PAPER

Quantum error correction in crossbar architectures

View the article online for updates and enhancements.

Quantum Science and Technology

PAPER

Quantum error correction in crossbar architectures

Jonas Helsen[1] , Mark Steudtner[1,2], Menno Veldhorst[1,3] and Stephanie Wehner[1]

[1] QuTech, Delft University of Technology, Lorentzweg 1, 2628 CJ Delft, The Netherlands
[2] Instituut-Lorentz, Universiteit Leiden, P.O. Box 9506, 2300 RA Leiden, The Netherlands
[3] Kavli Institute of Nanoscience, Delft University of Technology, P.O. Box 5046, 2600 GA Delft, The Netherlands

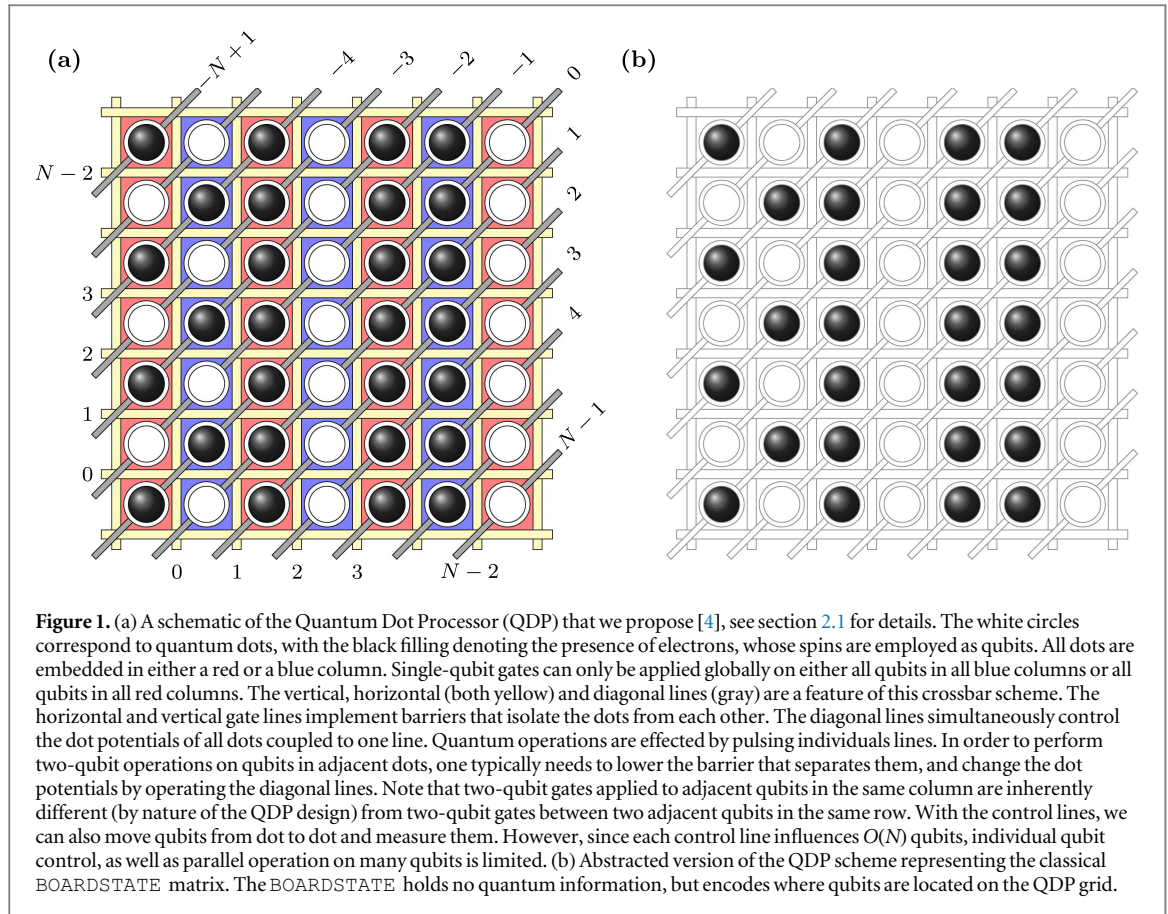E-mail: j.helsen@tudelft.nl

Abstract

A central challenge for the scaling of quantum computing systems is the need to control all qubits in the system without a large overhead. A solution for this problem in classical computing comes in the form of so-called crossbar architectures. Recently we made a proposal for a large-scale quantum processor (Li *et al* arXiv:1711.03807 (2017)) to be implemented in silicon quantum dots. This system features a crossbar control architecture which limits parallel single-qubit control, but allows the scheme to overcome control scaling issues that form a major hurdle to large-scale quantum computing systems. In this work, we develop a language that makes it possible to easily map quantum circuits to crossbar systems, taking into account their architecture and control limitations. Using this language we show how to map well known quantum error correction codes such as the planar surface and color codes in this limited control setting with only a small overhead in time. We analyze the logical error behavior of this surface code mapping for estimated experimental parameters of the crossbar system and conclude that logical error suppression to a level useful for real quantum computation is feasible.

1. Introduction

When attempting to build a large-scale quantum computing system a central problem, both from experimental and theoretical perspectives, is what might be called the interconnect problem. This problem, which also exists in classical computing, arises when computational units (e.g. qubits in quantum computers, transistors in classical computers) are densely packed such that there is not enough room to accommodate individual control lines to every unit. A solution to this problem, which is commonplace in classical computing systems, is a so-called 'crossbar architecture'. In this class of computing architecture we do not draw a control line to every qubit but rather organize computational units in a grid with control lines addressing full rows and columns of this grid. Control effects then happen at the intersection of column and row lines. In this way, using N control lines $O(N^2)$ computational units can be addressed. This makes it possible to scale the system to a large number of qubits. The price to pay for this is a reduced ability to perform operations on different units in the grid in parallel. For classical systems this is not a fundamental problem, but when the computational units are qubits, whose information decays over time, parallelism becomes absolutely essential. This introduces a formidable roadblock for the development of crossbar systems for quantum computing systems. Nevertheless various crossbar architectures for quantum computers have been proposed in the past [1–5]. Recently [4] we proposed a quantum computing platform based on spin qubits in silicon quantum dots featuring a crossbar architecture. This architecture features compatibility with modern silicon manufacturing techniques and in combination with recent advances in controlling quantum dot qubits and the inherent long coherence times of spin qubits in silicon we expect it to be a formidable step forwards in creating large-scale quantum computing devices.

Any realistic quantum computing device, including the one we propose in [4], will suffer from noise processes that degrade quantum information. This noise can be combated by quantum error correction [6, 7], where quantum information is encoded redundantly in such a way that errors can be diagnosed and remedied as

Figure 1. (a) A schematic of the Quantum Dot Processor (QDP) that we propose [4], see section 2.1 for details. The white circles correspond to quantum dots, with the black filling denoting the presence of electrons, whose spins are employed as qubits. All dots are embedded in either a red or a blue column. Single-qubit gates can only be applied globally on either all qubits in all blue columns or all qubits in all red columns. The vertical, horizontal (both yellow) and diagonal lines (gray) are a feature of this crossbar scheme. The horizontal and vertical gate lines implement barriers that isolate the dots from each other. The diagonal lines simultaneously control the dot potentials of all dots coupled to one line. Quantum operations are effected by pulsing individuals lines. In order to perform two-qubit operations on qubits in adjacent dots, one typically needs to lower the barrier that separates them, and change the dot potentials by operating the diagonal lines. Note that two-qubit gates applied to adjacent qubits in the same column are inherently different (by nature of the QDP design) from two-qubit gates between two adjacent qubits in the same row. With the control lines, we can also move qubits from dot to dot and measure them. However, since each control line influences $O(N)$ qubits, individual qubit control, as well as parallel operation on many qubits is limited. (b) Abstracted version of the QDP scheme representing the classical BOARDSTATE matrix. The BOARDSTATE holds no quantum information, but encodes where qubits are located on the QDP grid.

they happen without disturbing the encoded information. Many quantum error correction codes have been developed over the last two decades and several of them have desirable properties such as high noise tolerance, efficient decoders and reasonable implementation overhead. Of particular note are the planar surface [8] and color codes [9], which have the nice property that they can be implemented in quantum computing systems where only nearest-neighbor two-qubit gates are available.

However these codes, and all other quantum error correction codes, were developed under the (often implicit) assumption that all physical qubits participating in the code can be controlled individually and in parallel. For large (read: comprising many qubits) error correction codes this introduces a tension between the needs of the error correction code and the control limitations for large systems mentioned above. While practical large-scale quantum computers most likely pose control limitations, surprisingly little work has been done in this area [10]. Here we investigate the minimal amount of parallel control resources needed for quantum error correction and focus in particular on crossbar architectures. In figures 1 and 5 we summarize the layout and control limitations of the architecture in [4]. Overcoming these limitations motivates the current work.

1.1. Contributions

1.1.1. Analysis of the crossbar system

We analyze the crossbar architecture we propose in [4]. We give a full description of the layout and control characteristics of the architecture in a manner accessible to non-experts in quantum dots. We develop a language for describing operations in the crossbar system. Of particular interest here are the regular patterns (see e.g. section 3.4) that are implied by the crossbar structure. These configurations provide an abstraction on which we build mappings of quantum error correction codes (see below). This analysis is particular to the system in [4] but we believe many of the considerations to hold for more general crossbar architectures.

1.1.2. An efficient algorithms for control on crossbar architectures

We develop an algorithm for moving around qubits (shuttling) on crossbar architectures. We show that the task of shuttling qubits in parallel can be described using a matrix taking value in an idempotent monoid. The control algorithm then reduces to finding independent columns of this matrix, for a suitable notion of independence. This algorithm in principle allows the straightforward mapping of more complicated quantum algorithms which require long-range operations, with little operational overhead. We also expect this algorithm to be applicable to the control of more general crossbar architectures. We also sketch an algorithm for parallel

two-qubit interactions in crossbar systems which produce *optimal* control sequences. This algorithm is based on computing the Schmidt-normal form of matrices with entries in the principal ideal domains $\mathbb{Z}_2$ and $\mathbb{Z}_4$.

1.1.3. Mapping of surface and color codes

We map the planar surface code and the 6.6.6. (hexagonal) and 4.8.8. (square-octagonal) color codes [9] to the crossbar architecture, taking into account its limited ability to perform parallel quantum operations. The tools we develop for describing the mapping, in particular the configurations described in section 3.4, should be generalizable to other quantum error correction codes and general crossbar architectures.

1.1.4. Analysis of the surface code logical error

Due to experimental limitations the mappings mentioned above might not be attainable in near term devices. Therefore we adapt the above mappings to take into account practical limitations in the architecture [4]. In this version of the mapping the length of an error correction cycle scales with the distance of the mapped code. This means the mapping does not allow for arbitrary logical error rate suppression. Therefore we analyze the behavior of the logical error rate with respect to estimated experimental error parameters and find that the logical error rate can in principle be suppressed to below 10^{-20} (an error rate comparable to the error rate of classical computers [11]), allowing for practical quantum computation to take place.

Our work raises several interesting theoretical questions regarding the mapping of quantum algorithms to limited control settings, see section 6.

1.2. Outline

In section 2 we introduce the architecture we proposed in [4]. We forgo an explanation of the physics and focus on the abstract control aspects of the system (explaining them in a largely self-contained manner accessible to non-experts in quantum dot physics). We introduce classical helper objects such as the BOARDSTATE which will aid later developments. We discuss one- and two-qubit operations, measurements, and qubit shuttling. In section 3 we focus on parallel operations. We discuss difficulties inherent in parallel operation in a crossbar system and develop an algorithm for dealing with them efficiently. We also introduce several BOARDSTATE configurations which feature prominently in quantum error correction mappings and describe how to reach them efficiently by parallel shuttling. In section 4 we give a quick introduction to quantum error correction with a particular focus on the planar surface code and the 4.8.8. and 6.6.6. color code. In section 4.4 we bring together all previous sections and devise a mapping of the planar surface code to the crossbar architecture. This we continue in section 4.5 for the 6.6.6. and 4.8.8. color codes. Finally in section 5 we analyze in detail the logical error probability of the surface code mapping as a function of the code distance and estimated error parameters of the crossbar system.

2. The quantum dot processor

In this section we will give an overview of the quantum dot processor (QDP) architecture as proposed in [4]. We will use this architecture as a concrete realization of the more general idea of quantum crossbar architectures. We will focus not so much on the details of the implementation but rather focus on abstract operational properties of the system as they are relevant for our purposes. The basic organization of the QDP is that for an $N \times N$ grid of qubits interspersed with control lines that effect operations on the qubits. The most notable feature of the QDP (and crossbar architectures in general) is the fact that any classical control signal sent to a control line will be applied simultaneously to all qubits adjacent to that control line. This means that every possible classical instruction applied to the QDP will affect $O(N)$ qubits (these qubits will not necessarily be physically close to each other). This has important consequences for the running of quantum algorithms on the QDP (or any crossbar architecture) that must be taken into account when compiling these algorithms to hardware level instructions. Notably it places strong restrictions on performing quantum operations in parallel on the QDP. To deal with these restrictions it is important to have a good understanding of how operations are performed on the QDP. It is for this reason that we begin our study of the QDP with an examination of its control structure at the hardware level. We describe the physical layout of the system and develop nomenclature for the fundamental control operations. This nomenclature might be called the 'machine code' of the QDP. From these basic instructions we go on to construct all elementary operations that can be applied to qubits in the QDP. These are quantum operations, such as single-qubit gates, nearest-neighbor two-qubit gates and qubit measurements but also a non-quantum operation called coherent shuttling which does not affect the quantum state of the QDP qubits but changes their connectivity graph (i.e. which qubits can be entangled by two-qubit gates). All of these operations are restricted by the nature of the control architecture in a way that gives rise to interesting patterns (section 3.4) and which we will more fully examine in section 3.

2.1. Layout

A schematic overview of the QDP architecture is given in figure 1, where qubits (which are electrons, denoted by black balls) occupy an array of $N \times N$ quantum dots (hereafter often referred to as sites). The latter are denoted by white sites when empty, since they either are occupied by a qubit or not. We will label the dots by tuples containing row and column indices $(i, j) \in [0 : N - 1]^{\times 2}$ (beginning from the *bottom left* corner), such that a single-qubit state $|\psi\rangle$ living on the (i, j)'th site will be denoted by $|\psi\rangle_{(i,j)}$. We assume the qubits to be initialized in the state $|0\rangle$. For future reference we note that $|0\rangle$ corresponds to the spin-up state and $|1\rangle$ to the spin-down state of the electron constituting the qubit.

Typically we will work in a situation where half the sites are occupied by a qubit and half the sites are empty (as seen in figure 1(a)). Because (as we discuss in section 2.3.1) the qubits can be moved around on the grid and the two-qubit gates depend on the filling of the grid, it is important to keep track of which sites contain qubits and which ones do not. This can be done efficiently in classical side-processing. To this end we introduce the BOARDSTATE object. BOARDSTATE consists of a binary $N \times N$ matrix with a 1 in the (i, j)'th place if the (i, j)'th site contains an electron (qubit) and a 0 otherwise. The BOARDSTATE does not contain information about the qubit state $|\psi\rangle_{(i,j)}$, only about the electron occupation of the grid. A particular BOARDSTATE is illustrated in the left panel of figure 1.

We now turn to describing the control structures that are characteristic for this architecture. As a first feature, we would like to point out that each site is either located in a red or a blue region in figure 1 (left panel). The blue (red) columns correspond to regions of high (low) magnetic fields, which plays a role in the addressing of qubits for single-qubit gates. We will denote the set of qubits in blue columns (identified by their row and column indices) by $\mathcal{B}$ and the set of qubits in red columns by $\mathcal{R}$.

Much finer groups of sites can be addressed by the control lines that run through the grid. The crossbar architecture features control lines that are connected to $O(N)$ sites. At the intersections of these control lines individual sites and qubits can be addressed. This means that using $O(N)$ control lines $O(N^2)$ qubits can be controlled. As seen in figure 1 the rows and columns of the QDP are interspersed with horizontal and vertical lines (yellow), as a means to control the tunnel coupling between adjacent sites. We refer to those lines as barrier gates, or barriers for short. Each line can be controlled individually, but a pulse has an effect on all $O(N)$ qubits adjacent to the line. Another layer of control lines is used to address the dots itself rather than the spaces in between them. The diagonal gate lines (gray), are used to regulate the dot potential. We label the horizontal and vertical lines by an integer running from 0 to $N - 2$ and the diagonal lines with integers running from $-(N - 2)$ to $N - 2$ where the $-(N - 2)$'th line is the top left line and increments move towards the bottom right (see figure 1(a)). Next we describe how these control lines can be used to effect operations on the qubits occupying the QDP grid.

2.2. Control and addressing

As described above, the QDP consists of quantum dots interspersed with barriers and connected by diagonal lines. For our purposes these can be thought of as abstract control knobs that apply certain operations to the qubits. In this section we will describe what type of gates operations are possible on the QDP. We will not concern ourselves with the details of parallel operation until section 3.

There are three fundamental operations on the QDP which we will call the 'grid operations'. These operations are 'lower vertical barrier' (V), 'lower horizontal barrier' (H) and 'set diagonal line' (D). The first two operations are essentially binary (on-off) but the last one (D) can be set to a value $t \in [0: T]$ where T is a device parameter. (At the physical level this corresponds to how many clearly distinct voltages we can set the quantum dot plunger gates [4]). Although the actual pulses on those gates differ by amplitude and duration between the different gates and operations, this notation gives us a clear idea which lines are utilized. This can be done because realistically one will not interleave processes in which pulses have such different shapes. We can label the grid operations by mnemonics (which in a classical analogy we will call OPCODES) as seen in section 2.2. These OPCODES are indexed by an integer parameter that indicates which control line it applies to. We count horizontal and vertical lines starting at zero from the lower left corner of the grid (see figure 1). Note that the lines at the boundary of the grid are never addressed in our model and are thus not counted.

We indicate parallel operation of a collection of OPCODES by ampersands, e.g. D[1]&H[2]&D[5]. We also define inherently parallel versions (in section 2.2) of the basic OPCODES that take as input a binary vector V of length N (for the diagonal line this is a T-valued vector of length N)

OPCODE	Effect
V[i]	Lower vertical barrier at index i
H[i]	Lower horizontal barrier at index i
D[i][t]	Set diagonal line at index i to value t

Table 1. OPCODES for horizontal and vertical shuttling and measurement together with the control OPCODES required to implement these operations on the QDP.

OPCODE	Control OPCODES	Effect
HS[i,j,k]	V[i]&D[i-j][t-1/2-k/2] &D[i-j+1][t-1/2+k/2]	$(k=1)$: Shuttle from (i,j) to $(i,j+1)$ $(k=-1)$: Shuttle from $(i,j+1)$ to (i,j)
VS[i,j,k]	H[j]&D[i-j][t-1/2-k/2] &D[i-j-1][t-1/2 + k/2]	$(k=1)$: Shuttle from (i,j) to $(i+1,j)$ $(k=-1)$: Shuttle from $(i+1,j)$ to (i,j)
M[i,j,k]	HS[i,j+1/2+k/2,-k]	Measurement of qubit at (i,j) using the ancilla at $(i,j+k)$

OPCODE	Effect
V[V]	Set vertical barrier to V (i), $\forall\, i \in [0:N-2]$
H[V]	Set horizontal barrier to V (i), $\forall\, i \in [0:N-2]$
D[V]	Set diagonal at height V (i), $\forall\, i \in [-N+2:N-2]$

These grid operations can be used to induce some elementary quantum gates and operations on the qubits in the QDP. Below we describe these operations.

2.3. Elementary operations
Here we give a short overview of the elementary operations available in the QDP. We will describe basic single-qubit gates, two-qubit gates, the ability to move qubits around by coherent shuttling [12] and a measurement process through Pauli Spin Blockade (PSB) [13]. All of these operations are implemented by a combination of the grid operations defined in section 2.2, and always have a dependence on the BOARDSTATE.

2.3.1. Coherent qubit shuttling
An elementary operation of the QDP is the coherent qubit shuttling [12, 14], of one qubit to an adjacent, empty site. That means that an electron (qubit) is physically moved to the other dot (site) utilizing at least one diagonal line and the barrier between the two sites (see figure 2(a) for an illustration). It thereby does not play a role whether the shuttling is in horizontal (from a red to a blue column or the other way around) or vertical direction (inside the same column). However, the shuttling in between columns results in a *Z*-rotation, that must be compensated by timing operations correctly, see [4] for details. This *Z*-rotation can also by used as a local single-qubit gate, see section 2.3.3. The operation is dependent on the BOARDSTATE by the prerequisite that the site adjacent to the qubit to must be empty. Collisions of qubits are to be avoided, as those will lead to a collapse of the quantum state (see however the measurement process in section 2.3.2). We now describe the coherent shuttling as the combination of grid operations.

We lower the vertical (or horizontal) barrier in between the two sites and instigate a 'gradient' of the on-site potentials of the two dots. That is, the diagonal line of the site containing the qubit must be operated at $t \in [0:T]$ while the line overhead the empty site must have the potential $\hat{t} \in [0:T]$ with $\hat{t} = t - 1$. Note that this implies it might not be operated at all (if it is already at the right level). We will subsequently refer to the combination of a lowered barrier and such a gradient as a 'flow'. A flow will in general be into one of the four directions on the grid. We define the commands VS[i,j,k] (vertical shuttling) and HS[i,j,k] (horizontal shuttling). The command VS[i,j,k] shuttles a qubit at location (i,j) to $(i+1,j)$ for $k=1$ (upward flow) and shuttles a qubit at location $(i+1,j)$ to (i,j) for $k=-1$ (downward flow). Similarly, the command HS[i,j,k] shuttles a qubit at location (i,j) to $(i,j+1)$ for $k=1$ (rightward flow) and shuttles a qubit at location $(i,j+1)$ to (i,j) for $k=-1$ (leftward flow). See table 1 for a summary of these OPCODES.

Using only these control lines, we can individually select a single qubit to be shuttled. However, when attempting to shuttle in a parallel manner, we have to be carefully take into account the effect that the activation of several of those lines has on other locations. We will deal with this in more detail in section 3.1.

2.3.2. Measurement and readout
The QDP allows for local single-qubit measurements in the computational basis $|0\rangle$, $|1\rangle$. We can measure a qubit by attempting to shuttle it to a horizontally adjacent site that is already occupied by an ancilla qubit and then detecting whether the shuttling was successful. This process is called Pauli Spin Blockade (PSB) measurement [4, 13]. See figure 2(b) for an illustration. However, the QDP's ability to perform this type of qubit measurements is limited by three factors.

Firstly, the measurement requires an ancilla qubit horizontally adjacent to the qubit to be measured. This ancilla qubit must be in a known computational basis state. Moreover, if the ancilla qubit is in the state $|0\rangle$ the ancilla qubit must be in the set $\mathcal{B}$ (blue columns in figure 1) while the qubit to be measured must be in the set $\mathcal{R}$ (red columns in figure 1). On the other hand, if the ancilla qubit is in the state $|1\rangle$ the ancilla qubit must be in the set $\mathcal{R}$ while the qubit to be measured is in the set $\mathcal{B}$. This means that when an qubit-ancilla pair is in the wrong configuration we must first shuttle both qubits one step to the left (or both the the right). Note that this takes two additional shuttling operations, which means it is important to keep track at all times where on the BOARDSTATE the qubit and its ancilla are or else incur a shuttling overhead (which might become significant when dealing with large systems and many simultaneous measurements). We will deal with this problem of qubit-ancilla pair placement in more detail in section 3.3.

Secondly, assuming that the qubit-ancilla pair is in the right configuration to perform the PSB process one still needs to perform a shuttling-like operation to actually perform the measurement. On the technical level, the operation is different from coherent shuttling, but the use of the lines is similar with the difference that after the readout, the shuttling-like operation is undone by the use of the same lines as before—which are not necessarily the lines one would use to reverse a coherent shuttling operation. However, scheduling measurement events on the QDP is at least as hard as the scheduling of shuttle operations discussed above. Depending on the state the qubit is in, it will now assume one of two possible states that can be distinguished by their charge distribution.

Thirdly, the readout process requires to have a barrier line that borders to the qubit pair, with an empty dot is across the spot of the qubit to be measured. This is a consequence of the readout procedure.

In table 1 we introduce the measurement OPCODE M[i, j, k] with $k \in \{1, 1\}$ to denote a measurement of a qubit at location (i, j) with an ancilla located to the left $(k = 1)$ or to the right $(k = 1)$.

2.3.3. Single-qubit rotations

There are two ways in which single-qubit rotations can be performed on the QDP, both with drawbacks and advantages. The first method, which we call the semi-global qubit rotation, relies on electron-spin-resonance [15]. Its implementation in the QDP allows for any rotation in the single-qubit special unitary group $SU(2)$ [16] to be performed but we do not have parallel control of individual qubits. The control architecture of the QDP is such that we can merely apply the same single-qubit unitary rotation on all qubits in either $\mathcal{R}$ or $\mathcal{B}$ (even or odd numbered columns). Concretely we can perform in parallel the single-qubit unitaries

$$U_{\mathcal{R}} = \bigotimes_{(i,j) \in \mathcal{R}} U_{i,j} \qquad U \in SU(2) \tag{1}$$

$$U_{\mathcal{B}} = \bigotimes_{(i,j) \in \mathcal{B}} U_{i,j} \qquad U \in SU(2), \tag{2}$$

where $U_{i,j}$ means applying the same unitary U to the state carried by the qubit at location (i, j). In general the only way to apply an arbitrary single-qubit unitary on a single qubit in $\mathcal{B}$ (or $\mathcal{R}$) is by applying the unitary to all qubits in $\mathcal{B}$ ($\mathcal{R}$), moving the desired qubit into an adjacent column, i.e. from $\mathcal{B}$ to $\mathcal{R}$ ($\mathcal{R}$ to $\mathcal{B}$) and then applying the inverse of the target unitary to $\mathcal{R}$ ($\mathcal{B}$). This restores all qubits except for the target qubit to their original states and leaves the target qubit with the required unitary applied. The target qubit can then be shuttled to its original location. A graphical depiction of the BOARDSTATE associated with this maneuver can be found in figure 3. This means applying a single unitary to a single qubit takes a constant amount of grid operations regardless of grid size.
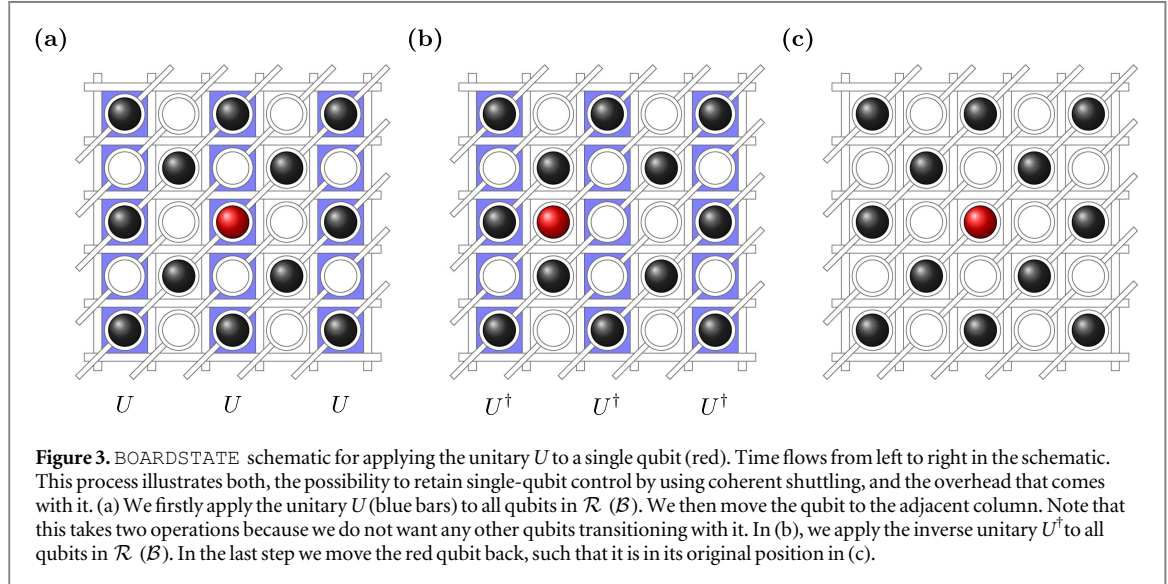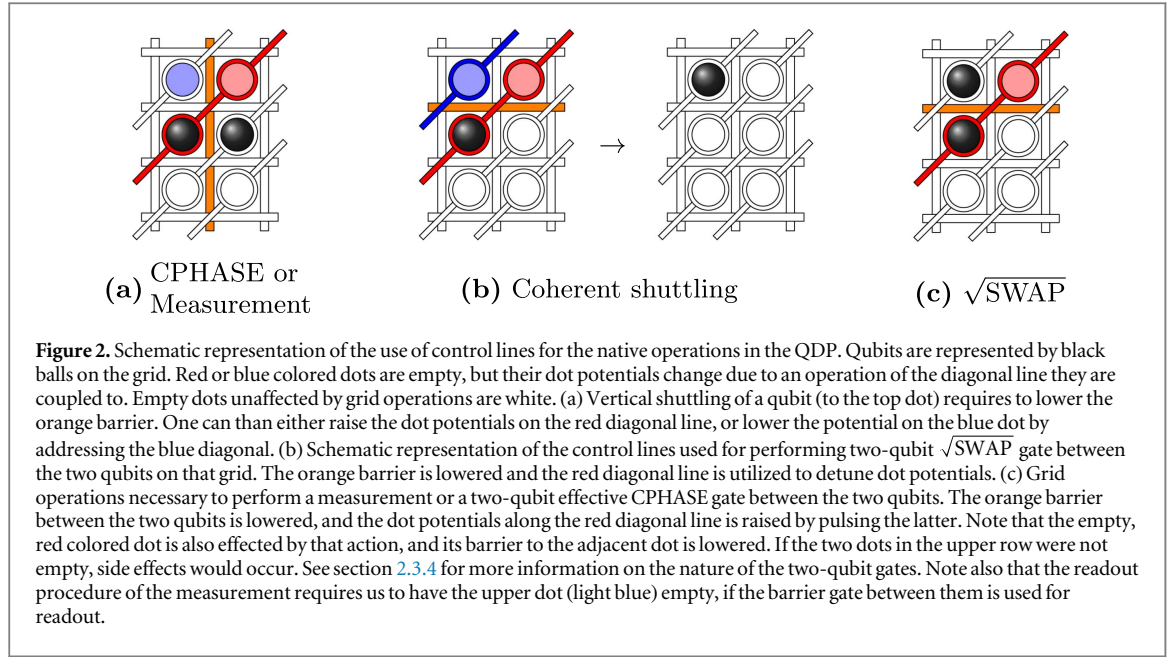
The second method does allow for individual single-qubit rotations but is limited to performing single-qubit rotations of the form

$$U(\phi) = e^{i\phi Z}, \qquad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad \phi \in [0, 2\pi) \tag{3}$$

This operation can be performed on a given qubit $|\psi\rangle_{(i,j)}$ by shuttling it from (i, j) to $(i, j \pm 1)$. When the qubit leaves the column it was originally defined ($\mathcal{B}$ to $\mathcal{R}$ or vice versa) it will effectively start precessing about its Z axis [4]. This effect is always present but it can be mitigated by timing subsequent operations such that a full rotation happens between every operation (effectively performing the identity transformation, see section 2.3.1). By changing the timing between subsequent operations any rotation of the form equation (3) can be effected. This technique will often be used to perform the Z-gate (defined above) and the $S = \sqrt{Z}$ phase gate in error correction sequences.

2.3.4. Two-qubit gates

As the last elementary tool, we have the ability to apply entangling two-qubit gates on adjacent qubits. The QDP can perform two different types of two-qubit gates. Inside one column, so between qubits at locations (i, j) and $(i \pm 1, j)$, a square-root of SWAP ($\sqrt{\text{SWAP}}$) can be realized [17]. This can be done by lowering the horizontal barrier between the two qubits and toggling the voltage on the diagonal lines overhead the two qubits. This

Figure 2. Schematic representation of the use of control lines for the native operations in the QDP. Qubits are represented by black balls on the grid. Red or blue colored dots are empty, but their dot potentials change due to an operation of the diagonal line they are coupled to. Empty dots unaffected by grid operations are white. (a) Vertical shuttling of a qubit (to the top dot) requires to lower the orange barrier. One can than either raise the dot potentials on the red diagonal line, or lower the potential on the blue dot by addressing the blue diagonal. (b) Schematic representation of the control lines used for performing two-qubit $\sqrt{\text{SWAP}}$ gate between the two qubits on that grid. The orange barrier is lowered and the red diagonal line is utilized to detune dot potentials. (c) Grid operations necessary to perform a measurement or a two-qubit effective CPHASE gate between the two qubits. The orange barrier between the two qubits is lowered, and the dot potentials along the red diagonal line is raised by pulsing the latter. Note that the empty, red colored dot is also effected by that action, and its barrier to the adjacent dot is lowered. If the two dots in the upper row were not empty, side effects would occur. See section 2.3.4 for more information on the nature of the two-qubit gates. Note also that the readout procedure of the measurement requires us to have the upper dot (light blue) empty, if the barrier gate between them is used for readout.

Figure 3. `BOARDSTATE` schematic for applying the unitary U to a single qubit (red). Time flows from left to right in the schematic. This process illustrates both, the possibility to retain single-qubit control by using coherent shuttling, and the overhead that comes with it. (a) We firstly apply the unitary U (blue bars) to all qubits in $\mathcal{R}$ ($\mathcal{B}$). We then move the qubit to the adjacent column. Note that this takes two operations because we do not want any other qubits transitioning with it. In (b), we apply the inverse unitary $U^\dagger$ to all qubits in $\mathcal{R}$ ($\mathcal{B}$). In the last step we move the red qubit back, such that it is in its original position in (c).

situation is illustrated in figure 2(c). The $\sqrt{\text{SWAP}}$ gate is defined as

$$\sqrt{\text{SWAP}} = \begin{pmatrix} 1 & & & \\ & (1+i)/2 & (1-i)/2 & \\ & (1-i)/2 & (1+i)/2 & \\ & & & 1 \end{pmatrix}, \tag{4}$$

in the computational basis. Alternatively, between horizontally adjacent qubits, e.g. between $(i, j) \in \mathcal{R}$ and $(i, j \pm 1) \in \mathcal{B}$ the native two-qubit gate is an effective CPHASE gate which has matrix representation

$$\text{CPHASE} = \begin{pmatrix} 1 & & & \\ & e^{i\phi_1} & & \\ & & e^{i\phi_2} & \\ & & & 1 \end{pmatrix}, \tag{5}$$

in the computational basis and with the two angles $\phi_1 + \phi_2 \bmod 2\pi = \pi$ (demonstrated in [18–20]). This gate can be performed between horizontally adjacent qubits by lowering the vertical barrier between them and toggling the overhead diagonal lines. This is illustrated in figure 2(b). In practice we expect the $\sqrt{\text{SWAP}}$ gate to have significantly higher fidelity than the CPHASE gate [4] so in any application (e.g. error correction) the $\sqrt{\text{SWAP}}$ gate is the preferred native two-qubit gate on the QDP. In table 2 we define OPCODES for the horizontal interaction (CPHASE) and the vertical interaction ($\sqrt{\text{SWAP}}$).
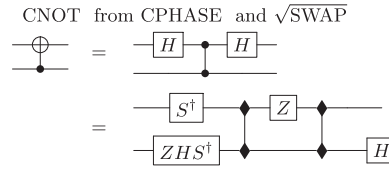
Figure 4. Construction of the CNOT gate out of the native CPHASE and $\sqrt{\text{SWAP}}$ gates. Note that one requires two $\sqrt{\text{SWAP}}$ gates to construct a CNOT gate [21]. When performing arbitrary algorithms it would be preferable to forgo this substitution and instead compile the algorithm directly into a gateset containing the $\sqrt{\text{SWAP}}$ gate.

Table 2. OPCODES for horizontal and vertical two-qubit operations on the QDP, respectively the CPHASE and $\sqrt{\text{SWAP}}$ gates. We also include OPCODES for the performing of CNOT gates composed of $\sqrt{\text{SWAP}}$ or CPHASE gates.

OPCODE	Effect	Parameter
`HI[(i,j)]`	Perform CPHASE gate between sites `(i,j)` and `(i,j+1)`	$(i,j) \in [0: N-2]^{\times 2}$
`VI[(i,j)]`	Perform $\sqrt{\text{SWAP}}$ gate between sites `(i,j)` and `(i+1,j)`	$(i,j) \in [0: N-2]^{\times 2}$
`HC[(i,j)]`	Perform CNOT (using CPHASE) between `(i,j)` and `(i,j+1)`	$(i,j) \in [0: N-2]^{\times 2}$
`VC[(i,j)]`	Perform CNOT (using $\sqrt{\text{SWAP}}$) between `(i,j)` and `(i+1,j)`	$(i,j) \in [0: N-2]^{\times 2}$

2.3.5. CNOT subroutine

Many quantum algorithms are conceived using the CNOT gate as the main two-qubit gate. However the QDP does not support the CNOT gate natively. It is easy to construct the CNOT gate from the CPHASE gate by dressing the CPHASE gate with single-qubit Hadamard rotations as seen in figure 4 (left). It is slightly more complicated to construct a CNOT gate using the $\sqrt{\text{SWAP}}$ but it can be done by performing two $\sqrt{\text{SWAP}}$ gates interspersed single-qubit rotations [19–21] as seen in figure 4 (right). If the control qubit is moved from an adjacent column on the QDP (as it is in most cases we will deal with) the Z and S-gates can be performed by the Z-rotation-by-waiting technique described in the last section. For completeness we also define an OPCODE for the CNOT operation in table 2.

3. Parallel operation of a crossbar architecture

In this section we focus on performing operations in parallel on the QDP (or more general crossbar architectures). Because of the limitations imposed by the shared control lines of the crossbar architecture, achieving as much parallelism as possible is a non-trivial task. We will discuss parallel shuttle operations, parallel two-qubit gates, parallel single-qubit gates and parallel measurement. As part of the focus on parallel shuttling we also include some special cases relevant to quantum error correction where full parallelism is possible.

Before we start our investigation however, we would like to put three issues into focus that are likely to be encountered when attempting parallel operations. Firstly, it must be understood that an operation on one location on a crossbar system can cause unwanted side effects in other locations (that might be far away). As indicated in section 2 many elementary operations on the grid in particular take place at the crossing points of control lines. This means that any parallel use of these grid operations must take into account 'spurious crossings' which may have such unintended side effects. We can illustrate this with an example. Imagine we want to perform the vertical shuttling operations `VS[i,j-1,1]` and `VS[i+2,j-1,1]` in parallel (see figure 5 for illustration). We can do this by lowering the horizontal barriers at rows i and $i + 2$ (orange in illustration) and elevating the on-site potentials on the diagonal lines $i - j + 1$ and $i + 2 - j + 1$ (red in illustration). This will open upwards flows at locations $(i, j - 1)$ and $(i + 2, j - 1)$. However it will also open an upward flow at the location $(i + 2, j + 1)$. This means, if a qubit is present at that location an unintended shuttling event will happen. To avoid this outcome we must either perform the operations `VS[i,j-1,1]` and `VS[i+2,j-1,1]` in sequence (taking two time-steps) or perform an operation `VS[i+2,j+1,-1]` to fix the mistake we made, again taking two time-steps. This is a general problem when considering parallel operations on the QDP.

Secondly, we would like to point out that in realistic setups, we expect a trade-off between parallelism (manifested in algorithmic depth) and operation fidelity (in particular this will be the case in the QDP system). In order to understand this, we have to be aware that most operations consist of applying the correct pulses for the right amount of time. These durations however can slightly vary from site to site (due to manufacturing imperfections), so we e.g. must be able to switch barriers back on again prematurely when accounting for a site with a shorter time required. If this is not possible (maybe because it would cause side effects) a loss in operation fidelity is a consequence of the resulting improperly timed operation. The most robust case is thus to schedule

Figure 5. Spurious shuttle operations. Here we illustrate an example of unintended side effects that occur due to the limited control. We again denote qubits by colored balls, and color barriers and lines that are operated. Empty dots with changed potentials are colored as well, white dots are unaffected. (a) The black qubits are to be shuttled from $(i, j - 1)$ to $(i + 1, j - 1)$ and from $(i + 2, j)$ to $(i + 3, j)$ respectively without moving the blue qubit. For that purpose, the (orange) barriers between the two-dot pairs are lowered, as well as the (red) diagonal lines through $(i, j - 1)$ and $(i + 2, j)$ are pulsed, such that the dot potentials on those sites are raised. (b) The qubit on $(i + 3, j + 1)$ has unintentionally moved to $(i + 2, j + 1)$. (c) To remedy this situation, we lower the barrier number $i + 2$ again (orange), and also raise the potential on $(i + 3, j + 1)$ and all other dots that are connected by the pulsed diagonal line (red). In (d), the desired situation is achieved.

operations line-by-line. By this we mean that we attempt to perform $O(N)$ grid operations in a time step while using every horizontal, diagonal or vertical line only once per individual grid operation. If we for instance schedule several vertical shuttle operations, we may choose to start by lowering one of horizontal barrier first and then detune the dot potentials of all qubits adjacent to that barrier, by pulsing the corresponding diagonal lines. To account for the variations, we reset the diagonal lines at slightly different times. Line-by-line operations work with either line types for every two-dot operation (measurement, shuttling and two-qubit gates). Note however that for shuttling operations individual control over one line is sufficient, whereas for measurement and two-qubit gates we would ideally like to be able to control two lines per qubit pair individually, where one line should be the barrier separating the two paired qubits. Results presented in the following take into account these constraints for quantum error correction. The parallel operation nonetheless remains one of the greatest challenges of the crossbar scheme. In this section we will assume all operations to be perfect (even when performed in parallel) but in section 5 we perform a more detailed analysis of the behavior of the QDP when operational errors are taken into account.

Thirdly, from a performance perspective it is important to separate the operations that have to be done on the qubits on the crossbar grid from operations that can be done by classical side computation (which for our purposes is essentially free). We will deal with this by including classical side computation in the OPCODES for parallel operation. This way the complexity of dealing with spurious operations is abstracted away. We devise algorithms that take in an arbitrary list of shuttling or two-qubit gate locations and work out a sequence of shuttling or two-qubit gate steps that achieve that list. We begin with discussing parallel shuttle operations.

3.1. Parallel shuttle operations
We define parallel versions of the shuttling OPCODES `HS[i, j, k]` and `VS[i, j, k]` as

OPCODE	Effect
`HS[L]`	Perform `HS[i,j,k]` for all $(i, j, k) \in$ `L`
`VS[L]`	Perform `HS[i,j,k]` for all $(i, j, k) \in$ `L`

This code takes in a set (denoted as `L`) of tuples (i, j, k) which denote 'locations at which shuttling happens' (i, j) and 'shuttling direction' (k). From these codes it is not immediately clear how many of the shuttling operations can be performed in a single grid operation, i.e. setting the diagonal lines to some configuration and lowering several horizontal or vertical barrier. If multiple grid operations are needed (such as in the example figure 5) we would like this sequence of grid operations to be as short as possible. However, given some initial `BOARDSTATE` and a parallel shuttling command `HS[L]` it is not clear what the sequence of parallel shuttling operations actualizing this command is. Below we analyze this problem of parallel shuttling in more detail and give a classical algorithm that produces, from an input `HS[L]` or `VS[L]` a sequence of parallel grid operations that performs this command. Ideally we would like this sequence to be as short as possible. This algorithm does

not perform optimally in all circumstances (i.e. it does not produce the shortest possible sequence of parallel shuttling operations) but for many relevant cases it performs quite well. Note that this is a technical section and the details are not needed to understand the quantum error correction results in section 4, 4.4 and 4.5. Readers interested only in those may skip ahead to section 3.2.

3.1.1. The flow matrix

We will only consider shuttling to the left and to the right but all mechanisms introduced work equally well for shuttling in the vertical directions. As will be seen in section 3.4 some BOARDSTATE configurations can be converted into each other in an amount of grid operations that is constant in the size of the grid. It can be seen that the problem of whether two shuttles can be performed in parallel is a problem with a matrix structure, as flows can only occur at the intersection open barriers and non-trivial diagonal line gradients. To capture this matrix intuition we construct, from the initial BOARDSTATE and the command HS[L] a matrix F which we call the flow matrix. This matrix will have entries corresponding to the crossing of the gradient line between two diagonal qubit lines and the vertical barrier lines. The flow matrix is defined with respect to a specific command HS[L] and its entries correspond to the locations on the grid where we want shuttling in certain directions to happen.

From a specific command HS[L] and a specific current BOARDSTATE we will define a flow matrix F. This matrix will have entries which take value in the set $\{r, l, e, re, le, *\}$. Each element of this set has a specific operational meaning. The elements r, l, e correspond to specific actions that can be taken on the qubit grid. They correspond specifically to 'shuttle to the right' (r), 'shuttle to the left' (l) and 'do nothing' (e). Note that these actions do not necessarily act on a fixed qubit. Rather they act on a specific location on the grid (where a qubit may or may not be present). The other three elements do not directly correspond to a shuttling action but rather signify that at this location we have a choice of different consistent actions. We will call these elements 'wildcards'. These wildcards signify the actions 'shuttle to the right or do nothing' (re), 'shuttle to the left or do nothing' (le), or 'any action is allowed' ($*$).

We fill in the matrix entry F_{ij} with a symbol r for every $(i, j, 1)$ in L. This indicates that at some point in time we want to perform the operation HS[i,j,1] at that location. Similarly we fill in a symbol l on every matrix entry F_{ij} for every $(i, j + 1, -1)$ in L. We place the symbols re, le respectively on the matrix entries $F_{i(j-1)}$ and F_{ij} for every occupied site (i, j) in the BOARDSTATE that has no corresponding entry in L. This indicates that we would like for no shuttle operations to happen on these crossing points (since we want the qubit to stay put) but that we do not mind a HS[i,j-1,1] happening on the crossing point to the left of the qubit at (i, j) (since it will not affect the qubit) or mind a HS[i,j,1] happening to the right of the qubit at (i, j). Lastly we fill in the symbol e on every matrix entry F_{ij} where we want no shuttling operation to happen at any time to the right of the site (i, j) (for instance on the crossing point between two qubits that are in horizontally adjacent sites). In every other matrix entry F_{ij} we fill in the wildcard symbol $*$ indicating that we do not care if any operation happens at this crossing point. Let's summarize the above construction by

$$
F_{ij} = \begin{cases}
r & \text{if} & (i, j, 1) \in \mathtt{L} \\
l & \text{if} & (i, j, 1) \in \mathtt{L} \\
e & \text{if} & (\mathtt{BOARDSTATE}\ (i, j) = 1 \wedge \mathtt{BOARDSTATE}\ (i, j + 1) = 1) \wedge ((i, j, k) \notin \mathtt{L}, k \in \{1, -1\}) \\
re & \text{if} & (\mathtt{BOARDSTATE}\ (i, j) = 0 \wedge \mathtt{BOARDSTATE}\ (i, j + 1) = 1) \wedge ((i, j, k) \notin \mathtt{L}, k \in \{1, -1\}) \\
le & \text{if} & (\mathtt{BOARDSTATE}\ (i, j) = 1 \wedge \mathtt{BOARDSTATE}\ (i, j + 1) = 0) \wedge ((i, j, k) \notin \mathtt{L}, k \in \{1, -1\}) \\
* & \text{if} & (\mathtt{BOARDSTATE}\ (i, j) = 0 \wedge \mathtt{BOARDSTATE}\ (i, j + 1) = 0) \wedge ((i, j, k) \notin \mathtt{L}, k \in \{1, -1\}).
\end{cases}
$$

The flow matrix F takes values in the set $\{r, l, e, re, le, *\}$. In supplementary material is available online at stacks.iop.org/QST/3/035005/mmedia we discuss the mathematical structure of this set in more detail. The above construction gives us a matrix of operations we would like to apply to the initial BOARDSTATE. You can see an example of a BOARDSTATE and HS[L] command with corresponding flow matrix F in figure 6.

3.1.2. An algorithm for parallel shuttling

The task is now to subdivide the flow matrix F into a sequence of shuttling operations that can be performed in parallel. Ideally we would like this sequence to be as short as possible. One simple way to generate a sequence of this form, as described in the beginning of the section, is to perform all operations one column at a time, i.e. lowering the first vertical barrier, setting the required gradients to shuttle every qubit adjacent to that vertical barrier and then move on to the second vertical barrier and so on. This yields a sequence of parallel shuttling operations of depth N. This solution is always possible for any flow matrix F. However, as can be seen in section 3.4 for some flow matrices this is far from an optimal solution. Below we set out in detail an algorithm
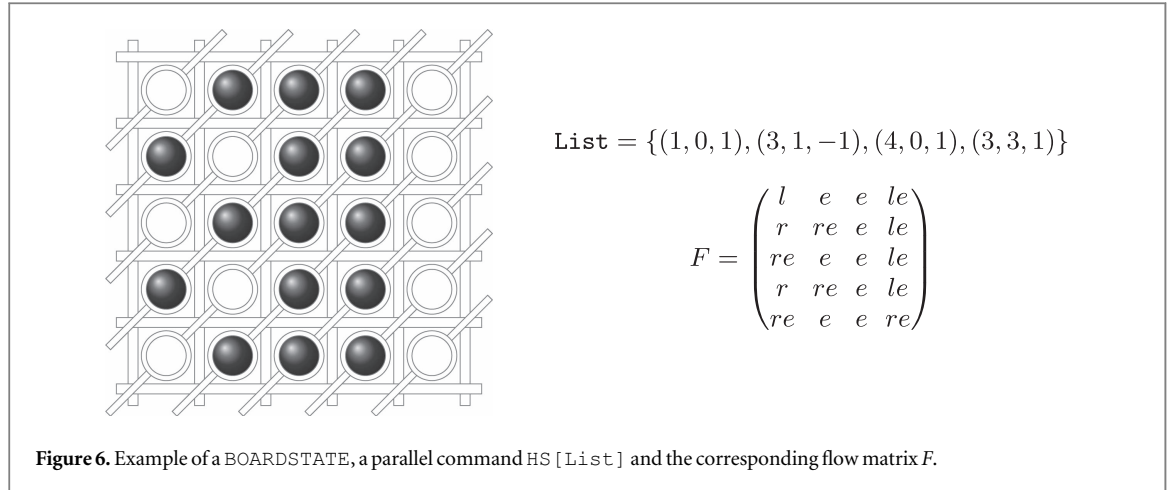
$$\text{List} = \{(1,0,1),(3,1,-1),(4,0,1),(3,3,1)\}$$

$$F = \begin{pmatrix} l & e & e & le \\ r & re & e & le \\ re & e & e & le \\ r & re & e & le \\ re & e & e & re \end{pmatrix}$$

Figure 6. Example of a `BOARDSTATE`, a parallel command `HS[List]` and the corresponding flow matrix *F*.

Table 3. Table listing the time complexity and relative power of the **CheckIndependence**( ) and **IndependenceSet**( ) for three different classes of subroutine. The parameters *N* and *M* are the size of the QDP grid and the size of the input set *S* respectively. The subroutine classes 'simple' and 'greedy commutative' can be run in polynomial time while the class 'k-commutative' is fixed-parameter-tractable, with independent parameter *k*. This subroutine yields increasingly better results (shorter shuttling sequences) for increasing *k* but the time complexity grows rapidly with *k*. See supplementary material for a detailed description of these subroutines. For an illustration of the advantages of these algorithms, one can consider the shuttle commands given in section 3.4. A naive line-by-line approach will take *N* time-steps while it is easy to see that the above algorithms find sequences of length one.

Name	Time Complexity	Relative power
Simple	$O(\textbf{CheckIndependence}(\ )) = O(NM)$	Shorter sequences than line-by-line.
	$O(\textbf{IndependenceSet}(\ )) = O(NM)$	
k-commutative	$O(\textbf{CheckIndependence}(\ )) = O(NMM^kk^4)$	Shorter sequences than 'Simple'.
	$O(\textbf{IndependenceSet}(\ )) = O(NMM^kk^4)$	Shorter sequences for increasing *k*.
Greedy commutative	$O(\textbf{CheckIndependence}(\ )) = O(NM^3)$	Shorter sequences than 'Simple'.
	$O(\textbf{IndependenceSet}(\ )) = O(NM^3)$	Relation to 'k-commutative' unknown.

that finds better (shorter sequences) solutions for many flow matrices. The algorithm is based on the idea that some columns of the flow matrix *F* can be 'dependent' on each other. For instance two columns could be composed of the exact same operations (up to a shift accounting for the fact that the diagonal lines do not run along the rows but diagonally). This means we can perform the shuttle operations in the two columns simultaneously by lowering barriers corresponding to these columns and setting the required gradient. More complicated forms of dependence are also possible. We can use dependence of columns to perform operations in parallel. For instance if a command `HS[L]` calls for exactly the same shuttling events to happen on two columns (up to a constant vertical shift proportional to the horizontal distance of the two columns) we can perform these shuttling operations in a single time step.

This notion of (in)dependence of columns is captured by a call to an 'independence subroutine'. We call these subroutines **CheckIndependence**(*S*, *v*) which takes in a set of columns *S* of the flow matrix *F* of and a column *v* of the flow matrix *F* and decides whether *v* is independent of the elements of *S* and **DependenceSet** (*S*, *v*) which takes in a set of columns *S* and a column *v* and returns a subset *A* of *S* containing all the columns on which *v* depends. We will discuss various versions of these subroutines leading to more or less refined notions of independence (and thus longer of shorter shuttling sequences) in supplementary material. We list all subroutines discussed in supplementary material in table 3 together with their relative power and time complexity. Here we just treat the subroutines as a given and build the algorithm around it. This algorithm does not always yield optimal sequences of parallel shuttling operations, but it can be run using a polynomial amount of classical side-resources given that the subroutine can be constructed efficiently, (see theorem 1) while we expect an algorithm that always produces optimal shuttling sequences to require exponential computational resources. Below we give a pseudo-code version of the algorithm. Note that this algorithm only produces sequences of parallel shuttling operations where the ordering of the operations does not matter. See supplementary material for more details on how this property is guaranteed.

Algorithm 1. Generate list of parallel shuttle operations

Input: Flow matrix F
Output: List of shuttle operations L
1:
2:   // We will consistently write columns of the flow matrix F as v_i where i indicates
3:   // the column index of v_i in F.
4:
5:   **Set** S to an empty list
6:
7:   // Below we construct a set of independent columns S and sets of dependence A_i for the dependent columns v_i.
8:
9:   **for** $i \in [0 : N - 2]$ **do**
10:      **Set** v_i to the i'th column of F
11:
12:         //Check if the column v_i is independent of the columns already in the set S. This requires a
13:         // subroutine call to **CheckIndependence**. See appendix for the construction of this subroutine.
14:
15:      **if CheckIndependence**(v_i, S) is TRUE **then**
16:
17:            // The function θ maps the symbols $*$, re, le to e. We must do this since we want to make an operation
18:            // out of v_i later and the wildcard elements $*$, re, le do not strictly correspond to operations. Other
19:            // choices are possible here but in keeping with the idea of doing a
20:            // minimal amount of operations, the mapping to e is a good choice.
21:
22:            **Add** $\theta(v_i)$ to S
23:            **Set** A_i to $\{v_i\}$
24:         **else**
25:            **Set** A_i to **DependenceSet**(S, v_i)
26:         **end if**
27:   **end for**
28:
29:   // Initialize an empty ordered set that will contain all HS[L] commands in sequence.
30:
31:   **Set** L to an empty ordered set
32:   **for** $v_i \in S$ **do**
33:
34:         // Initialize an empty set that will contain all tuples for a single HS[L] command.
35:
36:      **Set** L to an empty set
37:      **for** $j \in [0 : N - 2]$ **do**
38:
39:            // Check if v_i is in the dependence set A_j.
40:
41:         **if** $v_i \in A_j$ **then**
42:            // Loop over all components of v_i.
43:         **for** $k \in [0 : \text{length}(v_i) - 1]$ **do**
44:
45:               // ϕ maps the r, l, e valued column v to an $1, -1, 0$ valued vector as $\phi(r) = 1, \phi(l) = -1, \phi(e) = 0$.
46:
47:               **if** $\phi[(v_i)_k] \neq 0$ **then**
48:                  **Add** $(j, k - (i - j), \phi[(v_i)_k])$ to L
49:               **end if**
50:            **end for**
51:         **end if**
52:      **end for**
53:      **Add** HS[L] to L
54:   **end for**
55:   **return** L

Theorem 1. *The algorithm described in algorithm 1 has a time complexity upper bounded by*

$$O(N^4) + N \cdot O(\mathbf{CheckIndependence}(S, \ v_i)) + N \cdot O(\mathbf{DependenceSet}(S, \ v_i)), \tag{6}$$

where N is the number of columns in the input flow matrix F.

The subroutines **CheckIndependence**(S, v_i) *and* **DependenceSet**(S, v_i) *both take in a set S of independent columns of the flow matrix F and a column v_i of the flow matrix F and respectively check whether v is independent of the set S or produce a subset A of S on which v depends. Various versions of these subroutines are discussed in supplementary material and their time complexities are given in table 3.*

Proof. Begin by noting that the algorithm 1 consists of two independent **For**-loops. The first **For**-loop (lines 2-11) calls its body N times (ignoring constant factors). Calling the **For**-loop body (lines 3-10) in the worst case requires calling both **CheckIndependence**( ) and **DependenceSet**( ) plus some constant time instructions. This means the first **For**-loop has a worst case complexity of
$N \cdot O(\textbf{CheckIndependence}( )) + N \cdot O(\textbf{DependenceSet}( )).$

The second **For**-loop (lines 13-25) consists of three nested **For**-loops of length $O(N)$ with an **If**-clause inside the first two **For**-loops (line 16) constant time operation at the bottom (line 19). The first **For**-loop can be seen to be of order $O(N)$ by noting that the set of independent columns S can be no bigger than N in which case all columns are independent. The second **For**-loop (line 15) is $O(N)$ bounded by construction. Note that the **If**-clause on line 16 can take time $O(N)$ to complete since for any dependency set A_j we can only say that $|A_j| \leqslant N$ (since A_j is a subset of the set of all columns of F). The third loop is also $O(N)$ bounded since $length(v_i) \leqslant N$ for all columns v_i of F. Tallying up all contributions we arrive at equation (6), which completes the argument. ∎

This concludes our discussion of parallel shuttling operations. Before we move on however, it is worth pointing out an interesting example where this shuttling can be used a subroutine to perform more complicated operations. This example will also be of use later when discussing parallel measurement in section 3.3 and the mapping of quantum error correction codes in sections 4, 4.4, 4.5.

3.1.3. Selective parallel single-qubit rotations

In this section we will discuss a particular example that illustrates the use of abstracting away the complexity of parallel shuttling. Imagine a QDP grid initialized in the so-called *idle* configuration. This configuration can be seen in figure 7. We will focus on the qubit in the odd columns (i.e. the set $\mathcal{B}$). Imagine a subset S of these qubits to be in the state $|1\rangle$ and the remainder of these qubits to be in the state $|0\rangle$. The qubits on in the set $\mathcal{R}$ can be in some arbitrary (and possibly entangled) multiqubit state $|\Psi\rangle$. We would like to change the states states of the qubits in the set S to $|0\rangle$ without changing the state of any other qubit. Due to the limited single-qubit gates (see section 2.3.3) available in the QDP this is a non-trivial problem for some arbitrary set S. However using the power of parallel shuttling we can perform this task as follows. Begin by defining the set $\hat{S}$ to be the complement of S in $\mathcal{R}$. Now we begin by performing the parallel shuttling operation

$$\texttt{HS[L]}, \quad \texttt{L} = \{(i, j, 1) \parallel (i, j) \in \hat{S}\}. \tag{7}$$

Here we abuse notation a bit by referring to $\hat{S}$ as the set of locations of the qubits in $\hat{S}$. This operation in effect moves all qubits in $\hat{S}$ out of $\mathcal{R}$ (and into $\mathcal{B}$, note that the dots the qubits are being shuttled in are always empty because of the definition of the idle configuration). Now we can use a semi-global single-qubit rotation (as discussed in section 2.3.3) to perform an X-rotation on all qubits in $\mathcal{R}$, which is now just all qubits in the set S. This flips changes the states of the qubits in S from $|1\rangle$ to $|0\rangle$ without changing the state of any other qubit. Following this we can restore the BOARDSTATE to its original configuration by applying the parallel shuttling command
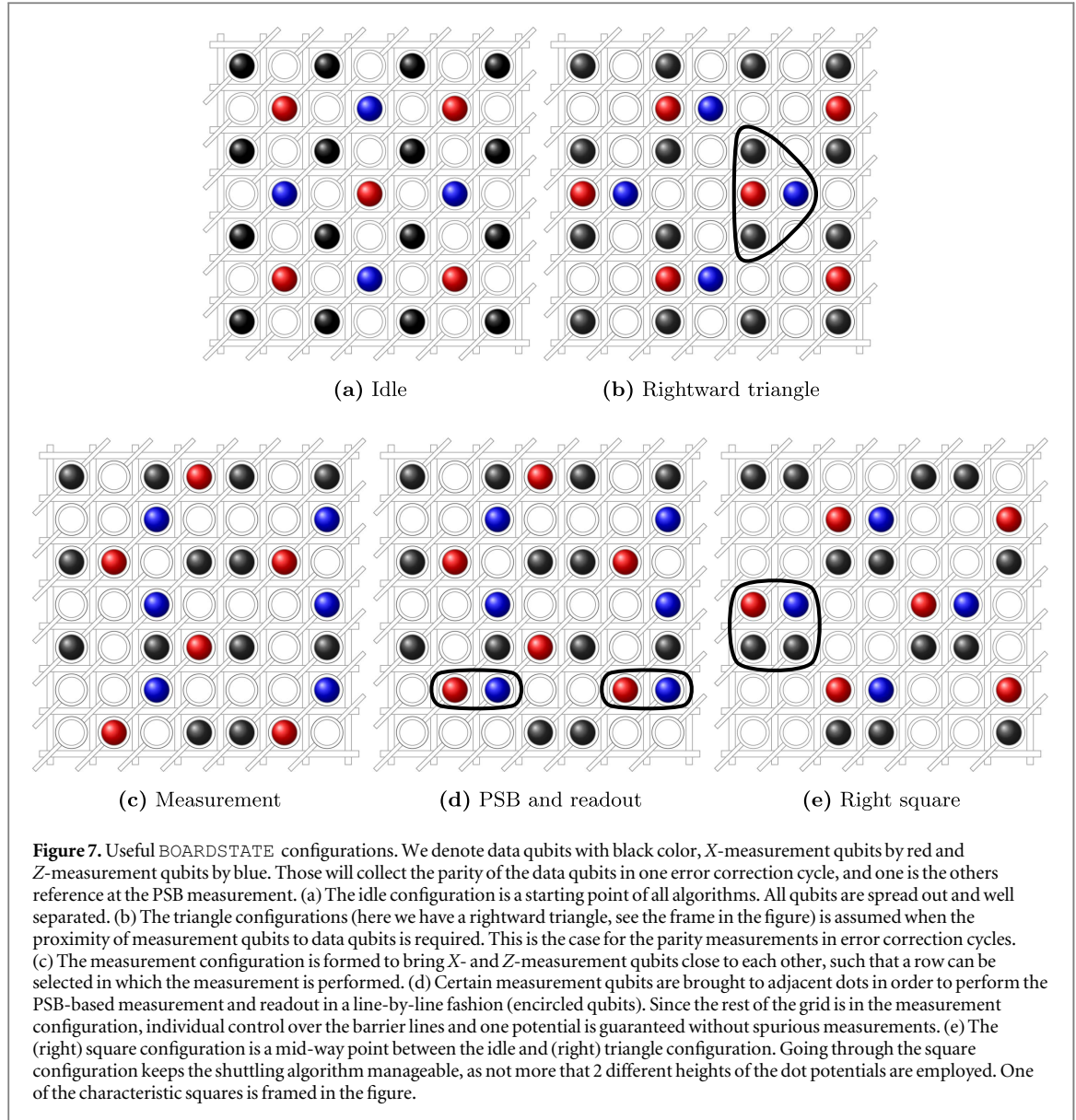
$$\texttt{HS[L]}, \quad \texttt{L} = \{(i, j, -1) \parallel (i, j) \in \hat{S}\}. \tag{8}$$

Now we have applied the required operation. Note that at no point we had to reason about the structure of the set S itself. This complexity was taken care of by the classical subroutines embedded in HS[L]. Next we discuss performing parallel two-qubit gates.

3.2. Parallel two-qubit gates

Similar to parallel shuttling it is in general rather involved to perform parallel two-qubit operations in the QDP. We can again define parallel versions of the OPCODES for two-qubit operations and then analyze how to perform them as parallel as possible (again having access to classical side computation).

OPCODE	Effect
HI[L]	Perform VI[(i,j)] for $(i,j) \in $ L
VI[L]	Perform HI[(i,j)] for $(i,j) \in $ L

Figure 7. Useful `BOARDSTATE` configurations. We denote data qubits with black color, *X*-measurement qubits by red and *Z*-measurement qubits by blue. Those will collect the parity of the data qubits in one error correction cycle, and one is the others reference at the PSB measurement. (a) The idle configuration is a starting point of all algorithms. All qubits are spread out and well separated. (b) The triangle configurations (here we have a rightward triangle, see the frame in the figure) is assumed when the proximity of measurement qubits to data qubits is required. This is the case for the parity measurements in error correction cycles. (c) The measurement configuration is formed to bring *X*- and *Z*-measurement qubits close to each other, such that a row can be selected in which the measurement is performed. (d) Certain measurement qubits are brought to adjacent dots in order to perform the PSB-based measurement and readout in a line-by-line fashion (encircled qubits). Since the rest of the grid is in the measurement configuration, individual control over the barrier lines and one potential is guaranteed without spurious measurements. (e) The (right) square configuration is a mid-way point between the idle and (right) triangle configuration. Going through the square configuration keeps the shuttling algorithm manageable, as not more that 2 different heights of the dot potentials are employed. One of the characteristic squares is framed in the figure.

Given an `BOARDSTATE` and a `HI[L]` command one could use an algorithm similar to the algorithm presented for shuttling. We can again construct a matrix F such that $F_{ij} = 1$ is for all tuples (i, j) in `L` indicating the locations where we desire a two-qubit operation to happen and $F_{ij} = 0$ everywhere else. Now we can use the algorithm presented above for shuttling to decompose the matrix F into a series of parallel `HI[L]` operations. However, since we have CPHASE$^2 = \mathbb{I}$ the independence subroutine reduces to linear independence of the columns of F modulo 2. This means we can find an *optimal* decomposition into parallel operations by finding the Schmidt-normal [22, Chapter 14] form of the matrix F (Note that we do have to 'tilt' the matrix F to account for the fact that as posed the diagonal lines of the matrix F are its 'rows'). We can make the same argument given a `BOARDSTATE` and a `VI[L]` command but now the Schmidt-normal form must be found modulo 4 as $(\sqrt{\text{SWAP}})^4 = \mathbb{I}$. As both addition modulo 2 ($\mathbb{Z}_2$) and addition modulo 4 ($\mathbb{Z}_4$) are principal ideal domains both of the Schmidt-normal forms can be found efficiently and generate optimal sequences of parallel two-qubit interactions. The depth of the sequence of operations is now proportional to the rank of the matrix F over $\mathbb{Z}_2$ (CPHASE) or $\mathbb{Z}_4$ ($\sqrt{\text{SWAP}}$). However, as mentioned before, the parallel operation of two-qubit gates in the QDP will mean taking a hit in operation fidelity vis-a-vis the more controllable line-by-line operation [4]. Since this operation fidelity is typically a much larger error source than the waiting-time-induced decoherence stemming from line-by line operation we will for the remainder of the paper assume line-by-line operation of the two-qubit gates. This will have an impact when performing quantum error correction on the QDP which we will discuss in more detail in section 5.

For the sake of completeness we also define a parallel version of the CNOT OPCODE. The same considerations of parallel operation hold for the parallel use of CNOT gates as they hold for the CPHASE and $\sqrt{\text{SWAP}}$ gates. We continue the discussion of parallelism in the QDP by analyzing parallel measurements.

OPCODE	Effect
`VC[L]`	Perform `VC[(i,j)]` for every (i, j) in `L`

3.3. Parallel measurements

Performing measurements on an arbitrary subset of qubits on the QDP is in general quite involved. Every qubit to be measured requires an ancilla qubit and this ancilla qubit must be in a known computational basis state, and an empty dot must be adjacent as a reference for the readout process. The qubits must then be shuttled such that they are horizontally adjacent to their respective ancilla qubits and must also be located in such a way such that they are in the right columns for the PSB process to take place (revisit section 2.3.2 for more information). This can be done using the algorithm for parallel shuttling presented above but in the worst case this will take a sequence of depth $O(N)$ parallel shuttle operations. On top of the required shuttling the PSB process itself (from a control perspective similar to shuttling) must be performed in a way that depends on the BOARDSTATE and the configuration of the qubit/ancilla pairs. In general this PSB process will be performed line-by-line (for the fidelity reasons mentioned in the beginning of the section) and hence requires a sequence of depth $O(N)$ parallel grid operations (plus the amount of shuttling operations needed to attain the right measurement configuration in the first place). Due to this complexity we will not analyze parallel measurement in detail but rather focus on a particular case relevant to the mapping of the surface code. But first we define a parallel measurement OPCODE `M[L]` which takes in a list of tuples (i, j, k) denoting locations of qubits to be measured (i, j) and whether the ancilla qubit is to the left ($k = -1$) or to the right ($k = 1$) of the qubit to be measured.

OPCODE	Effect
`M[L]`	Perform `M[(i, j, k)]` for every (i, j, k) in `L`

3.3.1. A specific parallel measurement example

Let us consider a specific example of a parallel measurement procedure that will be used in our discussion of error correction. We begin by imagining the BOARDSTATE to be in the *idle* configuration (figure 7 top left). We next perform the shuttle operations needed to change the BOARDSTATE to the *measurement* configuration. This configuration (and how to reach it by shuttling operations from the idle configuration) will be discussed section 3.4 and can be seen in figure 7(c). Next take the qubits to be measured in the parallel measurement operation to be the red qubits in figure 7. The qubits directly to the right or to the left of those qubits will be the required readout ancillas (blue in figure 7). We will assume that the readout ancillas are in the $|0\rangle$ state. If some ancilla qubits are in the $|1\rangle$ state instead we can always perform the procedure given in section 2.3.3 to rotate them to $|0\rangle$ without changing the state of the other qubits on the grid. Note that all the ancilla qubits are in the set $\mathcal{B}$ whereas the qubits to be read out are in the set $\mathcal{R}$. This means that we can perform the PSB process by attempting to shuttle the qubit to be measured (red) into the sites occupied by the ancilla qubits (blue). In principle we could perform this operations in parallel by executing the operations

$$\text{VS[L]}, \quad \text{L} = \{(i, j, 1) \,\|\, i = 0 \bmod 2, j = 1 \bmod 2, i + j = 1 \bmod 4\} \tag{9}$$

to bring the qubits to be measured (red) horizontally adjacent to the ancilla qubits (blue) and then

$$\text{M[L]}, \text{L} = \{(i, j, 1) \,\|\, i = 1 \bmod 4, j = 1 \bmod 4\} \tag{10}$$

and

$$\text{M[L]}, \text{L} = \{(i, j, -1) \,\|\, i = 3 \bmod 4, j = 3 \bmod 4\}. \tag{11}$$

All of these operations can be performed in a single time step. However for fidelity and control reasons laid out in the beginning we would prefer to perform these operations in a line-by-line manner. In particular we would like to perform these operations one row at a time since this gives us the ability to control both diagonal and vertical lines individually for each measurement. However we must take care to avoid spurious operations. For instance when performing measurements on the qubits at locations $(1, 1)$ and $(1, 5)$ we must avoid also performing a measurement on the qubit at location $(5, 5)$. To avoid this situation we will bring only the bottom row of qubits to be measured horizontally adjacent to the ancilla qubits, perform the PSB process and readout on

that row only and then shuttle the qubits to be measured back down again. This we repeat going up in rows until we reach the end of the grid. More formally we perform the following sequence of operations.

Algorithm 2. Loop over OPCODES to perform line-by-line measurements

---

1: **for** $i \in [0: N - 2]$ **do**
2:     **if** $i = 1 \bmod 4$ **then**
3:       VS[L],   L $= \{(i - 1, j, -1) \,\|\, j = 1 \bmod 4\}$
4:       M[L],   L $= \{(i, j, 1) \,\|\, j = 1 \bmod 4\}$
5:       VS[L],   L $= \{(i - 1, j, 1) \,\|\, j = 1 \bmod 4\}$
6:     **end if**
7:     **if** $i = 3 \bmod 4$ **then**
8:       VS[L],   L $= \{(i - 1, j, -1) \,\|\, j = 3 \bmod 4\}$
9:       M[L],   L $= \{(i, j, -1) \,\|\, j = 3 \bmod 4\}$
10:       VS[L],   L $= \{(i - 1, j, 1) \,\|\, j = 3 \bmod 4\}$
11:     **end if**
12: **end for**

We will use this particular procedure when performing the readout step in a surface code error correction cycle in section 4.4. This concludes our discussion of parallel operation on the QDP. We now move on to highlight some BOARDSTATE configurations that will feature prominently in the surface and color code mappings.

3.4. Some useful grid configurations

There are several configurations of the BOARDSTATE that show up frequently enough (for instance in the error correction codes in section 4.4) to merit some special attention. In this section we list these specific configurations and show how to construct them.

3.4.1. Idle configuration

The idle configuration is the configuration in which the QDP is initialized. As shown in figure 7 it has a checkerboard pattern of filled and unfilled sites. In this configuration no two-qubit gates can be applied between any qubit pair but since it minimizes unwanted crosstalk between qubits [4], it is good practice to bring the system back to this configuration when not performing any operations. For this reason we consider the idle configuration to be the starting point for the construction of all other configurations.

3.4.2. Square configuration

As seen in figure 7(e) the square configurations consist of alternating filled and unfilled 2×2 blocks of sites. The so-called right square configuration can be reached from the idle configuration by a shuttling operation HS[L] with the set L being

$$
\begin{aligned}
\text{L} = \{(i, j, 1) \,\|\, &i = 1 \bmod 2, \\
&j = 1 \bmod 2, i + j = 2 \bmod 4\} \cup \{(i, j, -1) \,\|\, i = 0 \bmod 2, \\
&j = 1 \bmod 2, i + j = 3 \bmod 4\}.
\end{aligned}
\tag{12}
$$

Note that this operation only takes a single time step, the square configuration is shown in figure 7(e). The right square configuration is characterized by the red (Z-) ancilla being in the left corner of every square. Another flavor of this configuration is the left square configuration, where the red ancilla is in the upper right corner, and the blue one in the left. The left square configuration can be reached from the idle configuration by a shuttling operation HS[L] with the set L being.

$$
\begin{aligned}
\text{L} = \{(i, j, 1) \,\|\, &i = 0 \bmod 2, \\
&j = 0 \bmod 2, i + j = 2 \bmod 4\} \cup \{(i, j, -1) \,\|\, i = 1 \bmod 2, \\
&j = 0 \bmod 2, i + j = 1 \bmod 4\}.
\end{aligned}
\tag{13}
$$

These configurations are used as an intermediate step for us to reach the triangle configurations.

3.4.3. Measurement Configuration

The measurement configuration can be reached from the idle configuration in three time-steps by the following sequence of parallel shuttling operations.

$$\text{HS[A]}, \quad \text{A} = \{(i, j, -1), \ (i-1, j-1, 1) \| i = 1 \bmod 4, \ j = 2 \bmod 4\},$$
$$\text{HS[B]}, \quad \text{B} = \{(i-1, j-1, 1) \| i = 3 \bmod 4, \ j = 1 \bmod 4\},$$
$$\text{VS[C]}, \quad \text{C} = \{(i, j, -1) \| i = 0 \bmod 2, \ j = 1 \bmod 2, \ i+j = 1 \bmod 4\}. \tag{14}$$

This configuration can be seen in figure 7(d) and it is an intermediate state in the measurement process of the blue qubits using the red qubits as ancillas. How this measurement protocol works in detail is described in section 3.3.

3.4.4. Triangle configurations

In order to collect the parity of the data qubits in the error correction cycles, we need to align the ancilla qubits with the data qubits, according to the two-qubit gates used. This is reflected in the use of triangle configurations. There are two triangle configurations that can be reached in a single parallel shuttling step from the right square configuration. The first one, seen in figure 7(b), is called the rightward triangle configuration. It can be reached from the square configuration by the grid operation HS [L] with the set L being

$$\text{L} = \{(i, j, -1) \| 0 = 1 \bmod 2, \ j = 1 \bmod 2, \ i+j = 3 \bmod 4\}, \tag{15}$$

which does as much as to shuttle the right data qubit of every square (framed squares in figure 7(e)) to the empty dot on its right. In this configuration, we are able to perform high-fidelity two-qubit gates between the two data qubits and the ancilla in every triangle. In order to reach the neighboring pair of data qubits with the same ancilla, we start from the left square configuration and shuttle the left data qubit to the left. Operationally, we would do HS [L] with

$$\text{L} = \{(i, j, 1) \ \| \ i = 0 \bmod 2, \ j = 0 \bmod 2, \ i+j = 2 \bmod 4\}. \tag{16}$$

Note again that these parallel shuttling operations can be performed in a single time step. From these configurations the idle configuration can also be reached in a single time step. In the next section these configurations will feature prominently in the mapping of several quantum error correction codes to the QDP architecture.

4. Error correction codes

In this section we will apply the techniques we developed in the previous sections to map several quantum error correction codes to the QDP.

4.1. Introduction

First we recall some basic facts about quantum error correction codes and topological stabilizer codes in particular. The focus will be on practical application, for a more in depth treatment of quantum error correction and topological error correction codes we refer to [7]. Recall first the Pauli operators on a single qubit:

$$X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{17}$$

Given a system of n qubits we denote by P_i the Pauli operator $P \in \{X, Z\}$ acting on the i'th qubit. With this definition we can see write the n qubit Pauli group $\mathcal{P}_n$ as the group generated by the operators $\{X_i, Z_j : \ i, j \in [1 : n]\}$ under matrix multiplication. A stabilizer quantum error correction code acting on n physical qubits and encoding k logical qubits can then be defined as the joint positive eigenspace of an abelian subgroup $\mathcal{S}$ of $\mathcal{P}_n$ generated by $n - k$ independent commuting Pauli operators. Operationally, this code is then defined by measuring the generators of $\mathcal{S}$ and if necessary perform corrections to bring the state of the system back into the positive joint eigenspace of these generators. This is a very general definition and it is not guaranteed that a code defined this way yields any protection against errors happening. Below we will see some common examples of stabilizer error correction codes that do have good protection against errors. On top of that, these codes have the desirable property that their stabilizers are in some sense 'local'. That is they can be implemented on qubits lying on a lattice such that the stabilizer generators can be measured by entangling a patch of qubits that is small with respect to the total lattice size. The most well known example of a code of this type is the so-called planar surface code.

4.2. Planar surface code

The planar surface code is probably the most well known practical quantum error correction code due to its high threshold [23], the availability of efficient decoding algorithms [24]. To construct the planar surface code (in
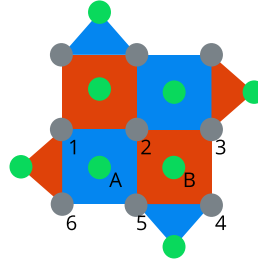
Figure 8. Schematic representation of a distance three rotated planar surface code [25]. The gray circles represent the data qubits supporting the code. The green circles represent ancilla qubits, which are used to perform the stabilizer measurements which define the code. These stabilizer measurements are represented by the red (*Z*-type stabilizers) and blue faces (*X*-type stabilizers). The ancilla qubit in the middle of a face will be used to perform a stabilizer measurement of the data qubits on the corners of that face. The actual quantum circuits used to perform these stabilizer measurements are shown in figure 9.
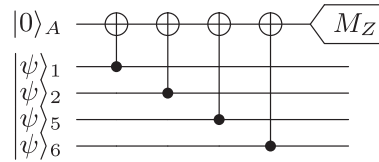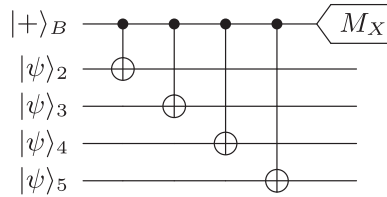
Figure 9. Quantum circuits for performing the *X*- and *Z*-stabilizer measurements of the planar surface code [6, 7, 26, 27]. The qubits *A* and *B* (see figure 8) are ancilla qubits used to perform stabilizer measurements on the the data qubits on the corners of the faces defining the code. The data qubits associated to the face of qubit *A* are $\{1, 2, 5, 6\}$ and likewise $\{2, 3, 4, 5\}$ for qubit *B*.

particular we will use the so-called rotated planar surface code [25], as it uses less physical qubits per logical qubit) we will consider a regular $n \times n$ square lattice of degree four (every node has four connected neighbors) and we will place qubits on each node. We will define the generators of the abelian group $\mathcal{C}$ that defines the surface code by alternately placing *X*- and *Z*-quartets on the faces of the lattice (in figure 8 the red faces correspond to *X*-stabilizer quarters while the green faces correspond to *Z*-stabilizer quartets). This $X(Z)$ will indicate that we pick the generator $X^{\otimes 4}$ ($Z^{\otimes 4}$) on the four qubits on the corners of the $X(Z)$ face. Note that this means that all of the generators commute with each other since they either act on disjoint sets of qubits or act on sets that have an overlap of exactly 2 qubits. Since $XZ = -ZX$ we have that $X^{\otimes 2}Z^{\otimes 2} = Z^{\otimes 2}X^{\otimes 2}$ which means that all generators commute. These generators (plus appropriate generators on the boundary of the lattice) define a stabilizer group which specifies a code space of dimension 2, i.e. a single logical qubit. We can locally measure these $X(Z)$ stabilizers by using the circuits [6, 7, 26, 27] illustrated in figure 9. This construction calls for one ancilla qubit per lattice face.

4.3. 2D color codes

Another important class of planar topological codes are the 2D color codes [9]. These codes are defined on 3-colorable tilings of the Euclidean plane. Two popular tilings are the so-called 6.6.6. and 4.8.8. tilings corresponding to hexagonal and square-octagonal tilings respectively. To construct the code qubits are places on all vertices of the tiling and *X*- and *Z*-stabilizers are associated to every tile by applying $X(Z)$ to every qubit on the corner of the tile. With suitable boundary conditions this construction encodes a single logical
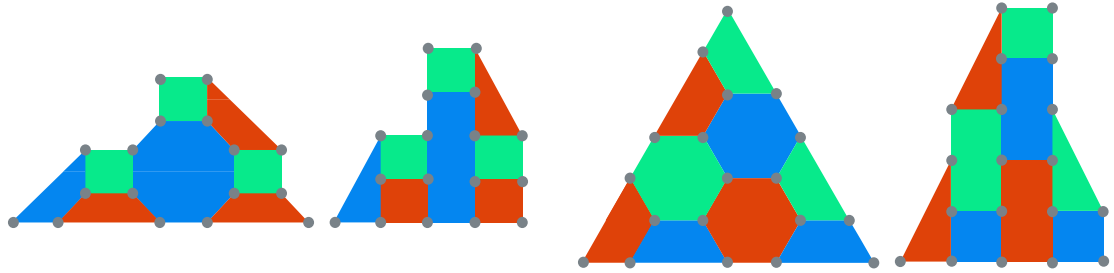
Figure 10. Distance 5 examples of the 4.8.8. (first from left) and 6.6.6. (third from left) color codes [9] and their deformed versions (second from left and fourth from left respectively). The vertices correspond to data qubits and every colored face corresponds to both an *X*- and a *Z*-stabilizer to be measured. These stabilizers can be measured by using weight 4, 6 and 8 versions of the circuits shown in figure 9. The deformation of the codes does not change the code properties at all. They are a visual guide that facilitates the mapping the the crossbar grid in section 4.5.
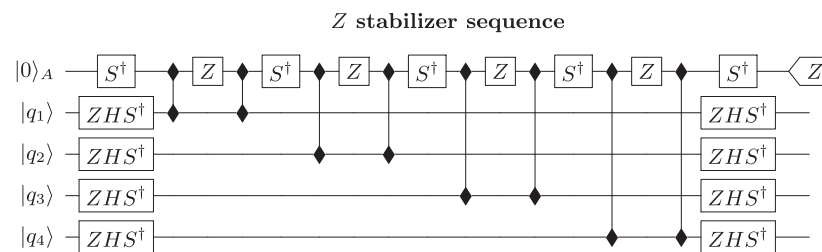
Figure 11. *Z*-stabilizer measurement circuit using the $\sqrt{\text{SWAP}}$ as the main two-qubit gate. The *Z*- and *S*-rotations can be performed by the timing procedure described in section 2.3.3.

qubit with a distance proportional to $\sqrt{n}$ with *n* the number of physical qubits. See figure 10 for examples of the 6.6.6. and 4.8.8. color codes of distance five. Note that these pictures do not include ancilla qubits for measuring the stabilizers. The planar color codes have lower thresholds than the planar surface code but are more versatile when it comes to fault-tolerant gates. The planar color codes support the full Clifford group as a transversal set, making quantum computation on color codes more efficient than on the surface code. In the next section we will focus on mapping these codes to the QDP using the concepts introduced in section 3.

4.4. Surface code mapping

We now describe a protocol that maps the surface code on the architecture described in section 2. The surface code layout has a straightforward mapping that places the data qubits on the even numbered columns and the *X*- and *Z*-ancillas on the odd columns. This means we have single-qubit control over all data qubits and all ancilla qubits separately. There are two ways to perform the surface code cycle; we could use either the $\sqrt{\text{SWAP}}$ gate or the CPHASE gate as the main two-qubit gate. Since in practice the $\sqrt{\text{SWAP}}$ gate has higher fidelity [4] we will use this gate. We begin by changing the circuits performing the *X*- and *Z*-stabilizer measurements to work with $\sqrt{\text{SWAP}}$ rather than CNOT. We can emulate a CNOT gate by using two $\sqrt{\text{SWAP}}$ gates interspersed with a *Z*-gate on the control plus some single-qubit gates. As described in section 2.3.5 the *Z*- and *S*-gates on the ancilla qubit can performed by waiting, which means they can be performed locally while the single-qubit operations on the data qubits can be performed in parallel using the global unitary rotations described in section 2.3.3. The *X*- and *Z*-circuits using $\sqrt{\text{SWAP}}$ are shown in figure 11.

We will split up the quantum error correction cycle by first performing all *X*-type stabilizers (the *X*-cycle) and then all *Z*-type stabilizers (*Z*-cycle). This means we can use the idle *Z*- (*X*-) ancilla to perform a measurement on the *X*- (*Z*-) ancilla at the end of the *X* (*Z*) cycle. For convenience we included a depiction of the surface code *Z*-cycle unit cell in figure 12 (right). The qubit labeled 'A' is the ancilla used for the *Z*-stabilizer circuit. The numbered qubits are data qubits and the qubit labeled 'B' is the qubit used for reading out the 'A' qubit. It is also the ancilla qubit for the *X*-cycle. We now describe the steps needed to perform the *Z*-cycle in parallel on the entire surface code sheet. For convenience we ignore the surface code boundary
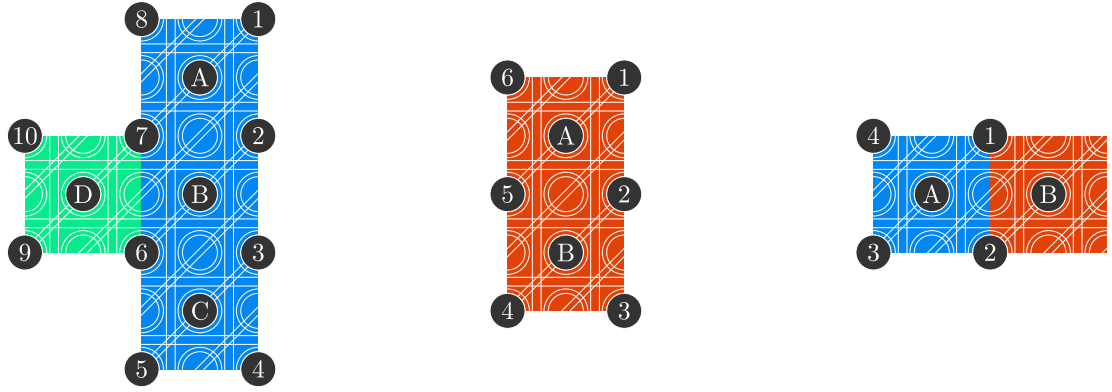
Figure 12. Unit cells of the deformed 4.8.8. and 6.6.6. codes (left and middle respectively) and the unit cell of the surface code (*Z*-cycle) with the gray circle corresponding to qubits. For the 4.8.8. unit cell the qubit labeled 'A' is the ancilla qubit for the octagon (now a rectangle) sub-cell while the qubit labeled 'D' is the ancilla for the square sub-cell. The qubit labeled 'B' is used to read out the qubit labeled 'D' and the qubit labeled 'C' is used to read out the ancilla qubit for the octagon cell directly below the square cell (not pictured). The qubits labeled by numbers are the data qubits. For the 6.6.6. unit cell the qubit labeled 'A' is the ancilla qubit used to perform the stabilizer measurement while the qubit labeled 'B' is used to read out the 'A' qubit for the unit cell directly to the bottom left (not pictured). The numbered qubits are again data qubits. For the surface code unit cell the qubit labeled 'A' is the ancilla used for the *Z*-cycle stabilizer measurement while the qubit labeled 'B' is the qubit used to read out the 'A' qubit. It is also the qubit used as the ancilla for the *X*-stabilizer cycle. The numbered qubits are again data qubits. Note that this unit cell mirrors when moving upwards. That is, the unit cell above the one pictured will have the ancilla qubit B to the right of qubit A instead of to the left as pictured.

conditions but these can be easily included. The *X*-cycle is equivalent up to different single-qubit gates ($XS^\dagger$ instead of $ZHS^\dagger$ on the data qubits, $HS^\dagger$ instead of $S^\dagger$ on the ancilla) and shifting every operation 2 steps up, e.g. setting i to $i + 2$.

The surface code *Z*-cycle

Step 1: Initialize in the idle configuration

Step 2: Apply $ZHS^\dagger$ to all qubits in $\mathcal{R}$ (data) and $S^\dagger$ to qubits in $\mathcal{B}$ (ancilla)

Step 3: Go to right square configuration

Step 4: Go to rightward triangle configuration

Step 5: Perform CNOT between qubits A and 1 by performing VC[L] with

$$\mathtt{L} = \{(i, j) \parallel i = 1 \bmod 2, j = 0 \bmod 2, i + j = 3 \bmod 4\}$$

Step 6: Perform CNOT between qubits A and 2 by performing VC[L] with

$$\mathtt{L} = \{(i, j) \parallel i = 0 \bmod 2, j = 0 \bmod 2, i + j = 2 \bmod 4\}$$

Step 7: Go to idle configuration

Step 8: Go to left square configuration

Step 9: Go to leftward triangle configuration

Step 10: Perform CNOT between qubits A and 3 by performing VC[L] with

$$\mathtt{L} = \{(i, j) \parallel i = 1 \bmod 2, j = 0 \bmod 2, i + j = 1 \bmod 4\}$$

Step 11: Perform CNOT between qubits A and 4 by performing VC[L] with

$$\mathtt{L} = \{(i, j) \parallel i = 0 \bmod 2, j = 0 \bmod 2, i + j = 0 \bmod 4\}$$

Step 12: Go to idle configuration

Step 13: Apply $ZHS^\dagger$ to all qubits in $\mathcal{R}$ (data) and $S^\dagger$ to qubits in $\mathcal{B}$ (ancilla)

Step 14: Apply measurement ancilla correction step for qubit B as described in section 3.1.3

Step 15: Go to measurement configuration

Step 16: Perform Pauli Spin Blockade measurement process as described in section 3.3 using qubit B as ancilla to qubit A

Step 17: Go to idle configuration

4.5. Color code mapping

The mapping of the color codes is largely analogous to that of the surface code. We begin with the 6.6.6. color code as it is easiest to map. We begin by deforming the tiling on which the color code is defined such that it is more amenable to the square grid structure of the QDP. This is fairly straightforward as can be seen from the $d = 5$ example in figure 10. In the deformed tiling it is clear how to map the code to the crossbar grid layout. We once again place all data qubits in the even columns and all ancilla qubits in the odd columns. This places the unit 'hexagon' seen in the deformed code in a 3×5 tile on the QDP (see figure 12 (right) for this unit tile). This places all data qubits in $\mathcal{R}$ and 2 extra qubits in $\mathcal{B}$, both of which could be used as an ancilla in the stabilizer circuit. We will always choose the top qubit (qubit 'A') of these two in the hexagon unit cell as the ancilla qubit for the error correction cycles. The extra (bottom) qubit (qubit 'B') in the unit cell will be used to perform the readout of the ancilla qubit of the unit hexagon to its direct left. This has the advantage of making the readout process independent of the measurement results of the previous cycles (as was the case in the surface code). Note also that the ancilla qubits are positioned along diagonal lines on the QDP grid. This makes the quantum error correction cycle very analogous to the surface code. We once again must split up the X- and Z-cycles (again due to the limited single-qubit rotations possible). Below we present the steps needed to perform the Z-cycle (which now measures a weight 6 operator). The X-cycle is identical up to differing single-qubit rotations on the data qubits.

The 6.6.6 color code *Z*-cycle

Step 1: Perform **Steps 1 to 11** in the surface code Z-cycle to perform CNOT s between the ancilla (qubit A) and the data qubits 1, 2, 5, 6 in the unit hexagon and end in the idle configuration

Step 2: Go to idle configuration but with all even columns up and all odd columns down by performing `VS[L]` with

$$L = \{(i, j, 1) \ \| \ i = 0 \bmod 2, \ j = 0 \bmod 2\}$$
$$\bigcup \{(i, j, -1) \ \| \ i = 1 \bmod 2, \ j = 1 \bmod 2\}$$

Step 3: Go to right square configuration
Step 4: Go to rightward triangle configuration
Step 5: Perform CNOT between qubits A and 3 by performing `VC[L]` with

$$L = \{(i, j) \ \| \ i = 1 \bmod 2, \ j = 0 \bmod 2, \ i + j = 1 \bmod 4\}$$

Step 6: Go to idle configuration
Step 7: Go to left square configuration
Step 8: Go to leftward triangle
Step 9: Perform CNOT by performing between qubits A and 4 `VC[L]` with

$$L = \{(i, j) \ \| \ i = 0 \bmod 2, \ j = 0 \bmod 2, \ i + j = 2 \bmod 4\}$$

Step 10: Go to idle configuration
Step 11: Invert **Step 6** by performing `VS[L]` with

$$L = \{(i, j, -1) \ \| \ i = 0 \bmod 2, \ j = 0 \bmod 2\}$$
$$\bigcup \{(i, j, 1) \ \| \ i = 1 \bmod 2, \ j = 1 \bmod 2\}$$

Step 12: Apply $ZHS^{\dagger}$ to all qubits in $\mathcal{R}$ (data) and $S^{\dagger}$ to qubits in $\mathcal{B}$ (ancilla)
Step 13: Go to measurement configuration
Step 14: Perform Pauli Spin Blockade measurement process as described in section 3.3 using qubit B as ancilla to read out qubit A (unit cell to the right)
Step 15: Go to idle configuration

Next up is the 4.8.8. color code. We deform the tiling on which the code is defined similarly to the 6.6.6. code. The deformed 4.8.8. code lattice can be seen in figure 12 (left). We again place the data qubits in the set $\mathcal{R}$ the ancilla qubits in the set $\mathcal{B}$. See figure 12 for a layout of the unit cell of the 4.8.8. code on the QDP. Note that there are two different types of tiles in this code. The square tile has one qubit (qubit 'D' in figure 12) in $\mathcal{B}$, which we will use as ancilla qubit for that tile. The deformed octagon tile has three qubits in $\mathcal{B}$. We will use the topmost qubit (qubit 'A') as the ancilla qubit for the tile while the middle one (qubit 'B') serves as the readout qubit for the square tile ancilla directly to its left and the bottommost one (qubit 'C') will be used to perform the readout of the octagon directly below the square tile (not pictured). Because the structure of the 4.8.8. code is less amenable to direct mapping the stepping process is a little more complicated. We will again only write down the Z-cycle with the X-cycle being the same up to initial and final single-qubit rotations on the data qubits.

<div style="text-align:center">

The 4.8.8 color code *Z*-cycle

</div>

Step 1: Initialize in the idle configuration

Step 2: Apply $ZHS^\dagger$ to all qubits in $\mathcal{R}$ (data) and $S^\dagger$ to qubits in $\mathcal{B}$ (ancilla)

Step 3: Go to right square configuration

Step 4: Go to rightward triangle configuration

Step 5: Perform CNOT between qubits A and 1 and D and 7 by performing VC[L] with

$$\mathtt{L} = \{(i,j) \ \| \ i = 1 \bmod 2, j = 0 \bmod 2, [i+j = 3 \vee 7 \bmod 16]\}$$

Step 6: Perform CNOT between qubits A and 2 and d and 6 by performing VC[L] with

$$\mathtt{L} = \{(i,j) \ \| \ i = 0 \bmod 2, j = 0 \bmod 2, [i+j = 2 \vee 6 \bmod 16]\}$$

Step 7: Go to left square configuration

Step 8: Go to left triangle configuration

Step 9: Perform CNOT between qubits A and 8 and D and 9 by performing VC[L] with

$$\mathtt{L} = \{(i,j) \ \| \ i = 1 \bmod 2, j = 0 \bmod 2, [i+j = 1 \vee 5 \bmod 16]\}$$

Step 10: Perform CNOT between qubits A and 7 and d and 10 by performing VC[L] with

$$\mathtt{L} = \{(i,j) \ \| \ i = 0 \bmod 2, j = 0 \bmod 2, [i+j = 0 \vee 4 \bmod 16]\}$$

Step 11: Go to idle configuration

Step 12: Go to idle configuration but with all even columns up and all odd columns down by performing VS[L] with

$$\mathtt{L} = \{(i,j,1) \ \| \ i = 0 \bmod 2, j = 0 \bmod 2\}$$
$$\cup \ \{(i,j,-1) \ \| \ i = 1 \bmod 2, j = 1 \bmod 2\}$$

Step 13: Go to right square configuration

Step 14: Go to rightward triangle configuration

Step 15: Perform CNOT between qubits A and 3 by performing VC[L] with

$$\mathtt{L} = \{(i,j) \ \| \ i = 1 \bmod 2, j = 0 \bmod 2, i+j = 3 \bmod 16\}$$

Step 16: Perform CNOT between qubits A and 4 by performing VC[L] with

$$\mathtt{L} = \{(i,j) \ \| \ i = 0 \bmod 2, j = 0 \bmod 2, i+j = 2 \bmod 16\}$$

Step 17: Go to idle configuration

Step 18: Go to left square configuration

Step 19: Go to leftward triangle configuration

Step 20: Perform CNOT between qubits A and 6 by performing VC[L] with

$$\mathtt{L} = \{(i,j) \ \| \ i = 1 \bmod 2, j = 0 \bmod 2, i+j = 1 \bmod 16\}$$

Step 21: Perform CNOT between qubits A and 5 by performing VC[L] with

$$\mathtt{L} = \{(i,j) \ \| \ i = 0 \bmod 2, j = 0 \bmod 2, i+j = 0 \bmod 16\}$$

Step 22: Go to idle configuration

Step 23: Invert **Step 6** by performing VS[L] with

$$\mathtt{L} = \{(i,j,-1) \ \| \ i = 0 \bmod 2, j = 0 \bmod 2\}$$
$$\cup \ \{(i,j,1) \ \| \ i = 1 \bmod 2, j = 1 \bmod 2\}$$

Step 24: Repeat **Steps 2-23** but setting *i to i* + 2 and *j to j* + 1

Step 25: Apply $ZHS^\dagger$ to all qubits in $\mathcal{R}$ (data) and $S^\dagger$ to qubits in $\mathcal{B}$ (ancilla)

Step 26: Go to measurement configuration

Step 27: Perform Pauli Spin Blockade measurement process as described in section 3.3 using qubit B (unit cell to the right) as ancilla for qubit A and using qubit C as ancilla for qubit D

Step 28: Go to idle configuration

---

5. Discussion

In this section we evaluate the mapping of the error corrections codes described above and argue numerically that it is possible to attain the error suppression needed for practical universal quantum computing. We will do this exercise for the planar surface code, as it is the most popular and best understood error correction code. The description given in section 4.4 assumes that all operations can be implemented perfectly in parallel. In practice though, for the reasons outlined in section 3 many operations that can in principle be done in parallel will be done in a line-by-line fashion. Note that for surface code in an array like this, the length of a quadratic grid scales linearly with the code distance as $N = 2d + 1$. This means that the time performing a surface code cycle and thus the number of errors affecting a logical qubit rises linearly with the code distance and hence this mapping of the surface code will not exhibit an error correction threshold. As a

Table 4. Time step count per step in terms of different types of possible gates for the line-by-line implementation of the surface code cycle described in section 4.4. The number of time-steps is quoted in terms of the code distance *d*. This table does not specify the exact order in which the operations happen, see section 5.1 for an explanation of the time flow. Note that the table shows the average of the time step counts for the *X*- and *Z*-cycles. The actual time count for the individual *X*- and *Z*-cycles is slightly different due to the boundary conditions of the surface code. The exact count for the *Z*-cycle can be obtained by replacing *d* by $d-1$ in every entry (except for the last column) whereas the exact count for the *X*-cycle is obtained by replacing *d* with $d+1$ in every column bar the last one. Since $(d+1)+(d-1)=2d=d+d$ this makes no difference for the full cycle count. Table cells that are left empty signify zero entries.

Steps	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	*X-Z* cycle average	Cycle total
$\sqrt{\text{SWAP}}$ gate					$2d$	$2d$				$2d$	$2d$							$8d$	$16d$
Z-rotation					$2d$	$2d$				$2d$	d							$7d$	$14d$
Shuttling			d	d			d	d	d			d			$5d$	$2d$	$3d$	$16d$	$32d$
Global rotation		1											1		1			3	6
Measurement																d		d	$2d$

consequence the error probability of the encoded qubit (the logical error probability) cannot be made arbitrarily small but rather will exhibit a minimum for some particular code distance after which the logical error probability will start rising with increasing code distance. The code distance which minimizes the error will depend non-trivially on the error probability of the code qubits. This is not a very satisfactory situation from a theoretical point of view, but from the point of view of practical quantum computation we are not so much interested in asymptotic statements but rather if the logical error probability can be made small enough to allow for realistic computation [27]. As a target logical error probability we choose $P_L = 10^{-20}$ as at this point the computation is essentially error free (for comparison, a modern classical processor has an error probability around 10^{-19} [11]). We will use this number as a benchmark to assess if and for what error parameters the surface code mapping in the QDP yields a 'practical' logical qubit. In order to assess this we must consider in more detail the sources of error afflicting the surface code operation on the QDP. We will begin by detailing how the surface code is likely to be implemented in practice on the QDP and afterward we will consider how this impacts the error behavior of the logical surface code qubit. We will distinguish two classes of error sources: operation induced errors and decoherence induced errors.

5.1. Practical implementation of the surface code

Here we present a mapping of the surface code based on the one presented in section 4.4 but differing in the amount of time-steps used to perform certain operations. In particular we choose to do all shuttle and two-qubit-gate operations in a line-by-line manner. This is a specific choice which we expect will work well but variations of this protocol are certainly possible. As mentioned above this will mean that the time an error correction cycler takes will scale will the code distance. This means it is important to keep careful track of the time needed to perform a cycle. We will do this while describing line-by-line operation of the surface code cycle in greater detail below.

In practice we will perform the protocol in section 4.4 in the following manner. We begin by performing step 1 and 2 for all qubits. Then we apply steps $3-7$ but only to the data and ancilla qubits in the columns 0 and 1. Note that after performing these steps on only the first two columns we are back in the *idle* configuration. Now we repeat the previous for columns 2 and 3 and so forth until we reach the end of the code surface. Having done these operations we are at the end of step 7 (go to *idle* configuration) and the grid is the *idle* configuration. We now repeat the same process to perform steps $8-12$ of section 4.4. Next we perform step 13 which can be done globally. Hereafter we perform step 14 (ancilla correction) in standard line-by-line fashion. Note that even in an ideal implementation step 14 has to be done line-by-line in the worst case. After this we perform step 15 (go to *measurement* configuration) in a line-by line manner and similarly for steps 16 (PSB/readout procedure) and 17 (go to *idle* configuration).

Note that in this line-by-line implementation there is a slight asymmetry between the *X*- and *Z*-cycles. Due to the boundary conditions of the surface code the *X*-cycle will involve $d+1$ columns pairs whereas the *Z*-cycle will involve $d-1$ column pairs. However since $(d+1)+(d-1)=2d$ this is mathematically equivalent to saying that the average cycle involves *d* column pairs. With this understanding we quite in table 4 how many time-steps every step in section 4.4 takes (split up by gates involved in that step) in this particular implementation of the protocol. Note that in this table we do not specify the order in which the operations happen, only to which step they are associated. We also calculate the amount of time-steps (for different gate types) needed for the full surface code error correction cycle.

Table 5. This table lists the total number of gates of a given type a data qubit and an ancilla qubit participate in over the course of a surface code cycle. In supplementary material we give a more detailed per step overview of the operations performed on data qubits and ancilla qubits. For clarity we have chosen qubit 1 in figure 12 (right) as a representative of the data qubits and qubit A in figure 12 (right) as a representative of the ancilla qubits. Other qubits in the code might have a different ordering of operations but there gate counts will be the same, except for the qubits located at the boundary of the code which will have a strictly lower gate count (we can thus upper bound their operation induced errors by those of the representative qubits).

	Data qubit			Z ancilla qubit			Average data/ancilla
	Z-cycle	X-cycle	Total	Z-cycle	X-cycle	Total	
$\sqrt{\text{SWAP}}$ gate	4	4	8	8	0	8	8
Z-rotation	0	0	0	7	0	7	3.5
Shuttling	2	4	6	10	4	14	10
Global rotation	2	2	4	2	3	5	4.5
Measurement	0	0	0	1	1	2	1

5.2. Decoherence induced errors

Decoherence induced errors are introduced into the computation by uncontrolled physical processes in the underlying system. The effect of these processes is called decoherence. Decoherence happens even if a qubit is not being operated upon and the amount of decoherence happening during a computation scales with the time that computation takes. Therefore, to account for decoherence induced errors during the error correction cycle we need to compute how long an error correction cycle takes. Generally any operation on the QDP takes a certain amount of time denoted by τ. We distinguish again five different operations: (1) two-qubit $\sqrt{\text{SWAP}}$ gates, (2) qubit shuttle operations, (3) single-qubit Z gates by waiting, (4) global single-qubit operations, (5) qubit measurements. The time they take we will denote by τ_{sw}, τ_{sh}, τ_z, τ_{gl} and τ_m respectively. In table 4 we performed a count of the total time taken by the surface code error correction cycle using the mapping described in sections 4.4, and 5.1. The table below summarizes the total number of time-steps for every gate type for a full surface code error correction cycle.

Symbol	Operation	Time-steps per cycle
τ_{sw}	$\sqrt{\text{SWAP}}$ gate	$16d$
τ_{sh}	Shuttling	$32d$
τ_z	Z-rotation by waiting	$14d$
τ_{gl}	Global qubit rotation	6
τ_m	Measurement	$2d$

We can now say the total time $\tau_{\text{total}}(d)$ as a function of the code distance d is given by

$$\tau_{\text{total}}(d) = 16d\tau_{sw} + 32d\tau_{sh} + 14d\tau_z + 6\tau_{gl} + 2d\tau_m. \tag{18}$$

This total time can be connected to an error probability by invoking the mean decoherence time of the qubits in the system, the so-called T_2 time [16, 28] (We ignore the influence of T_1 in this calculation as it is typically much larger than T_2 in silicon spin qubits [4, 29]). We can find the decoherence induced error probability P_{dec} [16], Page 384 as

$$P_{dec}(d) = \frac{\tau_{\text{total}}(d)}{2T_2}. \tag{19}$$

Next we investigate operation induced errors. These will typically be larger than decoherence induced errors but will not scale with the distance of the code.

5.3. Operation induced errors

Operation induced errors are caused by imperfect application of quantum operations to the qubit states. There are five operations performed on qubits in the surface code cycle. These are: (1) two-qubit $\sqrt{\text{SWAP}}$ gates, (2) qubit shuttle operations, (3) single-qubit Z gates by waiting, (4) global single-qubit operations, (5) qubit measurements. We will denote the probability of an error afflicting these operations by P_{sw}, P_{sh}, P_z, P_{gl} and P_m respectively. In table 5 we list the total number of gates of a given type a data qubit and an ancilla qubit participate in over the course of a surface code cycle. In supplementary material we give a more detailed per step overview of the operations performed on data qubits and ancilla qubits. For clarity we have chosen qubit 1 in figure 12 (right) as a representative of the data qubits and qubit A in figure 12 (right) as a representative of the ancilla qubits. Other qubits in the code might have a different ordering of operations but their gate counts will be the same, except for the qubits located at the boundary of the code which will have a strictly lower gate count (we can thus upper bound their operation induced errors by those of the representative qubits). For each gate we also
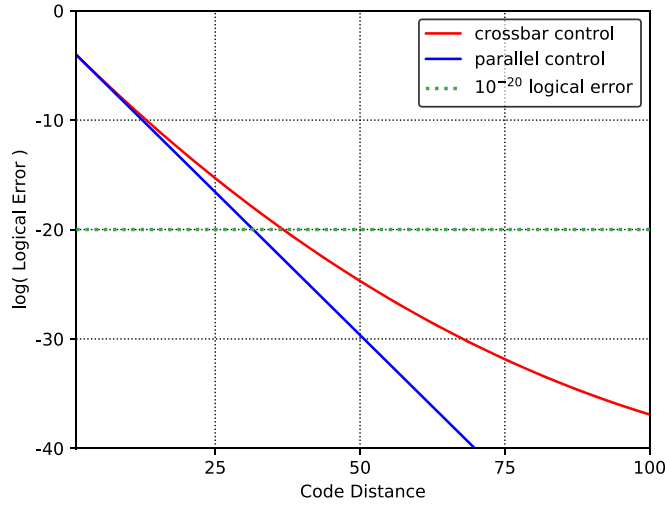
Figure 13. Plot of logical error probability versus code distance for the empirical model given in equation (21) with experimental parameters given in section 5.4. Note that the logical error probability for crossbar operation goes below $P_L = 10^{-20}$ for $d = 37$. This is only slightly slower than parallel operation, which reaches $P_L = 10^{-20}$ for $d = 31$. Due to the scaling of crossbar operation with the code distance the logical error probability bottoms out at some point. This however does not happen until $d = 155$ (not shown) for a logical error rate of $P_L = 10^{-41}$, which is not practically relevant. This rough model gives good indication it is possible to create very low logical error surface code logical qubits in the QDP.

calculate the average number of this gate data and ancilla qubits participate in. This average number will serve as our measure of operation induced error.

5.4. Surface code logical error probability

By tallying up the contributions from operational and decoherence induced errors we can construct a measure for the total error probability per QEC cycle experienced by all physical qubits that make up the code. Note that this a rather crude model that disregards possible influences from inter-qubit correlated errors and time-like correlated errors. Nevertheless it serves as a useful first approximation to the performance of the surface code on the QDP. We define the average per qubit per cycle error probability P_{tot} as

$$P_{\text{tot}}(d) = 8P_{sw} + 3.5P_{sh} + 10P_z + 4.5P_{gl} + P_m + P_{dec}(d). \tag{20}$$

Note that this quantity depends linearly on the code distance d. We can plug this total per cycle error probability P_{tot} into an empirical equation for the logical error probability P_L derived in [27].

$$P_L = 0.03\left(\frac{P_{tot}(d)}{8P_{th}}\right)^{\frac{d+1}{2}} \tag{21}$$

where P_{th} is the per step fault-tolerance threshold of the surface code, which we take to be $P_{th} = 0.0057$ following the result in [27]. The factor of 8 is inserted to account for the fact that the empirical relation derived in [27] is between the physical *per step* error rate and the logical *per cycle* error rate and the protocol analyzed in [27] requires 8 time-steps per surface code error correction cycle. This is an approximation but it will serve our purposes of getting a basic initial estimate of the logical error rate. The next step is to start plugging in experimental numbers into equation equation (20). In the table below we quote error probabilities and operation times for all relevant parameters. These numbers are projections from [4] and references therein.

Operation	Error probability	Time
two-qubit $\sqrt{\text{SWAP}}$ gate	$P_{sw} = 10^{-3}$	$\tau_{sw} = 20$ ns
qubit shuttle	$P_{sh} = 10^{-3}$	$\tau_{sh} = 10$ ns
Z-rotation by waiting	$P_z = 10^{-3}$	$\tau_z = 100$ ns
global qubit rotation	$P_{gl} = 10^{-3}$	$\tau_{gl} = 1000$ ns
measurement	$P_m = 10^{-3}$	$\tau_m = 100$ ns

To convert the operation times into decoherence induced error we use the estimated T_2 time of quantum dot spin qubits in ^{28}Si quoted as $T_2 = 10^9$ ns [4, 29] and equation (19). Plugging these numbers into equation (20) we get the following linear function of the code distance

$$P_{tot} = 2.7 \times 10^{-2} + 2.8d \times 10^{-5} \tag{22}$$

which we can plug into the empirical model equation (21). In figure 13 we plot the logical error probability P_L versus code distance. Note that for the experimental numbers provided the practical quantum computing

benchmarking $\log(P_L) = -20$ is reached for a code distance of $d = 37$. The maximal code distance for the experimental parameters is $d = 155$ for which the log-logical error probability reaches $\log(P_L) = -41$, after which it starts increasing again. For completeness we have also plotted what would happen if we had the power to operate the QDP (with quoted device parameters) completely in parallel. We estimate the physical per cycle error rate of this situation by setting $d = 1$ in equation (22). Note that the difference between parallel and crossbar style operation is not that big, the parallel version reaches $P_L = 10^{-20}$ for $d = 31$. This rough model provides some quantitative justification for the implementation of planar error correction codes in the QDP even in the absence of the ability to arbitrarily suppress logical error. Note also that, due to the long coherence times [4, 29] of the QDP spin qubits, the dominant terms in the expression for the total error probability P_{tot} are those associated with operation induced errors. This provides justification for the line-by-line application of two-qubit gates discussed in section 3.2, which takes a longer time to perform but improves gate quality. It also means that long coherence times and/or fast operation times are likely critical to the success of a crossbar based scheme. This concludes our discussion of the QDP mapping of the surface code. A similar exercise can be done for the 6.6.6. and 4.8.8. color codes but due to their lower thresholds [30], the results will likely be less positive for current experimental parameters.

6. Conclusion

We analyzed the architecture presented in [4], focusing on its crossbar control system. Building on this analysis we presented procedures for mapping the planar surface code and the 6.6.6. and 4.8.8. color codes. Because the line-by-line operation of the crossbar architecture means the noise in a single error correction cycle scales with the distance it is not possible to arbitrarily suppress the logical error rate by increasing the code distance. Instead there will be some 'optimal' code distance for which the logical error rate is the lowest. Using numbers for [4] and an empirical model taken from [27] we analyzed the logical error behavior of the surface code mapping and found that, for current experimental numbers, it is at least in principle possible to achieve logical error probabilities below $P_{log} = 10^{-20}$, making practical quantum computation possible. However, we strongly stress that this is a rather crude estimate and a more detailed answer would have to take into account the details of the dominant error processes in quantum dot qubits. It must also take into account that while it is possible to achieve certain low noise gates and good coherence times in quantum dots qubits in isolation this does not necessarily mean they will be practically achievable in the current QDP design. A future research direction would be to perform much more detailed simulations of this crossbar system, perhaps with input from future experiments. In such a simulation the effect of correlated errors (which might feasibly appear in a crossbar architecture) could be investigated.

Another possible research direction would be to use the currently developed machinery to map more exotic quantum error correction codes. A first step in this direction would be the implementation of variants of the surface code with more resistance to biased noise [31, 32]. Due to the possibility of qubit shuttling, also codes with long distance stabilizers could in principle be implemented. Codes such as the 3D gauge color codes might be prime candidates for this kind of treatment. However, barring some special cases, parallel shuttling is currently being performed in a line-by-line manner. A general classical algorithm for generating optimal (in time) shuttling-steps from an initial to a final `BOARDSTATE` would vastly simplify the task of mapping more exotic codes and also general quantum circuits. Such an algorithm would probably be useful for any future crossbar quantum architecture. In this work we constructed a non-optimal but classically efficient algorithm but finding an algorithm that generates optimal shuttling sequences and analyzing its resource use is still an open problem.

Lastly there are important aspects of quantum error correction that are not discussed in this paper. Two of these aspects are the ability to store multiple logical qubits simultaneously and the ability to perform quantum operations on the logical qubits. A popular way of performing these tasks is by encoding multiple logical qubits in a single surface code sheet by introducing topological defects into the surface code sheet [27]. This process involves not measuring stabilizers at certain points in the sheet, thus creating extra degrees of freedom which can store logical information. The code distance of the code is given by the physical distance (measured in number of physical qubits) between the defects. Operations can then be performed on these logical qubits by moving the defects around each other, a process known as braiding. We think this approach is not natural to the constraints of the crossbar architecture for the following reasons

- Encoding qubits as defects would mean the size of the surface code sheet would scale as the number of encoded qubits. Hence also, in our implementation, the physical error probability per QEC cycle would scale with the number of qubits. This would put an upper limit on the number of qubits that can be implemented.

- Creating and moving around defects requires turning on and of measurements for certain stabilizers in a local manner. This locality runs counter to the design ideas of the crossbar architecture

- Given that the size of the surface code sheet would scale with the number of logical qubits one would likely face significant issues involving uniformity of control parameters of the entire sheet. This would be a significant issue even if the scaling of the physical error probability can be avoided by clever implementation

However we can envision a mode of computation that we speculate is more amenable to this architecture by thinking of an architecture composed of separate modules containing a single logical qubit. We refer to figure 7 of [4] for a proposal of how this could be done. Inside each module our surface code protocol could be run with the ideal code distance given physical error parameters setting the size of these modules. We could then perform logical X and Z gates transversally within the modules and we could perform CNOT gates between adjacent modules via lattice surgery on the edges of the modules. Note that lattice surgery, which involves the turning on and off of stabilizer patches in regular patterns (see [25] for an introduction to lattice surgery), is very amenable to the constraints of the architecture, implying that a high degree of parallelization could be achieved when mapping lattice surgery techniques to the crossbar architecture.

Acknowledgments

ORCID iDs

Jonas Helsen ● https://orcid.org/0000-0001-7218-2585

References

[1] Colless J 2014 *PhD Thesis* University of Sydney
[2] Vandersypen L, Bluhm H, Clarke J, Dzurak A, Ishihara R, Morello A, Reilly D, Schreiber L and Veldhorst M 2017 *npj Quantum Information* **3** 34
[3] Hill C D, Peretz E, Hile S J, House M G, Fuechsle M, Rogge S, Simmons M Y and Hollenberg L C 2015 *Science Advances* **1** e1500707
[4] Li R *et al* 2017 arXiv:1711.03807
[5] Veldhorst M, Eenink H, Yang C and Dzurak A 2017 *Nat. Commun.* **8** 1766
[6] Gottesman D 1998 *Phys. Rev.* A **57** 127
[7] Lidar D A and Brun T A 2013 *Quantum Error Correction* (Cambridge: Cambridge University Press)
[8] Dennis E, Kitaev A, Landahl A and Preskill J 2002 *J. Math. Phys.* **43** 4452
[9] Bombin H and Martin-Delgado M A 2006 *Phys. Rev. Lett.* **97** 180501
[10] Versluis R, Poletto S, Khammassi N, Tarasinski B, Haider N, Michalak D, Bruno A, Bertels K and DiCarlo L 2017 *Physical Review Applied* **8** 034021
[11] Heijmen T 2010 Soft errors from space to ground: Historical overview, empirical evidence, and future trends *Soft Errors in Modern Electronic Systems* ed M Nicolaidis (Berlin: Springer) (https://doi.org/10.1007/978-1-4419-6993-4_1)
[12] Fujita T, Baart T A, Reichl C, Wegscheider W and Vandersypen L M K 2017 arXiv:1701.00815
[13] Hanson R, Kouwenhoven L P, Petta J R, Tarucha S and Vandersypen L M 2007 *Rev. Mod. Phys.* **79** 1217
[14] Taylor J, Engel H-A, Dür W, Yacoby A, Marcus C, Zoller P and Lukin M 2005 *Nat. Phys.* **1** 177
[15] Veldhorst M *et al* 2014 *Nat. Nanotechnol.* **9** 981
[16] Nielsen M A and Chuang I 2002 Quantum circuits *Quantum Computation and Quantum Information* (Cambridge: Cambridge University Press) (https://doi.org/10.1017/CBO9780511976667)
[17] Petta J R, Johnson A C, Taylor J M, Laird E A, Yacoby A, Lukin M D, Marcus C M, Hanson M P and Gossard A C 2005 *Science* **309** 2180
[18] Meunier T, Calado V and Vandersypen L 2011 *Phys. Rev.* B **83** 121403
[19] Watson T *et al* 2017 arXiv:1708.04214
[20] Veldhorst M *et al* 2015 *Nature* **526** 410
[21] Schuch N and Siewert J 2003 *Phys. Rev.* A **67** 032301
[22] Gorodentsev A L 2016 *Algebra I: Textbook for Students of Mathematics* (Berlin: Springer)
[23] Wang D S, Fowler A G and Hollenberg L C 2011 *Phys. Rev.* A **83** 020302
[24] Fowler A G, Stephens A M and Groszkowski P 2009 *Phys. Rev.* A **80** 052312
[25] Horsman C, Fowler A G, Devitt S and Van Meter R 2012 *New J. Phys.* **14** 123011
[26] Terhal B M 2015 *Rev. Mod. Phys.* **87** 307
[27] Fowler A G, Mariantoni M, Martinis J M and Cleland A N 2012 *Phys. Rev.* A **86** 032324
[28] Tomita Y and Svore K M 2014 *Phys. Rev.* A **90** 062320
[29] Tyryshkin A M 2011 *Nat. Mater.* **11** 143–47

[30] Landahl A J, Anderson J T and Rice P R 2011 arXiv:1108.5738

[31] Wootton J R, Peter A, Winkler J R and Loss D 2017 *Phys. Rev.* A **96** 032338

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

[33] West D B *et al* 2001 *Introduction to Graph Theory* vol 2 (Upper Saddle River: Prentice hall)

[34] Bron C and Kerbosch J 1973 *Commun. ACM* **16** 575

[35] Downey R G and Fellows M R 1995 *Theor. Comput. Sci.* **141** 109

[30] Landahl A J, Anderson J T and Rice P R 2011 arXiv:1108.5738

[31] Wootton J R, Peter A, Winkler J R and Loss D 2017 *Phys. Rev.* A **96** 032338

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

[33] West D B *et al* 2001 *Introduction to Graph Theory* vol 2 (Upper Saddle River: Prentice hall)

[34] Bron C and Kerbosch J 1973 *Commun. ACM* **16** 575