Exploring Quantum Error Correction with Surface Codes

CMPT 409 - Summer 2020 - Simon Fraser University

George Watkins[*], Hoang Minh Nguyen[†], Miaoxing Wang ,
Nicholas Tan [‡], Ava Nobakhtian

August 10, 2020

Abstract

In this report we motivate and introduce quantum error correction with surface codes. We start with an overview of the problem of errors in quantum computers and a general discussion of how to deal with it. Then we proceeded to motivate why consider surface codes, with an outlook on implementation with superconducting qubits. Then we treat the operation of surface codes as a simple scheme for detecting and then correcting errors in physical qubits. We build up on such scheme, working towards the construction of a single logical qubit. In the last section we discuss an implementation. We first analyze the available quantum technologies, in relation to their possibility of implementing surface codes. Then proceed to showcase our implementation of a circuit generator for the error detection cycle of a rotated surface code.

Contents

[*]github.com/gwwatkin
[†]github.com/alexnguyenn
[‡]github.com/Gaphodil

1 Quantum Error Correction

1.1 The Problem of Errors in Quantum Computing

Similarly to their classical counterparts, quantum computers follow a precise model of computation, which describes the operations they perform. This models utilizes the concepts of qubits, qubit registers, gates and measurements, all of which have a precise mathematical formulation[1]. When a quantum algorithm is developed, it is often formulated in these terms and quantum programs are written in languages based on these idealizations. However, unlike classical computers, where bits are usually persistent and their operations assumed to be reliable, quantum computers suffer from a significant amount of error. This is caused by the nature of quantum mechanical entities, which is prone to stray interaction with the environment (known as *decoherence* [30]) and faults in the control systems that manage qubits and apply gates [29]. This lack of fidelity is considered one of the most significant limitations of the current generation of quantum computers, which has been described as Noisy Intermediate Stage Quantum (NISQ) [26]. In 1995, Shor proposed the first scheme for mitigating quantum error [31]. Since then, more quantum error correcting codes have been developed [29]. These codes might play a crucial role in transitioning from NISQ devices to a new generation of machines that fulfills the promises of quantum.

1.2 Error Types

In this paper, we focus on a simple form of error called *coherent errors* and how they are treated when analyzing surface codes. Coherent errors can be seen as unwanted applications of random unitary transformations on a qubit or, equivalently, rotations of a qubit in the Bloch sphere without decoherence. A coherent error may thus belong to a continuous range of unwanted amplitude changes. Fortunately, similar to how we model classical errors with a simplified model, we can do the same with coherent errors. What we do is *digitize* coherent errors into two simpler kinds: Pauli-X and Pauli-Z errors [9]. Pauli-X type errors, also known as *bit flips*, apply an X gate to qubits, mapping $|0\rangle$ to $|1\rangle$ and vice-versa. Pauli-Z type errors, also known as *phase flips*, apply a Z gate to qubits, mapping $|0\rangle$ to $|0\rangle$ and, $|1\rangle$ to $-|1\rangle$ and vice versa. On a general state, these errors distribute to its basis states components. The intuition behind this simplification arises from the Bloch sphere representation of a qubit. Pauli-X errors are a purely vertical motion on the Bloch sphere, while Pauli-Z errors are purely horizontal one, making them orthogonal. Given that the Bloch sphere is two-dimensional, a general coherent error can be expressed as a linear combination of a Pauli-X error and Pauli-Z error [9]. Then the amplitude of the Pauli components of an error translate into the probability it occurring. These errors can happen anywhere in a quantum circuit, such as during the application of quantum gates or during measurement, and are often related to faults in the control mechanisms [29], something superconducting qubits, for example, do suffer from [21].

1.3 Quantum Error Correcting Codes

Let us introduce the main ideas behind quantum error correction in general and some of the technical terminology used in the field. A way of mitigating errors is to add redundancy to quantum information. That is, we use multiple physical qubits to represent a single one, which carries the meaning of the computation. We call this abstract qubit a *logical qubit*. In doing so, we effectively increase the dimensions of the Hilbert space qubits are encoded in for redundancy. Within this higher dimensional space, spanned by the physical qubits, error free values of the logical qubit will belong to a subspace called the *codespace*. We can determine the subspace which a logical qubit with a *projective* measurement, also called *stabilizer* measurement. Stabilizer measurement is useful, for instance, if the the codespace and the error subspace are orthogonal, so that we can tell if there was an error. We use helper qubits, often called *ancilla* qubits, to perform the stabilizer measurements. These ancilla qubits are entangled in such a way such that their measurement only extracts error information, called *syndrome* [29], without destroying the logical qubits state. The syndrome can be used to determine the most likely error which occurred, so that it is possible to apply gates that reverse it.

2 Surface Codes

2.1 Motivation

A natural question is why are surface codes interesting. A key reason is that they work with physical qubits in a simple two dimensional layout, hence the "surface" in the name. Suitable qubit layouts have already appeared in real quantum computers, most notably Google's Sycamore device, with which claims of quantum supremacy were made [1]. The authors of the experiment note that the device was intentionally created to be forward compatible with surface codes.

[1]We refer the reader to [23] for an introduction to these concepts.

From a performance perspective, surface codes are relatively tolerant to local errors compared to other quantum error correcting codes [12], and there are claims it can operate with physical qubit error rates above 1% [34; 3]. With more qubits better improvements in fidelity are possible; thus, surface codes result in a trade off between error rate and number of qubits [12]. Such a trade off might be advantageous, since it is possible that devices will increase in number of qubits following a patterns similar to Moore's law of semiconductors in classical computers [25].

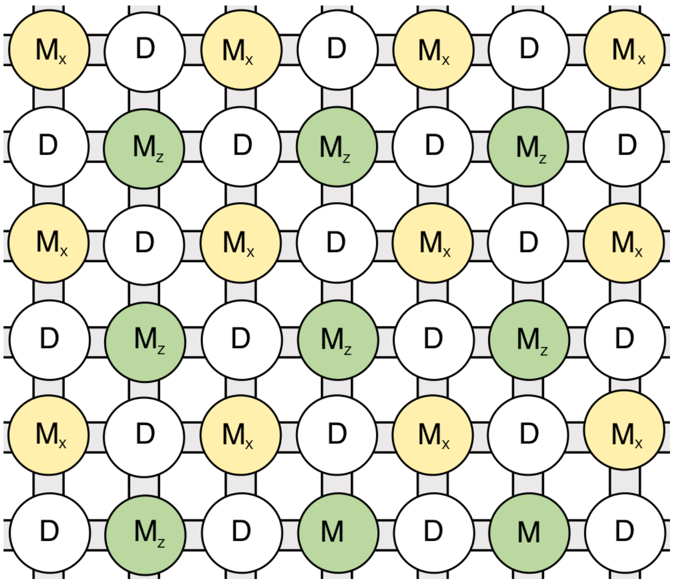
2.2 Superconducting Qubits

Superconducting qubits are one of the most promising technologies being explored for the creation of quantum computers [21], and have been considered for implementations of devices for operating surface codes [13; 2]. Unlike most other qubit technologies, which utilize microscopic entities, superconducting qubits exploit the quantum mechanical properties of macroscopic electrical elements [10]. Such electrical elements are fabricated using procedures similar to those employed in silicon CMOS manufacturing [21; 24]. As demonstration of this technology's potential, D-Wave's annealing machines, which will soon have up to 5000 qubits [8] and employ their quantum properties (albeit in a different form), are built with silicon based super conducting technology [7]. The Sycamore processor utilizes superconducting qubits as well [1]. It is reasonable from here to imagine that computers built with superconducting qubits are good candidates for scaling to large sizes, and therefore more capable of exploiting the trade off between qubit number and error rate brought by surface codes (Sec. 2.1). Additionally, superconducting qubits fall naturally into a two dimensional layout, which could work well with surface codes' topologies [21] (Sec. 2.3).

2.3 Operation

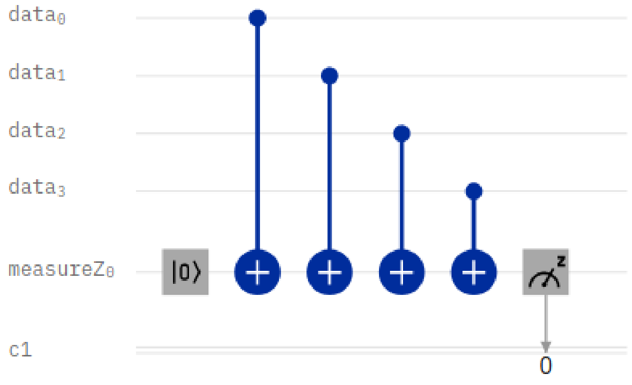
While surface codes with different topologies exist [19; 20; 35], for the purpose if illustrating the physical layout, we will focus on a square lattice code described by Fowler et al. [12] based on the one described by Bravyi and Kitaev [4]. This kind of surface code uses a two-dimensional lattice of measurement qubits and data qubits. We start by characterizing its physical qubit, then seeing how the code operates in cycles that detect and correct errors.

In a surface code's qubit layout there are *data qubits* and *measurement qubits*, which are divided into *measure-X* and *measure-Y*. A measurement qubit is surrounded by four data qubits, and each data qubit is surrounded by two measure-Z qubits and two measure-X qubits. Thus we have a two dimensional lattice of qubits,

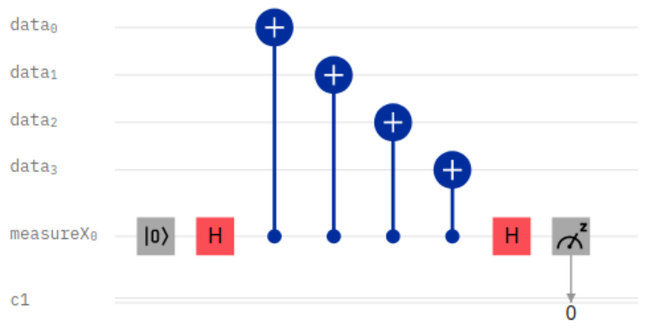


The origins of surface codes are in toric codes [12] first described by Kiatev [20]. Toric codes can be embedded on a torus and have connected left-right and top-bottom boundaries, thus it is not necessary to consider the edges to be special cases. Planar surface codes can be embedded on a 2D plane and have two smooth boundaries and two rough boundaries, with opposite edges having identical boundary types. Smooth boundaries end with measure-X qubits and rough boundaries end with measure-Z qubits [16].

A quantum program employing the code operates in cycles. For each cycle we focus on the operations of the measurement qubits. We examine first what happens to the measure-Z qubit, then the measure-X qubit during a single cycle of the surface code. In a cycle, for each measure-Z qubit, we apply four CNOT gates, with each surrounding data qubit as the control qubit and the measurement qubit as the target qubit, and then we perform a measurement of the measure-Z qubit. A circuit representation made with IBM Q Experience [11] follows,



At the same time, during a cycle a cycle, each measure-X qubit, we apply a Hadamard gate. We use the measure-X qubit as a control qubit for a CNOT gate, whose target qubits are the four surrounding data qubits. We apply a Hadamard gate to the measure-X qubit after applying all the CNOT gates, and then we perform a projective measurement. And a circuit representation,



The first cycle of the surface code uses the ground state $|0\rangle$, and subsequent cycles leave the qubits in a quiescent state $|\psi\rangle$, unless there is an error, in which case the state changes.

2.4 Error Detection

Next, we show how to use the above circuit structure and cycle to detect and correct some amount of digitized coherent errors (Sec. 1.2). We will limit ourselves to an intuitive understanding of the process, based on the review [12]. Extensive treatment of the error correcting ability of surface codes with the topology we present is given in [3]. While we focus on coherent errors, the code does also mitigate errors due to decoherence; however, the mathematical treatment of those errors in surface codes is rather involved [2].

In an $[[n, k, d]]$ surface code, n is the number of data qubits, k is the number of logical qubits represented by the surface code, and d is the code distance. The distance d of a code is the minimum number of physical operations required to change the logical qubit's state. A code with distance d can successfully correct $\left\lfloor \frac{d-1}{2} \right\rfloor$ errors [33]. In the code's most simple formulation, errors are detected as follows[3]:

- if two measure-Z qubits adjacent to a data qubit present a flip of their readouts, then a Pauli-X error occurred in the data qubit
- if two measure-X qubits adjacent to a data qubit present a flip of their readouts, then a Pauli-Z error occurred in the data qubit
- if a single measurement qubit encounters an flip, with no opposing counterpart, an error to has occurred in the measurement qubit

Let us first proceed with an informal explanation of why this works, and then a more formal discussion.

Suppose a Pauli-X error occurs in a data qubit - a bit flip. Thanks to the entanglement obtained with the CNOT operation, between the data qubit and the measurement qubits (Sec. 2.3), the qubits around the data qubit will be affected by this X flip. For the two measure-Z qubits, the phase bit of the data qubit implies a bit flip as well, which means that their measurement readouts are inverted. Therefore, we can detect if there was a Pauli-X error in a data qubit if the two measure-Z qubits surrounding the data qubit flip their measurement readouts. Now suppose a Pauli-Z error occurs in the data qubit. Similarly to the above case, entanglement will propagate the Z flip, a phase flip, to the surrounding qubits. The Hadamard gates combined with the CNOTs, between the data qubits and the measure-X qubit, have the following effect; they map flips in the Hadamard basis ($|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$ and $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$) for the data qubits to flips in the Standard Basis ($|0\rangle$ and $|1\rangle$) in the measure-X qubits. Therefore the phase flip of the data qubit results in a bit flip of the measure-Z qubit. From this interaction we can conclude that Pauli-Z errors are detected by checking if the two measure-X qubits surrounding the data qubit flip their measurement readouts.

Because of the orthogonality between bit flips and phase flips, measurement qubits only detect their designated error type. Measure-Z qubits are unaffected by Pauli-Z errors in data qubits, as they result in a phase flip, which is not detectable in measurement. Similarly, Pauli-X errors do not affect measure-X qubits because errors induce a phase flip in them as well.

Given the above intuition, we can expand our understanding with more rigor. Pauli-Z measurements project a qubit to the basis containing $|0\rangle$ and $|1\rangle$, while Pauli-X measurements project a qubit onto the basis containing $|+\rangle$ and $|-\rangle$. Pauli-X and Pauli-Z measurements on the same qubit are not commutative and will destroy the quantum state. Pauli-X and Pauli-Z gates applied to different qubits will

[2]Ghosh et al. [14] analyze the issue analytically and simulate a surface code with Monte Carlo methods, emulating superconducting qubits. A more intuitive approach to error correction and decoherence, though not specific to surface codes and still requiring a basic knowledge of quantum mechanics, is given in section IX.A of [9].

[3]Note how measurement qubits and error types have the operator types swapped. Though this may seem counter-intuitive, this is rooted in the operator interpretation of measurement; see [12], Appendix B.

commute, so they can be applied in any order. The stabilizers $X_a \otimes X_b$ and $_a \otimes Z_b$ commute, where a and b are qubits.

A careful reader might wonder what happens if two neighbouring data qubits encounter an error of the same type in the same cycle. In this event, the measurement qubit between them will present two bit flips, which cancel out. These are called *error chains* [12] and techniques for dealing with them have been proposed.[33]

2.5 Error Correction

A peculiar feature of being able to detect digitized coherent errors is that they do not need to be corrected as they occur. It suffices to track and correct them classically after measurement. Initially, one might see as the simplest way to deal with Pauli errors is to correct them as they are detected. For example, as soon as two measure-X qubits adjacent to a data qubit change measurement value, we could immediately apply a Pauli-Z gate to the data qubit to revert the error. However, this is unnecessary; because coherent errors only affect measurement, we can track which errors happened, and after receiving the readout of a data qubit, apply as many corrections as needed according to the recorded errors [12].

2.6 Towards Logical Qubits

We have seen how the code's cycles detect and corrects errors, but it does not do anything else yet. In this subsection we try to build an intuition for how we can go about making logical qubits from the techniques we've introduced.

It appears that the cycles of the surface code are way of protecting the data qubits from error. But quantum computation is usually based qubits and operators, therefore a way of representing those is needed. And the way it is done is by *logical operators*. Logical operators, through their operation, define a logical qubit, thus acting as gates on logical qubits. For example, a logical operator X_L or Z_L can be a chain of $X^{\otimes n}$ or $Z^{\otimes n}$ operations that spans two boundaries of the same type, for n data qubits [33]. An example of a surface code based scheme for a logical qubit is the *rotated surface code* [32] (with $d = 3$), which represents one logical qubit. Our circuit implementation (Sec 3) is based on such scheme. Another way of representing multiple logical qubits in a surface code by not applying quantum gates on some ancilla or data qubits, creating holes. This is technique is known as *lattice surgery* [22]. Implementing logical operations using To reduce the error rate, we can apply logical gates in a surface code by cancelling common logical operations in the circuit control software before their physical application [12].

3 Circuit implementation

3.1 Overview

For this project we wrote a tool that generates a quantum circuit for error detection using surface codes. The generated circuit runs iterations of a surface code and stores measurement outcomes in a classical register, where a control system could use them. The qubit layout is based on the rotated surface code (Sec 2.6). This tool builds an arbitrary size circuit that is meant to be run on a finite square lattice. Special attention was given to mapping the qubits of the circuit to the physical layout. For this reason a visualizer of the layout was written to facilitate the association of the circuit qubits to their physical position. After an extensive overview of the available quantum technologies, which we summarize in Sec 3.2, we concluded that it was best to aim for simulator executions and chose IBM's Qiskit[18] environment. The code is available at the following repository [4].

3.2 Platform

There are multiple platforms for quantum algorithm creation which are developed by various companies such as IBM [17], Microsoft [5], Google [15], and Rigetti [27]. In quantum computing, there is no standard technology and most often choosing the right platform will be based on the physical quantum computer that will be used at the end. Still, there are factors that can affect the decision making process and favour one platform over the other, which are briefly discussed below.

- Community Support

  Similar to any other computer tool, it is important to have good community support when using a particular quantum computing platform. The importance of community support becomes more crucial when we consider the fact the none of available platforms can be considered "mature" and are actively being improved. Therefore, it is common to run into various issues where the relevant community can help find proper resolutions.

- Tutorial Availability

[4]https://github.com/alexnguyenn/surface-code-generator

Quantum computer programming has a steep learning curve and many concepts are relatively new with few available resources. Therefore, it is important to have access to good tutorials and starting codes that help us get on board and start developing quantum algorithms.

- Tool Maturity

  Quantum computing is a young and fast growing area but still it is important to work with the tools that are relatively more mature than the others to increase code reliability and access resources if needed. In this regard, Microsoft Qsharp [6] and Google Cirq [15] are younger and less tested than IBM's qiskit [18] and Rigetti's Forest [28], which have been used in at international competitions, universities and research facilities.

- Development Environment

  Having access to widely used development platforms is a decisive factor in choosing the right platform, as it helps to reduce the gap between classical and quantum computers. To this end, leading quantum computer companies allow users to develop code in Python and Visual Studio, which are easy to learn and have good community support. Examples of such platforms are Rigetti's Forest [28], IBM's Qiskit [18], and Microsoft's Quantum Development Kit [6].

- Testing on Physical Quantum Computers

  Not all platforms provide access to physical quantum computers, so it is important to know whether the candidate platform provides easy deployment to a real machine. In this sense, Rigetti's Forest [28] and IBM's Qiskit [18] will be natural choices as they have built-in support for implementing the developed algorithm on a quantum computer. Unfortunately, we were unable to find freely accessible physical devices with surface code compatible circuit layouts on any of these platforms. It must be noted that Google's quantum processor Sycamore has a suitable layout (Sec. 2.1), but it was not clear how we would be able to access that device.

From the above factors, community support and access to valuable tutorials were more important in our decision making process. Considering that implementing a surface code was going to be challenging, it was important to have access to tutorials and community support as well as a user-friendly development environment. Due to technical challenges associated with practical surface codes, we decided to pursue a simulator implementation which provided more freedom in terms of qubit connectivity and layout, a primary concern for the code (Sec. 2.3). Considering all these factors, IBM's Qiskit [18] was selected in this project as it provided access to several good documentations and allowed us to develop the code in Python, which can be used to automatically generate the complex quantum circuitry required by the code. Moreover, Qiskit's Quantum Virtual Machine (QVM) was an easy-to-use tool to test the developed code and evaluate its effectiveness.

3.3 Demonstration

Let us proceed to showcasing the functionality of our surface code generating tool. To take full advantage of its functionality, a Python Jupiter Notebook with the Qiskit and Graphviz Python packages installed is advised. The IBM Q Experience [11] cloud platform constitutes a suitable environment once the source files in the repository are imported. We start by including the file, and constructing the main interface object of type `RotatedSurfaceCode`,

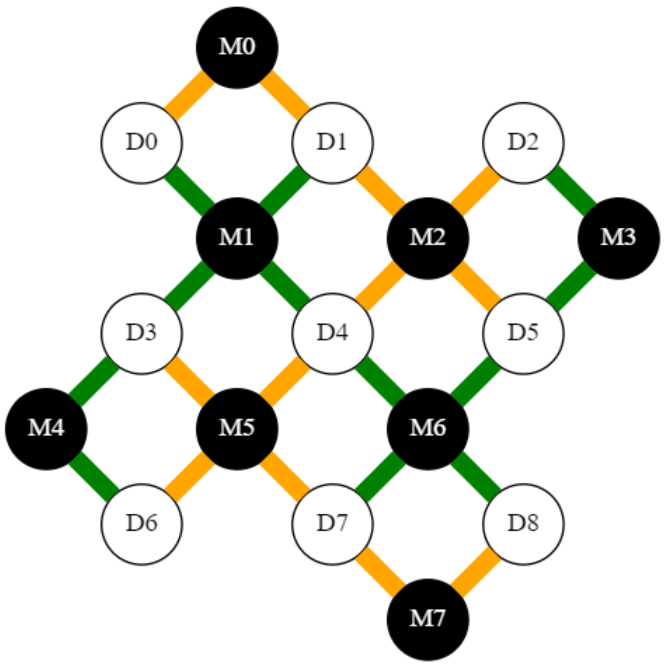
```python
from circuit import RotatedSurfaceCode
"""
d - Code Distance
T - # of Measurement Rounds
"""

d = 3
T = 3
code = RotatedSurfaceCode(d, T)
```

The object `code`, contains the Qiskit circuit representing the desired code in the member `circuit`. This member is a `qiskit.QuantumCircuit` object which can executed with the `qiskit.execute()` function or visualized with its `.draw()` method. However the regular 2-d, "music score" like display of the circuit may give little insight to the operation of the code on its own. For this reason, the object `code` has a method to show the qubits location in the physical layout,

```python
code.draw_lattice()
```

Which for this case produces the following output in a Notebook, the colors indicate the qubit types. White circles are data qubits and the black ones are measurement qubits. The colors of the connection indicate the measure qubit type, green is for measure-Z and orange for measure-X, as documented in the API,



4 Conclusion

4.1 Final Remarks

We begin with a general discussion of errors in quantum computers and strategies to tackle them, developing the main concepts of quantum error correction. Then we start focusing on surface codes by explaining the reasons for their appeal, planar layout and high error tolerance. We make particular note of how surface codes can exploit large numbers of qubits to improve fidelity, and how superconducting qubit technology could scale the number of qubits compatibly with surface codes. At this point we go over physical qubit layout and the surface code's cycle. This concept provides the fundamental mechanism for detecting quantum errors, which can then be corrected classically after measurement. With this framework of error detection as a basis, we give a brief overview of how logical qubits could take form. Having introduced the reader to surface codes, we dedicate the rest of the paper to the implementation of a generator for making quantum circuits that executes the surface code's error detecting cycle. To engage in such a task we scrutinized quantum technologies and chose Qiskit, focusing on simulator runs. The code layout we implement is called a rotated surface code, and the generated circuits can detect errors by storing measure wubit readouts in classical registers for later processing.

4.2 Ideas for Future Work

A natural next step could be to improve on the discussion of logical qubits, giving some practical examples of how to operate them in some simple codes. Lattice surgery techniques [22] could be employed. With such background logical qubit operations could be implemented in the circuit generator, and could be abstracted away into an API. To do that we would have to look further into the problem of error decoding, that is going from the readouts in the classical registers of the measure qubits to corrections to data qubit measurements. And generally we would have to research further into the classical machinery that accompanies the quantum circuit of a surface code.

References

[1] F. Arute, K. Arya, R. Babbush, D. Bacon, J. C. Bardin, R. Barends, R. Biswas, S. Boixo, F. G. Brandao, D. A. Buell, et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[2] R. Barends, J. Kelly, A. Megrant, A. Veitia, D. Sank, E. Jeffrey, T. C. White, J. Mutus, A. G. Fowler, B. Campbell, et al. Superconducting quantum circuits at the surface code threshold for fault tolerance. *Nature*, 508(7497):500–503, 2014.

[3] S. Bravyi, M. Englbrecht, R. König, and N. Peard. Correcting coherent errors with surface codes. *npj Quantum Information*, 4(1):1–6, 2018.

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

[5] M. Corporation. Microsoft quantum computing. https://www.microsoft.com/en-us/quantum, 2020.

[6] M. Corporation. Microsoft quantum development kit. https://www.microsoft.com/en-ca/quantum/development-kit, 2020.

[7] I. D-Wave Systems. Introduction to the d-wave quantum hardware. https://www.dwavesys.com/tutorials/background-reading-series/introduction-d-wave-quantum-hardware, 2020.

[8] I. D-Wave Systems. Techrepublic: D-wave announces 5,000-qubit fifth generation quantum annealer. https://www.dwavesys.com/media-coverage/

techrepublic-d-wave-announces-5000-\ qubit-fifth-generation-quantum-annealer, 2020.

[9] S. J. Devitt, W. J. Munro, and K. Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[10] M. H. Devoret, A. Wallraff, and J. M. Martinis. Superconducting qubits: A short review. *arXiv preprint cond-mat/0411174*, 2004.

[11] I. Q. Experience. https://quantum-computing.ibm.com/, 2020.

[12] A. G. Fowler, M. Mariantoni, J. M. Martinis, and A. N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[13] J. M. Gambetta, J. M. Chow, and M. Steffen. Building logical qubits in a superconducting quantum computing system. *npj Quantum Information*, 3(1):1–7, 2017.

[14] J. Ghosh, A. G. Fowler, and M. R. Geller. Surface code with decoherence: An analysis of three superconducting architectures. *Physical Review A*, 86(6):062318, 2012.

[15] Google. Cirq. https://en.wikipedia.org/wiki/Cirq, 2020.

[16] P. Groszkowski. Surface code thresholdcalculation and flux qubitcoupling. Master's thesis, the University of Waterloo, Waterloo, Ontario, Canada, 2009.

[17] IBM. Ibm quantum computing. https://www.ibm.com/quantum-computing/, 2020.

[18] IBM. Ibm's qiskit. https://qiskit.org/, 2020.

[19] M. S. Kesselring, F. Pastawski, J. Eisert, and B. J. Brown. The boundaries and twist defects of the color code and their applications to topological quantum computation. *Quantum*, 2:101, Oct. 2018.

[20] A. Y. Kitaev. Quantum communication, computing, and measurement. In *Proceedings of the 3rd International Conference of Quantum Communication and Measurement, New York: Plenum*, 1997.

[21] M. Kjaergaard, M. E. Schwartz, J. Braumüller, P. Krantz, J. I.-J. Wang, S. Gustavsson, and W. D. Oliver. Superconducting qubits: Current state of play. *Annual Review of Condensed Matter Physics*, 11:369–395, 2020.

[22] D. Litinski. A game of surface codes: Large-scale quantum computing with lattice surgery. *Quantum*, 3:128, 2019.

[23] M. A. Nielsen and I. Chuang. Quantum computation and quantum information, 2002.

[24] R. Pillarisetty, N. Thomas, H. George, K. Singh, J. Roberts, L. Lampert, P. Amin, T. Watson, G. Zheng, J. Torres, et al. Qubit device integration using advanced semiconductor manufacturing process technology. In *2018 IEEE International Electron Devices Meeting (IEDM)*, pages 6–3. IEEE, 2018.

[25] E. Prati, D. Rotta, F. Sebastiano, and E. Charbon. From the quantum moore's law toward silicon based universal quantum computing. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–4, 2017.

[26] J. Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[27] I. Rigetti & Co. Rigetti computing. https://www.rigetti.com/, 2020.

[28] I. Rigetti & Co. Rigetti forest sdk documentation. http://docs.rigetti.com/en/stable/, 2020.

[29] J. Roffe. Quantum error correction: an introductory guide. *Contemporary Physics*, 60(3):226–245, Jul 2019.

[30] M. Schlosshauer. Quantum decoherence. *Physics Reports*, 831:1–57, 2019.

[31] P. W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical review A*, 52(4):R2493, 1995.

[32] Y. Tomita and K. M. Svore. Low-distance surface codes under realistic quantum noise. *Physical Review A*, 90(6):062320, 2014.

[33] S. Varsamopoulos, K. Bertels, and C. G. Almudever. Decoding surface code with a distributed neural network–based decoder. *Quantum Machine Intelligence*, 2(1):1–12, June 2020.

[34] D. S. Wang, A. G. Fowler, and L. C. L. Hollenberg. Surface code quantum computing with error rates over 1 *Phys. Rev. A*, 83:020302, Feb 2011.

[35] T. J. Yoder and I. H. Kim. The surface code with a twist. *Quantum*, 1:2, 2017.