Quantum computation on a 19-qubit wide 2d nearest neighbour qubit array.

Alexis T. E. Shaw[*] and Michael J. Bremner
Centre for Quantum Computation and Communication Technology
Centre for Quantum Software and Information
School of Computer Science and
Faculty of Engineering & Information Technology,
University of Technology Sydney, NSW 2007, Australia

Alexandru Paler
Aalto University, 02150 Espoo, Finland

Daniel Herr
d-fine GmbH, An der Hauptwache 7, 60213, Frankfurt, Germany.

Simon J. Devitt
Centre for Quantum Software and Information
School of Computer Science and
Faculty of Engineering & Information Technology,
University of Technology Sydney, NSW 2007, Australia
(Dated: December 15, 2022)

In this paper, we explore the relationship between the width of a qubit lattice constrained in one dimension and physical thresholds for scalable, fault-tolerant quantum computation. To circumvent the traditionally low thresholds of small fixed-width arrays, we deliberately engineer an error bias at the lowest level of encoding using the surface code. We then address this engineered bias at a higher level of encoding using a lattice-surgery surface code bus that exploits this bias, or a repetition code to make logical qubits with unbiased errors out of biased surface code qubits. Arbitrarily low error rates can then be reached by further concatenating with other codes, such as Steane $[\![7,1,3]\!]$ code and the $[\![15,7,3]\!]$ CSS code. This enables a scalable fixed-width quantum computing architecture on a square qubit lattice that is only 19 qubits wide, given physical qubits with an error rate of 8.0×10^{-4}. This potentially eases engineering issues in systems with fine qubit pitches, such as quantum dots in silicon or gallium arsenide.

I. INTRODUCTION

Quantum processor architectures have evolved significantly since their first conception, just over a quarter of a century ago [8, 29, 39, 58]. They started with early discussions on how to build basic gates with two-level systems [39] and have evolved into recent plans for machines with millions of physical qubits [29, 36, 48]. Architecture design must work around the peculiarities of their constituent qubits. Physical limitations on qubit size, gate speed, decoherence rates, temperature, control wiring, and infrastruc-

ture all have a major effect on the potential scalability of a given architecture [8, 36, 58].

Quantum dot qubits in silicon and gallium arsenide (GaAs) present interesting constraints when investigating quantum architecture. They have demonstrated relatively low decoherence rates [61, 63], high operation temperature [62], and high gate speeds [27]. Significantly, they offer the potential of high qubit density and ease of manufacture for large systems on a single wafer [3, 57, 63]. Yet, the size and small qubit pitch that could allow for high qubit density come with significant drawbacks.

Running control wires into gates that have qubit spacings on the order of nanometers in a two-dimensional geometry is extremely challenging. This is especially important be-

---

[*] alexis@alexisshaw.com

cause architectures using the highest threshold quantum error correcting codes, such as the surface or honeycomb codes, tend to assume two-dimensional lattices of unrestricted size [37, 38, 58]. In response, quantum architectures that utilize three-dimensional fabrication have been proposed [58], even though the expected fabrication complexity is formidable. One approach to solving this is to restrict the width of the array, which limits the interconnect density because the system grows only in one dimension.

The idea of minimizing the width of a qubit array is certainly not new. Since the early days of fault tolerance, people have considered fixed or minimal-width arrays of qubits. However, previous approaches have always required undesirable tradeoffs, either increasing qubit requirements or requiring long-distance qubit interactions. For example, CSS codes were leveraged in the paper by Veldhorst et. al. [53], and the subsequent threshold was extremely small (less than 10^{-5}) due to the costs involved in interacting non-adjacent qubits in a nearest-neighbour array. Other examples include specific investigations into linear nearest neighbours in silicon [31] with a threshold of about 10^{-4} and the use of resonators to fold a square lattice into a bi-linear array [40]. Unfortunately, the latter approach required long resonators for interaction between qubits, and these resonators had unknown manufacturability and performance in spin systems given their length and complexity.

In this paper, we present a method of reducing the array width without requiring a lower error threshold or long-distance qubit interactions. Instead, we propose a coding structure that provides good reductions of width for realistic error rates, with a threshold array width as low as 19 qubits for a physical error rate of 8×10^{-4}. It locally has the same nearest neighbour interactivity and the same threshold error rate as the surface code. Of course, this comes at the cost of significantly more qubits; however, this may be an acceptable trade-off for certain technologies, such as silicon quantum dots, where qubits are hoped to be relatively cheap.

We primarily consider two schemes that are based on combining the surface code with techniques for long-range gate compilation and CSS codes. The basic idea is that the surface code can be used to decrease errors enough to allow techniques for longer distance interactions. Subsequently, CSS codes are used in a fixed width array. In section III, we examine the performance of the surface code for rectangular patches for each type of error. These results are then used in section IV to evaluate the performance of a lattice surgery scheme for remote qubit interaction using a very narrow width strip of surface code qubits that we call the surface code bus. In section V, we consider the performance of the Steane $[\![7, 1, 3]\!]$ and $[\![15, 7, 3]\!]$ CSS codes when using the flag qubit compilation approach of Reichardt [15, 47] and a shared long-distance interaction capacity, like that provided by the bus of section IV. We then combine the results of section IV and V and determine the performance of the concatenated scheme in section VI. Finally, in section VII, we examine the possibility of using the rectangular patches evaluated in section III to engineer a logical error bias that can then be concatenated with the repetition code to create a memory with unbiased errors and high enough logical fidelity that a CSS code could be concatenated above it. This enables us to reduce lattice width further, giving a minimum width of 27 for a threshold physical error rate of 1.5×10^{-3}, 19 for a threshold error rate of 8×10^{-4}, and 11 for a threshold error rate of 2×10^{-4}.

II. BACKGROUND

In this section, for the sake of completeness and in order to guide the reader, we summarise material and definitions and provide references to further resources.

II.1. Errors

The threshold and performance of a code are usually quoted in terms of an error rate given an error model. In this work, we assume a standard error model with balanced single-qubit depolarising noise channels. That is, each single-qubit physical operation, including

leaving qubits idle, is acted on by the single-qubit depolarising channel

$$\mathcal{E}_1 = (1-p)\rho + \frac{p}{3}(X\rho X + Y\rho Y + Z\rho Z) \quad (1)$$

before each gate and measurement, as well as after each initialisation. Two-qubit gates are assumed to be operated on by the two-qubit depolarising channel

$$\mathcal{E}_2(\rho) = (1-p)\rho + \\ \frac{p}{15} \sum_{P \in \{I,X,Y,Z\}^{\otimes 2} \setminus \{I \otimes I\}} P\rho P^\dagger \quad (2)$$

before each gate. We assume that there are no correlated higher-order interactions or correlated events apart from those generated by these channels.

A common assumption for existing architecture designs is that the physical error rate of a quantum computer should be approximately one order of magnitude below the threshold for the surface code, that is $p = 10^{-3}$, which we take as the highest value in our analysis [41, 44]. However, we also provide estimates for lower physical error rates, down to 10^{-4}, to inform the reader of the trade-offs between reducing physical errors and handling higher interconnect density.

In this paper, the logical error rate is the probability of a logical error when performing a single logical CNOT gate, unless otherwise stated. The maximum tolerable logical error rate for computation of several significant early quantum algorithms has been estimated in the order of $10^{-14} - 10^{-18}$ [38, 44, 48]. We use the logical error rate of 10^{-15} as a benchmark for each proposed architecture, as it reflects these estimates. An attempt has been made to create a rough equivalence for the difference in quantum operations; however, compilation differences between the examined options mean that this is still a loose comparison.

II.2. Stabilizer Codes

One of the key theoretical improvements toward quantum error correction was the development of the quantum stabilizer code [2, 12,

51, 52]. By measuring a collection of non-local multi-body Pauli operators, it is possible to "stabilise" a logical space that is not within the span of the space of those operators. In effect, the measurement of those operators forces the remaining state to be within a subspace of the total Hilbert space [32]. If low-weight errors are uniquely identifiable by the stabilising operators, then this provides a way to correct for those errors.

The repetition code is the simplest of the stabiliser codes, and perhaps the simplest error correction code. In the classical variant, n copies of the message are made and then compared against each other. The quantum bit-flip repetition code encodes the state $\alpha |0\rangle + \beta |1\rangle$ as $\alpha |0 \cdots 0\rangle + \beta |1 \cdots 1\rangle$ [42, Chapter 10.1.1]. A quantum repetition code cannot correct for all possible errors, as it can only correct for either bit-flip or phase-flip errors, but not for both. Even so, quantum repetition codes have the highest possible code capacity, with the ability to correct for at most $\frac{n-1}{2}$ errors of that one type. We use this to improve performance to the required level in Section IV and VII.

In order to implement our concatenation scheme efficiently, we need an efficient implementation of a block code that has good qubit density and a relatively high threshold. In order to obtain this, we leverage the recent advances in flagged syndrome extraction [13–15, 47], discussed in section II.3. We consider the Steane $[\![7,1,3]\!]$ code and the $[\![15,7,3]\!]$ code for use as higher level codes in this work as they appear to have a good mix of performance and density, as well as having flagged extraction circuits.

The Steane $[\![7,1,3]\!]$ code is a self-dual CSS code defined using the 7-bit hamming code [52], and has stabiliser generators listed in Table I. This code can correct for a single error, and is the smallest of the self-dual codes.

The $[\![15,7,3]\!]$ self-dual CSS code is defined using the 15-bit hamming code [13], and has stabiliser generators listed in Table II. This code can also correct for a single error, however it encodes logical qubits significantly more efficiently than the Steane code, at the expense of a greater number of more complicated stabilisers.

1	$IIIXXXX$
2	$IIIZZZZ$
3	$IXXIIXX$
4	$IZZIIZZ$
5	$XIXIXIX$
6	$ZIZIZIZ$

TABLE I: Stabiliser generators for the Steane $[\![7,1,3]\!]$ code

1	$IIIIIIIXXXXXXXX$
2	$IIIIIIIZZZZZZZZ$
3	$IIIXXXXIIIIXXXX$
4	$IIIZZZZIIIIZZZZ$
5	$IXXIIXXIIXXIIXX$
6	$IZZIIZZIIZZIIZZ$
7	$XIXIXIXIXIXIXIX$
8	$ZIZIZIZIZIZIZIZ$

TABLE II: Stabiliser generators for the Steane $[\![15,7,3]\!]$ code

II.3. Flaged Syndrome Extraction

Flagged syndrome extraction is a newly proposed technique for the fault-tolerant measurement of stabilisers in quantum error correcting codes [13–15, 45, 47]. It enables significantly lower extraction circuit complexity and lowers the number of ancillae required compared to previous techniques for fault-tolerant stabiliser measurement.

Earlier methods of fault-tolerant syndrome extraction assumed that error-amplification caused by stabiliser measurement led to severe degradation of code performance. It was assumed that an error with weight beyond the code capacity was uncorrectable, hence stringent measures were proposed in order to prevent error amplification during syndrome extraction. Shor's proposed method [50] was to create a GHZ state the size of the stabilizer to be measured, verify it by interacting each element with one of the qubits supported by the stabiliser, and then measure the GHZ state. This would require both increased circuit depth and a substantially greater number of ancilla qubits than what would be required by a non-fault-tolerant stabiliser measurement circuit. In addition to the increase

in resources, this method also leads to a lower threshold for the code, as there is more time and space for potential errors in each round of syndrome measurements.

Flagged syndrome extraction capitalises on a realisation that some temporary amplification of errors can be allowed, so long as any higher weight errors can be uniquely distinguished and corrected given enough rounds of syndrome extraction. This loosening of requirements allows a notable reduction in the number of ancilla qubits required, with only two ancilla qubits required for weight-3 codes. Further, in some cases, it is possible to perform this extraction with zero additional gates over a basic extraction (such as in the circuit described in Figure 13a). This is achieved by the careful ordering of otherwise commuting gates, as well as through inserting detection gadgets to disambiguate different error channels. The main downside of this approach is that the circuit and decoder designs are substantially more difficult than in earlier forms of fault-tolerant extraction. This is because the faults identified must be propagated forward through the extraction circuit so that the higher weight errors as well as the extraction of other syndromes may be corrected [13], whilst the earlier methods do not introduce additional data qubit errors during syndrome extraction, and so only need a simple hamming-code decoder.

II.4. The Surface Code

The surface code is one of a family of stabiliser codes that has physically local stabilisers. Here, the qubits that make up the stabilisers are supported by qubits that are physically close to each other. In a surface code such as in Figure 1 the plane is tiled with two sets of square plaquettes [9, 21]. The X plaquettes, and the Z plaquettes, are represented here as dark and pale diamonds. For historical reasons, the Z plaquettes are also called faces and the X plaquettes are called vertices [33]. However, the diamond presentation used here shows that they are completely symmetrical, with the exception of the ancilla initialisation and measurement.

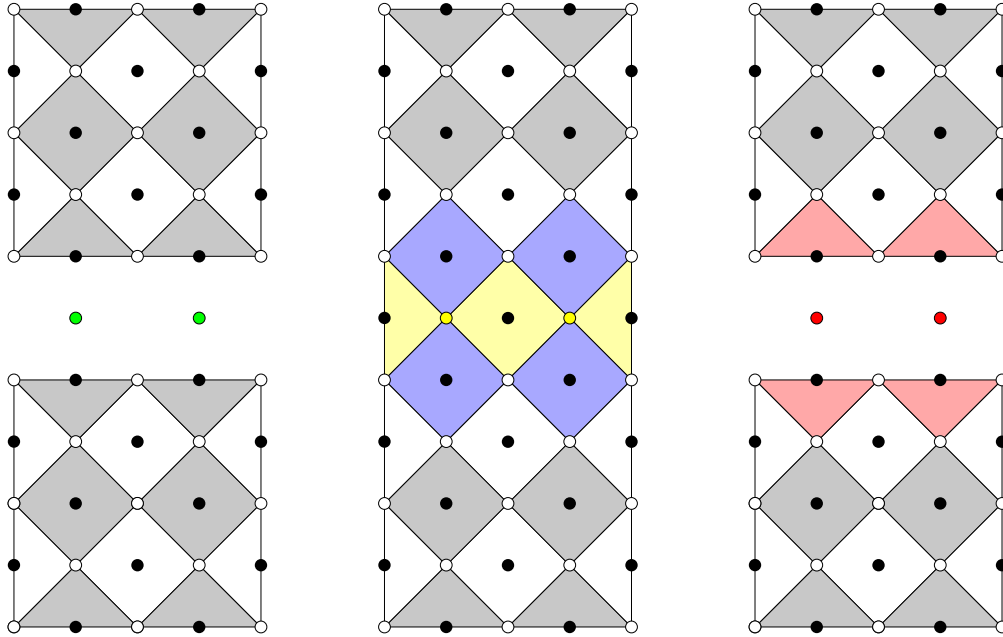The logical operators in the surface code

FIG. 1: A $d_X = 3$, $d_Z = 7$ surface code. One Z logical operator is highlighted in blue, and an X operator is highlighted in yellow.

run from one boundary of a type to the other boundary of that type. The type of the logical operator is determined by the plaquette type on the boundary, with an X-boundary having partial X-plaquettes on the boundary, and similarly for Z-boundaries. For the same reason used to determine plaquette names, Z-type boundaries are often called rough boundaries, and X-type logical boundaries are called smooth boundaries. To measure a logical operator, you measure qubits in the appropriate basis along that logical operator.

Physical errors are always correctable if the length of the longest error chain is less than half the distance between two boundaries of the same type. That is, the code distance is equal to the smallest number of code qubits between two boundaries of the same type. This makes these codes have a significantly lower density than other stabiliser codes (for example the $[\![7, 1, 3]\!]$ code needs 9 total qubits for fault tolerance [14], but the rotated $d = 3$ surface code uses at least 13 qubits, and the non-rotated surface code uses 25 [54], quantum LDPC codes have even higher densities [10, 29]). However, as a trade-off, their small stabiliser circuit sizes and physical locality means that syndrome extraction is extremely fast and relatively easy to perform. Further, the low circuit depth and locality of errors lead to one of the highest thresholds of any quantum error-correcting code.

In many ways, the discovery of the surface code is what may make the development of practical digital quantum computers possible. Simulations have put the logical error rate of a surface code of distance d at approximately

$$p_{l_{\mathrm{bus}}} \sim 0.3(70p)^{\lfloor \frac{d+1}{2} \rfloor} \qquad (3)$$

per round of syndrome extraction [18], and a threshold of over 1% [59].

5

II.5.   Lattice Surgery

Lattice surgery is the most space efficient manner of fault-tolerant 2-qubit interaction between surface code qubits that is currently known [29, 38]. When combined with faulty state injection and state distillation, it enables the surface code to perform fault-tolerant universal quantum computation [29, 37, 38].

The key operations used in lattice surgery are merging two adjacent surface code patches into one larger patch and taking a large patch and then splitting it into smaller ones.

Patches are merged by measuring new stabilisers between two surface code patches, as shown in the first operation of Figure 2. The logical state of the new patch is a function of the logical states of the two patches before the merger, the type of boundaries merged, and the parity of the new and modified stabilisers after measurement. The parity of these new measurements encodes a coherent logical parity measurement between the logical qubits. If the parity is zero after this measurement, then the operation encodes the mapping $|0\rangle_L \langle 00|_L + |1\rangle_L \langle 11|_L$. Otherwise, a correction must be made that is equivalent to choosing one of the patches and flipping it before the merge. The correction forces the measurement to be parity zero, and so the mapping for the parity zero measurement can be used.

A patch is split by modifying the stabilisers to separate the patches, as shown in the second operation of Figure 2, and then measuring the data-qubits that are between the patches. This operation creates an entangled state, which is described by the mapping $|00\rangle_L \langle 0|_L + |11\rangle_L \langle 1|_L$.

The combination of these two operations allow circuits to be compiled far more space-efficiently than other techniques, such as braiding [20]. Proposed compilation approaches have only a 50% space penalty when compared to quantum memory alone [37]. The combination of a merge followed by a split and a Pauli-correction results in a parity measurement between the two patches, with the measured operator determined by the types of boundaries merged. By combining two different parity measurements and one ancilla surface code patch, it is possible to implement either a CNOT or CZ gate (see Figure 3).

II.6.   The ZX-Calculus

The ZX-calculus was developed as a graphical system for reasoning about quantum linear maps. It represents quantum operations with tagged graphs that are manipulated using a collection of simplification rules [16, 60]. Every quantum circuit can be efficiently mapped to a ZX diagram, and the graphs can be used to prove the properties of those circuits.

In this work, we use ZX-calculus to show the correctness of the surface code bus in Section IV.2. It is especially useful for discussing the logical effect of lattice surgery operations [6], as the operations of merging and splitting of the surface code map neatly and naturally to the ZX-calculus. Some care is needed, however, to understand the action of joint measurements in the lattice surgery. The ZX-calculus can even be said to be complete, in the sense that the diagram re-writing rules are sufficient to convert any two diagrams that represent the same linear map to the same diagram [5, 30].

Here we reproduce some basic definitions and results that are used in this work. Specifically, we use the fact that state preparation and measurements are represented by degree-one nodes

$$|+\rangle = \text{◯} \quad , \quad \langle +| \, R_z(\alpha) = \text{——}\textcircled{\boldsymbol{\alpha}};$$

that a Z-flip, or other Z-rotation, is represented by a degree-2 Z-node

$$|0\rangle \langle 0| + e^{i\alpha} |1\rangle \langle 1| = \text{——}\textcircled{\boldsymbol{\alpha}}\text{——};$$

that a smooth split is represented by a Z-spider

$$S_S = |00\rangle \langle 0| + |11\rangle \langle 1| = \text{——}\text{◯}\langle ,$$

and that a rough merge is an X-spider with a correction on one leg that depends on the parity measurement result [6]

$$M_S = \text{⟨}\textcircled{b\pi}\text{◯} ,$$

6

FIG. 2: Illustration of a smooth lattice surgery merge followed by a smooth split, implementing an XX parity measurement. The qubits in green are initialised in the $|+\rangle$ state. Then the new stabilisers, shown in yellow, are measured, and the stabilisers in blue are expanded. Finally, the stabilisers in red are reduced, and the qubits in red are measured in the X basis.

(a) Circuit for CNOT

(b) Circuit for CZ

FIG. 3: CNOT and CZ gates implemented using parity measurements.

where b is the measurement outcome of the merge. Finally, we note that the CNOT gate can be represented as a Z-spider connected to an X-spider,

In addition to these definitions, we use the spider-merge simplification rule, which states nodes of the same type can be merged as long as their rotations are summed modulo 2π.

7

III. RECTANGULAR SURFACE CODES

III.1. Motivation

In rectangular surface codes with a small X or Z distance, edge effects necessarily have a greater impact than when dimensions get larger. Because of this, the simple scaling rules for taking the well-studied performance of square surface codes [11, 22, 26, 41, 46, 54, 59] to derive the performance of a rectangular surface code patch are expected to break down on high aspect ratio codes. Hence. direct analysis and simulations are required to study narrow rectangular surface code patches.

While there has been some recent work in this area [4, 35], this work does not separately evaluate how rectangular patches bias the rate of logical X and Z errors. In this paper, we manage the biassing effect of these rectangular patches on logical error rates to reduce the minimum required width for a scalable qubit array. This requires us to perform new simulations to understand and characterise these impacts, which is especially important for determining the performance of the surface code bus in Section IV.

III.2. Simulations of biased surface codes

We performed simulations of rectangular surface codes for d_Z surface code syndrome extraction cycles for $d_X \in \{3, 5, 7\}$ with increasing odd values of d_Z. We then measured the data qubits. These simulations were performed with the patches initialised in both the $|0\rangle$ and $|+\rangle$ logical states using the high-performance simulator *Stim* developed by Gidney [23]. This tool uses a tableau representation similar to that of the CHP simulator [1] of Aaronson et.al. which we re-implemented for our simulations of parity codes in section III.

In order to determine errors, the Pauli basis was updated for each round of stabiliser using a Minimum Weight Perfect Matching decoder, as described by Fowler [59], and then the measurements were interpreted accordingly. Although not an optimal decoder, the Minimum Weight Perfect Matching decoder is signifi-

cantly better than a hamming-distance lattice decoder, and it does not require significant extra implementation complexity. In our implementation, the matching graphs created were solved using the Blossom V maximum matching algorithm implementation of Kolmogorov [34].

To evaluate the performance of the decoder, simulations were also performed on square lattices and compared with pre-existing results [22, 59]. These results were largely the same with only some very minor differences. As such, we don't expect major improvements with better decoders. The results in figure 4 demonstrate the upper bound on the performance of the surface code given by this decoder.

III.3. Evaluation of results

A fitting function was necessary in order to extrapolate the performance of these surface codes and enable further analysis. These fitting functions needed to have a form that made sense given the expected theoretical behaviour of the codes, as well as provide a good fit for the simulation data. The available computing resources limited the number of simulations that could be performed and, in turn, restricted the number of widths (i.e. values of d_X) we could analyse. This made it difficult to determine a single fitting function, so we derived separate fits for each of the p_Z and p_X error rates, and for each d_X. This gave fits in good agreement with the simulation as shown by the dotted lines in Figure 4.

Our fits for the Z error rates p_Z are of the form $\alpha \frac{d_X - 0.5}{d_Z - 0.5} d_Z (\beta p)^{\lfloor \frac{d_z+1}{2} \rfloor}$. This was chosen by taking the expected $\alpha(\beta p)^{\lfloor \frac{d_z+1}{2} \rfloor}$ error per time step, multiplying it by d_Z to account for the number of syndrome extractions required to perform a lattice surgery operation, and then adding a correction factor of $\frac{d_X - 0.5}{d_Z - 0.5}$. The correction factor was empirically found to provide for a better fit. The fitting parameters α and β are given in Table IIIa for each d_X.

Fits for the X error rate p_X of the form $f_{d_X}(d_Z) \cdot p^{\lfloor \frac{d_X+1}{2} \rfloor}$ were also found, where $f(x)$ is some quadratic polynomial in d_Z. A
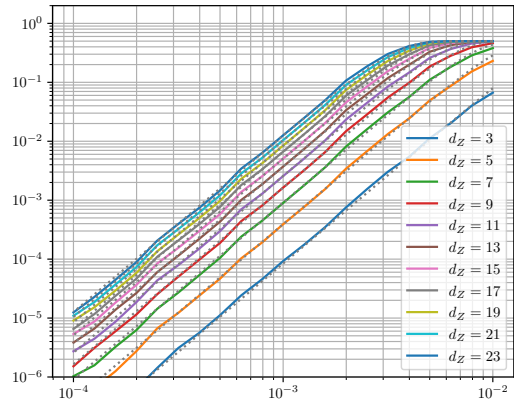
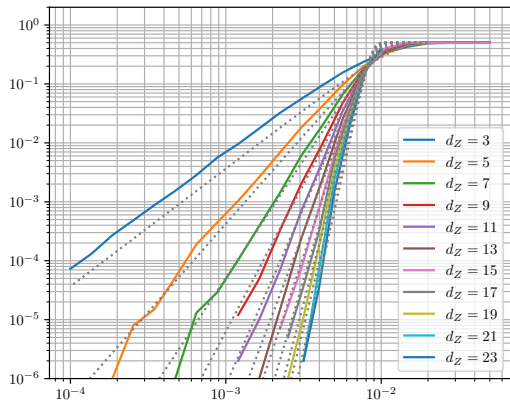(a) p_Z with $d_X = 3$, 10^9 samples per point.

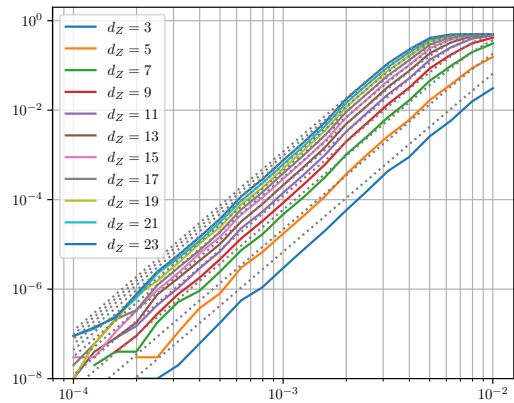(b) p_X with $d_X = 3$, 10^8 samples per point.

(c) p_Z with $d_X = 5$, 10^8 samples per point.

(d) p_X with $d_X = 5$, 10^6 samples per point.

(e) p_Z with $d_X = 7$, 10^6 samples per point.

(f) p_X with $d_X = 7$, 10^6 samples per point.

FIG. 4: Plots of the Logical error rates in both the Z and X directions over d_Z rounds of a biased rectangular surface code.

9

quadratic was chosen because the number of error channels grows approximately proportionally to the number of surface code extractions multiplied by the length of the surface code patch, that is as the square of d_Z. A generic quadratic was chosen as this improved the fit over a more restrictive function and seemed to be a physically plausible way to account for edge effects. These polynomial fits were found for each d_X, and are given in Table IIIb.

IV. THE SURFACE CODE BUS

IV.1. The GHZ state Bus

The preprint by Herr et. al. [28] proposed a method that used a narrow single-qubit wide bus to interconnect two $d \times d$ surface code patches to determine the parity between them, which they called the surface code data bus. This used the procedure we reproduce as Algorithm 1. A visual representation of their proposed bus is presented in Figure 5. Sadly, this procedure is not fault-tolerant (however we show a modified version that is fault tolerant in the next section), because at line 9 the parity measured is not protected by the repetition code, meaning any Z error that occurs on any of the data qubits will flip the X parity, taking the GHZ state prepared at line 3 from $|0 \cdots 0\rangle + |1 \cdots 1\rangle$ to $|0 \cdots 0\rangle - |1 \cdots 1\rangle$. As both of these have the same values for the XX operators checked in line 5, any such error in the time/space volume is uncorrectable. Further, any single measurement error on line 9 will also be undetectable.

The size of the GHZ state depends on the distance between the two qubit patches, as well as their code distance. Together, this means the GHZ state will be Nd qubits wide in the worst case, where N is the number of patches along the surface code bus (where the minimum is 2, in the case of adjacent patches), and d is the surface code distance. To detect X errors in the preparation of the GHZ state, it is necessary to perform step 5 d times, with each step taking $t \geq 4$ steps. The total time/space volume of the parity check must be then at least Ntd^2. This means that Ntd^2p needs

to be below 0.5 for the protocol to be fault-tolerant, as the code capacity for the repetition code is 0.5. This allows for $N \leq 13$ when $p = 0.1\%$ and $d = 3$; it allows $N \leq 4$ when $d = 5$, and only adjacent patches may be interacted fault-tollerantly when $d \geq 7$. Further, the number of repetitions in this bus could be substantial depending on how close the system is to the code capacity of the repetition code. An increase in code distances might be required in order to meet the required error rate for higher-level protocols and algorithms.

IV.2. The Folded Surface Code Bus

The above technique can be made fault tolerant for any code distance or spacing by using lattice surgery over biassed surface code patches to replace the GHZ state [43]. This modified fault-tolerant procedure is presented in Algorithm 2 with a visualisation of the logical operators shown in Figure 7. The visual representation is especially useful in understanding the possible error chains and the locations and interactions of various boundaries. Also useful in understanding the algorithm is the schematic view of the surface code regions shown in Figure 6. We call this the folded surface code bus because it can be understood as folding a temporally thin lattice surgery parity measurement along the time axis so that it is narrow in space.

The lattice-surgery operations in Algorithm 2 create a logical bell pair

$$|\alpha_1 \alpha_3\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

on lines 3-8. On lines 9 to 13, each half of this bell pair is merged with one of the input logical qubits, measuring both the parity between ϕ and α_1 and the parity between ψ and α_3. As α_1 and α_3 encode a bell pair, the parity between these measurements is the parity between the two input logical qubits. In addition to those lattice surgery operations, we have to measure the stabilisers for multiple periods to allow for error detection; compute an updated estimate of the Pauli frame, and correct the observed measurements before the final parity calculations are performed.

d_X	α	β
3	0.09	95
5	0.06	110
7	0.03	125

(a) Parameterd for the p_Z fit

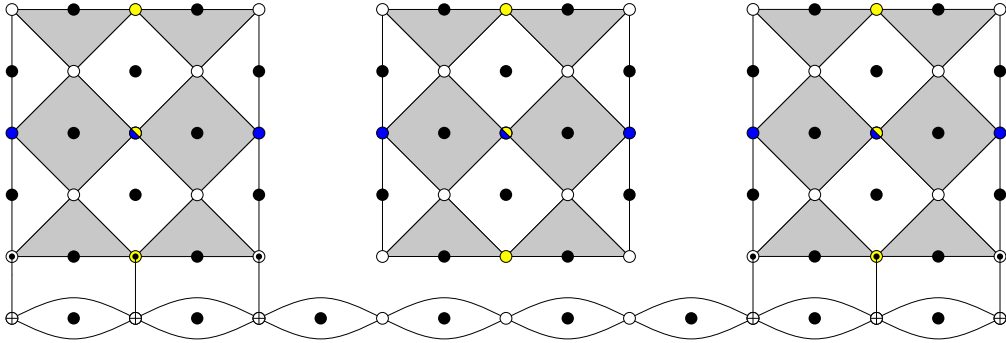| d_X	$f_{d_X}(d_Z)$
3	$f(x) = 500x^2 - 700x + 250$
5	$f(x) = 26399x^2 - 57295x + 18522$
7	$f(x) = 2.87 \times 10^6 x^2 - 1.55 \times 10^7 x + 2.75 \times 10^7$

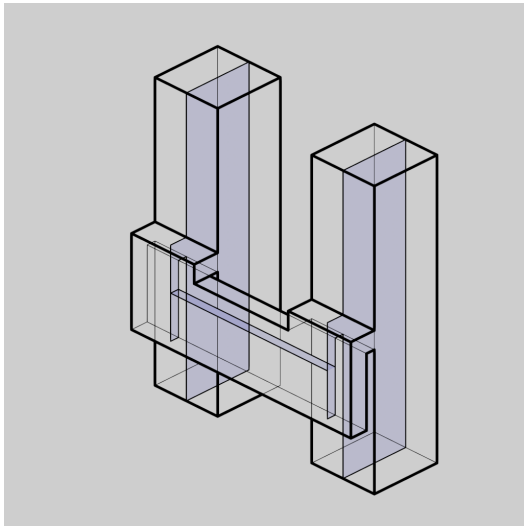(b) Functions for the p_X fit

TABLE III: Parameters for chosen fits.

Algorithm 1 GHZ Surface Code Bus

1: **procedure** UNPROTECTEDBUS($d \times d$ qubit SC patches ψ, ϕ indexed as $\theta_{[i,j]}$)
2:     **for** $i \in \{1 \ldots d\}$ **do**
3:         Create an $2d + m$ quantum register b which is initialised to the GHZ state $\frac{1}{\sqrt{2}}(|0 \cdots 0\rangle + |1 \cdots 1\rangle)$
4:         **for** $j \in \{1 \ldots d\}$ **do**
5:             Check for errors by measuring the operator $X_{b_{[k]}} X_{b_{[k+1]}}$
                for $k \in \{1 \ldots (2n + m - 1)\}$
6:         **end for**
7:         Update Pauli frame to correct errors in GHZ state.
8:         Perform a CNOT gate between the GHZ state and SC patch Z boundaries using

$$\bigotimes_{k=1}^{d} CNOT_{b_{[k]}, \psi_{[1,k]}} \otimes CNOT_{b_{[k]}, \phi_{[1,k]}}$$

9:         Measure b in the X basis and compute the parity p_i of the result.
10:     **end for**
11:     Compute the majority vote over all p_j, the parity XX over ψ and ϕ and return.
12: **end procedure**

FIG. 5: Visual representation of the interaction between the GHZ state, and the two $d = 3$ surface code patches in the Unprotected GHZ surface code bus ZZ measurement.

(a) Z Logical Operators   (b) X Logical Operators

FIG. 6: Logical operators for the Folded Surface Code Bus for XX. In these diagrams the vertical direction represents time, with the cross-section at each point of time representing the surface code at that point in time. The marked surfaces represent some of the possible logical operators at each point.

FIG. 7: Layout and position of surface code patches for the folded surface code bus.

The choice of period of stabiliser measurements within the loop at line 4 ensures that the ancilla patches are in either a $|\Phi^+\rangle$ or a $|\Psi^+\rangle$ bell state, so that no error can propagate to the input patches during the subsequent merges. Further, the choice of period of stabiliser measurements within the loop at line 11 determines that the uncertainty of the parity measurement due to errors in the merged parity is no greater than that due to the width of the bell preparation ancilla.

We now have to show that the method presented here is, in fact, fault-tolerant. There are two different approaches, and we present both here. The first is to note that the only surface code errors that can occur with a distance w probability are X errors in the creation of the bell state and errors in the determination of the parity between each of the biased logical bell-state patches. Figure 7 depicts a single iteration of the loop at Line 2, which shows that there are no other possible short error chains. The only short error chains are:

1. Z-error chains in the initialisation of the SC patch α;

2. Z-error chains between initialisation of SC patch α and the splitting of the SC patch into α_1, α_2, and α_3;

3. Z-error chains when measuring α_2;

4. parity measurement error-chains along the time axis in performing the rough merges between α_1 and ϕ and α_3 and ψ.

12

Algorithm 2 Fault-Tolerant Surface Code Bus XX measurement of width w

---

1: **procedure** PROTECTEDBUS-w($d \times d$ qubit SC patches ψ, ϕ indexed as $\theta_{[i,j]}$)
2:     **for** $i \in \{1 \ldots d\}$ **do**
3:         Create a $d_x = 2d + m$ and $d_z = w$ qubit surface code patch α, initialised to the logical
            $|+\rangle = \frac{1}{\sqrt{2}}(|1\rangle + |0\rangle)$ state. (if $w = 1$, this is exactly the GHZ state from Algorithm 1)
4:         **for** $j \in \{1 \ldots d\}$ **do**
5:             Measure and record the stabiliser operators of all 3 surface code patches.
6:         **end for**
7:         Split the surface code patch α into 3 patches with $d_z = w$, in order α_1 with $d_x = d$, α_2 with
            $d_x = m$ and α_3 with $d_x = d$, recording the measurements from the boundaries. (This leaves us
            in the logical GHZ state $\frac{1}{\sqrt{2}}(|111\rangle + |000\rangle)$).
8:         Measure the surface code patch α_2 in the Z basis leaving the logical state $\alpha_1\alpha_3 = \frac{1}{\sqrt{2}}(|11\rangle + |00\rangle)$
9:         Merge Surface code patch ψ with surface code patch α_1 along their rough boundaries creating
            $\psi\beta_1\alpha_1$. Where β_1 is initialised as $\bigotimes_{k=1}^{d-1} |+\rangle$ and is the boundary.
10:        Merge Surface code patch ϕ with surface code patch α_3 along their rough boundaries creating
            $\phi\beta_2\alpha_3$. Where β_2 id initialised as $\bigotimes_{k=1}^{d-1} |+\rangle$ is the boundary.
11:        **for** $j \in \{1 \ldots m\}$ **do**
12:            Measure and record the stabiliser operators of both surface code patches $\psi\alpha_1$ and $\psi\alpha_3$.
               and record all results.
13:        **end for**
14:        Restore the SC patches α and β to their original size by measuring out the area that was $\beta_1\alpha_1$
            and $\beta_2\alpha_3$.
15:        **for** $j \in \{1 \ldots d\}$ **do**   ▷ (You may merge this loop with that on line 4 of the
                                                             following enclosing loop
                                                             iteration (line 2) where it exists.)
16:            Measure and record the stabiliser operators of both ϕ and ψ.
17:        **end for**
18:        Update the Pauli frame predictions to correct for the split, and measure in lines 7-8 accounting
            for detectable errors.
19:        Determine the logical parity measurement p_{i1} associated with the merge on line 9 accounting for
            detectable errors.
20:        Determine the logical parity measurement p_{i2} associated with the merge on line 10 accounting
            for detectable errors.
21:        Compute $p_i = p_{i1} \oplus p_{i2}$
22:    **end for**
23:    Compute the majority vote over all p_i, that is the parity XX over ψ and ϕ; update the final Pauli
        Frame, and return.
24: **end procedure**

The ZX-calculus has been proposed as a language for the description of lattice surgery operations [6]. It can be used to show that this algorithm implements a parity measurement and that all errors appear as errors in the parity measurement, see the diagram in Figure 8.
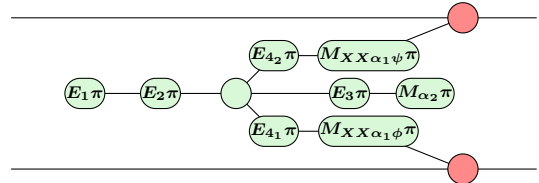

FIG. 8: A ZX-calculus diagram showing the effect of the possible short error chains on the result of the parity measurement.
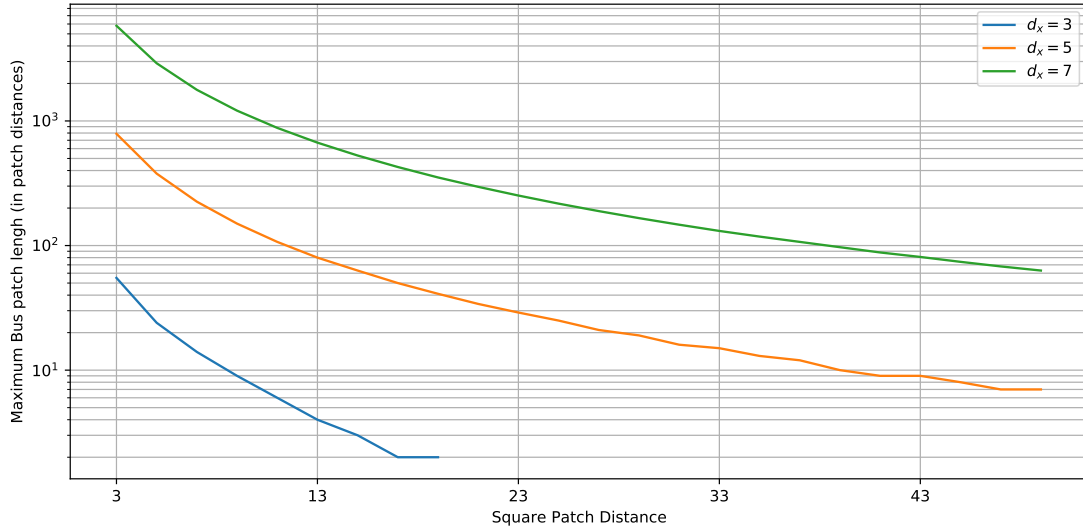
13

This diagram is equivalent to one of the loop iterations of the procedure in Algorithm 2, with all short error chain error possibilities included as rotations on the Z-nodes. These errors can be merged into a single Z-spider, with weight

$$(E_1 + E_2 + E_3 + E_{4_1} + E_{4_2} + \\ M_{\alpha_2} + M_{XX\alpha_1\phi} + M_{XX\alpha_1\psi})\pi. \quad (4)$$

Compare this to the diagram in Figure 9, which shows the ZX-diagram of an XX parity measurement with a possible error in measurement. Both these diagrams are equivalent under the mapping error

$$E = E_1 + E_2 + E_3 + E_{4_1} + E_{4_2} \mod 2$$

and parity measurement

$$M = M_{\alpha_2} + M_{XX\alpha_1\phi} + M_{XX\alpha_1\psi} \mod 2$$

because both can be contracted into a single Z-spider.

FIG. 9: ZX-calculus diagram of the measurement of the XX-parity between two qubits using CNOTs and an ancilla with a possible logical error.

The probability of an unrecoverable error in the quantum bus in any loop iteration is proportional to the probability of an uncorrectable short-edge error within the bus time/space volume during the course of a single bus cycle. Hence, the error is approximated as the probability of the second and fourth error channels, as the quantum space/time volume of this region is significantly larger than any other short error channel. Whether this error exceeds the repetition code's code capacity for a certain code distance, error likelihood or bus length determines the minimum practical width of the bus.

IV.3. Surface code bus performance

The performance of the surface code bus is understood through the analysis of the error behaviour of each inner parity measurement. This is done by determining the rates for each term in Equation 4. Terms E_1, E_2 and E_3 correspond to uncorrectable Z-errors in the rectangular surface code patch that spans the width of the bus. Likewise, Terms E_{4_1} and E_{4_2} correspond to uncorrectable errors in smaller rectangular surface code time/space volumes. The sum of the rates of these errors is then the probability of an inner parity measurement failing.

These error rates are exactly what was simulated in Section III. From these results, the rate of errors that occur in a single folded bus section can be determined. As the repetition code has a threshold of 0.5, it is trivial to check differing combinations of bus lengths, bus widths and surface code distances to determine if each bus operation is below this threshold. This check was performed, and the results are presented in Figure 10.

Whilst it is true that the total code presented in Algorithm 2 will be of distance d, the error rates on the bus will strongly depend on the width and the length of the bus. In order to ensure that the rate of errors on the bus is the same or lower than the rate of errors in the data logical qubits, we have evaluated the performance of the surface code bus at differing bus lengths, widths and surface code distances, and then determined the minimum number of repetitions that ensures the probability of an error on the bus is lower than the probability of a memory error. The results of this calculation are shown in Figure 11.

V. PARITY CODES FOR THE BUS

Our first attempt to reduce the width of lattice required to perform universal quantum computing by using the fault-tolerant surface code bus introduced in Section IV to mediate the interactions of logical qubits so that another code may be concatenated above it as illustrated in Figure 12.

With a single bus, such a layout should give

FIG. 10: Maximum bus lengths, for differing bus widths, and patch dimensions. Providing space between patches for bus access to each side as in Figure 12.

that code a threshold lower than when complete connectivity at the logical layer is possible but higher than what is required if the logical architecture of the qubits were to be nearest-neighbour linear interactions. This is because while we can interact any two qubits with each other, each linear section of the bus can only be used for one interaction at a time.

To determine whether this technique could have applicability, codes were selected to evaluate, for both performance and ease of implementation. A fault tolerance scheme was chosen for each code, and a decoder was designed as described below. Then the performance of each code was simulated using these choices under the timing constraints of the surface code bus lower layer. It is worth noting here, that while the bus can implement multi-qubit parity measurements, it is not known whether these can be used fault-tolerantly in this concatenation scheme, so parity measurements on the bus were limited to weight 2 in this consideration.

V.1. Choice of codes

There were three main considerations in choosing which block codes to test: the expected threshold of the different codes; the ease of implementing the codes, and the amount of scholarly literature that exists on the code performance. The flag-qubit method for fault-tolerance described in the section II.3 showed significant benefits in terms of reduced qubit count and circuit simplicity, which enabled a reduced threshold [14]. This choice left three codes with flag-qubit circuits within the published literature that could be easily tested: the $[\![5,1,3]\!]$ five qubit code; Steane's $[\![7,1,3]\!]$ code, and the $[\![15,7,3]\!]$ CSS code. As the five-qubit code historically demonstrated a much lower threshold than the other two codes [17], we analysed the threshold performance of both the $[\![7,1,3]\!]$ code and the $[\![15,7,3]\!]$ code to examine the tradeoff between qubit density and the total width of the processor array, w.

To choose which syndrome extraction circuit should be used for each code, we evaluated the total bus latency to extract all syndromes for each of the flagged extraction circuits found in the literature [14, 47]. The

15

(a) $w = 3$

FIG. 11: Performance of the Surface code bus for $w \in \{3, 5, 7\}$. In the top plot of each group we have the number of repetitions to make the short-edge error equal to the patch error for one bus cycle. In the middle Graph we have the total number of surface code cycles required for a single bus parity measurement. The lower graph shows the effective error rate on the surface code patch (solid), and bus operation(dotted) when using the number of repetitions.

16

(b) $w = 5$

FIG. 11: continued. Performance of the Surface code bus for $w \in \{3, 5, 7\}$. In the top plot of each group we have the number of repetitions to make the short-edge error equal to the patch error for one bus cycle. In the middle Graph we have the total number of surface code cycles required for a single bus parity measurement. The lower graph shows the effective error rate on the surface code patch (solid), and bus operation(dotted) when using the number of repetitions.

(c) $w = 7$

FIG. 11: continued. Performance of the Surface code bus for $w \in \{3, 5, 7\}$. In the top plot of each group we have the number of repetitions to make the short-edge error equal to the patch error for one bus cycle. In the middle Graph we have the total number of surface code cycles required for a single bus parity measurement. The lower graph shows the effective error rate on the surface code patch (solid), and bus operation(dotted) when using the number of repetitions.

18

FIG. 12: An descriptive illustration of what the Steane $[[7,1,3]]$ code on top of $d = 5$ surface code patches connected with a $w = 2$ bus might look like.

Steane $[[7,1,3]]$ code circuit in figure 13a had the lowest complexity by far, requiring only 18 bus cycles. There were two candidate circuits for the $[[15,7,3]]$. Both had the same bus latency, so the circuit in 13b was chosen because it was slightly easier to develop its decoder.

V.2. Decoder design

A decoder had to be designed for each of the top-level block codes after the codes and circuits had been chosen. As each code was distance three, a brute force analysis was feasible. We evaluated all weight-one Pauli errors after initialisation, on idle qubits, and before measurements, and all weight-two Pauli errors before CNOT gates for all gates in a round of fault-tolerant syndrome extraction. The measured error syndrome is recorded for each error, and the errors propagated through the syndrome extraction to create a decoder. The decoder has to provide a unique correction for each error syndrome, or at least leave an error that can be corrected in the next cycle.

For the Steane $[[7,1,3]]$ CSS code, a fault-tolerant round of syndrome extraction consists of either two or three complete sets of syndrome extraction. Each syndrome extraction is made up of three copies of the circuit in figure 13a, with the qubits permuted to extract syndromes 1 and 2 first, followed by syndromes 3 and 4, and finally syndromes 5 and 6. If two sets of measurements return identical syndromes, then the procedure is complete. Otherwise, an additional set of measurements is taken to complete the round of fault-tolerant extraction. The decoder for this fault-tolerant syndrome extraction was calculated manually, given the relatively short size of the decision tree required. The correctness of the decoder was checked automatically with an exhaustive search.

For the $[[15,7,3]]$ CSS code, a fault-tolerant round of syndrome extraction only involves one or two complete sets of syndrome extraction. Each is made up of four copies of the circuit in figure 13b, with the qubits permuted to extract the syndromes in ordered pairs, first with syndromes 1 and 2, then syndromes 3 and 4, and so on. In this case, a single set of mea-

(a) $[\![7,1,3]\!]$ flag circuit for one X and Z syndrome [47]

(b) $[\![15,7,3]\!]$ flag circuit for one X and Z syndrome [47]

FIG. 13: Flag qubit extraction circuits used for $[\![7,1,3]\!]$ and $[\![15,7,3]\!]$ codes

surements is required if the syndromes all return unchanged values (zero/plus), otherwise two sets of measurements were made. For this code, the decoder was too large to consider optimizing manually, so an automatic routine to extract and verify a decoding table was created.

V.3. Simulation Design

The codes were simulated with the balanced error model described in section II.1 to evaluate their performance. As the bus operations take d^2 time steps, the error rate in this simulation assumes that measurement and initialisation of logical ancilla patches are error-free operations, as they effectively take one round of syndrome extraction on the surface code layer. This is because initialization requires d rounds of syndrome extraction in the surface code to be fault-tolerant. However, these can occur simultaneously with the first bus operation, so do not introduce any additional errors or computation time.

The simulation for these codes was performed with a custom CHP simulator. We ran repeated syndrome extractions with perfect corrections applied between them until an uncorrectable error occurred. When evaluating the Steane $[\![7,1,3]\!]$ code after each round of syndrome extraction and correction, a copy of the simulated state was placed back into a $+1$

eigenstate of all the surface code stabilisers by performing syndrome extraction with errors disabled. The parity of the logical operator, in the appropriate basis, was then measured and compared to the initial state. To correct all order one errors, we determined that three rounds of syndrome extraction were required prior to measuring the eigenstate of the logical operator and compared it to the initial value for this implementation of the $[\![15,7,3]\!]$ code. In both cases, the mean and standard deviation of the number rounds before failure were computed and used to find the logical error rate.

The ± 1 eigenstates of Pauli-X, Pauli-Y or Pauli-Z were chosen as our initial encoded states for the $[\![7,1,3]\!]$ code. The states $|0\rangle^{\otimes 7}, |1\rangle^{\otimes 7}, |+\rangle^{\otimes 7}, |-\rangle^{\otimes 7}, |i+\rangle^{\otimes 7}$, and $|i-\rangle^{\otimes 7}$ were chosen as our initial encoded states for the $[\![15,7,3]\!]$ code, with the restriction of states here due to limited computing resources.

V.4. Simulation Results

For each physical error rate, 900 simulations to failure were computed. Then the mean of these gaussian samples was used to compute an estimator for the Bernoulli trials. The maximum likelihood estimator and the Bayesian posterior mean of the posterior distribution were both evaluated, and they pro-

FIG. 14: Flow chart for simulation of block code performance simulation.

duced almost identical results. The posterior mean and 3σ confidence interval were calculated assuming a uniform prior distribution, with the results presented in Figure 15. As initially expected, the 3σ confidence interval is approximately $\pm 10\%$ around the estimate. From these parameter estimates, a polynomial fit was manually extracted to degree-4 for each code. These are

$$
\begin{aligned}
p_{[\![7,1,3]\!]}(p) = \quad & 2.23 \times 10^4 p^2 - 3.5 \times 10^6 p^3 \\
& + 1.7 \times 10^8 p^4
\end{aligned}
$$
$$(5)$$

for the $[\![7,1,3]\!]$ code and

$$
\begin{aligned}
p_{[\![15,7,3]\!]}(p) = \quad & 8.00 \times 10^5 p^2 - 6.0 \times 10^8 p^3 \\
& + 14 \times 10^{10} p^4
\end{aligned}
$$
$$(6)$$

for the $[\![15,7,3]\!]$ code, which are also depicted in Figure 15.

From these plots, we can see that the Steane $[\![7,1,3]\!]$ code has a pseudo-threshold of approximately $p = 4.52 \times 10^{-5}$ when used with the surface code bus. Similarly the pseudo-threshold for the $[\![15,7,3]\!]$ code is approximately $p = 1.25 \times 10^{-6}$.

VI. PERFORMANCE OF THE CONCATENATED CODES

To evaluate the concatenated quantum error correcting codes, we must:

- Determine the performance of the base surface code layer, and evaluate the performance of two-qubit interactions.

- Determine a compilation scheme at the first layer for the $[\![7,1,3]\!]$ or $[\![15,7,3]\!]$ codes above the surface code.

- Determine a scheme to concatenate additional levels of the code above this in a self-similar manner.

- Calculate the performance of this scheme using these compilation strategies recursively.

We must set a target for final logical error performance to make width and area comparisons. We have chosen an error rate of 10^{-15} for a logical CNOT or parity measurement operation between two adjacent qubits as our target in this work.

The performance is evaluated at several physical error rates between 10^{-3} and 10^{-4}, which represent both near, and intermediate-term physical error targets. We then determine the required architecture size for these rates. These results are presented in Table IV and are calculated using the procedures described in the remainder of this section.

(a) $[\![7,1,3]\!]$ code

(b) $[\![15,7,3]\!]$ code

FIG. 15: The logical error rates for the $[\![7,1,3]\!]$ and $[\![15,7,3]\!]$ codes

	d_{sc}	d_b	w	p_l	block size	qubit density
Surface Code - No Bus	27	NA	107	5.49×10^{-16}	5778	5778
Surface Code - Bus	31	5	71	1.66×10^{-16}	5112	5112
1L $[\![7,1,3]\!]$ on SC + Bus	21	7	55	3.96×10^{-17}	3.39×10^4	3.39×10^4
2L $[\![7,1,3]\!]$ on SC + Bus	15	7	43	1.91×10^{-16}	1.87×10^5	1.87×10^5
3L $[\![7,1,3]\!]$ on SC + Bus	13	7	39	4.64×10^{-16}	1.39×10^6	1.39×10^6
7L $[\![7,1,3]\!]$ on SC + Bus	11	7	35	8.27×10^{-21}	7.37×10^9	7.37×10^9
1L $[\![15,7,3]\!]$ on SC + Bus	23	7	59	2.74×10^{-17}	7.08×10^4	1.01×10^4
2L $[\![15,7,3]\!]$ on SC + Bus	21	7	55	2.50×10^{-20}	1.11×10^6	2.26×10^4
3L $[\![15,7,3]\!]$ on SC + Bus	19	7	51	9.73×10^{-21}	1.72×10^6	5.01×10^4
4L $[\![15,7,3]\!]$ on SC + Bus	17	7	47	3.84×10^{-17}	2.63×10^8	1.10×10^5

(a) $p = 1.0 \times 10^{-3}$

	d_{sc}	d_b	w	p_l	block size	qubit density
Surface Code - No Bus	21	NA	83	3.23×10^{-16}	3652	3652
Surface Code - Bus	23	5	55	3.26×10^{-16}	2450	2450
1L $[\![7,1,3]\!]$ on SC + Bus	15	5	39	5.11×10^{-16}	1.72×10^4	1.72×10^4
2L $[\![7,1,3]\!]$ on SC + Bus	11	5	35	3.56×10^{-16}	9.82×10^4	9.82×10^4
4L $[\![7,1,3]\!]$ on SC + Bus	9	5	27	4.31×10^{-19}	6.06×10^6	6.06×10^6
1L $[\![15,7,3]\!]$ on SC + Bus	17	5	43	6.59×10^{-17}	3.78×10^4	5.41×10^3
2L $[\![15,7,3]\!]$ on SC + Bus	15	5	39	2.94×10^{-18}	5.62×10^5	1.15×10^4
4L $[\![15,7,3]\!]$ on SC + Bus	13	5	35	4.34×10^{-16}	8.16×10^6	2.38×10^4

(b) $p = 4.7 \times 10^{-4}$

	d_{sc}	d_b	w	p_l	block size	qubit density
Surface Code - No Bus	17	NA	67	2.48×10^{-16}	2278	2278
Surface Code - Bus	19	3	43	1.40×10^{-16}	1892	1892
1L $[\![7,1,3]\!]$ on SC + Bus	13	3	31	5.07×10^{-17}	1.09×10^4	1.09×10^4
2L $[\![7,1,3]\!]$ on SC + Bus	9	5	27	6.43×10^{-18}	7.48×10^4	7.48×10^4
4L $[\![7,1,3]\!]$ on SC + Bus	7	5	23	9.82×10^{-22}	4.43×10^6	4.43×10^6
1L $[\![15,7,3]\!]$ on SC + Bus	13	5	35	1.50×10^{-16}	2.52×10^4	3.60×10^3
2L $[\![15,7,3]\!]$ on SC + Bus	13	3	31	1.54×10^{-19}	3.57×10^5	7.29×10^3
3L $[\![15,7,3]\!]$ on SC + Bus	11	3	27	3.26×10^{-17}	4.90×10^6	1.43×10^4

(c) $p = 2.2 \times 10^{-4}$

	d_{sc}	d_b	w	p_l	block size	qubit density
Surface Code - No Bus	15	NA	59	2.59×10^{-17}	1770	1770
Surface Code - Bus	15	3	35	4.93×10^{-16}	1260	1260
1L $[\![7,1,3]\!]$ on SC + Bus	11	3	27	2.46×10^{-18}	8.32×10^3	8.32×10^3
2L $[\![7,1,3]\!]$ on SC + Bus	7	3	23	6.78×10^{-16}	3.76×10^4	3.76×10^4
9L $[\![7,1,3]\!]$ on SC + Bus	5	3	19	5.10×10^{-22}	1.14×10^{11}	1.14×10^{11}
1L $[\![15,7,3]\!]$ on SC + Bus	11	3	27	1.67×10^{-16}	1.51×10^4	2.16×10^3
3L $[\![15,7,3]\!]$ on SC + Bus	9	3	23	1.44×10^{-20}	3.58×10^5	1.04×10^4

(d) $p = 1.0 \times 10^{-4}$

TABLE IV: Error rates for different code concatenation configurations, parameters, and physical error rates. In this figure d_{sc} is the surface code distance, d_b is the bus width distance, w is the width of the lattice, and p_l is the logical error rate of the concatenated code.

VI.1. Performance of the surface code, and the surface code bus.

The performance of the surface code is determined to provide a base for comparison, which is required when establishing the performance of the concatenated codes. Here, we estimate the error rate of the surface code using the fit in Equation 3 multiplied by $d+1$, the number of rounds required for a lattice surgery operation.

We use the procedure described in Section IV.3 to calculate the estimate of the bus error rate using the simplifying assumption that all bus operations have the worst-case length. These bus operations are then used as the underlying physical error rate for concatenated codes, which will result in slightly higher logical error rates than a tighter analysis using exact lengths.

VI.2. Compilation of the Steane $[\![7,1,3]\!]$ code

A scheme for concatenation is required when concatenating multiple layers of the Steane $[\![7,1,3]\!]$ code with the surface code bus. Logical CNOTs are performed transversally between the two logical blocks at each level of concatenation below the surface code. This requires at least three CNOT gates between qubits in the two logical blocks. There are several ways of doing this including those shown in Figure 16.

1. For one level (1L) of concatenation of the Steane code above the surface code, we use the bus to perform logical CNOTs between surface code patches. We arrange the qubits as in Figure 12, with CNOT ancillae adjacent to each parity ancilla. This layout enables some minor parallel bus operations. We allocate additional time to enable CNOTS between patches in different logical blocks and perform transversal CNOTs gates between logical patches as in Figure 16. The existing ancillae, both CNOT ancillae and code ancillae, are reused for this purpose.

2. For two levels (2L) of concatenation, logical operations at both levels are performed using the bus. We allocate adequate time in each L1 syndrome extraction to perform a long-distance bus operation between any two of the L1 qubits within the block, as described above. CNOTs between two adjacent L2 CNOT qubits are performed using long bus operations, as are CNOT gates between adjacent 2L qubits. Swaps between 2L qubits are performed by using transversal quantum teleportation of 1L qubits. This ensures that no single bus error can create more than a weight-1 logical error at layer 3 of concatenation.

3. For three levels (3L) of concatenation or higher, a swap network of sufficient length is used to make interacting lower-level qubits adjacent to each other in each concatenated layer. This swap procedure is then applied recursively.

In our analysis, we increase the error rate of the code at each layer of concatenation in proportion to the additional time required to perform operations between blocks. This provides an estimate of the error rate with these interactions. We then apply the equation of fit for the performance of the $[\![7,1,3]\!]$ code. This procedure is performed recursively. To determine the minimum possible combined lattice width that meets the 10^{-15} performance target, we iteratively tried each possible bus width and code distance. For each of these, we then determined if it was possible to reach the target and, if so, how many layers of concatenation were required. The narrowest lattices for each number of layers of concatenation that reach our target were then recorded.

VI.3. Compilation of the $[\![15,7,3]\!]$ CSS code

A concatenation scheme for multiply concatenating the $[\![15,7,3]\!]$ code must be determined in order to concatenate the $[\![15,7,3]\!]$ code above the surface code with bus multiple times. We have chosen the simplest scheme in this work, where we concatenate fifteen (15)

(a) Patch interactions required to perform logical Steane operations using the $XXXIIII$ or $ZZZIIII$ logical operator, between adjacent Steane arrays.

(b) Patch interactions required to perform logical Steane operations using the $XIIIIXX$ or $ZIIIIZZ$ logical operator, between adjacent Steane arrays.

FIG. 16: Logical operators between order reversed Steane qubits on the bus.

level one (L1) code blocks together into a new L2 code block of $7 \times 7 = 49$ Level zero L0 logical surface code qubits. This is possible because a transversal CNOT operation of fifteen (15) pairs of qubits between two blocks is equivalent to performing a CNOT between all corresponding logical qubits in each block [13]. This work does consider CNOT gates between differing forty-nine (49) L0 logical qubit code blocks on the same qubit. However, we have not considered how to fault-tolerantly implement the internal Clifford group (that is the Clifford group on the seven qubits within a single $[\![15, 7, 3]\!]$ code block) at any level of concatenation, and the implementation of this may reduce the threshold for this code.

The concatenation scheme is then as follows.

1. For the first level of concatenation, we perform CNOT operations between code blocks by performing CNOTs in parallel, reusing the ancilla qubits for the syndrome extraction in order to increase bus utilization and perform a CNOT between two L1 code blocks. This can be done in eighteen (18) bus operations. When concatenated again for a higher level code, we perform swaps using the bus to teleport into the ancilla qubits between the blocks, this can be done in thirty-six (36) bus operations. This is done so that no operation can cause a

weight 2 error.

2. For the second and higher levels of concatenation, a swap network of sufficient length is then used to make interacting lower-level qubits adjacent in each concatenated layer. After swapping, the CNOT is applied transversally, and then the qubits are swapped back. This swap procedure is applied recursively at each successive layer.

As with the Steane code, to determine the performance of the concatenated code, the error rate of each layer of concatenation is increased proportionately with the additional time required to perform two qubit interactions before applying the equation of fit. After which, a search was performed to determine the performance of the code.

VI.4. Evaluation of results

It can be seen from the results in Table IV that, in most cases, the overall width w can be reduced by about half that required for the surface code with bus, or about a third of the surface code alone. This required the number of qubits per block to increase by about three orders of magnitude when the physical error rate is 10^{-3}. By increasing the width slightly,

to about 70% of that required for the surface code with bus, this penalty is reduced substantially, to only a factor of four over the standard approach. It is possible to have a reduction whilst retaining more density through the use of the $[\![15, 7, 3]\!]$, at the cost of more complex compilation and a larger smallest unit size. It may be possible to increase density or width further with different choices of block codes.

VII. REPETITION CODES ABOVE BIASSED SURFACE CODES

The approach of applying a quantum block code directly above the surface code appears to be insufficient for further reduction of the array width w, and so a new approach is required for further improvement. As you increase the length of a surface code patch, one error type is suppressed exponentially, while the other only grows linearly. It is known [7, 35, 55, 56] that highly biased error models can greatly increase the performance of error correction. A further reduction in width may be possible by engineering a highly biased logical error model using a highly asymmetric surface code patch and then concatenating it with a repetition code. These rectangular surface code patches can be much narrower, as the threshold of the repetition code is much higher than that of a code that corrects for all possible quantum errors.

Whilst we might think of looking at more complicated codes than the quantum repetition code because we wish to minimise array widths, we are limited by the complexity of more advanced codes. The most space-efficient method to mediate long-distance interactions between these patches is a single surface code bus. Such a bus must grow in width as interaction distance increases, and each bus can only perform one stabiliser measurement at a time. By measuring the parity of neighbouring qubits, the quantum repetition code can be implemented in constant time for increasing code distance. This reduces the requirements for the performance of the biased surface code patches in our proposed architecture, allowing the patches to be narrower.

VII.1. Rectangular surface codes and creating biased logical errors

The repetition code is a classical code, and so only corrects for one type of quantum error—either of type X, or of type Z. An error of the other type in any classical error-correcting code is never corrected. To be able to use the repetition code as a layer two code, one type of error must be suppressed well below the target rate of the layer three code. This is precisely what we engineer using a highly asymmetric surface code layer, forcing one of the error rates to be many orders of magnitude lower than the other.

This may be achieved using a highly rectangular surface code. The probability of a logical error in a surface code decreases exponentially with the length of the shortest possible uncorrectable error chain. As this length scales proportionally to the length of the shortest logical operator, we can engineer an exponential bias in logical error rates by using rectangular surface patches. Assume, without loss of generality, that the Z logical operator is along the longer edge of a rectangular surface code patch. The effect of increasing this longer patch is to reduce the possibility of a Z logical error exponentially due to the increased number of errors required to form an error chain. The probability of an X logical error only increases linearly though, as the length of the X error chains required to form a logical error does not change, only the number of possible chains does. This means that the probability of an X error is approximately $\frac{d_Z}{d_X} \cdot p_{l_X}(d_X, d_X)$ [38] for a rectangular surface code patch. This does, however, ignore some edge effects, so the simulations of III are required for more accurate analysis.

The rectangular surface codes may then be concatenated with the repetition code using lattice surgery. This is a delicate task, however, as it is essential that you never reduce the minimum distance between two Z boundaries when working with rectangular surface code patches, which increases the probability of a logical Z error exponentially. Nonetheless, it is possible to measure the logical Z parity of two adjacent surface code patches within a repetition code. In Appendix A we present a

scheme for performing such a parity measurement, and we also offer an analysis of the time needed to perform a parity measurement between two adjacent rectangular surface code qubits in approximately $9d_Z$ time steps. Using the surface code bus, each syndrome measurement takes at least $d_Z(d_Z + 1) + 1$ time steps. This results in a total code time of $2(d_Z^2 + d_Z + 1)$, as we require two sets of measurements to implement the repetition code.

As we can engineer rectangular surface code patches with biased logical errors and can perform logical parity measurements between them, we are able to implement a repetition code on top of these biased surface code patches. By then concatenating these logical repetition code qubits with other codes, such as the Steane $[\![7, 1, 3]\!]$ code, we are able to reach an arbitrary logical error rate. As Hadamard and phase gates are difficult to perform on these rectangular qubits in a fault-tolerant way, compilation above the error-correcting code level is expected to be performed using either gate teleportation or measurement-based techniques [37]. The repetition code level would result in a scheme depicted in Figure 17.

VII.2.   Evaluation of results

To evaluate the performance of this scheme, we must understand the effect of different values of d_Z and d_X on the time required to extract one round of parity measurements in the repetition code.

We assume that a lattice surgery is used when performing an operation between two repetition codes. To measure the ZZ parity between two repetition codes, we must move the patches so that the qubits of one repetition code are interdigitated with those of the other whilst simultaneously extracting the repetition code stabilizers. The two overlapping syndrome extractions are required when performing a logical CNOT gate or swapping two repetition code qubits.

For bussed extraction, this takes at least $5(d_Z^2 + d_Z + 1)$ cycles in the worst case, where two logical qubits must be moved past each other during a computation. (A computation consists of two parity extractions plus one quantum teleportation [19]). For the non-bussed method, it takes $18d_Z$ surface code cycles to perform a round of syndrome extraction in the same situation. This is due to the complicated series of moves that must be performed to enable qubits to move past each other while simultaneously performing each of the two repetition codes. A description of the lattice surgery operations can be found in Appendix A.

We then multiply the per-time-step rates for each surface code patch during the lattice surgery procedures to determine the expected logical rates for one cycle of syndrome extraction of the repetition code. We call this the effective p_X and p_Z for a given configuration of bussed/non-bussed operation, and code distances d_X and d_Z.

We then use these error rates to calculate the probability of an error in the repetition code of distance d_{rc}, which is made up of d_{rc} patches. To estimate the logical fidelity of the X errors after the repetition code, we use the CDF of the binomial probability for having more than $\lceil n/2 \rceil - 1$, or more errors within the repetition code block. We do this for each block size until the repetition code has a lower logical X error probability than the Z errors. To estimate the rate of Z errors, we multiply the effective p_Z by d_{rc}, as the repetition code is only correcting for X errors. Then, to get an effective error rate per CNOT gate, we have to multiply this by $2d_{rc} + 1$, as it takes that many repetition code cycles to perform a logical CNOT operation.

If we evaluate the results for each value of d_X, we can determine a physical error rate that enables both X and Z errors to be below the threshold for the Steane $[\![7, 1, 3]\!]$ code presented earlier. You can see the extrapolated logical CNOT error rate for the non-bussed implementation after the repetition code in Figure 18. If we take $p = 10^{-6}$ as the threshold on this architecture, then the threshold for $d_X = 3$ is approximately 2×10^{-4}; for $d_X = 5$, it is approximately 8×10^{-4}, and for $d_X = 7$, it is approximately 1.5×10^{-3}. Architectures for these can be implemented in arrays with widths of 11, 19, and 27 respectively. Whilst the number of qubits required is extremely

(a) Physical qubits, $p_X = 10^{-4}$, $p_Z = 10^{-4}$, Perhaps silicon quantum dot qubits similar to these [25] (Image used with the permission of the authors)

(b) A rectangular surface code patch with $d = 2$ bus, $d_X = 3$, $d_Z = 19$, 369 qubits total. With the qubits in Fig 17a $p_X < 0.158$, $p_Z < 6.91 \times 10^{-21}$.
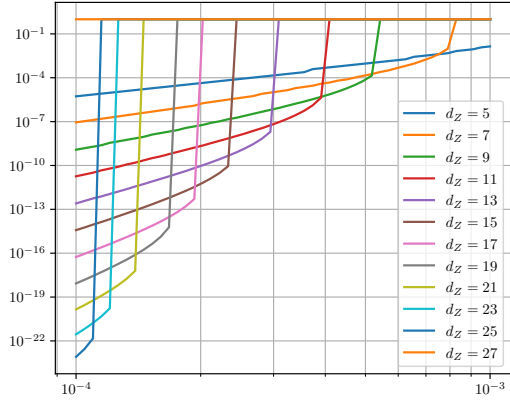

(c) A repetition code made up of surface code patches, with 123 surface patches as in Fig 17b, $p_x < 1.39 \times 10^{-16}$, $p_Z < 2.10 \times 10^{-16}$.

FIG. 17: An example of concatenation levels for biased repetition code, with one possible configuration of height and physical error rate.

high at the threshold, it comes down quickly as you move away from that point. If you compare these values to those estimated for standard CSS codes on square surface code patches presented in Section VI, you can see that narrower widths are possible at a given physical error rate and, at certain widths, fewer qubits are required for the same logical error rate. This is traded against the unknown complexity of compiling for this architecture.

VIII. CONCLUSION

In this work, we have investigated the possibility of reducing interconnect density by restricting the maximum width w of the required qubit array that is needed to accommodate the surface code lattice. Previous attempts to deal with the problem of interconnect density either required more complicated long-distance interactions [40] or had a much lower code threshold [31, 53] than found in this work.

(a) Regularized p_l for $d_X = 3$

(b) Qubit counts for $d_X = 3$

(c) Regularized p_l for $d_X = 5$.

(d) Qubit counts for $d_X = 5$.

(e) Regularized p_l for $d_X = 7$

(f) Qubit counts for $d_X = 7$

FIG. 18: Plots of the regularized CNOT fidelity and qubit counts after the repetition code for differing biassed patch distances.

29

Here we chose to investigate using the surface code as a base layer for other codes.

In order to evaluate this, we first simulated the performance of rectangular surface code patches, giving a collection of polynomial fits. We then, in section IV, presented a fully fault-tolerant version of a folded surface code bus that enables long-distance qubit parity measurements with much lower overhead than other methods. We used the rectangular simulations of Section III to evaluate the performance of this in differing contexts.

We then determined the performance of both the $[[15, 7, 3]]$ and Steane $[[7, 1, 3]]$ codes with the recently described flag-qubit fault-tolerance technique. We simulated these codes with randomised depolarised noise. We then evaluated the results to determine the expected performance of architectures using these codes directly concatenated with the surface code, where long-distance CNOTs between surface code patches are mediated by the folded surface code bus. This gave an array width of 35 when the error rate $p = 10^{-3}$, and 19 when $p = 10^{-4}$, compared to widths of 71 and 35 respectively without concatenation.

Finally, we explored the performance when deliberately biassing the logical error rate and then regularising the logical errors, estimating the width of the minimum qubit performance required to meet the threshold of higher level codes. This showed that at $p < 1.5 \times 10^{-3}$ and $p < 0.8 \times 10^{-3}$ array widths of 27 and 19 respectively should suffice.

We also developed several tools for the evaluation of quantum error correcting codes. While better tools have become available during the progress of this work, especially the excellent stim tool and the increasingly good pymatching decoder, the tools used along with all data we have generated are available from the authors upon request[49].

The authors would have liked to evaluate other codes, especially the $[[23, 1, 7]]$ Golay code, LDPC codes, and concatenated $[[4, 1, 2]]$

subsystem codes. However, the difficulty of defining efficient flagged fault-tolerant extractions meant that this would have to be done in future work. Similarly, it would be interesting to investigate the compilation and evaluation of the performance of the logical codes above the repetition code above deliberately biased surface codes, as well as the evaluation of the performance of other topological codes in place of the surface code, such as the XZZX code [7] and the honeycomb code [24], and their impact on minimum bus width and interconnect density.

[1] Aaronson, S.and Gottesman, D., Phys. Rev. A **70**, 052328 (2004).

[2] Aharonov, D.and Ben-Or, M., in *Proceedings of the Twenty-Ninth Annual ACM Sym-*

posium on Theory of Computing, STOC '97 (Association for Computing Machinery, New York, NY, USA, 1997) p. 176–188.

[3] Ansaloni, F., Chatterjee, A., Bohuslavskyi, H., Bertrand, B., Hutin, L., Vinet, M., and Kuemmeth, F., Nature Communications **11**, 6399 (2020).

[4] Azad, U., Lipińska, A., Mahato, S., Sachdeva, R., Bhoumik, D., and Majumdar, R., IET Quantum Communication **3**, 174 (2022).

[5] Backens, M., New Journal of Physics **16**, 093021 (2014).

[6] de Beaudrap, N.and Horsman, D., Quantum **4**, 218 (2020).

[7] Bonilla Ataides, J. P., Tuckett, D. K., Bartlett, S. D., Flammia, S. T., and Brown, B. J., Nature Communications **12**, 2172 (2021).

[8] Bourassa, J. E., Alexander, R. N., Vasmer, M., Patil, A., Tzitrin, I., Matsuura, T., Su, D., Baragiola, B. Q., Guha, S., Dauphinais, G., Sabapathy, K. K., Menicucci, N. C., and Dhand, I., Quantum **5**, 392 (2021).

[9] Bravyi, S., Englbrecht, M., König, R., and Peard, N., npj Quantum Information **4**, 55 (2018).

[10] Breuckmann, N. P.and Eberhardt, J. N., PRX Quantum **2**, 040101 (2021).

[11] Cai, Z., Fogarty, M. A., Schaal, S., Patomäki, S., Benjamin, S. C., and Morton, J. J. L., Quantum **3**, 212 (2019).

[12] Calderbank, A. R.and Shor, P. W., Phys. Rev. A **54**, 1098 (1996).

[13] Chao, R.and Reichardt, B. W., npj Quantum Information **4**, 42 (2018).

[14] Chao, R.and Reichardt, B. W., Phys. Rev. Lett. **121**, 050502 (2018).

[15] Chao, R.and Reichardt, B. W., PRX Quantum **1**, 010302 (2020).

[16] Coecke, B.and Duncan, R., in *Automata, Languages and Programming*, edited by L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz (Springer Berlin Heidelberg, Berlin, Heidelberg, 2008) pp. 298–310.

[17] Cross, A. W., Divincenzo, D. P., and Terhal, B. M., Quantum Info. Comput. **9**, 541–572 (2009).

[18] Devitt, S. J., Greentree, A. D., Stephens, A. M., and Van Meter, R., Scientific Reports **6**, 36163 (2016).

[19] Erhard, A., Poulsen Nautrup, H., Meth, M., Postler, L., Stricker, R., Stadler, M., Negnevitsky, V., Ringbauer, M., Schindler, P., Briegel, H. J., Blatt, R., Friis, N., and Monz, T., Nature **589**, 220 (2021).

[20] Fowler, A. G.and Gidney, C., "Low overhead quantum computation using lattice surgery," (2018).

[21] Fowler, A. G., Mariantoni, M., Martinis, J. M., and Cleland, A. N., Phys. Rev. A **86**, 032324 (2012).

[22] Fowler, A. G., Whiteside, A. C., McInnes, A. L., and Rabbani, A., Phys. Rev. X **2**, 041003 (2012).

[23] Gidney, C., Quantum **5**, 497 (2021).

[24] Gidney, C., Newman, M., Fowler, A., and Broughton, M., Quantum **5**, 605 (2021).

[25] Gilbert, W., Tanttu, T., Lim, W. H., Feng, M., Huang, J. Y., Cifuentes, J. D., Serrano, S., Mai, P. Y., Leon, R. C. C., Escott, C. C., Itoh, K. M., Abrosimov, N. V., Pohl, H.-J., Thewalt, M. L. W., Hudson, F. E., Morello, A., Laucht, A., Yang, C. H., Saraiva, A., and Dzurak, A. S., "On-demand electrical control of spin qubits," (2022).

[26] Hakkaku, S., Mitarai, K., and Fujii, K., Phys. Rev. Research **3**, 043130 (2021).

[27] He, Y., Gorman, S. K., Keith, D., Kranz, L., Keizer, J. G., and Simmons, M. Y., Nature **571**, 371 (2019).

[28] Herr, D., Paler, A., Devitt, S. J., and Nori, F., "Time versus hardware: Reducing qubit counts with a (surface code) data bus," (2019), arxiv:1902.08117, arXiv:1902.08117 [quant-ph].

[29] Horsman, C., Fowler, A. G., Devitt, S., and Meter, R. V., New Journal of Physics **14**, 123011 (2012).

[30] Jeandel, E., Perdrix, S., and Vilmart, R., in *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18 (Association for Computing Machinery, New York, NY, USA, 2018) p. 559–568.

[31] Jones, C., Fogarty, M. A., Morello, A., Gyure, M. F., Dzurak, A. S., and Ladd, T. D., Phys. Rev. X **8**, 021058 (2018).

[32] Kempe, J., Bacon, D., Lidar, D. A., and Whaley, K. B., Phys. Rev. A **63**, 042307 (2001).

[33] Kitaev, A., Annals of Physics **303**, 2 (2003).

[34] Kolmogorov, V., Mathematical Programming Computation **1**, 43 (2009).

[35] Lee, J., Park, J., and Heo, J., Quantum Information Processing **20**, 231 (2021).

[36] Lekitsch, B., Weidt, S., Fowler, A. G., Mølmer, K., Devitt, S. J., Wunderlich, C., and Hensinger, W. K., Science Advances **3**, e1601540 (2017).

[37] Litinski, D., Quantum **3**, 128 (2019).

[38] Litinski, D., Quantum **3**, 205 (2019).

[39] Lloyd, S., Science **261**, 1569 (1993).

[40] Mohiyaddin, F., Li, R., Brebels, S., Simion, G., Dumoulin Stuyck, N. I., Godfrin, C., Shehata, M., Elsayed, A., Gys, B., Kubicek, S., Jussot, J., Canvel, Y., Massar, S., Weckx, P., Matagne, P., Mongillo, M., Govoreanu, B., and Radu, I. P., in *2021 IEEE International Electron Devices Meeting (IEDM)* (2021) pp. 27.5.1–27.5.4.

[41] Nagayama, S., Fowler, A. G., Horsman, D., Devitt, S. J., and Meter, R. V., New Journal of Physics **19**, 023050 (2017).

[42] Nielsen, M. A.and Chuang, I. L., *Quantum Computation and Quantum Information: 10th Anniversary Edition* (Cambridge University Press, 2010).

[43] This method was determined in discussions involving the authors of the Herr paper and Craig Gidney, and is presented here with their consent.

[44] O'Gorman, J.and Campbell, E. T., Phys. Rev. A **95**, 032338 (2017).

[45] Prabhu, P.and Reichardt, B. W., in *16th Conference on the Theory of Quantum Computation, Communication and Cryptography (TQC 2021)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 197, edited by M.-H. Hsieh (Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2021) pp. 5:1–5:13.

[46] Raussendorf, R.and Harrington, J., Phys. Rev. Lett. **98**, 190504 (2007).

[47] Reichardt, B. W., Quantum Science and Technology **6**, 015007 (2020).

[48] Roetteler, M., Naehrig, M., Svore, K. M., and Lauter, K., in *Advances in Cryptology – ASIACRYPT 2017*, edited by T. Takagi and T. Peyrin (Springer International Publishing, Cham, 2017) pp. 241–270.

[49] Shaw, A., "Tools and data for the paper "quantum computation on a 19-qubit wide 2d nearest neighbour qubit array."." `https://github.com/alexisshaw/RibbonPaper` (2022).

[50] Shor, P., in *Proceedings of 37th Conference on Foundations of Computer Science* (1996) pp. 56–65.

[51] Shor, P. W., Phys. Rev. A **52**, R2493 (1995).

[52] Steane, A. M., Phys. Rev. Lett. **77**, 793 (1996).

[53] Stephens, A. M.and Evans, Z. W. E., Phys. Rev. A **80**, 022313 (2009).

[54] Tomita, Y.and Svore, K. M., Phys. Rev. A **90**, 062320 (2014).

[55] Tuckett, D. K., Bartlett, S. D., and Flammia, S. T., Phys. Rev. Lett. **120**, 050505 (2018).

[56] Tuckett, D. K., Darmawan, A. S., Chubb, C. T., Bravyi, S., Bartlett, S. D., and Flammia, S. T., Phys. Rev. X **9**, 041031 (2019).

[57] Vandersypen, L. M. K., Bluhm, H., Clarke, J. S., Dzurak, A. S., Ishihara, R., Morello, A., Reilly, D. J., Schreiber, L. R., and Veldhorst, M., npj Quantum Information **3**, 34 (2017).

[58] Veldhorst, M., Eenink, H. G. J., Yang, C. H., and Dzurak, A. S., Nature Communications **8**, 1766 (2017).

[59] Wang, D. S., Fowler, A. G., and Hollenberg, L. C. L., Phys. Rev. A **83**, 020302 (2011).

[60] van de Wetering, J., "Zx-calculus for the working quantum computer scientist," (2020), arxiv:2012.13966, arXiv:2012.13966 [quant-ph].

[61] Xue, X., Russ, M., Samkharadze, N., Undseth, B., Sammak, A., Scappucci, G., and Vandersypen, L. M. K., Nature **601**, 343 (2022).

[62] Yang, C. H., Leon, R. C. C., Hwang, J. C. C., Saraiva, A., Tanttu, T., Huang, W., Camirand Lemyre, J., Chan, K. W., Tan, K. Y., Hudson, F. E., Itoh, K. M., Morello, A., Pioro-Ladrière, M., Laucht, A., and Dzurak, A. S., Nature **580**, 350 (2020).

[63] Zwerver, A. M. J., Krähenmann, T., Watson, T. F., Lampert, L., George, H. C., Pillarisetty, R., Bojarski, S. A., Amin, P., Amitonov, S. V., Boter, J. M., Caudillo, R., Correas-Serrano, D., Dehollain, J. P., Droulers, G., Henry, E. M., Kotlyar, R., Lodari, M., Lüthi, F., Michalak, D. J., Mueller, B. K., Neyens, S., Roberts, J., Samkharadze, N., Zheng, G., Zietz, O. K., Scappucci, G., Veldhorst, M., Vandersypen, L. M. K., and Clarke, J. S., Nature Electronics **5**, 184 (2022).

Appendix A: A lattice surgery construction for non-bus repetiton codes on rect patches.

In order to perform non-bussed extraction of the repetition code in approximately $9d_Z + 4$ time steps × qubit area per qubit patches , one can use the series of lattice surgery in the figure below, one logical qubit is indexed by roman numerals, the other with letters. The sequence has the effect of shifting the repetition code qubits to the right every extraction. By taking the mirror image of the sequence a shift to the left could be achieved. To have

no-net movement extractions should alternate between these two sets. The sequence also only extracts the repetition code for one of the two sets of qubits, to perform the extraction for both one should perform the sequence once for each of the repetition code qubits this then takes $18d_Z + 8$.

To perform a cnot one then inter-digitates two repetition codes using the lattice surgery procedure, and performs either an XX or ZZ parity measurement between each of the individual sub-patches in the lattice.
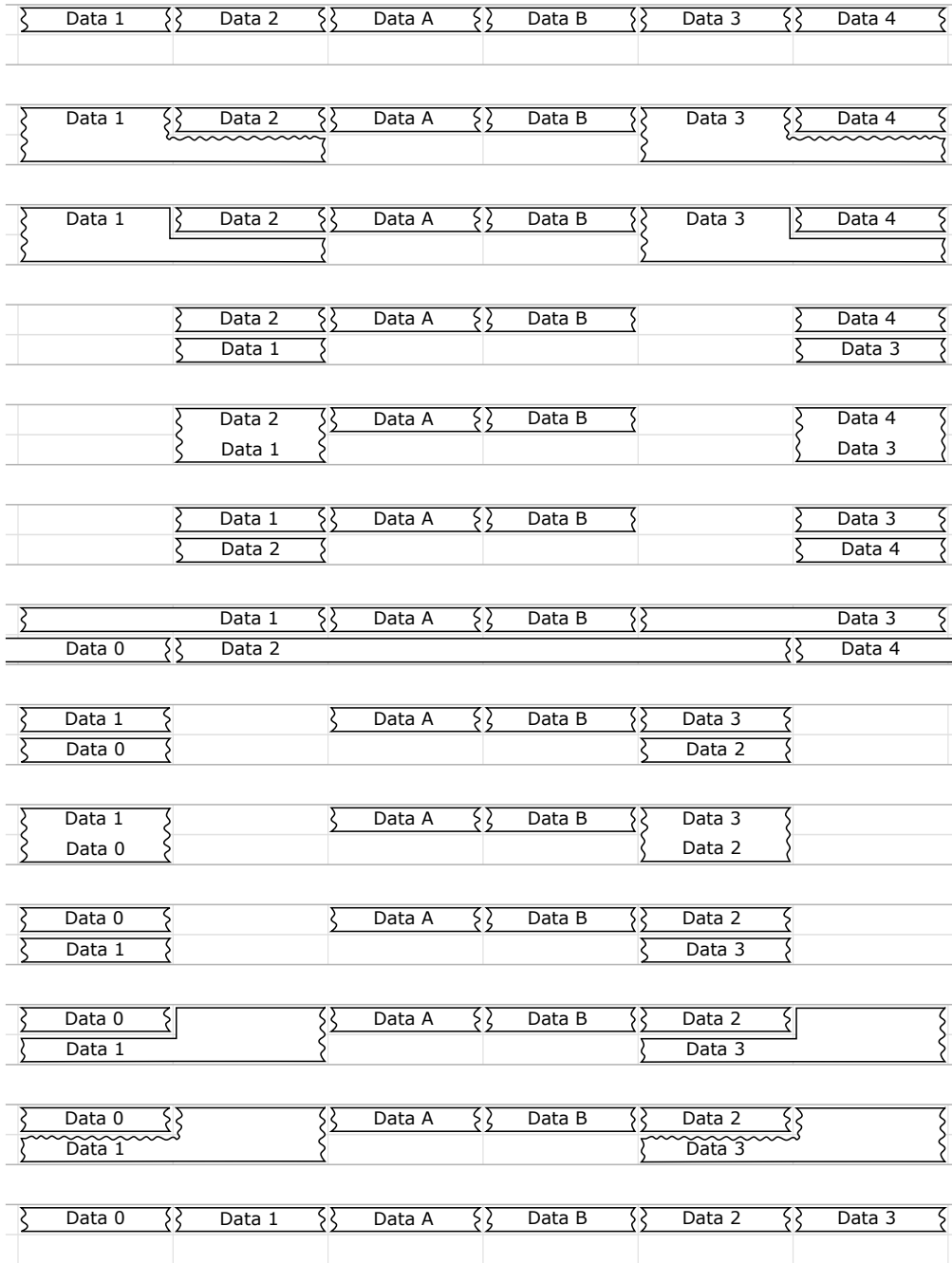
FIG. 19: Lattice surgery instructions pt 1