2013.

[20] Simon Forest, David Gosset, Vadym Kliuchnikov, and David McKinnon. Exact synthesis of single-qubit unitaries over Clifford-cyclotomic gate sets. *Journal of Mathematical Physics*, 56(8), 2015.

[21] 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, January 2016.

[22] Peter Selinger. Efficient Clifford+T approximation of single-qubit operators. *Quantum Info. Comput.*, 15(1-2):159–180, January 2015.

[23] Mikko Möttönen, Juha J. Vartiainen, Ville Bergholm, and Martti M. Salomaa. Quantum Circuits for General Multiqubit Gates. *Physical Review Letters*, 93(13), September 2004.

[24] Ville Bergholm, Juha J. Vartiainen, Mikko Möttönen, and Martti M. Salomaa. Quantum circuits with uniformly controlled one-qubit gates. *Physical Review A*, 71(5), May 2005.

[25] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. Synthesis of Quantum Logic Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6):1000–1010, June 2006. arXiv: quant-ph/0406176.

[26] Jonathan Welch, Alex Bocharov, and Krysta M. Svore. Efficient approximation of diagonal unitaries over the Clifford+T basis. *QIC*, 16(1 & 2):87–104, January 2016.

[27] Brett Giles and Peter Selinger. Exact synthesis of multiqubit Clifford+T circuits. *Phys. Rev. A*, 87:032332, Mar 2013.

[28] Vadym Kliuchnikov and Jon Yard. A framework for exact synthesis. *http://arxiv.org/abs/1504.04350*, April 2015.

[29] 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, june 2013.

[30] David Gosset, Vadym Kliuchnikov, Michele Mosca, and Vincent Russo. An algorithm for the T-count. *Quantum Info. Comput.*, 14(15-16):1261–1276, November 2014.

[31] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Quantum random access memory. *Phys. Rev. Lett.*, 100:160501, Apr 2008.

[32] Vittorio Giovannetti, Seth Lloyd, and Lorenzo Maccone. Architectures for a quantum random access memory. *Phys. Rev. A*, 78:052310, Nov 2008.

[33] Lov K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, 79:325–328, Jul 1997.

[34] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. *SIGACT News*, 28(2):14–19, June 1997.

[35] Andris Ambainis. Quantum walk algorithm for element distinctness. *SIAM J. Comput.*, 37(1):210–239, April 2007.

[36] Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. *SIAM J. Comput.*, 35(1):170–188, July 2005.

[37] I. Kerenidis and A. Prakash. Quantum Recommendation Systems. *ArXiv e-prints*, March 2016.

[38] C. Wang and L. Wossnig. A quantum algorithm for simulating non-sparse Hamiltonians. *ArXiv e-prints*, March 2018.

[39] Z. Zhao, J. K. Fitzsimons, M. A. Osborne, S. J. Roberts, and J. F. Fitzsimons. Quantum algorithms for training Gaussian Processes. *ArXiv e-prints*, March 2018.

[40] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd. Quantum algorithm for linear systems of equations. *Phys. Rev. Lett.*, 103:150502, Oct 2009.

[41] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A*, 86:032324, Sep 2012.

[42] Stephen Brierley and Stefan Weigert. Maximal sets of mutually unbiased quantum states in dimension 6. *Physical Review A*, 78(4), October 2008.

[43] Ingemar Bengtsson, Wojciech Bruzda, Asa Ericsson, Jan-Ake Larsson, Wojciech Tadej, and Karol Zyczkowski. Mubs and Hadamards of Order Six. *arXiv:quant-ph/0610161*, October 2006. arXiv: quant-ph/0610161.

[44] Philippe Raynal, Xin Lü, and Berthold-Georg Englert. Mutually unbiased bases in six dimensions: The four most distant bases. *Physical Review A*, 83(6), June 2011.

[45] Paul Butterley and William Hall. Numerical evidence for the maximum number of mutually unbiased bases in dimension six. *Physics Letters A*, 369(1-2):5–8, September 2007.

[46] Markus Grassl. On SIC-POVMs and MUBs in Dimension 6. *arXiv:quant-ph/0406175*, June 2004. arXiv: quant-ph/0406175.

[47] Daniel McNulty and Stefan Weigert. On the Impossibility to Extend Triples of Mutually Unbiased Product Bases in Dimension Six. *International Journal of Quantum Information*, 10(05):1250056, August 2012. arXiv: 1203.6887.

[48] Daniel McNulty and Stefan Weigert. All mutually unbiased product bases in dimension 6. *Journal of Physics A: Mathematical and Theoretical*, 45(13):135307, April 2012.

[49] Daniel McNulty and Stefan Weigert. The limited role of mutually unbiased product bases in dimension 6. *Journal of Physics A: Mathematical and Theoretical*, 45(10):102001, March 2012.

[50] Stephen Brierley and Stefan Weigert. Constructing mutually unbiased bases in dimension six. *Physical Review A*, 79(5), May 2009.

[51] I. D. Ivanović. Geometrical description of quantal state determination. *Journal of Physics A: Mathematical and General*, 14(12):3241, 1981.

[52] W. K. Wootters and B. D. Fields. Optimal state-determination by mutually unbiased measurements. *Ann. Phys.*, 191(2):363–381, 1989.

[53] A. B. Klimov, L. L. Sánchez-Soto, and H. de Guise. Multicomplementary operators via finite Fourier transform. *J. Phys. A*, 38(12):2747–2760, 2005.

[54] Yong Siah Teo, Berthold-Georg Englert, Jaroslav Řeháček, and Zden ěk Hradil. Adaptive schemes for incomplete quantum process tomography. *Phys. Rev. A*, 84:062125, Dec 2011.

[55] Christopher Granade, Joshua Combes, and D G Cory. Practical Bayesian tomography. *New J. Phys.*, 18(3):033024, 2016.

[56] F. Huszár and N. M. T. Houlsby. Adaptive Bayesian quantum tomography. *Phys. Rev. A*, 85(5):052120–, 05 2012.

[57] Jiangwei Shang, Zhengyun Zhang, and Hui Khoon Ng. Superfast maximum-likelihood reconstruction for quantum tomography. *Phys. Rev. A*, 95:062336, Jun 2017.

[58] J. Rehacek, Z. Hradil, Y. S. Teo, L. L. Sanchez-Soto, H. K. Ng, J. H. Chai, and B.-G. Englert. Least-bias state estimation with incomplete unbiased measurements. *Physical Review A*, 92(5), November 2015. arXiv: 1509.07614.

[59] 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, October 2014.

[60] Nabila Abdessaied, Mathias Soeken, and Rolf Drechsler. Quantum Circuit Optimization by Hadamard Gate Reduction. In Shigeru Yamashita and Shin-ichi Minato, editors, *Reversible Computation*, pages 149–162, Cham, 2014. Springer International Publishing.

[61] Matthew Amy, Parsiad Azimzadeh, and Michele Mosca. On the controlled-not complexity of controlled-not–phase circuits. *Quantum Science and Technology*, 4(1):015002, 2018.

[62] Alex Bocharov, Martin Roetteler, and Krysta M. Svore. Efficient synthesis of universal repeat-until-success circuits. *Physical Review Letters*, 114(080502), February 2015.

[63] Nathan Wiebe and Martin Roetteler. Quantum arithmetic and numerical analysis using repeat-until-success circuits. *Quantum Information and Computation*, 16:134–178, 2016.

[64] Philippe Oechslin. Making a faster cryptanalytic time-memory trade-off. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, volume 2729 of *Lecture Notes in Computer Science*, pages 617–630. Springer, 2003.

[65] Paul C. van Oorschot and Michael J. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[66] Paul C. van Oorschot and Michael J. Wiener. *Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings*, chapter Improving Implementable Meet-in-the-Middle Attacks by Orders of Magnitude, pages 229–236. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[67] Boost C++ Libraries. `http://www.boost.org/`.

[68] Matthew Amy. Personal communication, 2016. Established using techniques in http://arxiv.org/abs/1601.07363.

[69] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Phys. Rev. A*, 70:052328, Nov 2004.

[70] Thomas G. Draper. Personal communication, 2016.

[71] R. Craigen and H. Kharaghani. *Hadamard Matrices and Hadamard Designs. In Handbook of Combinatorial Designs.* Discrete Mathematics and Its Applications. CRC Press, 2010.

[72] R. Mathon. Symmetric conference matrices of order $pq^2 + 1$. *Canad. J. Math.*, 30:321–331, 1978.

[73] Jennifer Seberry N. A. Balonin. A review and new symmetric conference matrices. *Informatsionno-upravliaiushchie sistemy*, 71(4):2–7, 2014.

[74] Jennifer Seberry N. A. Balonin. Visualizing hadamard matrices: the propus construction. *Preprint, mathscinet.ru*, page 15pp, 2014.

[75] Dragomir Ž Đoković and Ilias S. Kotsireas. New results on d-optimal matrices. *Journal of Combinatorial Designs*, 20(6):278–289.

[76] Mingyuan Xia, Tianbing Xia, Jennifer Seberry, and Jing Wu. An infinite family of goethals–seidel arrays. *Discrete Applied Mathematics*, 145(3):498 – 504, 2005.

[77] Yury J. Ionin and Mohan S. Shrikhande. *Combinatorics of Symmetric Designs*. New Mathematical Monographs. Cambridge University Press, 2006.

[78] J. Seberry Wallis. *Hadamard Matrices*, volume 292 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin-New York, 1972.

[79] Richard J Turyn. An infinite class of williamson matrices. *Journal of Combinatorial Theory, Series A*, 12(3):319 – 321, 1972.

[80] Dragomir Ž. Đoković and Ilias S. Kotsireas. Compression of periodic complementary sequences and applications. *Designs, Codes and Cryptography*, 74(2):365–377, jul 2013.

[81] Srinivasan Arunachalam. Quantum speed-ups for boolean satisfiability and derivative-free optimization. Master's Thesis, University of Waterloo, 2014.

[82] Thijs Laarhoven, Michele Mosca, and Joop van de Pol. Solving the shortest vector problem in lattices faster using quantum search. In Philippe Gaborit, editor, *Post-Quantum Cryptography*, pages 83–101, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[83] Brian W. Kernighan and Dennis Ritchie. *The C programming language (2nd edition)*. Prentice Hall Software Series, 1988.

[84] Peter Selinger. Quantum circuits of t-depth one. *Phys. Rev. A*, 87:042302, Apr 2013.

[85] M. Amy, D. Maslov, and M. 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, Oct 2014.

[86] Alan Mishchenko and Marek Perkowski. Fast heuristic minimization of exclusive-sums-of-products. 09 2001.

[87] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Reversible logic synthesis of k-input, m-output lookup tables. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '13, pages 1235–1240, San Jose, CA, USA, 2013. EDA Consortium.

[88] Nabila Abdessaied, Matthew Amy, Rolf Drechsler, and Mathias Soeken. Complexity of reversible circuits and their quantum implementations. *Theoretical Computer Science*, 618:85 – 106, 2016.

[89] N. Abdessaied, M. Amy, M. Soeken, and R. Drechsler. Technology mapping of reversible circuits to clifford+t quantum circuits. In *2016 IEEE 46th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 150–155, May 2016.

[90] 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. *Phys. Rev. A*, 52:3457–3467, Nov 1995.

[91] 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.

[92] Daniel Litinski. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *arXiv:1808.02892 [cond-mat, physics:quant-ph]*, August 2018. arXiv: 1808.02892.

[93] Austin G. Fowler and Craig Gidney. Low overhead quantum computation using lattice surgery. *arXiv:1808.06709 [quant-ph]*, August 2018. arXiv: 1808.06709.

[94] Leo P. Kadanoff. Innovations in statistical physics. *Annual Review of Condensed Matter Physics*, 6(1):1–14, 2015.

[95] Oleg Kabernik. Quantum coarse graining, symmetries, and reducibility of dynamics. *Phys. Rev. A*, 97:052130, May 2018.

[96] J. L. E. Silva, S. Glancy, and H. M. Vasconcelos. Quadrature histograms in maximum-likelihood quantum state tomography. *Phys. Rev. A*, 98:022325, Aug 2018.

[97] Dale Scerri, Erik M. Gauger, and George C. Knee. Coarse-graining in retrodictive quantum state tomography. *arXiv e-prints*, page arXiv:1809.07970, September 2018.

[98] Yong Siah Teo, Jaroslav Řeháček, and Zden ěk Hradil. Coarse-grained quantum state estimation for noisy measurements. *Phys. Rev. A*, 88:022111, Aug 2013.

[99] I. Bloch, J. Dalibard, and W. Zwerger. Many-body physics with ultracold gases. *Rev. Mod. Phys.*, 80(3):885–964, 07 2008.

[100] R. Blatt and C. F. Roos. Quantum simulations with trapped ions. *Nat. Phys.*, 8(4):277–284, 04 2012.

[101] M. G. A. Paris and J. Řeháček, editors. *Quantum State Estimation*, volume 649 of *Lect. Not. Phys.* Springer, Berlin, 2004.

[102] V. Bužek, R. Derka, G. Adam, and P. L. Knight. Reconstruction of quantum states of spin systems: From quantum Bayesian inference to quantum tomography. *Ann. Phys.*, 266(2):454–496, 1998.

[103] R. Schack, T. A. Brun, and C. M. Caves. Quantum Bayes rule. *Phys. Rev. A*, 64(1):014305, 06 2001.

[104] D. Gross, Y.-K. Liu, S. T. Flammia, S. Becker, and J. Eisert. Quantum state tomography via compressed sensing. *Phys. Rev. Lett.*, 105(15):150401, 10 2010.

[105] M. Cramer, M. B. Plenio, S. T. Flammia, R. Somma, D. Gross, S. D. Bartlett, O. Landon-Cardinal, D. Poulin, and Y. K. Liu. Efficient quantum state tomography. *Nat. Commun.*, 1:149 EP, 12 2010.

[106] S. T. Flammia, D. Gross, Y.-K. Liu, and J. Eisert. Quantum tomography via compressed sensing: error bounds, sample complexity and efficient estimators. *New J. Phys.*, 14(9):095022, 2012.

[107] O. Landon-Cardinal and D. Poulin. Practical learning method for multi-scale entangled states. *New J. Phys.*, 14(9):085004, 2012.

[108] T. Baumgratz, D. Gross, M. Cramer, and M. B. Plenio. Scalable reconstruction of density matrices. *Phys. Rev. Lett.*, 111(2):020401, 07 2013.

[109] G. Tóth, W. Wieczorek, D. Gross, R. Krischek, C. Schwemmer, and H. Weinfurter. Permutationally invariant quantum tomography. *Phys. Rev. Lett.*, 105(25):250403, 12 2010.

[110] T. Moroder, P. Hyllus, G. Tóth, C. Schwemmer, A. Niggebaum, S. Gaile, O. Gühne, and H. Weinfurter. Permutationally invariant state reconstruction. *New J. Phys.*, 14:105001, 2012.

[111] A. B. Klimov, G. Björk, and L. L. Sánchez-Soto. Optimal quantum tomography of permutationally invariant qubits. *Phys. Rev. A*, 87(1):012109, 01 2013.

[112] J. Řeháček, S. Olivares, D. Mogilevtsev, Z. Hradil, M. G. A. Paris, S. Fornaro, V. D'Auria, A. Porzio, and S. Solimeno. Effective method to estimate multidimensional Gaussian states. *Phys. Rev. A*, 79(3):032111, 03 2009.

[113] P. Castiglione, M. Falcioni, A. Lesne, and A. Vulpiani. *Chaos and Coarse Graining in Statistical Mechanics*. Cambridge University Press, Cambridge, 2008.

[114] S. R. White. Density matrix formulation for quantum renormalization groups. *Phys. Rev. Lett.*, 69(19):2863–2866, 11 1992.

[115] I. Chuang and M. Nielsen. *Quantum Computation and Quantum Information*. Cambridge University Press, Cambridge, 2000.

[116] M. Grassl, M. Rötteler, and T. Beth. Efficient quantum circuits for non-qubit quantum error-correction codes. *Int. J. Found. Comput. Sci.*, 14(05):757–775, 2016/11/19 2003.

[117] A. Vourdas. Quantum systems with finite Hilbert space. *Rep. Prog. Phys.*, 67(3):267–320, march 2004.

[118] A. Vourdas. Quantum systems with finite Hilbert space: Galois fields in quantum mechanics. *J. Phys. A*, 40(33):R285–R331, 2007.

[119] E. Binz and S. Pods. *The Geometry of Heisenberg Groups*. American Mathematical Society, Providence, 2008.

[120] W. K. Wootters. Picturing qubits in phase space. *IBM J. Res. Dev.*, 48(1):99–110, Jan 2004.

[121] K. S. Gibbons, M. J. Hoffman, and W. K. Wootters. Discrete phase space based on finite fields. *Phys. Rev. A*, 70(6):062101, 12 2004.

[122] A. B. Klimov, J. L. Romero, G. Björk, and L. L. Sánchez-Soto. Geometrical approach to mutually unbiased bases. *J. Phys. A*, 40(14):3987–3998, 2007.

[123] A. B. Klimov, J. L. Romero, G. Björk, and L. L. Sánchez-Soto. Discrete phase-space structure of n-qubit mutually unbiased bases. *Ann. Phys.*, 324(1):53–72, 1 2009.

[124] C. Muñoz, A. B. Klimov, and L L Sánchez-Soto. Symmetric discrete coherent states for n-qubits. *J. Phys. A*, 45(24):244014, 2012.

[125] S. Bandyopadhyay, P. O. Boykin, V. Roychowdhury, and F. Vatan. A new proof for the existence of mutually unbiased bases. *Algorithmica*, 34:512–528, November 2002.

[126] G. Björk, A. B. Klimov, and L. L. Sánchez-Soto. The discrete Wigner function. *Prog. Opt.*, 51:469–516, 2008.

[127] R. L. Stratonovich. On distributions in representation space. *Sov. Phys. JETP*, 31:1012–1020, 1956.

[128] https://github.com/glassnotes/balthasar.

[129] M. Pedram and A. Shafaei. Layout Optimization for Quantum Circuits with Linear Nearest Neighbor Architectures. *IEEE Circuits and Systems Magazine*, 16(2):62–74, 2016.

[130] Marcos Yukio Siraichi, Vinícius Fernandes dos Santos, Sylvain Collange, and Fernando Magno Quintao Pereira. Qubit allocation. In *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, CGO 2018, pages 113–125, New York, NY, USA, 2018. ACM.

[131] A. Zulehner, A. Paler, and R. Wille. An Efficient Methodology for Mapping Quantum Circuits to the IBM QX Architectures. *ArXiv e-prints*, December 2017.

[132] S. S. Tannu and M. K. Qureshi. A Case for Variability-Aware Policies for NISQ-Era Quantum Computers. *ArXiv e-prints*, May 2018.

[133] G. Li, Y. Ding, and Y. Xie. Tackling the Qubit Mapping Problem for NISQ-Era Quantum Devices. *ArXiv e-prints*, September 2018.

[134] Seth T. Merkel, Jay M. Gambetta, John A. Smolin, Stefano Poletto, Antonio D. Córcoles, Blake R. Johnson, Colm A. Ryan, and Matthias Steffen. Self-consistent quantum process tomography. *Phys. Rev. A*, 87:062119, Jun 2013.

[135] R. Blume-Kohout, J. King Gamble, E. Nielsen, J. Mizrahi, J. D. Sterk, and P. Maunz. Robust, self-consistent, closed-form tomography of quantum logic gates on a trapped ion qubit. *ArXiv e-prints*, October 2013.

[136] R. Blume-Kohout, J. K. Gamble, E. Nielsen, K. Rudinger, J. Mizrahi, K. Fortier, and P. Maunz. Demonstration of qubit operations below a rigorous fault tolerance threshold with gate set tomography. *Nature Communications*, 8, February 2017.

[137] T. Xin, S. Lu, N. Cao, G. Anikeeva, D. Lu, J. Li, G. Long, and B. Zeng. Local-measurement-based quantum state tomography via neural networks. *ArXiv e-prints*, July 2018.

[138] Giacomo Torlai and Roger G. Melko. Latent Space Purification via Neural Density Operators. *arXiv:1801.09684 [quant-ph]*, January 2018. arXiv: 1801.09684.

[139] Giacomo Torlai, Guglielmo Mazzola, Juan Carrasquilla, Matthias Troyer, Roger Melko, and Giuseppe Carleo. Many-body quantum state tomography with neural networks. *Nature Physics*, 14(5):447–450, May 2018. arXiv: 1703.05334.

[140] C. Granade, C. Ferrie, I. Hincks, S. Casagrande, T. Alexander, J. Gross, M. Kononenko, and Y. Sanders. QInfer: Statistical inference software for quantum applications. *ArXiv e-prints*, October 2016.

[141] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and their Applications*. Cambridge University Press, Cambridge, 1986.

Appendix A

pQCS runtime analysis

The runtimes presented in Equation (3.8) and Equation (3.9) stem from a so-called 'flawed' runtime analysis originally presented in [65]. Suppose we are searching for a collision in a space of size N, and that the available memory is full with w distinguished points. The number of steps required to find a single collision in this case is

$$\frac{N\theta}{w} + \frac{2}{\theta}, \tag{A.1}$$

where θ is the fraction of points which are distinguished. The first term comes from the fact that to fill the memory with w distinguished points, w/θ elements in the space will be traversed on average, and any given point in a new trail has a $1/N$ probability of landing on a previously seen point; the second term comes from the need to trace back through both trails to locate the collision, and each trail has length $1/\theta$ on average.

The assumption is made that there is a single 'golden' collision. In this case $N/2$ 'bad' collisions will be found on average before the golden one is found. If we parallelize using m processors and assume each step in a trail takes time τ, then we obtain a runtime

$$T \propto \frac{1}{m}\left(\frac{N^2\theta}{2w} + \frac{N}{\theta}\right)\tau \tag{A.2}$$

The next step taken in [65] is to differentiate and find θ such that Equation (A.1) is optimized, which is what results in the inverse-square-root dependence on w. They then performed computational experiments for a range of w and N in order to find optimal prefactors.

However, the optimal θ is expressed in terms of w/N, which when $w >> N$ (as is the case when we synthesize the Toffoli on the BG/Q) would not result in a fractional θ. So let us continue a hypothetical analysis of this form without finding the optimal θ. Consider the case where we are optimizing for T-count. In the most general case, the two halves of the MITM equation will be different sizes N_1 and N_2 where $N_1 = 4^{n\lceil\frac{t}{2}\rceil}$ and $N_2 = 4^{n\lfloor\frac{t}{2}\rfloor}$ (t being the T-count). Since when we have an odd depth we partition the larger space and search sequentially (in theory this could also be done in parallel), we must add a prefactor

of N_1/N_2 in front of the runtime, and the N becomes $2N_2$, because the full space we're searching is that of $N_2 \times \{1, 2\}$. As for τ, let's assume $\tau = \lceil \frac{t}{2} \rceil 4^{\alpha n}$ where α is a constant which reflects the complexity of the matrix multiplication algorithm. Then we have that

$$T \propto \frac{1}{m} 4^{n\left(\lceil \frac{t}{2} \rceil - \lfloor \frac{t}{2} \rfloor\right)} \left(\frac{4^{2n\lfloor \frac{t}{2} \rfloor + 1}\theta}{2w} + \frac{2 \cdot 4^{n\lfloor \frac{t}{2} \rfloor}}{\theta} \right) \left\lceil \frac{t}{2} \right\rceil 4^{\alpha n} \tag{A.3}$$

$$= \left\lceil \frac{t}{2} \right\rceil \frac{1}{2m} 4^{n\left(\alpha + 1 + \lceil \frac{t}{2} \rceil\right)} \left(\frac{4^{n\lfloor \frac{t}{2} \rfloor}\theta}{w} + \frac{1}{\theta} \right) \tag{A.4}$$

When $w >> N_2$, the first term disappears and the expression reduces to

$$T \propto \left\lceil \frac{t}{2} \right\rceil \frac{4^{n\left(\alpha + 1 + \lceil \frac{t}{2} \rceil\right)}}{m\theta}, \tag{A.5}$$

which is exponential in both n and t, and inversely proportional to both m and θ, precisely what we have observed in practice.

Appendix B

Finite fields

In this appendix we briefly recall some background needed for this paper. The reader interested in more mathematical details is referred, e.g., to the excellent monograph by Lidl and Niederreiter [141].

A commutative ring is a nonempty set R with two binary operations, called addition and multiplication, such that it is an Abelian group with respect to addition, and the multiplication is associative. The most typical example is the ring of integers $\mathbb{Z}$, with the standard sum and multiplication. On the other hand, the simplest example of a finite ring is the set $\mathbb{Z}_n$ of integers modulo n, which has exactly n elements.

A field F is a commutative ring with division, i.e., such that 0 does not equal 1 and all elements of F except 0 have a multiplicative inverse (note that 0 and 1 here stand for the identity elements for the addition and multiplication, respectively, which may differ from the familiar real numbers 0 and 1). Elements of a field form Abelian groups with respect to addition and multiplication (in this latter case, the zero element is excluded). Note that the finite ring $\mathbb{Z}_d$ is a field if and only if d is a prime number.

The characteristic of a finite field is the smallest positive integer d such that

$$\underbrace{1+1+\ldots+1}_{d \text{ times}} = 0 \tag{B.1}$$

and it is always a prime number. Any finite field contains a prime subfield $\mathbb{Z}_d$ and has d^n elements, where n is a natural number. Moreover, the finite field containing d^n elements is unique up to isomorphism and is called the Galois field $\mathbb{F}_{d^n}$.

We denote as $\mathbb{Z}_d[x]$ the ring of polynomials with coefficients in $\mathbb{Z}_d$. If $P(x)$ is an irreducible polynomial of degree n (that is, one that cannot be factorized over $\mathbb{Z}_d$), the quotient space $\mathbb{Z}_d[X]/P(x)$ provides an adequate representation of $\mathbb{F}_{d^n}$. Its elements can be written as polynomials that are defined modulo the irreducible polynomial $P(x)$. The multiplicative group of $\mathbb{F}_{d^n}$ is cyclic and its generator is called a primitive element of the field.

As a trivial example of a nonprime field, we consider the polynomial $x^2 + x + 1 = 0$, which is irreducible over $\mathbb{Z}_2$. If σ is a root of this polynomial, the elements $\{0, 1, \sigma, \sigma^2 =$

$\sigma + 1 = \sigma^{-1}\}$ form the finite field $\mathbb{F}_{2^2}$ and σ is a primitive element.

A basic map is the trace

$$\mathrm{tr}(\alpha) = \alpha + \alpha^d + \ldots + \alpha^{d^{n-1}} \,. \tag{B.2}$$

The image of the trace is always in the prime field $\mathbb{Z}_d$ and satisfies

$$\mathrm{tr}(\alpha + \alpha') = \mathrm{tr}(\alpha) + \mathrm{tr}(\alpha') \,. \tag{B.3}$$

In terms of it we define an additive character as

$$\chi(\alpha) = \exp\left[\frac{2\pi i}{d}\,\mathrm{tr}(\alpha)\right] \,, \tag{B.4}$$

which possesses two important properties:

$$\chi(\alpha + \alpha') = \chi(\alpha)\chi(\alpha'), \qquad \sum_{\alpha' \in \mathbb{F}_{d^n}} \chi(\alpha\alpha') = d^n \delta_{0,\alpha} \,. \tag{B.5}$$

Any finite field $\mathbb{F}_{d^n}$ can be also considered as an n-dimensional linear vector space over its prime field $\mathbb{F}_d$. Given a basis $\{\theta_j\}$, $(j = 1, \ldots, n)$ in this vector space, any field element can be represented as

$$\alpha = \sum_{j=1}^{n} a_j\, \theta_j, \tag{B.6}$$

with $a_j \in \mathbb{Z}_d$. In this way, we map each element of $\mathbb{F}_{d^n}$ onto an ordered set of natural numbers $\alpha \Leftrightarrow (a_1, \ldots, a_n)$.

Two bases $\{\theta_1, \ldots, \theta_n\}$ and $\{\theta'_1, \ldots, \theta'_n\}$ are dual when

$$\mathrm{tr}(\theta_k \theta'_l) = \delta_{k,l}. \tag{B.7}$$

A basis that is dual to itself is called self-dual. A self-dual basis exists if and only if either d is even or both n and d are odd.

There are several natural bases in $\mathbb{F}_{d^n}$. One is the polynomial basis, defined as

$$\{1, \sigma, \sigma^2, \ldots, \sigma^{n-1}\}, \tag{B.8}$$

where σ is a primitive element. An alternative is a normal basis, constituted of

$$\{\sigma, \sigma^d, \ldots, \sigma^{d^{n-1}}\}. \tag{B.9}$$

The appropriate choice of basis depends on the specific problem at hand. For example, in $\mathbb{F}_{2^2}$ the elements $\{\sigma, \sigma^2\}$ are both roots of the irreducible polynomial. The polynomial basis is $\{1, \sigma\}$ and its dual is $\{\sigma^2, 1\}$, while the normal basis $\{\sigma, \sigma^2\}$ is self-dual.

Appendix C

Derivation of equation for line operators

Here we present the derivation of our equation for the surviving displacement operators. We begin by considering the projectors for the rays,

$$|\ell_0^{(\lambda)}\rangle\langle\ell_0^{(\lambda)}| = \frac{1}{2^{mn}}\sum_{\alpha} D(\alpha,\lambda\alpha) = \frac{1}{2^{mn}}\sum_{\alpha}\Phi(\alpha,\lambda\alpha)Z_\alpha X_{\lambda\alpha}\,. \tag{C.1}$$

As mentioned in Section 5.3, the projectors for the shifted lines can be obtained by applying an appropriate displacement operator to induce a transformation. Let us ignore for now the ray with infinite slope, $\alpha = 0$. Then for the rest of the rays, we can shift them vertically by applying the displacement operators of the form $D(0,\gamma)$:

$$\begin{aligned} |\ell_\gamma^{(\lambda)}\rangle\langle\ell_\gamma^{(\lambda)}| &= \frac{1}{2^{mn}}\sum_{\alpha} D(0,\gamma)D(\alpha,\lambda\alpha)D^\dagger(0,\gamma) \\ &= \frac{1}{2^{mn}}\sum_{\alpha}\Phi(\alpha,\lambda\alpha)X_\gamma Z_\alpha X_{\lambda\alpha}X_\gamma\,, \end{aligned} \tag{C.2}$$

where we recall the convention that all the phases $\Phi(0,\gamma) = 1$.

Here, we can make further use of the commutation relation in Equation (5.3). We obtain

$$\begin{aligned} |\ell_\gamma^{(\lambda)}\rangle\langle\ell_\gamma^{(\lambda)}| &= \frac{1}{2^{mn}}\sum_{\alpha}\Phi(\alpha,\lambda\alpha)\chi(\gamma\alpha)Z_\alpha X_{\lambda\alpha} \\ &= \frac{1}{2^{mn}}\sum_{\alpha}\chi(\gamma\alpha)D\left(\alpha,\lambda\alpha\right). \end{aligned} \tag{C.3}$$

It is then straightforward to see that the thick rays, which are obtained by summing over

all intercepts γ in coset $\mathfrak{C}_o$, can be written as

$$|L_{\mathfrak{C}_0}^{(\lambda)}\rangle\langle L_{\mathfrak{C}_0}^{(\lambda)}| = \frac{1}{2^{mn}} \sum_{\lambda} \left[\sum_{\gamma \in \mathfrak{C}_0} \chi(\gamma\alpha) \right] D(\alpha, \lambda\alpha). \tag{C.4}$$

Finally, we mention that for the infinite slope the analysis proceeds in exactly the same way, but that the lines are translated by displacement operators of the form $D(\gamma, 0)$ and Equation (5.3) gives us $\chi(\gamma\beta)$ instead.

Only those operators which have a non-zero term in the sum will contribute, thus we consider them as the effective displacement operators in the coarse phase space.