New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG IOP Institute of Physics

PAPER • OPEN ACCESS

Surface code error correction on a defective lattice

To cite this article: Shota Nagayama *et al* 2017 *New J. Phys.* **19** 023050

New Journal of Physics
The open access journal at the forefront of physics

Deutsche Physikalische Gesellschaft DPG
IOP Institute of Physics

Published in partnership with: Deutsche Physikalische Gesellschaft and the Institute of Physics

OPEN ACCESS

RECEIVED
20 July 2016

REVISED
10 November 2016

ACCEPTED FOR PUBLICATION
12 January 2017

PUBLISHED
23 February 2017

PAPER

Surface code error correction on a defective lattice

Shota Nagayama[1], **Austin G Fowler**[2], **Dominic Horsman**[3], **Simon J Devitt**[4] **and Rodney Van Meter**[1]

[1] Keio University, 5322 Endo, Fujisawa-shi, Kanagawa 252-0882 Japan
[2] Google Inc., Santa Barbara, CA 93117, United States of America
[3] Department of Physics, Durham University, South Road, Durham DH1 3LE, United Kingdom
[4] Center for Emergent Matter Science, RIKEN, Wakoshi, Saitama 315-0198, Japan

E-mail: kurosagi@sfc.wide.ad.jp

Keywords: quantum error correction, surface code, topological quantum error correction, qubit loss, fault tolerant quantum computation

Supplementary material for this article is available online

Abstract

The yield of physical qubits fabricated in the laboratory is much lower than that of classical transistors in production semiconductor fabrication. Actual implementations of quantum computers will be susceptible to loss in the form of physically faulty qubits. Though these physical faults must negatively affect the computation, we can deal with them by adapting error-correction schemes. In this paper we have simulated statically placed single-fault lattices and lattices with randomly placed faults at functional qubit yields of 80%, 90%, and 95%, showing practical performance of a defective surface code by employing actual circuit constructions and realistic errors on every gate, including identity gates. We extend Stace *et al*'s superplaquettes solution against dynamic losses for the surface code to handle static losses such as physically faulty qubits [1]. The single-fault analysis shows that a static loss at the periphery of the lattice has less negative effect than a static loss at the center. The randomly faulty analysis shows that 95% yield is good enough to build a large-scale quantum computer. The local gate error rate threshold is $\sim$0.3%, and a code distance of seven suppresses the residual error rate below the original error rate at $p = 0.1\%$. 90% yield is also good enough when we discard badly fabricated quantum computation chips, while 80% yield does not show enough error suppression even when discarding 90% of the chips. We evaluated several metrics for predicting chip performance, and found that the average of the product of the number of data qubits and the cycle time of a stabilizer measurement of stabilizers gave the strongest correlation with logical error rates. Our analysis will help with selecting usable quantum computation chips from among the pool of all fabricated chips.

1. Introduction

Fully scalable quantum computers are required in order to solve meaningful problems [2–7]. For example, processing Shor's algorithm to factor a number described with N bits requires a quantum register with at least $2N + 2$ high-quality qubits [8, 9]. Many architectures have been proposed for a scalable quantum computer and their feasibility depends on the physical systems in which they are implemented and the physical operations they use [10–13]. The coherence time of a quantum state is limited as the quantum state is easily changed by noise; fault-tolerant quantum computation (FTQC) is therefore required [14–18]. The surface code is one of the most feasible current proposals for FTQC, requiring a 2D square-lattice of qubits and interaction only between nearest neighbors, maintaining good scalability and having a higher threshold than other codes on equivalently constrained architectures [19–23]. The surface code qubits are grouped in 'plaquettes' which, in the absence of faulty components, consist of four neighboring qubits in the lattice. Each plaquette is associated with a stabilizer measurement. There are two types of stabilizers, Z stabilizers and X stabilizers, enabling the correction of arbitrary errors. Error syndromes are associated with pairs of sequential stabilizer measurements that differ.

In reality, the problems we face include not only state errors but also losses of quanta. Some examples of loss mechanisms are static loss such as devices incapable of trapping single electrons for use as qubits, and dynamic

loss such as photon generation failure or dynamic loss of other qubit carriers. There are many proposals for 2D nearest-neighbor architectures on which the surface code runs. However, each of them suffers from the problems we just mentioned; if a qubit is missing, there will be a hole in the code. DiVincenzo offered an architecture of superconducting hardware for the surface code [16], in which a superconducting loop which does not show the appropriate quantum effect will be a hole. Jones *et al* proposed an architecture for scalable quantum computation with self-assembled quantum dots used to trap electrons, which are used as qubits [11]. There very likely will be defective quantum dots which cannot trap a single electron, leaving holes in the code. During the initial boot stage, qubits are calibrated; if qubits cannot be tuned to hold a single quantum, or if they cannot be tuned to match their neighbors, they can be declared not working.

The surface code is robust against unintended changes of quantum state provided these changes are local in space and time, but it does not address loss. To resolve this problem, we have two choices: design a microarchitecture to work around missing qubits or adapt the syndrome collection and processing to tolerate loss. Van Meter *et al* proposed a system in which the microarchitecture can create the regular 2D lattice even when some qubits are faulty [10]. However, this requires the ability to couple qubits across a distance spanning several qubit sites. Stace *et al* showed that qubit loss is acceptable when performing the surface code and that there is a tradeoff between the loss rate and the state error rate [1]. They introduced the concept of a 'superplaquette' ('superstar') which consists of several plaquettes (stars) that surround defective qubits. They showed that, under the assumption that the superplaquette operators can be measured perfectly, a threshold error rate exists for qubit loss rates below 50%. Barrett *et al* showed that dynamic loss in the 3D topological quantum computation is acceptable up to $p_{\mathrm{loss}} = 24.9\%$ [24]. This latter approach, however, cannot be used if a device used to bond together qubits in the 3D topological lattice in the quantum computer is permanently faulty, leading to a column in time of lost qubits. We measure the six face qubits in a unit cell for syndrome extraction in the 3D topological computation because the six qubits are the output of $\Pi_i X_i \otimes_{q_j \in ngbr(q_i)} Z_j$, where q_i are the face qubits. A lost qubit merges $\Pi_i X_i \otimes_{q_j \in ngbr(q_i)} Z_j$ of two unit cells [25, chapter 20]. A column in time of lost qubits from the beginning of the computation to the end works as a logical qubit because we do not have stabilizer $X_0 \otimes_{q_j \in ngbr(q_0)} Z_j$ on the first 2D surface of the 3D cluster state where the lost column starts from q_0 and $X_0' \otimes_{q_j \in ngbr(q_0')} Z_j$ on the last 2D surface; therefore, the merged $\Pi_i X_i \otimes_{q_j \in ngbr(q_i)} Z_j$ of the merged unit cells is not closed. Hence, another solution is required for faulty devices in 3D topological code.

In this paper, we examine these theoretical limits in the context of errors in the state and stabilizer measurement process. We give the realistic relationship of the 2D surface code between static loss tolerance and state error tolerance by employing explicit stabilizer circuit constructions. For simplicity, we mainly discuss superplaquettes in this paper; however, we can apply similar statements to superstars since they have symmetry. Additionally, we introduce the concept of a defective stabilizer whose syndrome qubit—the ancilla qubit in the center of the stabilizer used to measure the error syndrome—is defective. Finally, we show the acceptability of such special stabilizer units with examples of some stabilizer circuits and graphs of the relationships between qubit yield, code distance, and effective code distance. The effective code distance is the value characterizing how many gate errors the code can truly handle given the presence of defective qubits. We analyze the correlation between the logical error rates we find on each defective lattice and the characteristics of those defective lattices to find indicators to judge whether a defective lattice is good enough to use. We simulate yields of 0.95, 0.90, and 0.80. Our results show that once the fabrication yield reaches 90%, it becomes possible to build large-scale systems by culling the poorer 50% of chips during post-fabrication testing. Yield 0.80 is not usable even when discarding 90% of generated lattices. We find that the deepest depth among the stabilizer circuits has the largest correlation with the logical error rate of the lattice and the biggest number of data qubits owned by a stabilizer has the next largest correlation. Large-scale quantum computation will require an ensemble of sufficiently fault-tolerant quantum computation chips, coupled either by their proximity or through the use of quantum communication links [26–32]. Our results will contribute to guiding the construction of this ensemble.

2. Surface code quantum computation

The surface code is a means for encoding logical qubits on a form of entangled 2D lattice, consisting of many qubits, made by interaction only between nearest neighbor qubits. This fact makes it potentially possible to fabricate devices using planar photolithography, including quantum dot, superconducting, and planar ion trap structures. It gives the quantum processor extensibility by adding one more row of qubits and control devices along the outside edge of the lattice, making it one of the most feasible current proposals for building a scalable quantum computer.

The lattice is divided into many plaquettes and the state of the lattice is maintained by repeatedly measuring sets of stabilizers. By definition, a stabilizer generator is a set of Pauli operators U that do not change a state $|\Psi\rangle$, such as $|\Psi\rangle = U|\Psi\rangle$. We check a stabilizer by extracting its eigenvalue. X (Z) errors are checked using the Z (X)

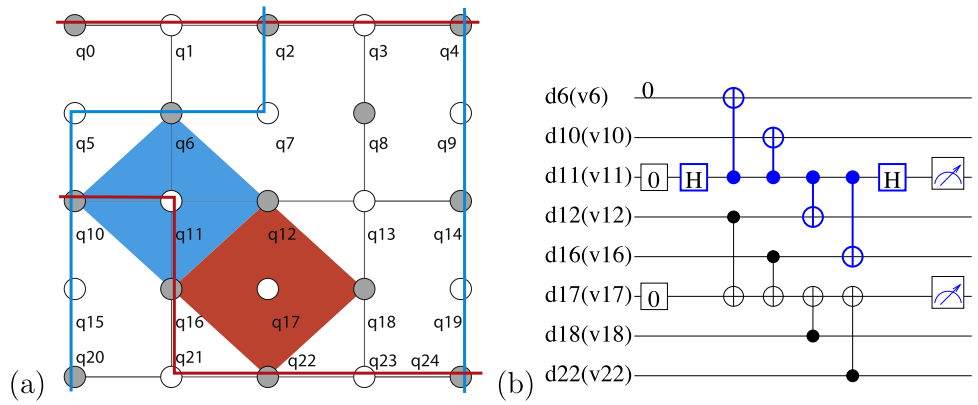

Figure 1. (a) Example of a surface code encoding a single logical qubit, describing a Z stabilizer (red diamond), an X stabilizer (blue diamond), two instances of the Z operator (red lines) and two instances of the X operator (blue lines). The gray (white) circles are data (syndrome) qubits. Qubits q12, q16, q18, and q22 (q6, q10, q12, and q16) are included in the Z (X) stabilizer. Other qubits are also involved in other corresponding stabilizers. The west and the east boundaries of the lattice are for the Z operator. Other possible lines between the west and the east boundaries also have the same Z operator. The X operator runs between the north and the south boundaries. Other equivalent logical operators are formed by multiplying a line by associated Z (X) stabilizers. (b) Circuits for each stabilizer colored in figure 1(a). The '0' on the top of the figure is step number. The top (bottom) half is an X (Z) stabilizer. Each face and vertex of the lattice in figure 1(a) has the same stabilizer circuit. The only gates required for stabilizer operation are INIT in Z basis, CNOT, SWAP, and H gates, and MEASUREMENT in Z basis. Boxes containing 0 are the INIT gates.

Table 1. Stabilizer representation of the stabilizers in figure 1(b). The upper line is a Z stabilizer and the lower is an X stabilizer.

q6	q10	q12	q16	q18	q22
		Z	Z	Z	Z
X	X	X	X		

stabilizers. We refer to these normal stabilizers involving four data qubits and one syndrome qubit as 'unit stabilizers'.

The surface code corrects errors in each unit and the code space is protected as a whole. Figure 1(a) shows the layout of normal unit stabilizers of a planar code, which is the form of the surface code we employ for our simulation [21]. The black lines in the figure are just a visual guide demarking plaquettes; each syndrome qubit is actually physically coupled to four neighbors. Figure 1(b) and table 1 show the stabilizer representation and the circuits of the stabilizers marked in figure 1(a), respectively. The stabilizers measure the parity of the data qubits involved. Normally, the parity is even (+1 eigenstate). When the states of an odd number of qubits that belong to the stabilizer are flipped, the parity becomes odd (−1 eigenstate).

The planar code defines a logical operator, using the degree of freedom introduced at a set of lattice boundaries. A lattice boundary is a terminal of a logical operator; hence a pair of boundaries introduces a logical operator and two pairs of different boundaries can generate a set of a logical X operator and a logical Z operator so that a single logical qubit is introduced. Any path between a pair of boundaries defines the same logical operator. The planar code performs logical two-qubit operations by transversal operations or lattice surgery [33]. To measure a logical qubit, take the parity of the measurement results on the physical qubits composing a logical operator. Parities of measurements on any path should have the same value, so that the logical measurement has redundancy against measurement failure as shown in figure 1(a).

A change in the error syndrome of a stabilizer indicates that the stabilizer is the termination of an error chain. Therefore, we execute minimum weight perfect matching to find a likely error pattern that results in the observed error syndromes. See appendix A for details of the error correction of the surface code. The distance between the two boundaries for an operator is the code distance of the surface code, as shown with the blue line on the right and with the red line on the top in figure 1(a). The larger the code distance, the higher the tolerance against errors. If the two boundaries were farther apart, more errors would be required to cause the error correction to fail.

A *nest* is created and used as a network for minimum weight perfect matching. Each vertex of the nest corresponds to a stabilizer value and each edge corresponds to a possible error chain. Details are in appendix B.4.

Table 2. Stabilizers of a set of modified plaquettes and a set of modified stars. (a) Superplaquette and superstar are adopted both to the Z stabilizer and to the X stabilizer (figure 3(a)). (b) Two triangular Z stabilizers are used along with a single superstar X stabilizer (figure 3(b)).

(a)							
1	2	3	4	5	6	7	8
Z	Z	Z	Z	Z	Z		
	X	X	X	X		X	X
(b)							
1	2	3	4	5	6	7	8
Z	Z	Z					
			Z	Z	Z		
	X	X	X	X		X	X

Table 3. Stabilizers of four triangular stabilizers as in figure 4. Some pairs anti-commute.

1	2	3	4	5	6	7	8
Z	Z	Z					
			Z	Z	Z		
	X		X			X	
		X		X			X

3. The structure of the surface code on a defective lattice

The difficulty in working around faulty devices arises from the nearest neighbor architecture and the two separate roles for qubits. Distant qubits have to interact around faulty devices but the nearest neighbor architecture does not provide the capability for such qubits to interact directly. SWAP gates are brought in to solve this problem. The solutions for faulty syndrome qubits and for faulty data qubits differ. To tolerate faulty data qubits, we use the 'superplaquette' after Stace *et al* [1]. The idea is to maintain error correction by modifying the shape of stabilizers around lost data qubits (faulty devices). On the other hand, we do not have to modify the unit of stabilizers when syndrome qubits are faulty. We can gather error syndromes onto another syndrome qubit instead of the faulty syndrome qubit by using SWAP gates. Stabilizers which do not involve faulty devices remain as normal stabilizers.

3.1. Stabilizer reconfiguration

There are two ways to reconfigure around a faulty device. The first is to introduce two triangular stabilizers by purging the broken qubit from stabilizers that involve it, leaving two stabilizers composed of three data qubits and one syndrome qubit, as depicted in figure 2(a). It is impossible to adopt triangular stabilizers for both stabilizers around a faulty device since neighboring Z triangular stabilizers and X triangular stabilizers do not commute when they have only one qubit in common, as shown in figure 4. Note that those four triangles cannot be stabilizers but can be gauge operators for the subsystem code [34, 35].

The second approach is to generate a superplaquette by merging the two broken plaquettes depicted in figure 2(b). At the least, either the plaquette or star must adopt a superplaquette or a superstar. In this paper, we form superplaquettes and superstars for both stabilizers after Stace *et al* and Barrett *et al* [1, 24]. Forming superplaquettes and superstars for both stabilizers around a faulty device produces a degree of freedom which results in a logical qubit by code deformation [36], which is also called a 'junk qubit' [1]. However, operations specifically targeted at this qubit are required to execute two qubit gates between this new logical qubit and the logical qubits of the planar code and defect-based code, so that the effect of its presence in this paper is just reducing the minimum distance between the boundaries.

3.2. Stabilizer circuits around faulty devices

Stabilizer-measurement circuits working around faulty devices have different shape and depth from the circuits of normal stabilizers. Figure 5(b) shows the shape of a superplaquette in which two plaquettes are connected by a faulty device and its circuit. We call a circuit for an individual stabilizer a 'stabilizer circuit' and the circuit for a

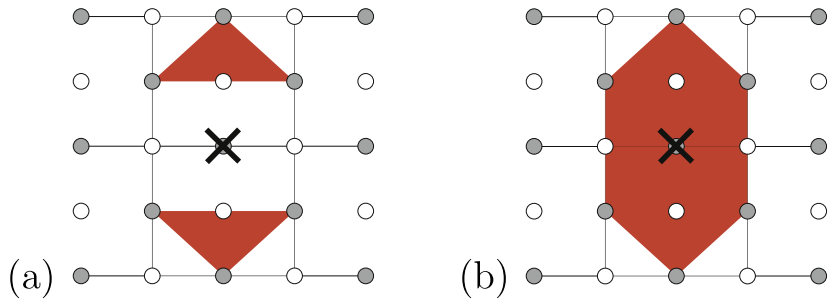

Figure 2. Modified stabilizers around a faulty device marked with the black cross. (a) A pair of Z triangular stabilizers. (b) A superplaquette.

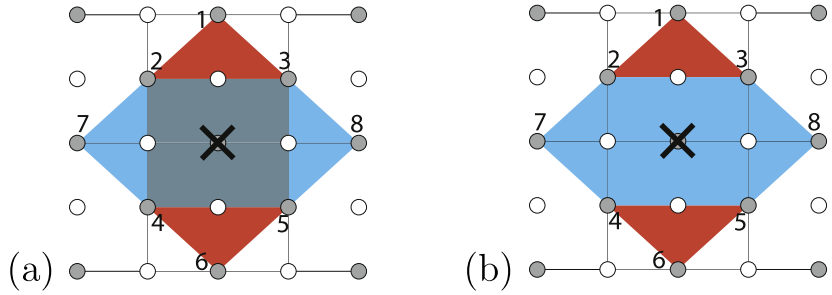

Figure 3. Two sets of modified stabilizers that commute. The corresponding stabilizers are shown in table 2. (a) Superplaquette and superstar are adopted both for the Z stabilizer and the X stabilizer. (b) Triangular stabilizer is adopted for Z stabilizer and superstar is adopted for the X stabilizer.

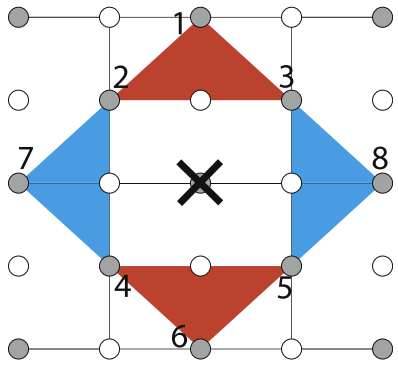

Figure 4. A set of modified stabilizers which anti-commute. The corresponding stabilizers are shown in table 3.

complete lattice the 'whole circuit'. We define two terms, 'qubit device' and 'qubit variable'. A qubit device is the physical structure that holds the qubit variable, such as the semiconductor quantum dot or loop of superconducting wire. A qubit variable is the information encoded on a qubit device. This distinction corresponds to the difference between a register or memory location in a classical computer and the program variable held in that location. In figure 5(a), the horizontal lines correspond to qubit devices, distinguished with the 'd' labels (numbers). The 'v' labels of qubit variables share the same number as the label of the qubit device of the qubit variable's original position.

In figure 5(b), the qubit device labeled d40 is faulty, and hence the variable v40 does not exist. The data qubit variables v58, v48, v50, v32, v22, and v30, initially held respectively in the qubit devices d58, d48, d50, d32, d22, and d30, are stabilized by the red stabilizer. The syndrome qubit variable v49 is initialized while residing in d49, and then moves around using SWAP gates to gather error syndromes of those data qubits. After gathering three error syndromes from v58, v48, and v50, v49 moves into d41 via d50. The data qubit variable v50 is moved onto d49 by the first SWAP gate between d49 and d50. After moving v49 from d50 to d41, the data qubit v50 on d49 is moved back to d50 by the second SWAP gate. v41, now in d49, is disentangled from other qubits, and hence we

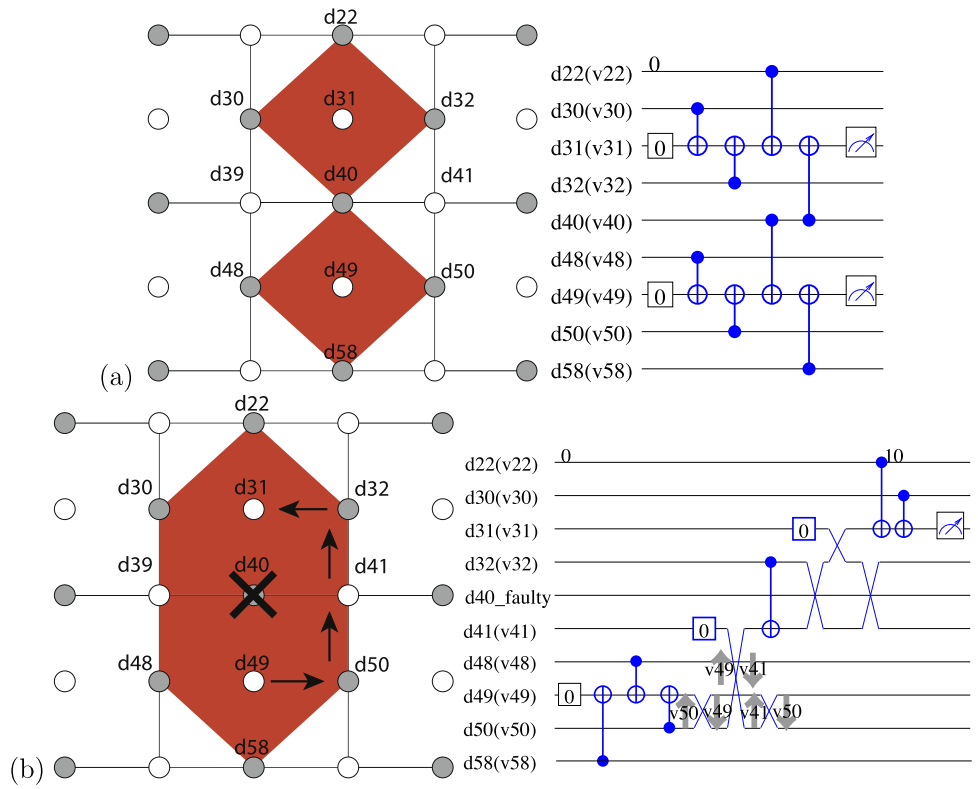

Figure 5. Stabilizers and their circuits. (a) A set of normal Z stabilizer circuits around d40 for the case where d40 is properly functional. The stabilizers are $Z_{v58}Z_{v48}Z_{v50}Z_{v40}$ and $Z_{v40}Z_{v32}Z_{v22}Z_{v30}$. (b) An example of a superplaquette stabilizer circuit. The qubit device d40 is faulty and two plaquettes are connected. The new stabilizer is $Z_{v58}Z_{v48}Z_{v50}Z_{v32}Z_{v22}Z_{v30}$. The circuit measuring only this stabilizer is shown on the right.

can initialize d49 any time. v49 eventually moves to d31, finishes gathering all error syndromes, and gets measured. Figure 5(b) summarizes the move of v49 from d49 to d31.

To find the optimized stabilizer circuit, first we search the smallest set of syndrome qubit devices which are neighbored to all the data qubits of the stabilizer. By moving a syndrome qubit variable around the syndrome qubit devices in the set, all data qubits are neighbored to the syndrome qubit variable during the move and error syndromes can be collected onto the syndrome qubit variable with nearest-neighbor interaction. For optimization, we solve a traveling salesman problem to find the shortest move of the syndrome qubit variable of the stabilizer among the syndrome qubit devices. If there are several smallest sets of syndrome qubit devices, we compare the moves of each set and adopt the set with the shortest move. This procedure gives the optimized circuit for single stabilizer reconfiguration.

In figure 5, we see that the superplaquette circuit is deeper than the normal plaquette circuit. In general, superplaquettes require more steps to gather error syndromes than normal stabilizers. Obviously, the deeper stabilizer will have more opportunities to accumulate physical errors. Thus, an important engineering goal is to create stabilizer circuits that are as shallow as possible.

We present a basic algorithm for composing a stabilizer circuit, shown in algorithm 1 in appendix B.2. A syndrome qubit variable travels one way to gather error syndromes. In this algorithm, we search for the shortest traversable path in which error syndromes can be gathered from all data qubits.

In our scheme, a single syndrome qubit is used to collect error syndromes from all data qubits in a stabilizer. Hence, if a Z (X) error occurs on the syndrome qubit in Z (X) stabilizer, the error propagates to data qubits whose error syndromes are collected after the error occurrence. This error propagation is correctly incorporated in our model.

3.3. Building a whole circuit from stabilizer circuits

On a perfect lattice, the stabilizer circuits are highly synchronous and easily scheduled efficiently. The circuits for a defective lattice must be asynchronous on account of the different depth of stabilizers. Such asynchronicity introduces a problem when several stabilizers try to access a qubit at the same time. We have to assign priorities to stabilizers. Stabilizers with lower priority have to wait, so that they have more opportunities to accumulate physical errors on data qubits and ancilla qubits. Therefore we give higher priority to stabilizers which have

deeper stabilizer circuits to deter error opportunities from concentrating there, since a shorter error chain is obviously preferred for error correction. The scheduling algorithm, shown in algorithm 2 in appendix B.2, is

(i) Sort stabilizers in order of depth, longest first. If they tie, stabilizers in the upper left of the lattice have priority. (lines 1–2)

(ii) The deepest stabilizer is scheduled. The step when the deepest stabilizer finishes is saved (currentCeil). (line 9)

(iii) Each non-deepest stabilizer is scheduled once, in order of decreasing depth. (lines 10–13)

(iv) Each non-deepest stabilizer which does not exceed the currentCeil is scheduled once again, in order of depth. Short ones may be scheduled twice or more before the loop terminates. (lines 14–21)

(v) If all of the non-deepest stabilizers exceed the currentCeil, return to step (ii). Otherwise, return to step (iv). (lines 21–22)

Our algorithm must enforce an important restriction, which the completely synchronous circuits of the perfect lattice fulfill without explicitly being stated. Different types of stabilizers which share an even number of data qubits must access those qubits in the same order. For example, if we have two stabilizers X_1X_2 and Z_1Z_2 on qubits 1 and 2, we have to execute them as X_1X_2 then Z_1Z_2 (or reverse order). X_1Z_2 then Z_1X_2 is not allowed because it will entangle syndrome qubits. We postpone stabilizers of low priority to resolve conflicts by simply adding identity gates. See appendix B.3 for details. Optimization around this strategy is left for later research.

3.4. Adapting matching to asynchronous operation

Irregular stabilizer circuits degrade the parallelism of stabilizer measurements of the whole circuit, so that the surface code on a defective lattice has irregular error matching nests. A superplaquette is measured in a longer cycle than normal stabilizers and a vertex corresponding to a superplaquette has many edges. Additionally, the weights of edges are generated by tracing propagation, along the stabilizer circuits, of virtually created errors on every qubit at every physical step. Therefore the weights of edges reflect the possibilities of errors accurately, allowing Blossom V to achieve a result likely to match the actual physical error set.

4. Numerical results

In our simulations,

- static loss placement is accurately determined by scanning before quantum computation;
- static loss does not occur after fabrication; and
- stabilizer circuits are created before quantum computation.

We assume a circuit-based error-occurrence model, including imperfect syndrome extraction, summarized by Landahl *et al* [37]. This circuit-based error model assumes that each gate acts ideally, followed by a noisy channel, and that each measurement acts ideally after a noisy channel. Errors may occur at every gate in the circuit. Our error channel for a single-qubit gate has error probability p, meaning that each error (X, Z, or Y) occurs with probability $p/3$. In a similar fashion, for two-qubit gates, our error model has probability $p/15$ for each two-qubit error (IX, IZ, IY, XI, XX, XZ, XY, ZI, ZX, ZZ, ZY, YI, YX, YZ, YY). We assume that the set of physical gates available includes CNOT, SWAP, and Hadamard gates. We assume that INIT and measurement in Z basis have X error probability p. All operations require one time step.

Our circuit is asynchronous in the sense that stabilizers are measured at different frequencies. Stabilizers whose circuits have shallower depth may be measured more times than those whose circuits have deeper depth. To achieve proper syndrome matching, the surface code requires that the lattice be covered by stabilizers. Otherwise, an unstabilized area works as a defect-based qubit which may serve as an end of error chains, leading to undetectable logical errors. Hence, after all stabilizers covering the lattice have been measured at least once since the last execution of the matching algorithm, the matching algorithm is re-executed. Typically, this timing is dependent on the deepest stabilizer circuit. From the result of matching, we make a map of Pauli frames which describes where Pauli frame corrections should be applied for error correction. Because our circuit is asynchronous and there might be SWAP gates, we must keep track of the location of qubit variables to combine the error information about data qubits and the map to check the result of error correction.

We have conducted extensive simulations, beginning with a perfect lattice and then extending to imperfect ones. First, we show the numerical results of several basic test cases: only a single faulty device exists, in the center of the lattice; only a single faulty device exists, in the west of the lattice; and only a single faulty device exists, in the northwest of the lattice, all for the distances 5, 7, 9, and 13. Our simulation holds d temporal rounds of measured stabilizer values for error correction. Hence, d measurements are saved for the stabilizer with the deepest circuit and more measurements are saved for normal stabilizers because of the scheduling algorithm shown in subsection 3.3. After finishing an error-correction cycle, the oldest round is discarded, a new round is created by new measurements, and error correction is re-executed. Next, we show the numerical results for randomly generated lattices for three different yields: 80%, 90%, and 95%. We generated 30 lattices for each pair of yield and code distance of 5, 7, 9, 13, 17, and 21. Some defective lattices cannot encode a logical qubit, so the code distance becomes 0 as a result of merging stabilizers, so that ultimately we simulated 474 randomly generated lattices (details described in section 4.3) for physical error rates of 0.1%, 0.2%, 0.3%, 0.4%, 0.5%, 0.6%, 0.7%, 0.8%, 0.9%, 1%, and 2%. It is hard to collect enough logical errors in Monte Carlo simulation as the logical error rate is exponentially suppressed, and therefore we choose 0.1% as the lowest physical error rate for our simulation. Thus, we simulated 5214 parameter combinations.

The computational resource devoted to circuit simulation, excluding chip generations and circuit constructions, was more than 100,000 CPU days, executed on the StarBED project testbed [38]. Each preparation of stabilizer circuits which solves the traveling salesman problem required up to 1 CPU day. After construction of the nest, for example, the simulation of $d = 5$ of single-faulty-northwest for $p = 10^{-3}$ consisted of 370945 rounds of error correction to find 500 logical X errors in 1424.98 seconds. The simulation of $d = 13$ of single-faulty-northwest for $p = 10^{-3}$ consisted of 315550 rounds of error correction in 5.8 days but found 0 logical X errors.

Peak memory sizes are estimated to be 30 GB for 316 lattices, 63 GB for 133 lattices, and more than 100 GB for 25 lattices. The greatest memory consumption is during nest building, shown in figures B5, B6, and B7. To give accurate weights to the edges of the 'nests', Autotune virtually creates errors on every qubit at every physical step and traces their propagation. Roughly speaking, the size of the error structure is 136 bytes. A lattice includes 1089 qubits for distance 17. Let us assume 200 physical steps per error correction cycle due to the asynchronous stabilizers; each error propagates to 10 physical qubits on average; each error remains for 100 physical steps on average. Then memory consumption is $136 * 1089 * 200 * 10 * 100 = 29620800000$ bytes, roughly 30 GB. Several factors affect this rough estimate. Faulty devices reduce the number of qubits and other structures generated to create the nests, but the memory consumption remains on the order of tens of gigabytes.

These peak memory sizes are big; however, they do not affect the quantum computation in practice. This is because the heavy operations whereby Autotune virtually creates errors on every qubit at every physical step, and traces their propagation by the circuits to give accurate weights to the edges of the nests, can be executed preliminarily. To avoid redundant execution of heavy creation of nests, all 11 physical gate error rates for a single lattice are simulated in parallel on a single simulation node, allowing us to share a single in-memory copy of the nest. We attempted to simulate distance 21 but failed because we cannot accumulate enough logical errors to have valid data points, for one of several reasons: good lattices have strong tolerance against errors; even bad lattices have strong error tolerance at lower physical error rate; at higher physical error rates, simulating an error correction cycle takes too much computation time because many physical errors occur in our extended asynchronous error-correction cycle, taxing the scalability of the matching algorithm; or because the simulation requires more than 128 GB memory, the maximum available in our system.

4.1. Perfect lattice

Figure 6 depicts the results of simulation of perfect lattices, used as our baseline for comparison. Each curve represents a set of simulations for a lattice of a particular code distance for varying physical gate error rates. Points below the break-even line are conditions in which the logical error rate in the logical state is below that of a bare, unencoded physical qubit for a single physical gate time. Distance 9 achieves break-even at $p = 0.3\%$. The crossing point of the curves, each of which describes a code distance, is called the *threshold*, the physical error rate below which the larger code distance has the lower logical error rate. Above the threshold, the error-correction process introduces more errors than it corrects, and the higher code distance has the higher logical error rate.

The threshold indicated by this simulation is around 0.58%, similar to the 0.60% reported in [19]. This related work employs the assumptions most similar to our perfect lattice simulation, other than the asynchronous scheduling of stabilizers. Our error-correction circuits are designed to omit identity gates to shrink the asynchronous circuit depth, whereas circuits of related work achieve perfect synchronization and parallelism through careful insertion of identity gates. For example, identity gates on the qubit d17 in figure 1(b) between the initialization and the CNOT gates or between the CNOT gates and the measurement are omitted in

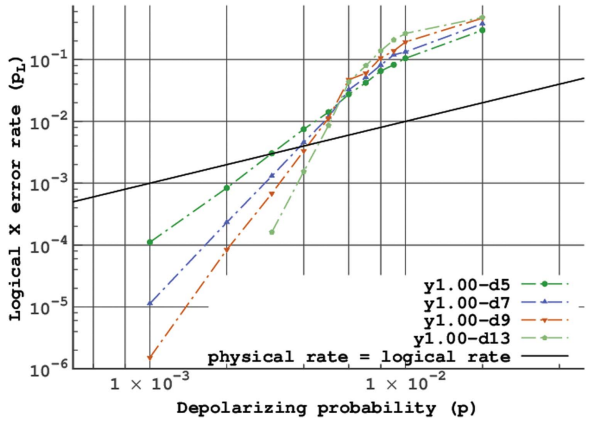

Figure 6. Results of baseline simulations of perfect lattices of code distance 5, 7, 9, 13, and 17. The average number of steps per error-correction cycle for every code distance is 8.1, 8.0, 8.0, 8.0, and 8.0, respectively. The black line is the break-even line. The threshold seems to be around 0.58%. Each data point has 50 $\sim$ 1500 logical errors. The irregularity of the point at $p = 0.6\%$ of distance 9 may come from statistical variance.

our simulation. We infer that our baseline simulation follows the related work and is valid, and the effect of asynchronicity on the perfect lattice is small.

4.2. Lattice with a single faulty device

Figures 7(a), (b), and (c) depict the results of simulations to investigate the effect of a single faulty device in the center, on the west edge, and on the northwest corner of the lattice, respectively. The plots show that our approach works properly because the larger code distance has the lower logical error rate at lower physical error rates.

Each single-fault residual error rate is worse than that of the corresponding perfect lattice. The slope of each code distance of single-fault chips is lower than that of the corresponding perfect lattice. The gap grows slightly as the physical error rate is reduced, visible as the less-steep curve for the defective lattice.

There are differences depending on the single-fault location. Comparing the points $d = 9$ of the perfect lattice with those of single-faulty-center, single-faulty-west, and single-faulty-northwest at $p = 0.1\%$, faulty lattices are 10.9×, 7.60× and 7.20× worse than the perfect lattice, respectively. Single-faulty-northwest has a lower residual error rate than the others. This may be because the big stabilizer that causes asynchronous scheduling of stabilizers is on the periphery, so that the number of stabilizers that are close to the big stabilizer and hence have stronger scheduling restrictions than more remote stabilizers is smaller than other single-faulty chips. Across the range of our simulations, the negative impact is $6 \times \sim 11\times$ depending on location, distance, and error rate.

From the point of view of absolute logical error rate, the penalty for having a defect is greater at lower physical error rates. An 'effective' code distance is the code distance at the same physical error rate of the perfect lattice which has the closest logical error rate to the defective lattices. For single-faulty-center, at $p = 0.3\%$, faulty $d = 9$ is 1.5× worse than perfect $d = 5$, and hence the effective code distance of faulty $d = 9$ at $p = 0.3\%$ is $\approx$5. The effective code distance is useful when considering the resource overhead of modifications. In the preceding example, to achieve a logical error rate equivalent to that of $d = 5$ on the perfect lattice at $p = 0.3\%$, we at least need $d = 9$ for the defective lattice. This indicates that 3.5× the number of physical qubits are required.

From the point of view of the effective code distance, the penalty for having a defect is smaller at lower physical error rates. At $p = 0.3\%$, faulty $d = 9$ is 1.5× worse than perfect $d = 5$, and at $p = 0.1\%$, faulty $d = 9$ is 14.9× better than perfect $d = 5$ while faulty $d = 7$ is 1.4× worse than perfect $d = 5$. Hence, to exceed the effective code distance 5, $p = 0.3\%$ requires us to use $d = 11$ while $p = 0.1\%$ only requires us to use $d = 9$. The trend of the penalty of the effective code distance and that of the absolute logical error rate differ. This difference is caused by the difference of slopes of each code distance of each lattice. We have to be mindful of those trends when designing a quantum computer to achieve an adequate logical error rate.

Because the proportional impact of a single fault should lessen as the code distance increases, the crossing point of the curves is not a good measure of performance here. Figure 7 shows that the crossing points of two distances would differ. The crossing point of distances 9 and 13 appears to be around 0.6%, which is the threshold for the perfect lattice as shown in figure 6, whereas the crossing point of distances 5 and 7 is around 0.8%.

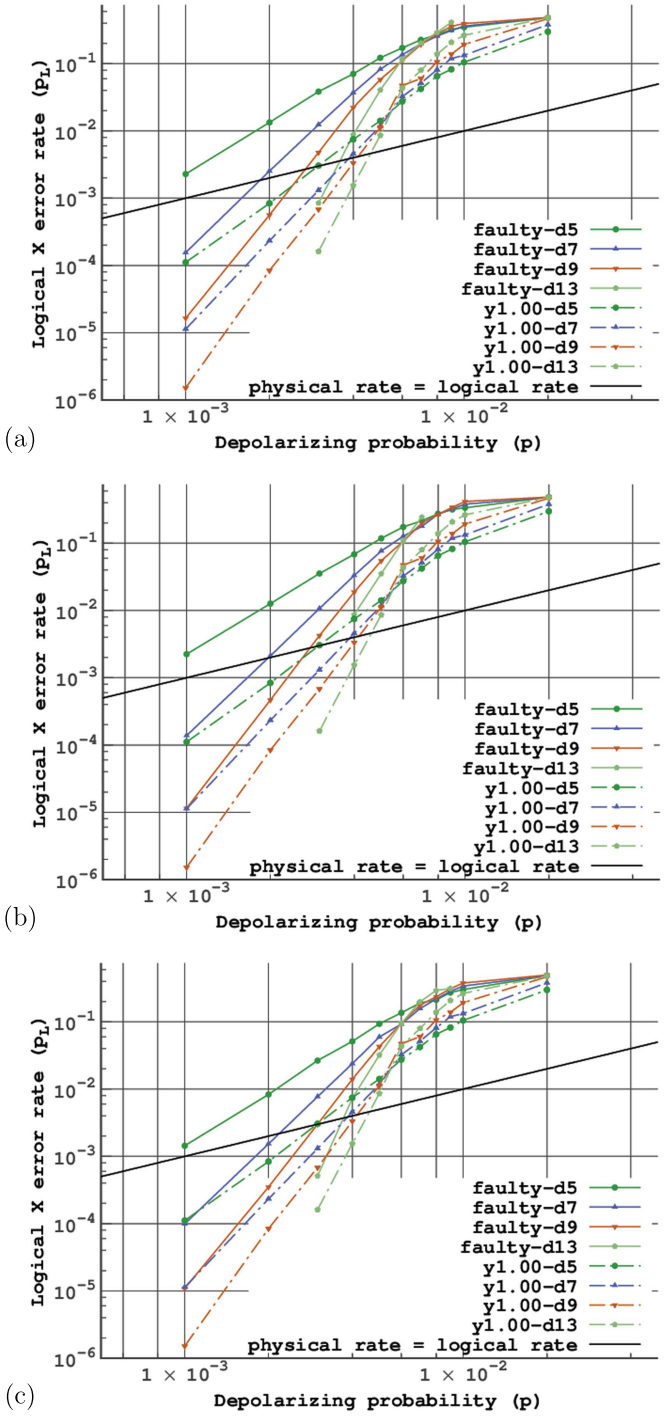

Figure 7. Results of simulations of defective lattices that have a single faulty device (a) in the center of the lattice, (b) in the west of the lattice, and (c) in the northwest of the lattice, respectively. Dashed lines are of the perfect lattices for reference. The code distances are 5, 7, 9, and 13. The average number of steps per error correction cycle is 32.5 for every code distance and fault location.

Table 4 shows the data of the single-faulty lattice simulations. The reduced code distance is the minimum distance between corresponding boundaries shortened by merging stabilizers. The naive hypothesis would be that reduced code distance is a good metric to predict the logical error rate of the lattice since the number of physical errors required to cause a logical error is a minimum on the shortest logical operator, which is the minimum distance between corresponding boundaries. However, the effect is more complex. We will explore this further in sections 4.4 and 5.

4.3. Random multiple faulty devices

We generated 30 randomly defective lattices for each combination of three yields, 80%, 90%, and 95%, and five code distances, 5, 7, 9, 13, 17, and 21, so that we generated 540 lattices. 66 lattices cannot hold a logical qubit and

Table 4. The *X* error rate of single-faulty lattices and corresponding *Z* stabilizer data. Averages here are arithmetic means. 'Faulty location' is the location of the faulty device (static loss). '#X and Z stabs' stands for the total number of *X* stabilizers and *Z* stabilizers. 'Reduced distance' is the minimum distance between corresponding boundaries shortened by merging stabilizers. '#Z stabs' is the number of *Z* stabilizers. 'Biggest #dataq of Z stabs' is the largest number of data qubits in a *Z* stabilizer. 'Ave. *CDQ* of Z stabs' is the average of *CDQ*s (metric is the space-time product of an error-correction circuit: the number of data qubits *DQ* involved, multiplied by the 'cycle', the sum of the circuit depth, and the waiting time for next stabilization *C*, after [39]) of *Z* stabilizers. 'Biggest Z *CDQ*' is the largest *CDQ* for any stabilizer circuit of the chip. 'Ave. #dataq of Z stabs' is the average of the number of data qubits in Z stabilizers.

Faulty Location	Code Distance	#X and Z stabs	Residual X Error Rate	Reduced Distance	#Z Stabs	Biggest #dataq of Z stabs	Steps per Error Correction Cycle	Ave. *CDQ* of Z stabs	Biggest Z *CDQ*	Ave. Steps per Error Correction of Z stabs	Ave. #dataq of Z stabs
center	5	38	7.038E-02	4	19	6	32	35.871	195.194	8.844	3.684
west	5	38	6.843E-02	4	19	6	32	35.803	195.194	8.822	3.684
northwest	5	38	3.335E-02	4	19	6	32	35.648	195.194	8.791	3.684
center	7	82	3.704E-02	6	41	6	32	32.249	195.194	8.149	3.756
west	7	82	3.319E-02	6	41	6	32	32.270	195.194	8.153	3.756
northwest	7	82	1.969E-02	6	41	6	32	32.505	195.194	8.228	3.756
center	9	142	2.224E-02	8	71	6	32	31.215	195.194	7.943	3.803
west	9	142	1.873E-02	8	71	6	32	31.456	195.194	8.016	3.803
northwest	9	142	1.046E-02	8	71	6	32	31.539	195.194	8.033	3.803
center	13	310	8.776E-03	12	155	6	32	30.711	195.194	7.824	3.858
west	13	310	8.662E-03	12	155	6	32	31.028	195.194	7.910	3.858
northwest	13	310	4.869E-03	12	155	6	32	31.084	195.194	7.925	3.858

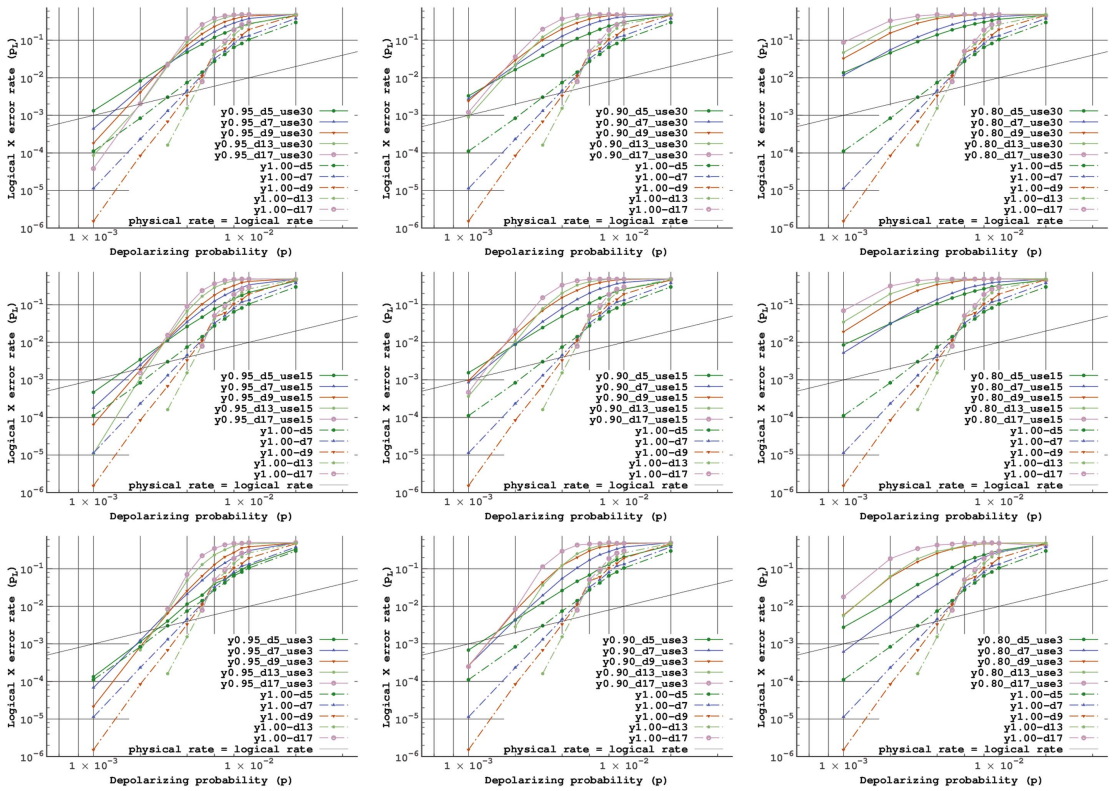

Figure 8. Graphs of randomly defective lattices. Dashed lines are of the perfect lattices for reference. The left column is of $y = 95\%$, the middle column is of $y = 90\%$, and the right column is of $y = 80\%$. Top row is of all generated lattices, middle row is of culling the worst 50%, and bottom row is of culling the worst 90%.

Table 5. Number of defective lattices generated and simulated.

Yield	0.80					0.90					0.95				
code distance	5	7	9	13	17	5	7	9	13	17	5	7	9	13	17
#encodable	20	24	22	19	19	29	29	30	30	30	28	30	30	30	30
#unencodable	10	6	8	11	11	1	1	0	0	0	2	0	0	0	0

we were unable to simulate distance 21; hence, we ultimately simulated 474 lattices. Table 5 shows the number of defective lattices generated and simulated. On some defective lattices, by chance the faulty qubit placement results in a lattice for which we are unable to build an effective circuit for encoding a logical qubit, so they are not simulated. Our software successfully built circuits for almost all lattices at $y = 0.90$ and above, but only about two-thirds at $y = 0.80$.

This unencodable condition occurs when a defective data qubit chain stretches from a boundary of the lattice to the other boundary of the same type (south and north for Z stabilizer boundary, or west and east for X stabilizer boundary). For instance, if a faulty qubit is on a boundary—say, the *qubit*1 which is stabilized by Z_1 of the stabilizer $Z_1Z_2Z_3Z_4$—and the qubit is not stabilized by another Z stabilizer, then $Z_1Z_2Z_3Z_4$ cannot be merged with another stabilizer to work around Z_1. Hence, we remove $Z_1Z_2Z_3Z_4$ with *qubit*1 and eventually *qubit*2, *qubit*3, and *qubit*4 become a part of the boundary instead. In general, this adaptation reduces the code distance (shown in tables 7 and 8). Therefore, a lattice of lower yield and lower code distance has a higher probability of being unencodable. Though only 30 instances for each condition are too few to collect statistics of useful accuracy, table 5 shows this trend at yields of 90% and 95%. $Y = 80\%$ might be saturated in terms of the percentage of encodable lattices because the code distances do not show meaningful differences.

Figure 8 is the main result of this work, showing the geometric mean of all encodable lattices, plotting physical error rates versus logical error rates. Appendix C.2 shows the scatter plots of raw data.

The left column in figure 8 show the graphs of $y = 95\%$, describing the geometric mean of all encodable lattices (top row), the better 50% (middle row), and the best 10% (bottom row) of generated lattices. Note that those cull percentages are based on the original set of 30 generated lattices, not the smaller number of the

Table 6. The average number of faulty qubits in all generated lattices after culling the weakest 50% and weakest 90%, respectively. The numbers of qubits in a perfect lattice of distance 5, 7, 9, 13, and 17 are 81, 169, 289, 625, and 1089, respectively. The correlations between this number and the logical error rate is 0.25 to 0.68, as shown in tables 7 and 8—a strong correlation but not as strong as the best metrics in tables 7 and 8.

Yield	Code Distance	All	50%	90%
	5	13.9	13.2	12.3
	7	29.7	28.9	29
0.80	9	54	55.1	55
	13	122.6	122.6	121.3
	17	215.6	214.5	221.7
	5	8.4	7.3	5.7
	7	16.2	14.2	9.3
0.90	9	29.6	26.7	23.7
	13	61	56.3	48
	17	107.3	101.7	96
	5	3.8	2.9	1.7
	7	8	7.1	4.7
0.95	9	14.7	11.9	9
	13	31.3	28.5	25.3
	17	53.7	51.3	51.3

encodable lattices. Some points of longer distance at lower physical error rate are not plotted since not enough logical errors are accumulated because of the very low logical error rates.

At 95% functional qubit yield, we see many chips beating break-even at $p = 10^{-3}$. The threshold is about 0.3%, about half of the threshold error rate for a perfect lattice. The significant penalty in both threshold and residual error rate can be dramatically reduced by culling poorer chips and discarding them. At 50% cull at $p = 10^{-3}$, the residual error rate for $d = 7$ is about that of $d = 5$ with a perfect lattice, and $d = 13$ is about that of a perfect $d = 7$.

Naturally, the logical error rates get better as we discard more of the poorest lattices. At $p = 0.2\%$, $y = 95\%$ in unculled $y = 95\%$ shows that even distance 17 is just on the break-even line and 90% culled $y = 95\%$ shows that all five distances exceed break-even. The steepness of the slope of the curves of culled defective lattices exceeds that of the curves of lower code distances on the perfect lattice, though it does not match the perfect lattice of the same distance. Thus, an appropriate culling strategy reduces the penalty for a 5% fault rate to a manageable level, allowing us to achieve a desired level of error suppression by using a slightly larger code distance. At $p = 0.1\%$, by culling 90%, the penalty against the perfect lattices changes from 12.0× to 1.2× at $d = 5$, from 39.0× to 6.1× at $d = 7$, and from 119.9× to 14.2× at $d = 9$. We do not have data points for 90% discarding at $d = 13$ and $d = 17$ since not enough logical errors are accumulated on the best 10% of these lattices. The smaller code distance gets closer to the perfect lattice because it has fewer qubits, and therefore good outliers may be generated with higher probability, as shown in table 6. Table 6 summarizes the average number of static losses on all the generated lattices, on the 50%-cull lattices and on the 90%-cull lattices. The remaining three lattices of 90%-cull distance 5 have 1, 2, and 2 static losses, respectively. Table 6 also allows us to see the importance of static loss placement because the number of static losses does not decrease much but all the logical error rates improve.

The middle column in figure 8 is the graphs of $y = 90\%$, under the same conditions with those of $y = 95\%$. The threshold is about 0.15%, about a quarter of the threshold for a perfect lattice. At 90% cull (in the middle-bottom of figure 8) at $p = 10^{-3}$, the residual error rates for $d = 7$, $d = 9$, and $d = 17$ are about twice those of $d = 5$ with a perfect lattice. $d = 13$ would be better than that of perfect $d = 5$, but it is missing since the logical error rate may be too low to accumulate enough logical errors. At $p = 0.1\%$ of $y = 90\%$, unculled (middle-top) shows that only distance 13 exceeds the break-even, but 90%-cull (middle-bottom) shows all five distances exceeding the break-even.

The right column in figure 8 is the graphs of $y = 80\%$. At $y = 80\%$, we have already seen that only two-thirds of the chips can even be encoded. Our simulations indicate that even those chips for which we could create a circuit are unusable. Even at $p = 10^{-3}$, there is no evidence of a correctable threshold, and although residual error rates do decline as the physical error rate is reduced, only a single data point reaches break-even. We conclude that $y = 80\%$ is not good enough to build a computer.

Unculled $y = 95\%$ shows that distances 13 and 17 are approximately identical at $p = 0.2\%$, while other distances show that longer is better. Unculled distances 13 and 17 for $y = 90\%$ do not show that longer is better, though other distances do. Unculled $y = 80\%$ shows that distance 7 exceeds distance 5 at $p = 0.1\%$, while other distances do not show an improvement for the longer distance. Those indicate that the longer code distances cross at lower physical error rate. We need to consider this fact when deciding the code distance to use.

4.4. Metrics for selecting good chips

Both to improve our understanding of the root causes of the error rate penalty and to provide a simple means of selecting good chips, we evaluated the correlation between a set of easy-to-calculate metrics and the simulated residual error rate. Tables 7 and 8 describe the correlations between 18 metrics and logical error rates or log (logical error rates), respectively, for $p = 0.002$ for each combination of yield and code distance.

The simplest possible metrics, just counting numbers of qubits in various categories, show only modest correlation.

Steane's KQ metric is the space-time product of a circuit: the number of qubits Q involved, multiplied by the circuit depth K [39].

The CDQ and CQ are the product of the 'cycle', which is the average number of steps in a stabilizer measurement and the number of data qubits or the total number of qubits including ancillae involved in the stabilizer, respectively. The CDQ and CQ reflect the total probabilities of possible physical errors which occur in a measurement of the stabilizer. Both tables 7 and 8 indicate that the average of the CDQ and the average of the CQ of Z stabilizers have the strongest and second strongest correlations with the logical X error rate. The average number of qubits in a Z stabilizer and the average 'cycle' of Z stabilizers show the next strongest correlations. Those mean that the accumulation of possible errors in a stabilizer may be the factor most strongly correlated to the logical error rate.

Somewhat to our surprise, both the KQ of the largest stabilizer and the average across the entire lattice do not have good correlations. This may be because this form of KQ does not correctly capture the total probabilities of possible physical errors which occur in a measurement of the stabilizer.

Table 6 implies that the number of faulty devices is correlated with the logical error rate. By culling bad lattices, table 6 shows that the average number of faulty devices on a lattice is reduced and figure 8 shows that the logical error rate gets better. However, the average CDQ of Z stabilizers has significantly higher correlation with logical X error rate, 0.76, than that of the number of faulty devices, 0.43. We calculated the cross-correlation of elements for $y = 0.95$ and $d = 9$. The correlation between the number of faulty devices and the average CDQ of Z stabilizers is 0.79.

The number of faulty ancilla qubits is the most weakly correlated to the logical error rate. This fact indicates that even if the number of faulty ancilla qubits increases, the logical error rate does not decline rapidly. For a given yield, the placement of faults matters more than the exact number.

5. Discussion

We have proposed and analyzed an adaptation of the surface code for static losses, which are manifested as faulty devices on quantum computation chips occurring during fabrication. With this fundamental analysis of static loss and its influence, independent analysis has now been conducted for the three major imperfections of quantum computation for the surface code: state error, dynamic loss, and static loss. The ultimate goal of investigating faulty devices is to support collection of a large pool of sufficiently fault-tolerant quantum computation chips, because a realistic large-scale quantum computer must be assembled from many quantum computation chips, coupled by their proximity or via distributed quantum computation [10]. We analyzed our approach against faulty losses by simulation to investigate the relationship between the logical error rates and lattice characteristics of simulated defective lattices. Our approach is to merge stabilizers broken by faulty data qubits to a superplaquette and to work around faulty ancilla qubits using SWAP gates, without changing the original role of the qubits.

Our simulation with a single faulty device revealed that faulty qubits at the periphery reduce the logical error rate less than those in the center. Even a single fault has a large impact on the residual error rate.

Our simulation with randomly placed faulty devices showed that at 95% yield, the impact on net error rate is significant but many of the chips still achieve break-even by $p = 10^{-3}$ and therefore could be used in a real-world setting. At 90% yield, very few chips achieve break-even. At 80% yield, almost no chips are usable. These facts establish the goals for experimental research to build the surface code quantum computer.

The simulation of randomly placed faulty devices also showed that discarding bad lattices makes the ensemble better, showing the trade-off between the cost of culling and the strength of fault tolerance of an ensemble. Culling increases our effective error suppression, such that $d_e^{culled} \approx d_e^{unculled} + 2$, where d_e is the effective code distance, for lattices with physical distance 9 and 13 at 95% yield. 90% yield shows error

Table 7. Linear Correlation between candidate metrics for lattice quality factors and *X* logical error rates for each combination of yield and code distance. Boldface is the strongest correlation in each row. 'Dist.', 'flty', 'bgst', 'syn.', 'stab', and 'msmt' stand for distance, faulty, biggest, syndrome, stabilizer, and measurement, respectively. Averages here are arithmetic means. 'Reduced dist.' is the minimum distance between corresponding boundaries shortened by merging stabilizers. '#Z stabs' is the number of *Z* stabilizers. 'Bgst #qubit of Z stabs' and 'ave. #qubit of Z stabs' are the biggest and the average number of qubits involving data qubits and syndrome qubits in *Z* stabilizer circuits. 'Bgst #data qubit of Z stabs' and 'ave. #data qubit of Z stabs' are the biggest and the average number of data qubits in a *Z* stabilizer. 'Dpst depth of Z stabs' is the depth of the deepest *Z* stabilizer circuit. 'Ave. depth of Z stabs' is the average depth of Z stabilizer circuits. 'Bgst *KQ* of Z stabs' and 'Ave. *KQ* of Z stabs' are the biggest and average *KQ* (metric is the space-time product of a circuit: the number of qubits *Q* involved, multiplied by the circuit depth *K* [39]) of *Z* stabilizer circuits. 'Bgst *KDQ* of Z stabs' and 'Ave. *KDQ* of Z stabs' are the biggest and average *KDQ* which is the product of the number of data qubits *DQ* involved and of the circuit depth *K* of *Z* stabilizer circuits. 'Cycle' here indicates every how many steps a stabilizer is measured, including waiting time by scheduling of stabilizers. 'Bgst Z cycle' and 'Ave. Z cycle' are the biggest and average cycle of *Z* stabilizers. 'CQ' is the product of the cycle *C* and the number of qubits *Q*, similar to *KQ*. 'Bgst *CQ*' and 'Ave. *CQ*' are the biggest and verage *CQ* of stabilizers. 'Bgst *CDQ*' and 'Ave. *CDQ*' are the biggest and average *CDQ*, which are the product of the number of data qubits *DQ* involved and of the cycle *C* of Z stabilizer circuits. 'Ave. #Z stabs per step' is how many stabilizers are measured in a step on average.

Yield	Code dist.	#stab	#flty qubit	#flty data qubit	#flty syn. qubit	*Z* reduced dist.	#*Z* stab	Bgst #qubit of *Z* stabs	Ave. #qubit of *Z* stabs	Bgst #data qubit of *Z* stabs	Ave. #data qubit of *Z* stabs	Dpst depth of *Z* stabs	Ave. depth of *Z* stabs	Bgst *KQ* of *Z* stabs	Ave. *KQ* of *Z* stabs	Bgst *KDQ* of *Z* stabs	Ave. *KDQ* of *Z* stabs	Bgst *Z* cycle	Ave. *Z* cycle	Bgst *CQ* of *Z* stabs	Ave. *CQ* of *Z* stabs	Bgst *CDQ* of *Z* stabs	Ave. *CDQ* of *Z* stabs	Ave. #*Z* stab msmts /step
0.95	5	−0.64	0.59	0.70	0.08	−0.67	−0.71	0.81	0.83	0.79	0.71	0.26	−0.01	0.24	0.37	0.23	0.16	0.69	0.71	0.74	**0.88**	0.76	0.86	−0.69
0.95	7	−0.40	0.45	0.65	−0.13	−0.62	−0.46	0.65	0.44	0.70	0.44	0.05	0.00	0.07	0.11	0.08	0.04	0.78	0.55	0.76	0.78	**0.79**	**0.79**	−0.44
0.95	9	−0.50	0.68	0.72	0.33	−0.60	−0.48	0.63	0.81	0.64	0.79	0.42	−0.07	0.46	0.48	0.40	0.24	0.62	0.81	0.67	0.88	0.68	**0.89**	−0.69
0.95	13	−0.28	0.48	0.55	0.11	−0.19	−0.27	0.36	0.56	0.37	0.52	−0.02	−0.47	0.02	−0.14	0.03	−0.25	0.30	0.59	0.33	**0.71**	0.34	0.69	−0.40
0.95	17	−0.18	0.34	0.27	0.25	−0.10	−0.18	0.20	0.39	0.15	0.40	0.05	−0.32	0.17	−0.06	0.05	−0.16	0.13	0.50	0.11	**0.56**	0.10	0.55	−0.40
0.90	5	−0.19	0.35	0.26	0.29	−0.48	−0.24	**0.62**	0.20	0.55	0.16	−0.23	−0.21	0.06	−0.13	−0.11	−0.16	0.49	0.28	0.69	0.48	**0.62**	0.46	−0.21
0.90	7	−0.31	0.37	0.43	0.07	−0.59	−0.33	0.50	0.53	0.49	0.45	0.20	−0.33	0.59	0.23	0.43	−0.01	0.51	0.58	0.71	0.71	0.70	**0.74**	−0.46
0.90	9	−0.39	0.40	0.54	0.09	−0.65	−0.42	0.67	0.65	0.63	0.58	−0.29	−0.54	0.01	−0.12	−0.18	−0.32	0.28	0.39	0.43	**0.68**	0.43	0.66	−0.45
0.90	13	−0.36	0.47	0.63	−0.01	−0.30	−0.37	0.47	0.77	0.52	**0.81**	−0.26	−0.25	−0.10	0.24	−0.28	0.04	0.46	0.57	0.60	**0.81**	0.62	**0.81**	−0.51
0.90	17	−0.55	0.58	0.62	0.10	−0.62	−0.56	0.56	0.76	0.56	0.74	−0.05	−0.25	0.34	0.33	0.10	0.08	0.17	0.65	0.35	0.86	0.37	**0.87**	−0.74
0.80	5	−0.20	0.30	0.22	0.17	−0.43	−0.22	**0.78**	0.59	0.46	0.40	−0.11	−0.13	0.43	0.30	−0.19	0.05	0.64	0.72	**0.78**	0.75	0.63	0.67	−0.49
0.80	7	0.22	0.35	0.10	0.40	−0.51	0.13	0.56	0.81	0.44	0.75	0.07	−0.34	0.29	0.58	0.03	0.24	0.56	0.74	0.64	0.79	0.69	**0.83**	−0.48
0.80	9	−0.39	0.37	0.31	0.15	−0.47	−0.38	0.63	0.88	0.68	0.68	−0.42	−0.45	0.41	0.58	0.12	−0.01	0.50	0.77	0.58	0.86	0.66	**0.92**	−0.53
0.80	13	0.02	0.34	0.18	0.20	−0.43	−0.05	0.65	0.79	0.66	0.80	0.21	0.09	0.45	0.68	0.22	0.52	0.70	0.63	0.65	0.79	0.70	**0.84**	−0.35
0.80	17	0.39	0.25	0.19	0.19	−0.24	0.34	0.37	0.60	0.29	0.57	−0.15	−0.41	0.28	0.37	0.04	0.07	0.51	0.55	0.34	0.61	0.33	**0.62**	0.16
average		−0.25	0.42	0.42	0.15	−0.46	−0.28	0.56	0.64	0.53	0.59	−0.02	−0.25	0.25	0.25	0.06	0.04	0.49	0.60	0.56	0.74	0.56	**0.75**	−0.44

Table 8. Logarithmic correlation between candidate metrics for lattice quality factors and *X* logical error rates for each combination of yield and code distance. See table 7 for column heading definitions.

Yield	Code dist.	#stab	#flty qubit	#flty data qubit	#flty syn. qubit	*Z* reduced dist.	#*Z* stab	Bgst #qubit of *Z* stabs	Ave. #qubit of *Z* stabs	Bgst #data qubit of *Z* stabs	Ave. #data qubit of *Z* stabs	Dpst depth of *Z* stabs	Ave. depth of *Z* stabs	Bgst *KQ* of *Z* stabs	Ave. *KQ* of *Z* stabs	Bgst *KDQ* of *Z* stabs	Ave. *KDQ* of *Z* stabs	Bgst *Z* cycle	Ave. *Z* cycle	Bgst *CQ* of *Z* stabs	Ave. *CQ* of *Z* stabs	Bgst *CDQ* of *Z* stabs	Ave. *CDQ* of *Z* stabs	Ave. #*Z* stab msmts /step
0.95	5	−0.63	0.66	0.71	0.18	−0.74	−0.68	0.87	0.76	**0.88**	0.64	0.44	−0.05	0.40	0.33	0.39	0.12	0.72	0.75	0.72	0.82	0.76	0.83	−0.75
0.95	7	−0.49	0.62	0.79	−0.05	−0.67	−0.53	0.57	0.59	0.65	0.55	0.27	0.06	0.29	0.26	0.30	0.15	0.79	0.69	0.69	0.85	0.73	**0.87**	−0.54
0.95	9	−0.63	0.81	0.83	0.42	−0.63	−0.62	0.70	0.77	0.69	0.72	0.39	−0.15	0.43	0.38	0.36	0.14	0.66	0.83	0.69	0.87	0.70	**0.88**	−0.79
0.95	13	−0.31	0.58	0.66	0.15	−0.29	−0.31	0.43	0.71	0.43	0.68	−0.05	−0.30	0.01	0.09	0.03	−0.03	0.31	0.70	0.41	**0.82**	0.40	0.81	−0.52
0.95	17	−0.16	0.18	0.17	0.12	−0.08	−0.16	0.25	0.24	0.18	0.29	−0.02	−0.24	0.08	−0.08	−0.02	−0.13	0.15	0.34	0.17	**0.41**	0.16	0.40	−0.28
0.90	5	−0.10	0.41	0.28	0.36	−0.42	−0.15	0.62	0.32	0.53	0.19	−0.14	−0.32	0.13	−0.17	−0.01	−0.24	0.52	0.45	**0.66**	0.60	0.59	0.58	−0.24
0.90	7	−0.34	0.45	0.39	0.30	−0.65	−0.35	0.51	0.54	0.46	0.40	0.29	−0.39	0.51	0.16	0.39	−0.11	0.51	0.65	0.64	0.72	0.61	**0.73**	−0.50
0.90	9	−0.49	0.51	0.58	0.19	−0.67	−0.52	0.66	0.63	0.63	0.53	−0.22	−0.58	0.11	−0.15	−0.09	−0.36	0.42	0.52	0.46	**0.74**	0.48	0.73	−0.58
0.90	13	−0.43	0.69	0.74	0.21	−0.48	−0.46	0.46	0.87	0.49	0.86	−0.14	−0.28	0.03	0.29	−0.15	0.04	0.47	0.77	0.54	0.92	0.55	**0.93**	−0.69
0.90	17	−0.50	0.75	0.67	0.30	−0.63	−0.51	0.49	0.76	0.49	0.67	0.05	−0.29	0.40	0.31	0.18	0.03	0.25	0.73	0.35	**0.87**	0.36	0.86	−0.76
0.80	5	−0.20	0.29	0.24	0.14	−0.51	−0.26	0.69	0.65	0.49	0.55	0.01	0.07	0.39	0.48	−0.09	0.28	0.77	**0.81**	0.75	0.79	0.68	0.77	−0.63
0.80	7	0.19	0.36	−0.06	0.51	−0.62	0.03	0.65	0.78	0.56	**0.83**	−0.04	−0.31	0.31	0.59	0.01	0.33	0.44	0.67	0.58	0.69	0.60	0.75	−0.62
0.80	9	−0.36	0.27	0.24	0.09	−0.54	−0.35	0.59	0.80	0.58	0.60	−0.35	−0.40	0.19	0.46	−0.07	−0.06	0.55	0.77	0.61	0.82	0.64	**0.87**	−0.49
0.80	13	0.13	0.20	0.01	0.19	−0.36	0.05	0.59	0.78	0.64	**0.80**	0.27	0.17	0.50	0.73	0.32	0.58	0.56	0.61	0.54	0.71	0.60	0.79	−0.28
0.80	17	0.43	0.19	0.14	0.15	−0.13	0.40	0.41	0.59	0.33	0.54	−0.18	−0.40	0.29	0.35	0.01	0.05	0.49	0.50	0.36	**0.61**	0.35	**0.61**	0.25
average		−0.26	0.47	0.43	0.22	−0.50	−0.30	0.57	0.65	0.54	0.59	0.04	−0.23	0.27	0.27	0.10	0.05	0.51	0.65	0.54	0.75	0.55	**0.76**	−0.49

suppression at practical physical error rates, at around $p = 0.1\%$, and culling works for 90% yield. With a low physical error rate, 90% yield may be sufficient to build a quantum computer. At 80% yield, only very weak error suppression is observed even at $p = 0.1\%$, even when discarding the weakest 90% of the chips. We conclude that 80% yield is not suitable for building a quantum computer, using the surface code without additional architectural support.

The randomly faulty lattice simulation also revealed that the average of the *CDQ* and the average of the *CQ* of *Z* stabilizers show the strongest correlations to simulated residual error rate among a set of proposed metrics for chip evaluation. The *CDQ* and *CQ* are the product of the 'cycle', which is the average number of steps in the stabilizer measurement and the number of data qubits/number of qubits involved in the stabilizer, respectively. Therefore, the accumulated error possibilities in a stabilizer may be the factor most strongly correlated to the logical error rate.

Faulty data qubits result in merging plaquettes and deepen the stabilizer circuit, hence lengthening the 'cycle'. Faulty ancilla qubits result in requiring more SWAP gates to walk through data qubits and ancilla qubits surrounding the faulty ancilla qubits. However, our data also shows that the number of faulty ancilla qubits has weak correlation to the residual error rate. This finding will also contribute to efforts to build a large-scale quantum computer.

Acknowledgments

This work is supported by JSPS KAKENHI Grant Number 25:4103, Kiban B (25280034 & 16H02812), the JSPS Grant for Challenging Exploratory Research, and the JST ImPACT project. This work is supported by the StarBED project [40].

Appendix A. Surface code error correction details

In the common case, an isolated *X* (or *Z*) error, two neighboring stabilizers will both show -1 eigenstates, and the error is easily isolated as shown in figure A1(a). Because two errors on any plaquette cancel and leave the plaquette in the +1 eigenstate, a series of errors in a neighborhood likely results in two -1 plaquettes separated by some distance, surrounded by +1 plaquettes. If an error chain is connected to the boundary of the lattice, the termination will be hidden. Therefore, an error chain running between the two boundaries will be a logical error.

Applying the same flip operation as the original error is the obvious means of correcting errors, because it fixes the states of each stabilizer (figure A1(a)). To achieve this, we have to identify pairs of error terminations by decoding the detected error information. This problem can be mapped to the graph theory problem known as 'minimum weight perfect matching', a common solution for which is the Blossom V algorithm [41].

However, many different possible chains can connect two units with -1 eigenvalues, as depicted in figure A1(b). Fortunately, any chain works equally well. If the algorithm does not choose the original exact error path, a cycle of errors appears. Such a trivial error cycle does not affect logical states (figure A1(c)). Thus, the choice of a chain between -1 units is not a problem. The important problem in error correction is to pair up the most probable sets of units. Longer chains of errors occur with lower probability, and the matching algorithm weights such possibilities accordingly.

The distance between the two boundaries for an operator is the code distance of a surface code, shown in figure A1(d). The longer the code distance, the higher the tolerance against errors. In the figure, four errors between the two boundaries for the *X* operator are fatal, because the matching algorithm fails to pair them properly. If the two boundaries were farther apart, a longer chain of errors would be required to cause the error correction to fail.

In deleting the oldest round of error syndromes, there can be an error syndrome which is temporally matched to a syndrome which is not to be deleted. If this error syndrome is deleted, the left pair will be matched to another syndrome, leading to unintended behaviors. To avoid this behavior, Autotune employs a means by which the syndrome to be deleted is retained until its pair is deleted.

Appendix B. Details of the implementation

B.1. Irregular whole circuit on account of a fault

Figure B1 show an example of a defective lattice in which the central qubit d40 is faulty. Figure B2 shows the first few tens of steps of the whole circuit of the lattice. We can see that the circuit becomes irregular around the faulty device.

B.2. Algorithms

Algorithm 1, discussed in section 3.2, comprises our stabilizer circuits for the individual superunits.

Algorithm 2, discussed in section 3.3, schedules the stabilizer circuits into a whole circuit for the chip.

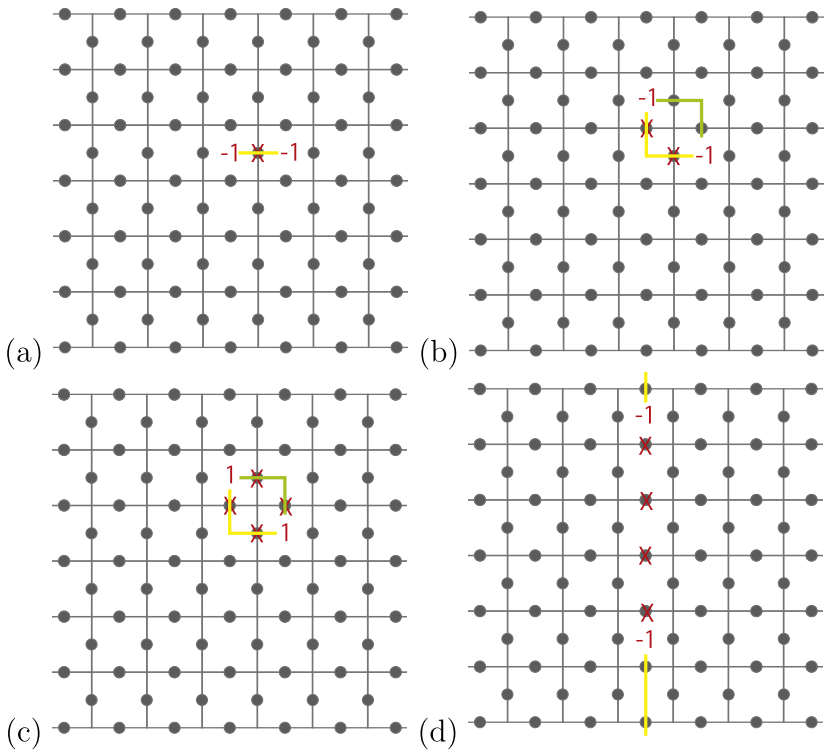

Figure A1. Correctable and uncorrectable errors. Data qubits are described with dots and the lines indicate plaquettes. Syndrome qubits are omitted for visibility. (a) A correctable single error. The red '*X*' indicates that an *X* error occurs on the underlying data qubit. The corresponding two *Z* stabilizers which share the data qubit return -1 eigenvalues. It is easy to interpret the error chain from the eigenvalues, as seen where the yellow line passes through the errored data qubit. (b) A correctable case with two errors. We can consider two error chains from the eigenvalues of stabilizer measurement, the yellow one and the green one. Either of them is valid. Obviously the yellow one is valid; executing *X* gates on qubits underlying the green one generates the trivial error cycle described in (c). (c) Topologically trivial error cycle. This does not affect the logical state of the surface code or the states of data qubits on boundaries. (d) Example of a mis-correcting error chain. Four *X* errors occur in the center of the lattice. The matching algorithm can pair the two -1 plaquettes, for a distance of 4, or pair each -1 with the neighboring boundary of the *X* operator for a total distance of 3. Because three errors are more probable than four errors, the matching expects that three errors occur and chooses the yellow error chains. After applying *X* gates on the data qubits under the error chains, a logical *X* operator is generated, connecting the two boundaries. This is a logical error.

Algorithm 1. Stabilizer Circuit Composition

Input: *Dat*: Set of data qubits belonging to the stabilizer (typically 4 or 6)
Input: *Anc*: Set of ancilla qubits around the stabilizer (typically 1 or 8)
Input: *G*: Graph of qubits, describing qubits' neighbor relationships
Output: Stabilizer circuit of shallowest depth

```
MinCost = __INT_MAX__; MinPath = None;
/* Search for the smallest set of ancilla qubits which neighbor all data qubits. */
for n ∈ (1..Num(Anc)) do
    for ancs ∈ Combination(Anc, n) do
        if d ∈ Neighbors(ancs), ∀ d ∈ Dat then
            /* Search for the shortest path involving data qubits. */
            cost, path = SolveTravelingSalesman(ancs);
            if cost < MinCost then
                MinCost = cost;
                MinPath = path;
            end
        end
    end
    if MinPath ≠ None then
        break;
    end
```

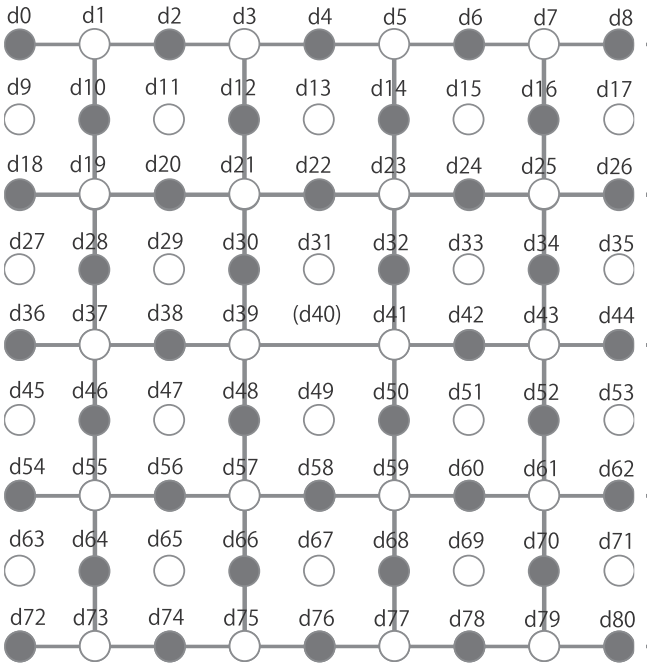

Figure B1. Picture of a lattice corresponding to figure B2. Gray dots are data qubits and white dots are syndrome qubits. The data qubit labeled (d40) is faulty.

(Continued.)

```
end
/* Add operations along the path found. */
AddOp(Initializations(MinPath.ancillas));
for q ∈ MinPath do
    if q ∈ Anc then
        foreach d ∈ [d|d ∈ Dat, d ∈ Neighbors(q)] do
            if IsNotAlreadyGathered(d) then
                AddOp(GatherSyndrome(d, q));
            end
        end
    end
    /* SWAP gate which moves the ancilla qubit variable which holds error syndrome to the next hop in MinPath. */
    if q.next ≠ Null then
        AddOp(SWAP(q, q.next));
    end
    /* SWAP gate which returns the data qubit variable which was swapped to the previous hop to the original positions. */
    if q ∈ Dat then
        AddOp(SWAP(q.prev, q));
    end
end
/* Measure the ancilla qubit which holds error syndrome. */
AddOp(Measurement(q));
```

Algorithm 2. Scheduling algorithm

Input: *SC*: The set of stabilizer circuits
Input: *MaxStep*: The number of time steps to output
Output: *WholeCircuit*

```
/* Sort stabilizers in order of depth, longest first. If they tie, stabilizers on top-left of the lattice have priority. */
sortedSC = Sort(SC);
deepest = Head(sortedSC);
afterHead = AfterHead(sortedSC);
WholeCircuit = NULL;
wholeCeil = 0;
```

Figure B2. The first 60 steps of a whole circuit for a lattice of $d = 5$ and in which the qubit device d40 in the center is faulty. The lattice condition is shown in figure B1. We can see swap gates around d40. The detail of the irregular stabilizer circuit is shown in figure 5.

(Continued.)

while *wholeCeil* $\leqslant$ *MaxStep* **do**
  /* wholeCeil is the step when the deepest stabilizer last scheduled finishes. */
  deepest.ceil = *wholeCeil* = `Schedule`(*deepest*, *WholeCircuit*);
  foreach $s \in$ *afterHead* **do**
    /* schedule every stabilizer once */
    s.ceil = `Schedule`(*s*)
  end
  /* loop until every s.Ceil > wholeCeil */;
  while `Any`(*s.ceil* $\leqslant$ *wholeCeil*|$s \in$ *afterHead*) **do**
    foreach $s \in$ *afterHead* **do**
      if *s.ceil* $\leqslant$ *wholeCeil* **then**

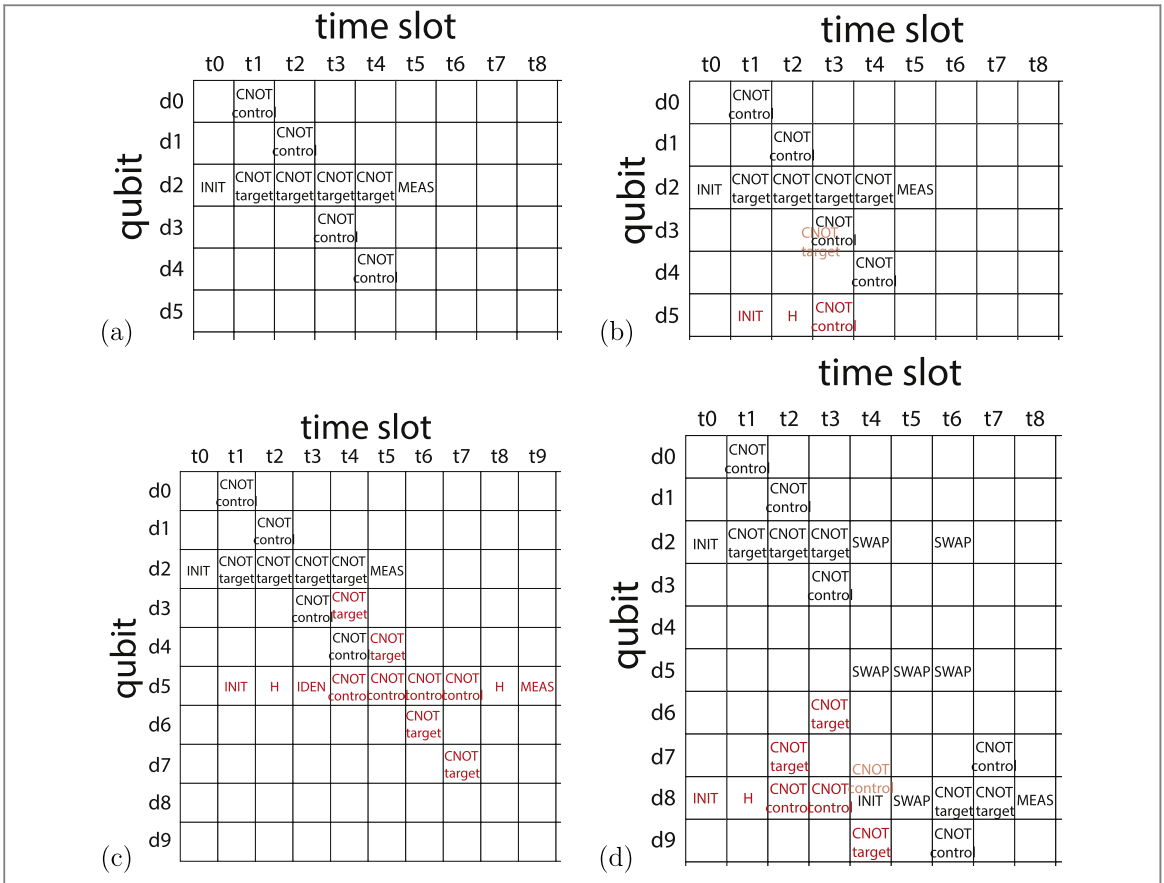

Figure B3. (a) Scheduling chart for the six gates comprising a *Z* stabilizer circuit. Each box is called a slot and NULL slots are vacant. (b) Example of resource allocation conflict occurring between two stabilizers, with the higher-priority one in black and the lower-priority one in red. d3-t3 is allocated to the black stabilizer. The red stabilizer cannot lock the slot d3-t3, as indicated by the light-colored CNOT target. (c) Solution to the contention for a data qubit. The red stabilizer waits for the slot of d3-t3 to become unlocked. (d) The slot of d8-t4 is allocated to the black stabilizer. The black SWAP gate already locks the slot of d8-t4 and the red CNOT cannot lock it.

(Continued.)

```
18            s.ceil = Schedule(s, WholeCircuit)
19          end
20        end
21    end
22 end
```

B.3. Solving conflicts in scheduling

Figures B3 and B4 illustrate various conflicts that occur during scheduling and our solutions. Our scheduling is implemented to allocate 'slots' to gates of stabilizer circuits, as shown in figure B3(a). Each qubit on each step has a slot and only one gate can operate in a slot. When a gate is set in a slot, the slot gets locked. When a swap gate is set in the slot of a data qubit, the data qubit is locked until the data qubit variable returns to the original data qubit device. If conflicts occur, we add identity gates as shown in figures B3(b) and B3(c). This method does not work for conflict on syndrome qubits; this is because a data qubit may be unlocked after a single time step but a syndrome qubit may not be unlocked for several steps, as shown in figure B3(d). Any sequence of gates on syndrome qubits in a stabilizer starts with an initialization gate, which removes the error syndromes which have already been gathered by the syndrome qubit, as illustrated in d8-t4 in figure B3(d). If the stabilizer currently being scheduled (red gates) waits for the syndrome qubit to be unlocked, the initialization gate deletes the syndrome qubit variable with some error syndromes of the stabilizer as shown at d8-t4 in figure B4(e). To avoid this problem, the currently scheduled stabilizer is completely rescheduled after the previously scheduled stabilizer finishes, as shown in figure B4(f).

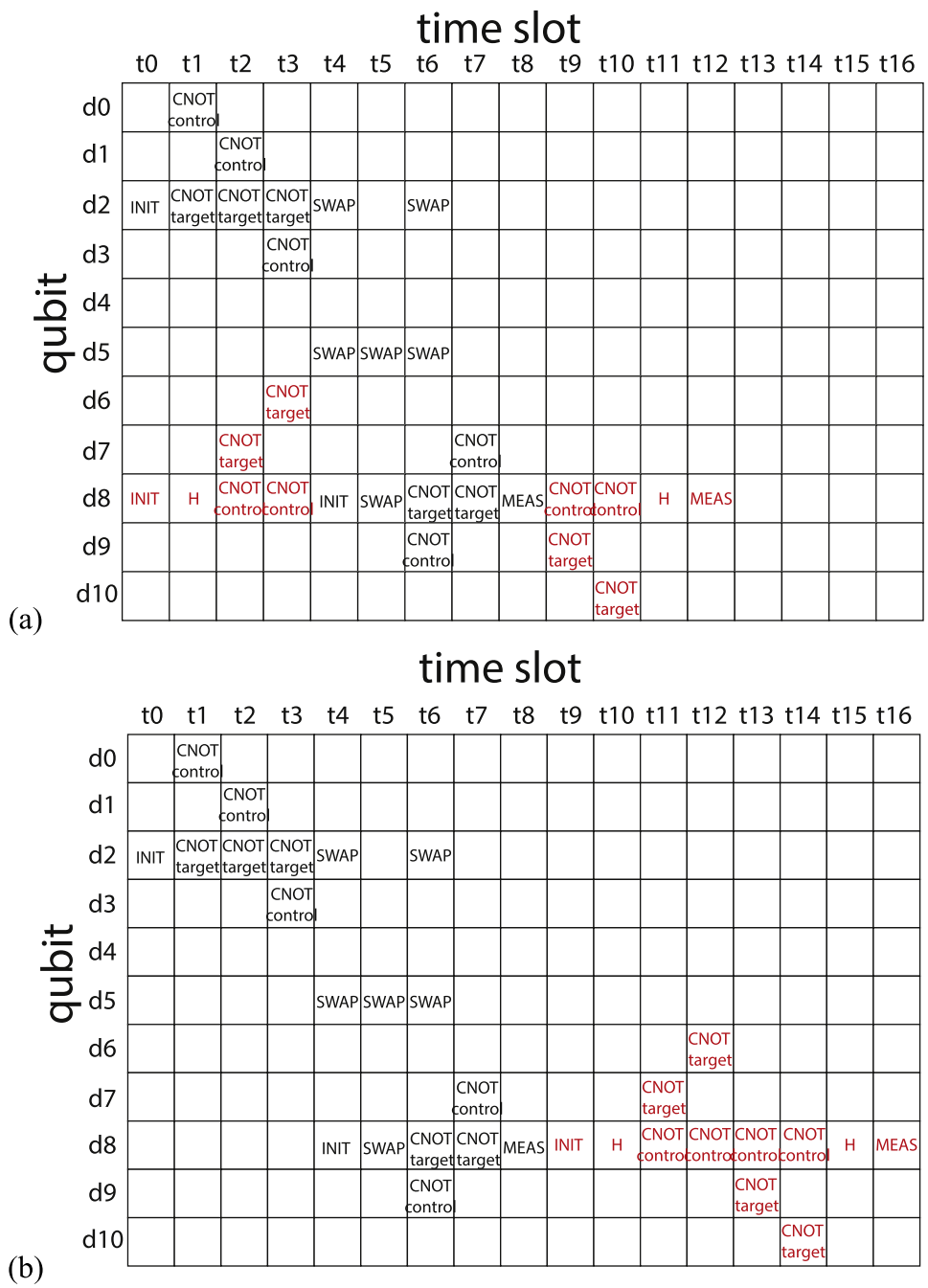

Figure B4. (e) Attempt to solve the competition for a syndrome qubit by waiting. The red stabilizer is split and the former half of its error syndromes are deleted by the initialization gate in d8-t4. This is invalid. (f) Solution to the competition for a syndrome qubit. The red gates are all rescheduled after the black stabilizer.

B.4. Change of the matching nest

A *nest* is used to prepare a network for minimum weight perfect matching. Figure B5 depicts a nest of a perfect lattice of the surface code, output by the Autotune Software created by Fowler *et al* [42]. Each vertex of the nest corresponds to a stabilizer value and each edge corresponds to a possible error. Edges which do not have two vertices are at the boundaries of the lattice. As time advances, the nest expands along the Z axis, creating new vertices and edges when measuring stabilizers. A stabilizer which measures a different eigenvalue from the last stabilizer measurement creates and holds a node on the corresponding vertex. Because an ancilla error (or measurement error) will happen only once, three cycles with an error on the middle cycle would produce the eigenvalue sequence $+1, -1, +1$. The two transitions will be recorded in the nest as two nodes. A data qubit error results in errors on two neighboring stabilizers, so that an error after the first measurement would give the sequence $+1, -1, -1$ in two separate places (or only one if the qubit is on a boundary). In this case, the two transitions result in the creation of two horizontal neighbor nodes in the nest. The matching algorithm will

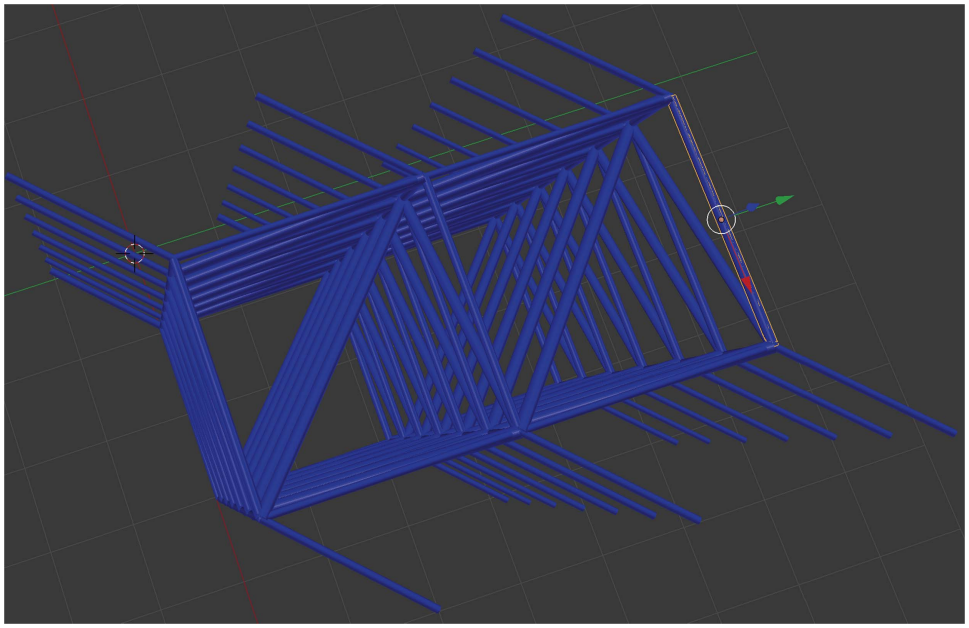

Figure B5. The matching nest for the distance 3 surface code. A stabilizer measurement corresponds to a vertex and a qubit error corresponds to a edge. The ends of the nest correspond to the boundaries of the lattice, and hence they do not have measurement values. If an error occurs on a data qubit, the stabilizers the data qubit is stabilized by will get a different measurement result than the prior round. Then the corresponding vertices create and hold nodes for the minimum-weight perfect matching. Lines for the minimum-weight perfect matching are created by Dijkstra's algorithm on this nest, searching from a node for other nodes [43]. The weight of a line is defined by the weights of edges that the search goes through to create the line, which corresponds to the possibility of the errors which result in the line.

match the two vertical nodes of a stabilizer error or the two horizontal nodes of a qubit error. Lines for the matching between nodes are created with Dijkstra's algorithm on the nest [43]. The weight of a line is given by the sum of weights of the edges which compose the line. Minimum weight perfect matching based on those weights selects the most probable physical errors, therefore it works as error correction.

Irregular stabilizer circuits degrade the parallelism of stabilizer measurements of the whole circuit so that the surface code on a defective lattice has irregular error-matching nests, as shown in figures B6 and B7. These figures are output by Autotune during simulation of our surface code on a defective lattice [42]. We can see the irregularity of the nest. A superunit stabilizer is measured in a longer cycle than normal stabilizers and the vertex of a superunit stabilizer has many edges, some of which are thick. This thickness is proportional to the error probability. The network on which Blossom V runs is generated on this adapted nest to find the best possible solution.

B.5. System architecture

Figure B8 shows the major software components for compiling a circuit for the surface code and simulating its behavior on a defective lattice. Subaru produces a whole circuit for a defective lattice and Autotune simulates the operation of the surface code along the circuit [42]. Subaru performs the tasks described in section 3. Subaru can have alternative inputs—yield or a lattice. The yield is the probability of fabricating qubits which work properly. Instead of a yield, a complete lattice can be input into Subaru. This enables us to investigate particular conditions using hand-constructed lattices.

Appendix C. Supplemental graphs

This appendix shows supplemental graphs to visualize the effect of culling and raw data.

C.1. Graphs to compare culled pools

Figure C1 shows the graphs between yields and logical error rates at specific physical error rates.

C.2. Scatterplots of randomly defective lattices

Figures C2–C6 show the scatter plots of raw data of randomly defective lattices. Figure C3 shows an outlier chip. Actually, the lattice on the chip has 40× worse logical Z error rate as logical X error rate. The lattice by chance has

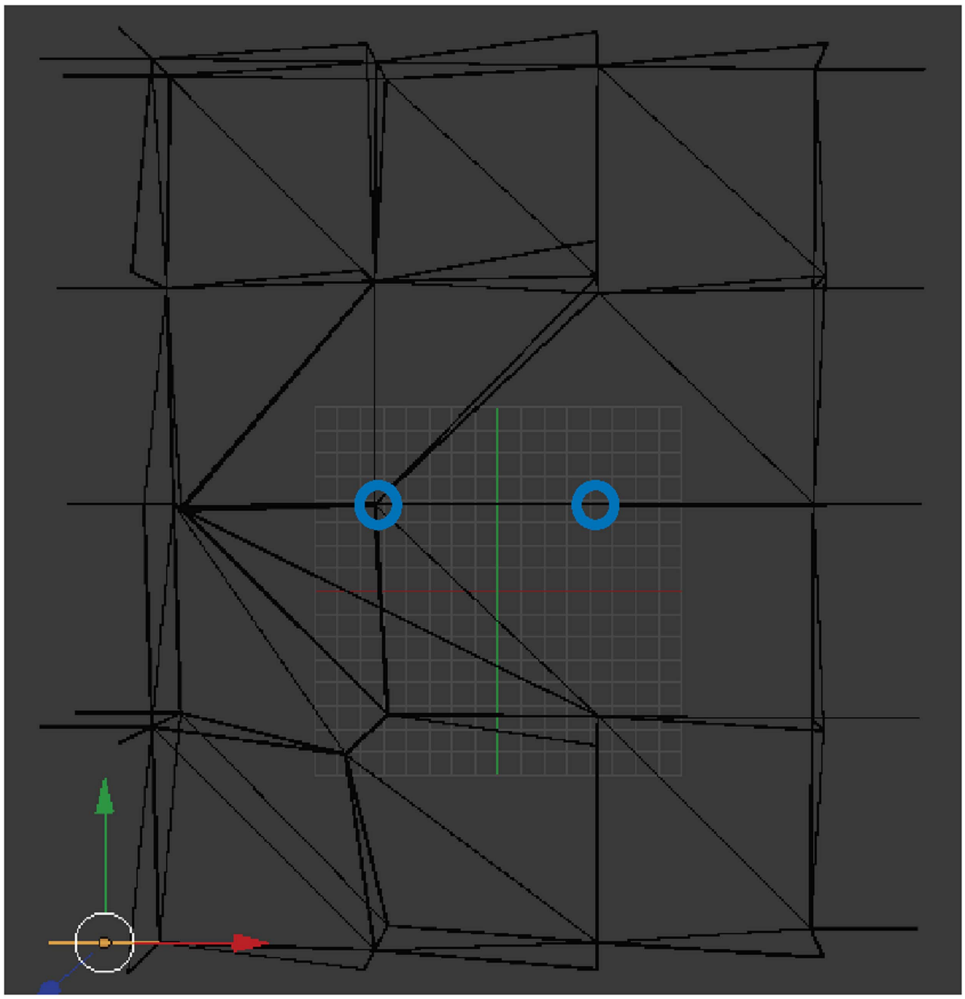

Figure B6. A top view of a visualization of an asynchronous nest. Each vertex describes a syndrome measurement and each edge connects two syndrome measurements which might be changed by the same error. The two blue circles indicate the positions where two unit stabilizers originally existed and now are merged into a superunit stabilizer placed in the left blue circle.

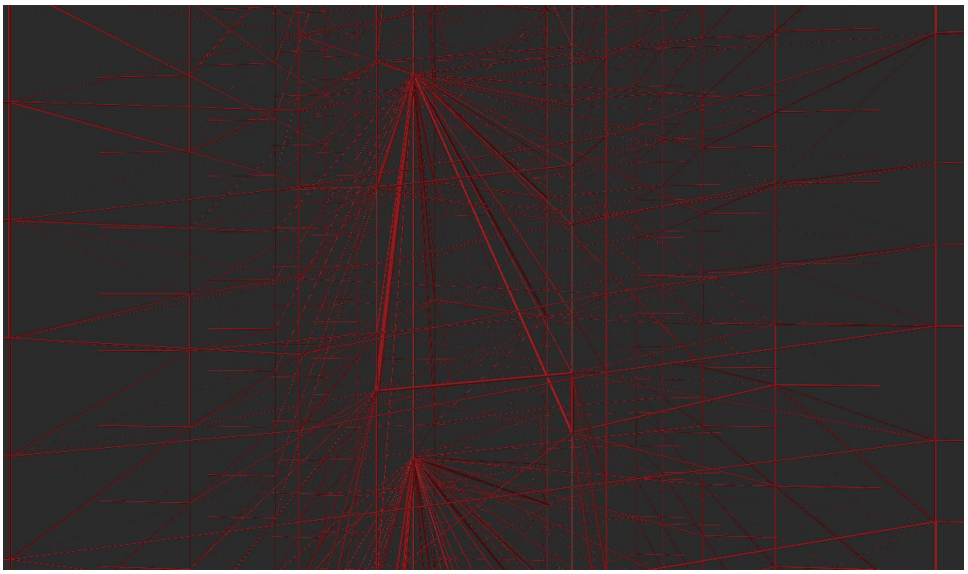

Figure B7. A front-view of a visualization of an asynchronous nest. The diameter of each edge is proportional to the probability of observing detection events at the endpoints of the edge.

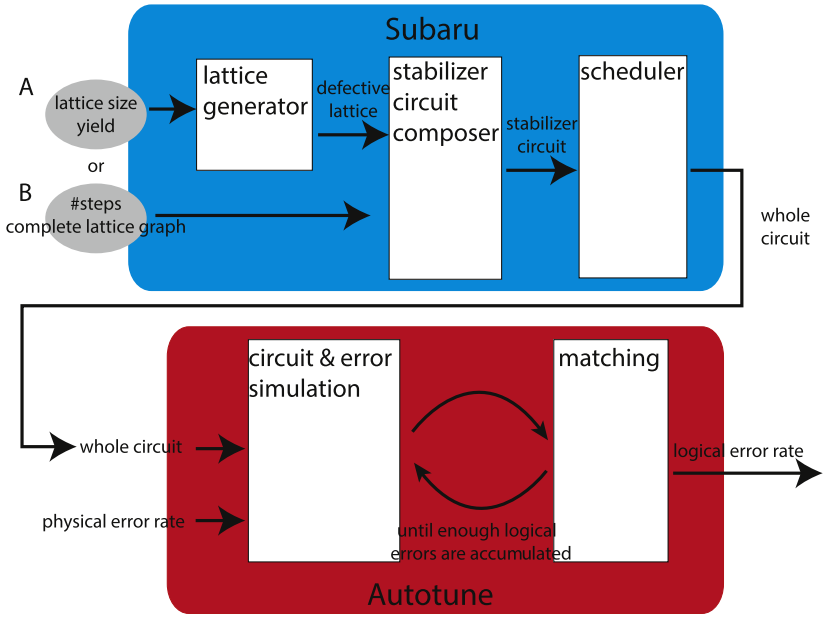

Figure B8. Simulation system design. Subaru can have one of two mutually exclusive inputs—a pair of yield and a lattice size (labeled 'A') or a full description of a lattice (labeled 'B'). With the former input, it randomly generates a defective lattice and then builds circuits to suit. With the latter input, it builds circuits for the provided defective lattice. It outputs a whole circuit of the requested number of steps. The circuit and physical error rate are input into Autotune, and Autotune outputs the logical error rate corresponding to the inputs.

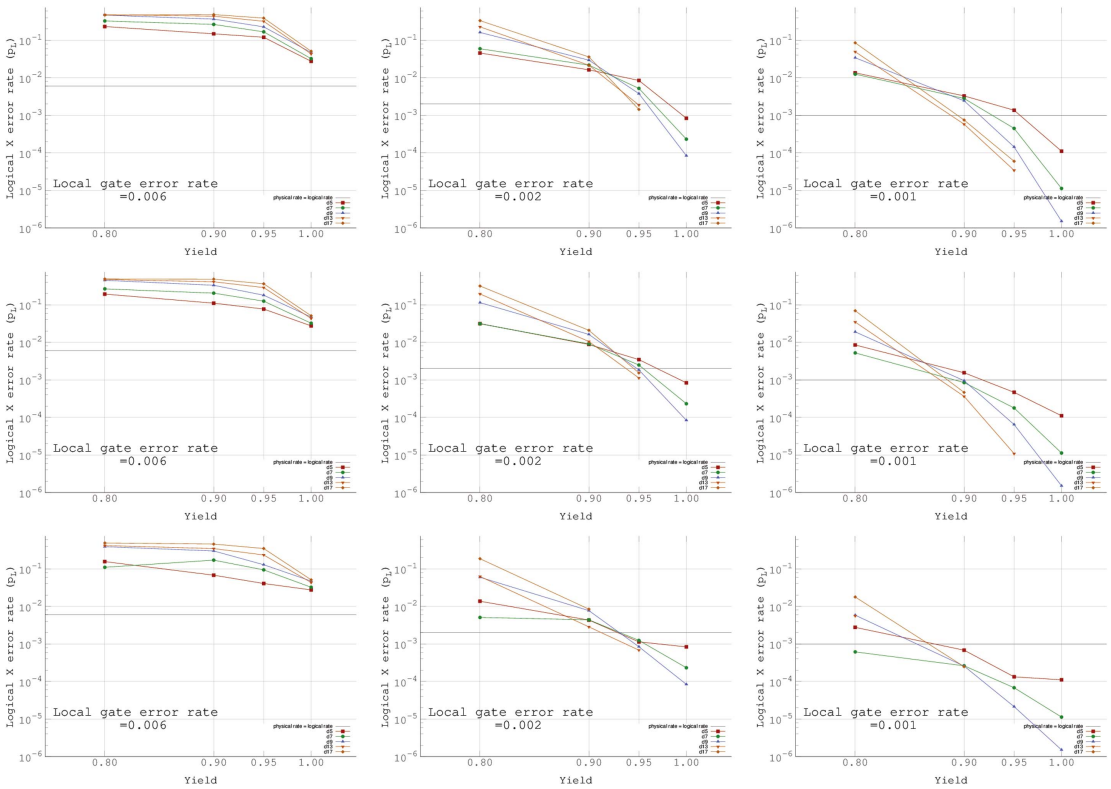

Figure C1. Graphs between yields and logical error rates at specific physical error rates. The top row is of no-culled graphs, the middle row is of 50%-culled graphs, and the bottom row is of 90%-culled graphs. The left column is for $p = 0.6\%$, the middle column is for $p = 0.2\%$, and the right column is for $p = 0.1\%$.

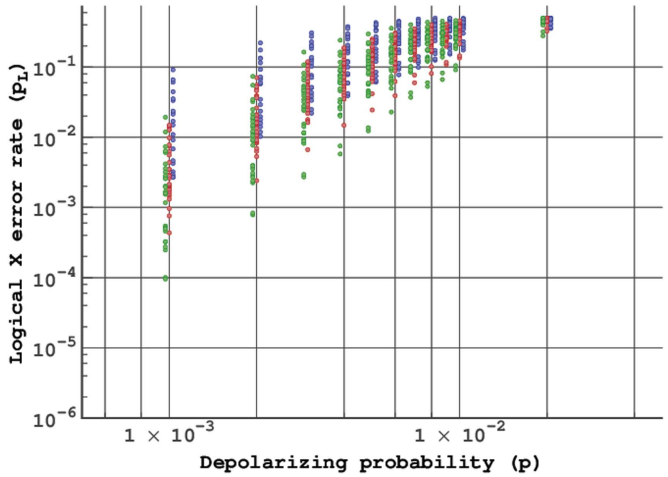

Figure C2. Scatterplot of $d = 5$ with one dot per chip. Green dots are of $y = 95\%$, red dots are of $y = 90\%$, and blue dots are of $y = 80\%$. Blue and green data are offset from the vertical line for visibility.

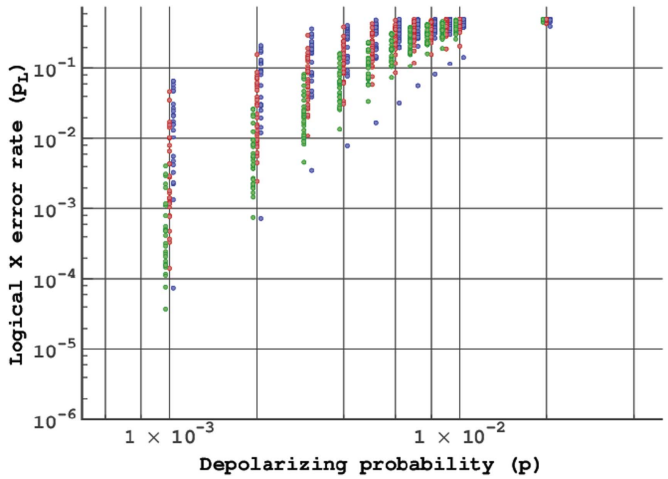

Figure C3. Scatterplot of $d = 7$ with one dot per chip. Green dots are of $y = 95\%$, red dots are of $y = 90\%$, and blue dots are of $y = 80\%$. Blue and green data are offset from the vertical line for visibility.

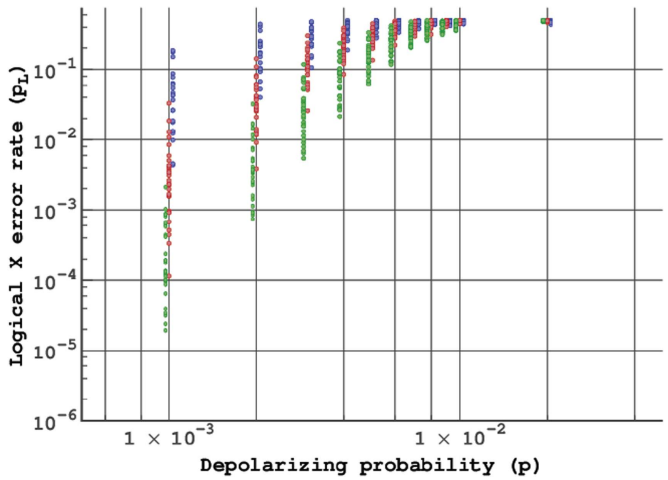

Figure C4. Scatterplot of $d = 9$ with one dot per chip. Green dots are of $y = 95\%$, red dots are of $y = 90\%$, and blue dots are of $y = 80\%$. Blue and green data are offset from the vertical line for visibility.

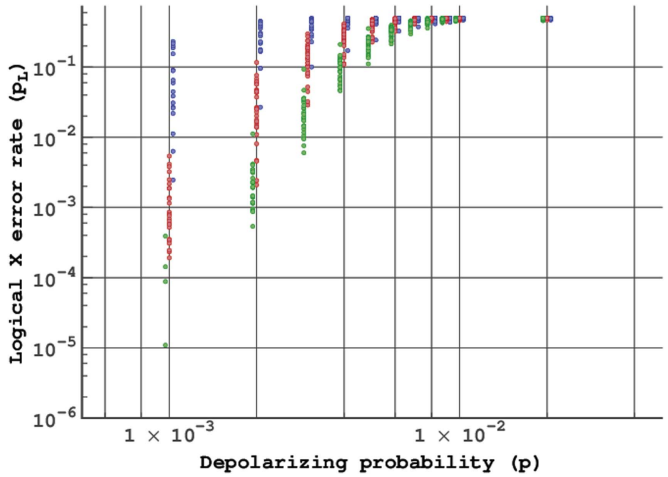

Figure C5. Scatterplot of $d = 13$ with one dot per chip. Green dots are of $y = 95\%$, red dots are of $y = 90\%$, and blue dots are of $y = 80\%$. Blue and green data are offset from the vertical line for visibility.

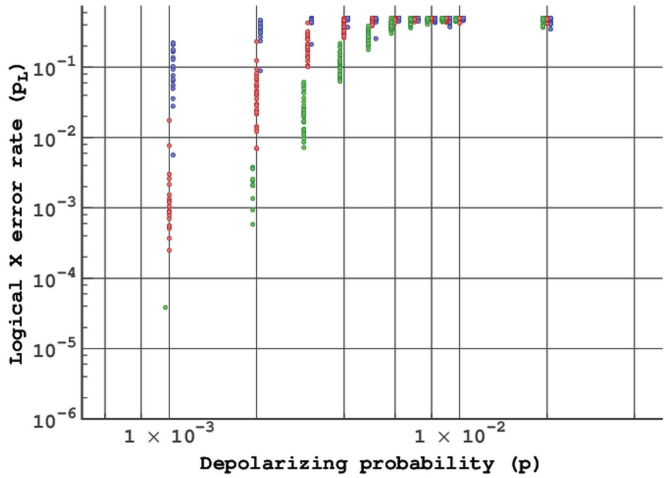

Figure C6. Scatterplot of $d = 17$ with one dot per chip. Green dots are of $y = 95\%$, red dots are of $y = 90\%$, and blue dots are of $y = 80\%$. Blue and green data are offset from the vertical line for visibility.

faulty devices which deform the left and the right boundaries to be close with preserving the top and the bottom boundaries apart. Because of the largely deformed shape of the lattice, the usable area of the lattice is narrow and there are only a few faulty devices on the usable area which increase logical error rates. Hence, the chip exhibits stronger tolerance against logical X error than others.

References

[1] Stace T M, Barrett S D and Doherty A C 2009 Thresholds for topological codes in the presence of loss *Phys. Rev. Lett.* **102** 200501
[2] Nielsen M A and Chuang I L 2000 *Quantum Computation and Quantum Information* (Cambridge: Cambridge University Press)
[3] Van Meter R and Horsman C 2013 A blueprint for building a quantum computer *Commun. ACM* **56** 84–93
[4] Cirac J I *et al* 2000 A scalable quantum computer with ions in an array of microtraps *Nature* **404** 579–81
[5] Yao N Y, Jiang L, Gorshkov A V, Maurer P C, Giedke G, Cirac J I and Lukin M D 2012 Scalable architecture for a room temperature solid-state quantum information processor *Nat. Commun.* **3** 800
[6] Chiaverini J *et al* 2005 Implementation of the semiclassical quantum Fourier transform in a scalable system *Science* **308** 997–1000
[7] Fowler A G, Thompson W F, Yan Z, Stephens A M, Plourde B L T and Wilhelm F K 2007 Long-range coupling and scalable architecture for superconducting flux qubits *Phys. Rev.* B **76** 174507
[8] Shor P W 1994 Algorithms for quantum computation: Discrete logarithms and factoring *Proc. 35th Symposium on Foundations of Computer Science* (Los Alamitos, CA: IEEE Computer Society Press) pp 124–34
[9] Takahashi Y and Kunihiro N 2006 A quantum circuit for Shor's algorithm using 2n + 2 qubits *Quantum Inf. Comput.* **6** 184–92
[10] Van Meter R, Ladd T D, Fowler A G and Yamamoto Y 2010 Distributed quantum computation architecture using semiconductor nanophotonics *Int. J. Quantum Inf.* **8** 295–323

[11] Jones N C, Van Meter R, Fowler A G, McMahon P L, Kim J, Ladd T D and Yamamoto Y 2012 Layered architecture for quantum computing *Phys. Rev.* X **2** 031007
[12] Devitt S J, Fowler A G, Stephens A M, Greentree A D, Hollenberg L C L, Munro W J and Nemoto K 2009 Architectural design for a topological cluster state quantum computer *New J. Phys.* **11** 083032
[13] Whitney M G, Isailovic N, Patel Y and Kubiatowicz J 2009 A fault tolerant, area efficient architecture for shor's factoring algorithm *SIGARCH Comput. Archit. News* **37** 383–94
[14] Shor P W 1996 Fault-tolerant quantum computation *In Proceedings of the 37th Annual Symposium on Foundations of Computer Science* (Washington DC: IEEE Computer Society) p 56–65
[15] Preskill J 1998 Reliable quantum computers *Proc. Roy. Soc. Lond.* A **454** 385–410
[16] DiVincenzo D P 2009 Fault-tolerant architectures for superconducting qubits *Phys. Scr.* **2009** 014020
[17] Aliferis P, Brito F, DiVincenzo D P, Preskill J, Steffen M and Terhal B M 2009 Fault-tolerant computing with biased-noise superconducting qubits: a case study *New J. Phys.* **11** 013061
[18] Svore K M, DiVincenzo D P and Terhal B M 2007 Noise threshold for a fault-tolerant two-dimensional lattice architecture *Quant. Inf. Comput.* **7** 297–318
[19] Fowler A G, Stephens A M and Groszkowski P 2009 High-threshold universal quantum computation on the surface code *Phys. Rev.* A **80** 052312
[20] Kitaev A Y 2003 Fault-tolerant quantum computation by anyons *Ann. Phys., NY* **303** 2–30
[21] Bravyi S B and Kitaev A Y 1998 Quantum codes on a lattice with boundary arxiv:9811052
[22] Raussendorf R and Harrington J 2007 Fault-tolerant quantum computation with high threshold in two dimensions *Phys. Rev. Lett.* **98** 190504
[23] Raussendorf R, Harrington J and Goyal K 2007 Topological fault-tolerance in cluster state quantum computation *New J. Phys.* **9** 199
[24] Barrett S D and Stace T M 2010 Fault tolerant quantum computation with very high threshold for loss errors *Phys. Rev. Lett.* **105** 200502
[25] Lidar D A and Brun T A (ed) 2013 *Quantum Error Correction* (Cambridge: Cambridge University Press)
[26] Buhrman H, Cleve R, Massar S and de Wolf R 2010 Nonlocalty and communication complexity *Rev. Mod. Phys.* **82** 665–98
[27] Buhrman H and Röhrig H 2003 Distributed quantum computing *Mathematical Foundations of Computer Science 2003* (Berlin: Springer) pp 1–20
[28] Buhrman H, Cleve R and Wigderson A 1998 Quantum versus classical communication and computation *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing* (ACM) pp 63–8
[29] Monroe C, Raussendorf R, Ruthven A, Brown K R, Maunz P, Duan L-M and Kim J 2014 Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects *Phys. Rev.* A **89** 022317
[30] Chien C-H, Van Meter R and Kuo Sy-Y 2015 Fault-tolerant operations for universal blind quantum computation *J. Emerg. Technol. Comput. Syst.* **12** 9
[31] Broadbent A, Fitzsimons J and Kashefi E 2010 Measurement-based and universal blind quantum computation *in Formal Methods for Quantitative Aspects of Programming Languages* (Berlin: Springer) pp 43–86
[32] Crep´eau C, Gottesman D and Smith A 2002 Secure multi-party quantum computation *Proc. STOC 2002* (New York: ACM) pp 643–52
[33] Horsman C, Fowler A G, Devitt S and Van Meter R 2012 Surface code quantum computing by lattice surgery *New J. Phys.* **14** 123011
[34] Bacon D 2006 Operator quantum error-correcting subsystems for self-correcting quantum memories *Phys. Rev.* A **73** 12340
[35] Poulin D 2005 Stabilizer formalism for operator quantum error correction *Phys. Rev. Lett.* **95** 230504
[36] Bombin H and Martin-Delgado M A 2009 Quantum measurements and gates by code deformation *J. Phys. A: Math. Theor.* **42** 095302
[37] Landahl A J, Anderson J T and Rice P R 2011 Fault-tolerant quantum computing with color codes arXiv:1108.5738
[38] Miyachi T, Chinen K-I and Shinoda Y 2006 StarBED and SpringOS: large-scale General Purpose Network Testbed and Supporting Software *Proceedings of the 1st International Conference on Performance Evaluation Methodologies and Tools (Valuetools '06)* (New York: ACM) (https://doi.org/10.1145/1190095.1190133)
[39] Steane A M 2003 Overhead and noise threshold of fault-tolerant quantum error correction *Phys. Rev.* A **68** 042322
[40] Miyachi T, Nakagawa T, Chinen K-I, Miwa S and Shinoda Y 2012 *StarBED and SpringOS Architectures and Their Performance* (Berlin: Springer) pp 43–58
[41] Kolmogorov V 2009 Blossom V: a new implementation of a minimum cost perfect matching algorithm *Math. Program. Comput.* **1** 43–67
[42] Fowler A, Whiteside A, McInnes A and Rabbani A 2012 Topological code autotune *Phys. Rev.* X **2** 041003
[43] Dijkstra E W 1959 A note on two problems in connexion with graphs *Numer. Math.* **1** 269–71