Towards Distributed Quantum Algorithms

Pablo Andres-Martinez

Master of Science by Research

Laboratory for Foundations of Computer Science
School of Informatics
University of Edinburgh

2018

Abstract

Acknowledgements

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Pablo Andres-Martinez*)

Table of Contents

Chapter 1

Introduction

Chapter 2

Quantum Computing: A brief overview

Quantum computing aims to take advantage of quantum mechanics to speed-up computations. There is strong theoretical evidence that quantum computers are capable of solving some problems substantially faster than standard (classical) computers. Well known examples are:

- Shor's algorithm for *polynomial-time* large number factorisation (Shor, 1999). There is no known algorithm on classical computers that can perform this task in polynomial time, and it is suspected not to be possible[1]. Quantum computers theoretically provide an exponential speed-up on this task.

- Grover's algorithm for efficient unstructured search (Grover, 1996). The algorithm performs a brute-force search (i.e. requiring no knowledge about the search space) over a N data-points in time proportional to $\sqrt{N}$. A brute-force search in a classical computer should always take time proportional to N.

Besides, in May of the present year, Raz and Tal (Ran Raz, 2018) gave formal proof of the existence of problems that a classical computer may never solve in polynomial time, but are solvable in polynomial time on a quantum one. Nevertheless, all of these results are theoretical in nature and there are some caveats on their practical implementation, discussed in § 2.2.1. Giving experimental evidence of this time-efficiency separation between quantum and classical computers is a highly active area of research, known as *quantum supremacy*.

Quantum computing would be very valuable in many areas of research where classical computers are unable to solve problems efficiently. Some of the main applications

[1] RSA, a widely used encryption system, relies its security on the assumption that factorisation of large numbers can not be computed efficiently.

that have been discussed in the literature are:

- *Chemistry, medicine and material sciences*: Calculating molecular properties on complex systems is an intractable problem for classical computers. However, polynomial algorithms for this problem are known for quantum computers (Lanyon et al., 2010). Hence, quantum computers are likely to trigger a revolution on areas of science that need to model molecules and their interactions.

- Machine learning: Finding patterns in a large pool of data is the essence of machine learning. Multiple quantum algorithms have been shown to be able to detect patterns that are believed not to be efficiently attainable classically (Biamonte et al., 2017).

- Engineering: Optimization and search problems are common in almost every area of engineering. Quantum computers are particularly well suited for these tasks, with Grover's algorithm being a clear example.

For any of these applications we will require large scale quantum computers. Due to the obstacles in the way of building a large mainframe quantum computer (see § 2.2.1), some authors have advocated the alternative of building a grid of smaller quantum computer units that cooperate in performing an overall computation (Van Meter et al., 2010). In this work, particularly in Chapter **??**, we contribute to this perspective, providing a method for efficiently distributing any quantum program originally designed for a monolithic quantum computer.

2.1 The principles of Quantum Computing

The advantage of using quantum mechanics to perform computations is usually traced down to the following three principles:

- *Superposition*: In classical computing, the unit of information is the *bit*, which may take one of two values: 0 or 1. In quantum computing, the *bit*'s counterpart is the *qubit*, whose value may be *any linear combination* of 0 and 1, known as a *superposition*, and usually written as:

$$|qubit\rangle = \alpha|0\rangle + \beta|1\rangle$$

where α and β are complex numbers that must satisfy: $\alpha^2 + \beta^2 = 1$.

A popular analogy of a qubit's superposition is a coin spinning[2]: the classical states (0 and 1) are *heads* and *tails*, but when the coin is spinning, its state is neither of them. If we knew exactly how the coin was set spinning, we would be able to describe the probability distribution of seeing heads or tails when it stops; these would be our α^2 and β^2 values. We may *measure* a qubit, and doing so corresponds in our analogy to stopping the coin: we will get either 0 or 1 as outcome.

The essential aspect of this analogy is that, before measurement, the *qubit*'s state is neither 0 nor 1. Through certain operations (that would correspond to altering the axis of spin of the coin), we may change the coefficients α and β of the superposition. Interestingly, in quantum computing, we encode input and read output (after measurement) as standard classical binary strings, and thus, *for input/output we use as many qubits as bits would be required*. What superposition provides is the ability to – during mid-computation – maintain a superposition of all potential solutions to the problem, and update all of them simultaneously with a single operation to the qubits. In some sense, superposition allows us to explore multiple choices/paths of the computation, using only the resources required to explore a single one of those paths.

- *Interference*: Superposition gives us the ability to simultaneously explore different paths to solve a problem. However, in the end we will need to measure the qubits – stop the coins, in order to read heads or tails – and the result will be intrinsically random. For quantum computing to be any better than a probabilistic classical computer, we require the ability to prune the paths that have led to a dead-end. This is precisely what *interference* provides: some operations on the qubits may make different classical states in the superposition cancel each other out. Interference is at the core of any speed-up achieved by a quantum algorithm, and taking advantage of it is the main challenge when designing quantum algorithms.

- *Entanglement*: Quantum mechanics allows us to have a pair of qubits *a* and *b* in

[2]Note this is just an analogy, and while a coin spinning can be perfectly modelled using classical physics, a qubit can not. In fact, superposition is key in the (even weirder) two other principles that differentiate quantum and classical computing.

a superposition such as:

$$|a,b\rangle = \frac{1}{\sqrt{2}}|0,0\rangle + \frac{1}{\sqrt{2}}|1,1\rangle$$

This implies that, when we measure the qubits, we may only read $a = 0, b = 0$ or $a = 1, b = 1$ as outcome (the coefficients for $|0,1\rangle$ and $|1,0\rangle$) are both 0). Then, what happens if we only measure a? In this case, we would also know b's outcome, without measuring it. More surprisingly, if we measured both a and b at the same instant, we would obtain $a = 0, b = 0$ half of the times and $a = 1, b = 1$ the other half. In short, it seems like acting on one qubit has an instantaneous effect on the other. Whenever a group of qubits exhibits this property, we say they are *entangled*. Entanglement holds regardless how far apart a is from b; for instance, they could be on two different quantum processing units of a distributed grid. Indeed, entanglement will be key in our discussion of distributed quantum algorithms, and we explain how to use it to perform non-local operations in § 4.1.

2.2 Building Quantum Computers

Physicists have come up with different ways of realising qubits in labs. The key idea is to find a physical system that displays non-classical behaviour, and put it under the appropriate circumstances so we can manage its quantum properties, but noise in the environment may not interfere with these. Examples of such successful technologies are:

- *Optics*: The state of a qubit is represented in the properties of photons, for instance, their polarization (Kok et al., 2007). A great advantage of this technology is that photons can be easily sent over long distances, while preserving the quantum state. Thus, protocols in quantum information that heavily rely on communication, such as Quantum Key Distribution (Shor and Preskill, 2000), are usually discussed and experimented with using photonics. The downside of photonics is that it is very difficult to make photons interact, which is required for other than single qubit operations.

- *Ion-traps*: Each qubit is embodied as an ion, confined inside a chamber by means of an electric or magnetic fields. The qubit is acted upon by hitting the ion with electromagnetic pulses (e.g. laser light or microwave radiation). This is one of the most promising technologies for the future of quantum computing,

with groups of experimentalists having proposed how to scale up the technology (Weidt et al., 2016).

- *Superconductors*: Small circuits, similar to classical electrical circuits, are cooled down to near absolute zero so the quantum interactions of electrons are not obscured by other perturbations. Then, different parts of the circuit encode different qubits, which can be acted upon by applying different electric potentials. One of the main advantages of this technology is that the challenges for scaling up such circuits, apart from the cooling system, are similar to the challenges we have encountered over the years for classical computers. Therefore, this technology seems to be the most feasible in the near future and evidence of that is the fact that, using it, both IBM and Intel have already built small generic-purpose quantum computers of 17-20 qubits.

However, for quantum computers to be useful in real world applications, their qubit count should raise up, at the very least, one order of magnitude. And, unfortunately, increasing the amount of qubits in a quantum computer is particularly difficult, due to some caveats we will now discuss.

2.2.1 Scalability challenges

There are two main challenges to overcome in order to build large scale quantum computers:

- *Decoherence*: In § 2.1 we discussed the importance of having superposition in quantum computing, and we compared a qubit in superposition with a coin spinning. For the same reason why a coin spinning will eventually stop, a qubit in superposition will eventually degenerate into a classical state (i.e. either $|0\rangle$ or $|1\rangle$): physical systems have a tendency towards the state at which they are most stable, for the coin it is laying flat, for the qubit it is losing its superposition. This phenomenon is known as *decoherence* and it will always occur in any given technology[3], in some faster than others. Experimentalists attempt to increase the time it takes for the state of the qubit to degenerate, which in both ion-trap and superconductor technologies it is in the order of microseconds. Decoherence

[3]Some experts have discussed that encoding qubits in *Majorana fermions* would avoid decoherence (Kitaev, 2001). Although promising, currently this proposal has little experimental underpinning, and it is not regarded as attainable in the near future.

constitutes the main constraint to scalability of quantum computers, as it dictates the lifespan of qubits, limiting the number of operations that can be applied in a single program.

Certainly, the state of bits also degenerates in classical computers. However, in their case this is easier to account for: intuitively, we can keep monitoring the bits, and make sure to correct any unwanted change. This is not so simple in quantum computers, as monitoring a qubit would require *measuring* it, and that destroys any quantum superposition. Nevertheless, it is still to some extent possible to protect our quantum state from errors – either due to decoherence or imperfect hardware – through quantum error correction routines (Knill et al., 2000). This is a very active area of research, and it will be key for the implementation of reliable large scale quantum computers.

- *Connectivity*: In order to run complex computations on qubits, we will need to be able to apply apply multi-qubit operations on any subset of the available qubits. However, it is not realistic to expect that quantum computers will have fast connectivity between all qubits, due to spatial separation of these in the hardware. In classical systems, this problematic is solved by a memory hierarchy, with a ceaseless flow of data going up and down of it, from main memory to registers and back. However, the memory hierarchy model works because most of the data can stay idly in main memory while computation on the registers data is carried out. In quantum computers, we must avoid qubits being idle, as decoherence prevents the existence long-lasting memory. An alternative found in classical computers is to distribute the computation across different processing units, each having its own local memory which they use intensively, and communicating – through message passing – as little as possible. In § 3.1, we discuss an abstract distributed quantum architecture in detail.

2.2.2 Models of computation

In this section, we give a brief introduction to some models of quantum computation relevant to the this thesis.

- *Circuit model*: Also known as the network model. Any operation on n qubits – as long as measurement (i.e. destruction of information) is not involved – can be represented as square matrix on complex numbers, of dimension 2^n. These ma-

trices are always unitary, which means that a matrix U satisfies $UU^\dagger = I = U^\dagger U$, where I is the identity matrix and $A^\dagger$ is the conjugate transpose of A. Essentially, unitarity ensures that any operation on qubits can be reversed (i.e. undone), reason why this model is sometimes called the reversible model. Multiplying matrices AB corresponds to applying the operation described by B first, then A, on the same qubits. Application of two operations on disjoint set of qubits corresponds to the Kronecker product of the matrices $A \otimes B$. The fundamental concept is that any matrix can be represented as a product of other matrices, so we may decompose any operation into smaller building blocks: quantum gates.

Qubits are pictured as wires to which quantum gates are applied, similarly to a classical digital circuit. The set of quantum gates used is dependent on the architecture. There exist an (uncountable) infinite amount of different quantum operations, but a small finite set of them is enough to approximate any of them, up to a desired error factor. The most common choice of such a universal gate-set is `Clifford+T`, which contains six one-qubit gates, and a single two-qubit gate. The depiction of such gates and some of their most important properties are shown in Figures **TODO**. Circuits are read from left to right.

TODO: A figure depicting, and giving the matrix of, each of CNOT, X, Y, Z, H, S and T. Figures showing HXH = Z; XX = I, ZZ = I, YY = I, HH = I; SS = Z; TT = S; H2 CNOT H2 = NOTC, and also their algebraic notation.

The CNOT gate (Figure **??**) is particularly interesting. The qubit where the filled dot is acts as the 'control', and the qubit with $\oplus$ acts as 'target'. Whenever the control is $|0\rangle$, the target is unaffected; but if it is $|1\rangle$, an X gate (Figure **??**) is applied, flipping the value of the qubit. This works in any superposition, so if in

$$|c,t\rangle = \alpha|0,0\rangle + \beta|0,1\rangle + \gamma|1,0\rangle + \delta|1,1\rangle$$

$|c\rangle$ were acting as control and $|t\rangle$ as target, the outcome would be:

$$CNOT|c,t\rangle = \alpha|0,0\rangle + \beta|0,1\rangle + \gamma|1,1\rangle + \delta|1,0\rangle$$

TODO: A figure showing an abstract quantum circuit.

- *MBQC model*: Initials stand for Measurement Based Quantum Computing. Unlike the circuit model, where measurements are done at the very end of the circuit, MBQC carries out computations by means of repeatedly measuring an initial entangled resource. The process, sketched in Figure **??** can be thought of as

sculpting a statue from a block of granite. The initial entangled resource, which is a collection of entangled qubits forming a lattice structure, corresponds to the granite block. By measuring some qubits in the lattice – hitting the rock with a chisel – we remove some of the excess qubits, changing the overall state in the process. The outcome of measurements is probabilistic so, in order to provide deterministic computation, we must apply corrections on the neighbouring qubits whenever the measurement outcome deviated from the desired result. After multiple iterations of measurements and corrections, we end up with a set of qubits encoding the result. The input was incorporated into the lattice at the beginning of the process.

TODO: Figure from slides

In this way, any computation may be performed by applying 1-qubit measurement and 1-qubit correcting gates (controlled by classical signals). The initial resource state contains all the entanglement that is required, which may be prepared experimentally through multi-qubit interactions, such as Ising interactions (Raussendorf and Briegel, 2001), which are within our experimental capabilities. Hence, in some sense this model solves the problem of connectivity by applying a single large operation at the beginning of the process, and then only requiring cheap single qubit operations. The main drawback of MBQC is the large amount of qubits that are required for even the simplest of operations, but given this might be surmountable considering the rest of the architecture is greatly simplified. The MBQC model was presented for the first time by Raussendorf and Briegel (Raussendorf and Briegel, 2001) under the name of *one-way quantum computer*, highlighting its main difference with the circuit (reversible) approach.

- *Distributed model*: We may find a balance between the circuit model and MBQC, where multiple small quantum processing units run fragments of the overall circuit, and communication is achieved through a shared entangled resource. This model has been discussed in detail in the literature (Van Meter et al., 2010) and it is at the core of the Networked Quantum Information Technologies Hub (NQIT)[4]. In § 3.1, we discuss an abstract distributed quantum architecture in detail.

[4] a project supported by the UK National Quantum Technology program, aiming to provide scalable quantum computing

2.3 Programming on Quantum Computers

As of today, most quantum programming languages are high level circuit descriptors: they provide the means to define circuits gate by gate, or build them up from combinations of smaller circuits. In this category fall all the well-known languages, such as *QCL* (Ömer, 2003) (imperative paradigm, and one of the first quantum programming languages ever implemented), *Q#* (Svore et al., 2018) (imperative, designed by Microsoft), and *Quipper* (Green et al., 2013) (functional, built on top of Haskell). Besides, there are attempts at designing quantum programming languages that are completely hardware agnostic, meaning they aim to describe the computation, rather than a particular circuit that implements it. Examples of these are the different attempts at defining a quantum lambda calculus, for instance van Tonder's (Van Tonder, 2004) or Diaz-Caro's (Díaz-Caro, 2017)). However, these are still early in their development and tend to be particularly verbose.

Additionally, most of the literature on quantum algorithms describes these by explicitly giving circuits that implement them. There is a constructive procedure, given by the Solovay-Kitaev theorem (Dawson and Nielsen, 2005), that takes any circuit and a choice of universal gate-set and outputs an efficient equivalent circuit using only those gates. Hence, programmers do not need to worry about the gates they are using when describing their circuits.

Unfortunately, the fact that algorithms are almost exclusively defined in the circuit model implies that other models of quantum computing (introduced in § 2.2.2) are disregarded by a large portion of the community. In order to make other models of computation accessible, we need to provide automated procedures for transforming algorithms from the circuit models to these (and vice versa). Work has been done on the transformation from circuit to MBQC and backwards, the latter being the most challenging (Duncan and Perdrix, 2010). However, there is little amount of literature describing how to go from the circuit model to the distributed model. In § 3.2 we give an overview of the existent work on that aspect, and identify the gap on the literature we aim to answer in this thesis.

2.4 Summary

As a wrap up, here are the key concepts to keep in mind while reading the rest of this thesis:

- Quantum computers provide a computing power well beyond the capabilities of classical computers, which would be exploitable in many areas of science.
- Small quantum computers are already available.
- Scaling up is a challenging problem due to: *decoherence*, which may be overcome by the joint effort of error-correction, physics and engineering communities; and *connectivity*, which may be solved using distributed architectures.
- There is practically no programming support for distributed architectures.

Chapter 3

Distributed Quantum Computing

3.1 Architecture

Resources for communication are ebits and classical messages.

Show how to implement an ebit. Say we will represent it as a cup and show the transposing trick on it.

Explain that we will assume that what is costly is the ebits, and we will also like to have balanced amount of qubits in each block.

3.2 Distributing circuits

As we have already explained, we aim to split a given circuit into distributed blocks. The gates that operate over qubits on different blocks are known as *non-local* gates. Given that any circuit can be converted to Clifford+T circuit – using, for instance, Solovay-Kitaev's algorithm – The only gate that operates on more than one qubit will be the CNOT. Hence, we only need to understand how CNOTs can executed non-locally, in order to have universal distributed computation.

TODO: A figure with a simple circuit and dashed lines identifying how we want to split it.

The construction we will use is a slight variation of what was proposed by Yimsiriwattana and Lomonaco Jr (2004), and it is shown in Figures **TODO**. In principle, we will use an *ebit* per non-local CNOT. Each half of the ebit is sent to a different block. We will call the block that holds the target qubit (the one with a $\oplus$) the 'target block' and similarly for the control qubit.

TODO: showing the overall scheme, with cat-entangler and cat-disentangler as black boxes.

We must first apply what the authors refer to as the *cat-entangler* (Figure **??**), which 'copies'[1] the state of the control qubit into the ebit half in the target block. To do so, the ebit half in the control block is measured (and thus destroyed), and the outcome is used to correct the other half, in the same spirit as in the MBQC model (see § 2.2.2). Notice that the only information physically crossing the boundary between blocks is the *classical* outcome of the measurement (a bit, either 0 or 1).

TODO: Show the circuit for the cat-entangler and the resulting state

Then, the CNOT gate may be applied between the ebit half in the target block and the target qubit itself. After it, the *cat-disentangler* must be applied (Figure **??**), which simply destroys – with a measurement – the remaining ebit half and then corrects the control qubit, so the randomness of the measurement is counteracted. Once again, only classical information crosses the boundary.

TODO: Show the circuit for the cat-disentangler and the resulting state

In this way, we have implemented a non-local CNOT gate using one ebit and two classical single bit messages between blocks. However, the true advantage of this approach is attained when multiple non-local CNOTs are implementing using a single ebit. The original paper (Yimsiriwattana and Lomonaco Jr, 2004) proposed one way of doing this. We will extend it on § 4.1. What they propose simply consists in realising that, after the cat-entangler is applied, any number of CNOTs that are controlled by the same qubit, and that target different places in a single target block, may all be implemented by using the ebit as control, as shown in Figure **??**.

TODO: Show the circuit for many CNOTs, now with cat-entangler and everything

Now, depending of how we choose to partition the code, there will be different groups of CNOTs that we can implement with a single qubit. We will then wish to find the partition that requires the least amount of ebits to implement all of its CNOT gates. This optimization problem is not discussed in the original paper, nor in any other work, as far as we know. It will be our contribution in this thesis, along with an extension of the results just explained, both found in Chapter 4.

[1] Note that there is no such thing as 'copying' a quantum state (due to the non-cloning theorem). What we mean here by 'copying' is generating, from $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, the state $|\psi\rangle = \alpha|0,0\rangle + \beta|1,1\rangle$ which is fundamentally different from $|\psi,\psi\rangle = \alpha^2|0,0\rangle + \alpha\beta|0,1\rangle + \alpha\beta|1,0\rangle + \beta^2|1,1\rangle$.

Chapter 4

Towards Distributed Quantum Algorithms

4.1 Implementing non-local CNOTs

In § 3.2, we explained the proposal by Yimsiriwattana and Lomonaco Jr (2004) of how to implent a non-local CNOT. We will now extend their results.

The first thing to notice is that the CNOTs need not immediately follow one on the control qubit. Assuming the circuit uses only the Clifford+T gate-set, all the 1-qubit gates but H can be pushed through the CNOT, and we may transform a circuit so the controls of different CNOTs are indeed together. The way they are pushed through is shown in Figures **TODO**. All of these can be checked by calculating the corresponding matrices and seeing they match.

TODO: Figures of how to push gates through control.

The second improvement comes by realising that the trick used to implement multiple CNOTs controlled by the same qubit can also be applied if multiple CNOTs target the same qubit. The derivation is shown in Figure **??**, which uses some of the properties listed in § 2.2.2.

TODO: CNOTtargetProof

Finally, we will be interested in also pushing gates through the target of a CNOT, in order to bunch them up together too. Doing so is slightly more elaborate, and the relevant rules are derived in Figures **TODO**.

TODO: Derivation of pushing gates through target (playing with Hadamard).

4.2 Finding the best partition

In this section we explain how we search for a suitable partition of the circuit. As discussed in § 3.1, we will be interested in achieving the minimal count of required ebits possible, while maintaining a balance of the number of qubits in each block. The problem is very similar to the $(k,\varepsilon$ graph partitioning problem, where a graph partitioning in k subgraphs has to be found, minimising the number of edges that have their incident vertices in different blocks, and ensuring the number of vertices in each block is within the ε tolerance factor: $(1\pm\varepsilon)\frac{N}{k}$, where N is the total number of vertices in the graph. Essentially, the qubits of the circuit would become vertices of a graph, whose edges would correspond to each CNOT in the circuit. Whenever an edge is 'cut' (i.e. its vertices are in different subgraphs), the CNOT it represents would be non-local.

But there is a caveat. If we use graph partitioning naively, we will not be exploiting the fact that multiple CNOTs may be implemented using a single qubit. In what follows, we will explain how to make use of hypergraph partitioning, instead of simple graph partitioning, to account for this aspect. A more detailed review of hypergraph partition is given in Appendix A, here we summarise the key concepts:

- Hypergraphs are the result of extending the definition of graphs to accommodate edges that may have a number of incident vertices other than two. More formally, a hypergraph is a pair of sets (V,H), where V is the set of vertices and $H\subseteq 2^V$ is the set of hyperedges. Each hyperedge is represented as the subset of vertices from V it connects. We will not consider any notion of directionality, i.e. all vertices of a hyperedge play the same role.

- Hypergraph partitioning follows the same premise as graph partitioning. The user provides the two parameters (k,ε), which have the exact same meaning as before, and a hypergraph. What the problem now attempts to minimise is a metric known as $\lambda-1$, which is defined as follows: given a partition of the hypergraph, the function $\lambda: H\to\mathbb{N}$ pairs each hyperedge with the number of different blocks its vertices are in. Then, $\lambda-1=\sum_{h\in H}\lambda(h)-1$ provides a measure of not only how many hyperedges are 'cut' but also across how many blocks[1].

[1] Simply minimising the number of hyperedges 'cut' is also an often used approach, but it is not as useful for our problem.

4.2.1 Vanilla algorithm

The one where only hyperedges from controlled qubits.

4.2.2 Gate pushed extension

The one where gates are pushed.

4.2.3 Both ends are useful extension

The one where we can also make hyperedges from target qubits.

Appendix A

Hypergraph Partitioning

Bibliography

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., and Lloyd, S. (2017). Quantum machine learning. *Nature*, 549(7671):195.

Dawson, C. M. and Nielsen, M. A. (2005). The solovay-kitaev algorithm. *arXiv preprint quant-ph/0505030*.

Díaz-Caro, A. (2017). A lambda calculus for density matrices withclassical and probabilistic controls. In Chang, B.-Y. E., editor, *Programming Languages and Systems*, pages 448–467, Cham. Springer International Publishing.

Duncan, R. and Perdrix, S. (2010). Rewriting measurement-based quantum computations with generalised flow. In *International Colloquium on Automata, Languages, and Programming*, pages 285–296. Springer.

Green, A. S., Lumsdaine, P. L., Ross, N. J., Selinger, P., and Valiron, B. (2013). Quipper: a scalable quantum programming language. In *ACM SIGPLAN Notices*, volume 48, pages 333–342. ACM.

Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. In *ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING*, pages 212–219. ACM.

Kitaev, A. Y. (2001). Unpaired majorana fermions in quantum wires. *Physics-Uspekhi*, 44(10S):131.

Knill, E., Laflamme, R., and Viola, L. (2000). Theory of quantum error correction for general noise. *Physical Review Letters*, 84(11):2525.

Kok, P., Munro, W. J., Nemoto, K., Ralph, T. C., Dowling, J. P., and Milburn, G. J. (2007). Linear optical quantum computing with photonic qubits. *Reviews of Modern Physics*, 79(1):135.

Lanyon, B. P., Whitfield, J. D., Gillett, G. G., Goggin, M. E., Almeida, M. P., Kassal, I., Biamonte, J. D., Mohseni, M., Powell, B. J., Barbieri, M., et al. (2010). Towards quantum chemistry on a quantum computer. *Nature chemistry*, 2(2):106.

Ömer, B. (2003). *Structured quantum programming*. na.

Ran Raz, A. T. (2018). Oracle separation of bqp and ph. In *Electronic Colloquium on Computational Complexity*.

Raussendorf, R. and Briegel, H. J. (2001). A one-way quantum computer. *Physical Review Letters*, 86(22):5188.

Shor, P. (1999). Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332.

Shor, P. W. and Preskill, J. (2000). Simple proof of security of the bb84 quantum key distribution protocol. *Physical review letters*, 85(2):441.

Svore, K., Geller, A., Troyer, M., Azariah, J., Granade, C., Heim, B., Kliuchnikov, V., Mykhailova, M., Paz, A., and Roetteler, M. (2018). Q#: Enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop 2018*, page 7. ACM.

Van Meter, R., Ladd, T. D., Fowler, A. G., and Yamamoto, Y. (2010). Distributed quantum computation architecture using semiconductor nanophotonics. *International Journal of Quantum Information*, 8(01n02):295–323.

Van Tonder, A. (2004). A lambda calculus for quantum computation. *SIAM Journal on Computing*, 33(5):1109–1135.

Weidt, S., Randall, J., Webster, S., Lake, K., Webb, A., Cohen, I., Navickas, T., Lekitsch, B., Retzker, A., and Hensinger, W. (2016). Trapped-ion quantum logic with global radiation fields. *Physical review letters*, 117(22):220501.

Yimsiriwattana, A. and Lomonaco Jr, S. J. (2004). Generalized ghz states and distributed quantum computing. *arXiv preprint quant-ph/0402148*.