Protecting quantum memories using coherent parity check codes

Protecting quantum memories using coherent parity check codes

Joschka Roffe, David Headley, Nicholas Chancellor, Dominic Horsman & Viv Kendon

Joint Quantum Centre (JQC) Durham-Newcastle, Department of Physics, Durham University, South Road, Durham DH1 3LE, United Kingdom

E-mail: joshua.roffe@durham.ac.uk

8 May 2018

Abstract. Coherent parity check (CPC) codes are a new framework for the construction of quantum error correction codes that encode multiple qubits per logical block. CPC codes have a canonical structure involving successive rounds of bit and phase parity checks, supplemented by cross-checks to fix the code distance. In this paper, we provide a detailed introduction to CPC codes using conventional quantum circuit notation. We demonstrate the implementation of a CPC code on real hardware, by designing a $[[4, 2, 2]]$ detection code for the IBM 5Q superconducting qubit device. Whilst the individual gate-error rates on the IBM device are too high to realise a fault tolerant quantum detection code, our results show that the syndrome information from a full encode-decode cycle of the $[[4, 2, 2]]$ CPC code can be used to increase the output state fidelity by post-selection. Following this, we generalise CPC codes to other quantum technologies by showing that their structure allows them to be efficiently compiled using any experimentally realistic native two-qubit gate. We introduce a three-stage CPC design process for the construction of hardware-optimised quantum memories. As a proof-of-concept example, we apply our design process to an idealised linear seven-qubit ion trap. In the first stage of the process, we use exhaustive search methods to find a large set of $[[7, 3, 3]]$ codes that saturate the quantum Hamming bound for seven qubits. We then optimise over the discovered set of codes to meet the hardware and layout demands of the ion trap device. We also discuss how the CPC design process will generalise to larger-scale codes and other qubit technologies.

1. Introduction

Quantum computing experiments have now matured to the extent to which we can realistically expect to see a medium-scale circuit-model device within the next decade [1, 2]. It is hoped these near-future quantum computers will be sufficient for simple algorithms, possibly beyond what can be solved classically. However, the fulfilment of these aims will usually depend upon the efficacy of the adopted quantum error correction (QEC) code and the ease with which it can be compiled onto the chosen quantum technology platform.

Recently, Chancellor et al. [3] introduced the *coherent parity check* (CPC) framework as a toolset for the construction of a versatile new class of QEC codes. CPC codes have a canonical structure that allows any sequence of parity checks to be performed on a quantum register without risk of inducing decoherence. This is in contrast to most traditional QEC protocols, where the choice of parity checks is limited to stabilizers of the encoded quantum data. The freedom in the choice of parity checks therefore affords the CPC framework multiple advantages over conventional QEC.

In the original CPC paper [3], graphical methods based on the ZX *calculus* [4, 5] were used to give a construction for re-purposing general classical error correction codes for QEC. This opens the possibility of constructing QEC codes inspired by highly-optimised classical codes, such as *low density parity check* codes [6]. Furthermore, as the CPC formalism allows for complete freedom in the choice of parity checks, new CPC codes can be discovered numerically, either via brute-force or more sophisticated search techniques.

In this work, we demonstrate a further feature of CPC codes with regards to their implementation on physical hardware. In a theoretical setting, QEC codes are usually formulated in terms of idealised controlled-not (CNOT) gates. However, the native two-qubit entangling gates provided by various qubit technologies are usually of a different form. Consequently, one of the challenges in realising quantum codes is developing efficient methods by which QEC circuits can be realised using the native interaction of the chosen experiment. Here, we show that the symmetric structure of CPC codes enables efficient mapping from the theoretical representation of the code to the hardware-compiled. In particular we show that CPC codes can be implemented with any realistic maximally entangling Clifford native gate, meaning they will be suitable for deployment across a broad range of quantum hardware.

As a simple first example on real hardware, we implement a $[[n = 4,\ k = 2,\ d = 2]]$ CPC quantum code on the IBM Q five-qubit superconducting device (where we have adopted the usual convention whereby n represents the number of physical qubits, k the number of data qubits and d the code distance) [2]. We demonstrate that, for a simple known input state, a version of the $[[4, 2, 2]]$ circuit can be compiled to accommodate the connectivity constraints of the IBM chip. By analysing the experimental data using quantum state tomography, we show that the $[[4, 2, 2]]$ code's syndrome information can be used to improve the fidelity of the output state by post-selection.

There is currently no preferred qubit technology and the first quantum computers will likely be hybrid devices that interface multiple qubit types [7, 8]. In order to realise their full potential, these hybrid schemes will require tailor-made QEC strategies. To this end, we outline a three-stage CPC design process for the construction of hardware-optimised QEC memories.

As a proof-of-concept example, we demonstrate the use of the CPC design process in creating a quantum code for a seven qubit linear ion trap. In the first stage of this process, we show that exhaustive search techniques can be used to discover a large set of $[[7, 3, 3]]$ CPC codes. These $[[7, 3, 3]]$ codes have the highest possible information density

for a non-degenerate QEC code, as dictated by the quantum Hamming bound [9].

The second stage of the code design process involves implementing strategies to select the best CPC code from the discovered set. For the purposes of the ion trap device, we seek to identify the circuits in which the total two-qubit gate count is minimised. This involves consideration of the additional SWAP interactions that must be introduced to mediate interactions between spatially separated qubits.

The final hardware optimisation we consider in the CPC design process is compilation of CPC codes with a device's native two-qubit gate. For the ion trap under consideration, we assume the native interaction is of the form of a maximally entangling symmetrised phase (SP) gate [10]. A CNOT interaction can be implemented from an SP gate, but this requires addition of local single-qubit gates which increases the code overhead. As an example of the native gate compilation, we demonstrate that for many of the $[[7, 3, 3]]$ CPC circuits, constructive simplifications can be applied to reduce the total number of local corrections required.

The $[[7, 3, 3]]$ CPC codes outlined in this paper should be adaptable to many existing ion trap experiments [11, 12, 13, 14]. The ability to encode three data qubits in a seven qubit trap would mark an improvement over the current most widley adopted protocol for quantum memories, the surface code, which requires a minimum of 13 qubits per encoded data qubit [15, 16]. There have been many proposals for quantum codes promising high encoding densities [17, 18, 19, 20, 21]. The CPC construction provides a framework to allow for the automated discovery of high-density codes which are optimised for the requirements of the chosen experiment. Note, however, that the specific CPC code implementations presented in this work are not yet fault tolerant and that making them such will result in additional overhead. As this work covers quantum memories, we do not include discussion of encoded computation. Steps towards developing fault tolerant CPC gates are outlined in [3], and this remains an interesting area for future work.

The paper is structured as follows. In section 2, we give a detailed introduction to the CPC framework, and explain how it can be used to construct full QEC codes. This is followed, in section 3, by the presentation of experimental results obtained by running a simple CPC detection code on the IBM Q quantum computer. In section 4, we provide an overview of the ion trap hardware for quantum computing. In section 5, we demonstrate that the fundamental structure of CPC codes allow them to be efficiently compiled using a wide range of native gates. Section 6 describes the CPC design process and how it can be used to construct hardware-optimised $[[7, 3, 3]]$ codes for the ion trap device. Finally, in section 7, we discuss possible improvements to the CPC design process and how it might be applied in the discovery of larger quantum codes.

2. Coherent parity check (CPC) codes

The signature feature of CPC codes is the ability to implement QEC routines with any sequence of parity checks. This is possible due to a fail-safe code structure that ensures syndrome measurements cannot decohere the register. This freedom in the choice of parity checks gives the CPC framework multiple advantages over traditional QEC techniques. First, it is possible to directly translate the parity checking sequences from classical codes into a CPC code, which allows the derivation of dense QEC codes that encode multiple data qubits per logical block. Second, the CPC framework does not require quantum data to be initially redundantly encoded. Third, the space of possible CPC codes can be searched numerically, meaning code discovery can be automated.

In this section, we outline the tools of the CPC framework, starting with the fundamental CPC gadget. This gadget has a symmetric *encode-error-decode* structure that amounts to an extended measurement of the identity operator. We prove that the CPC gadget is inherently non-disturbing and can be implemented using any parity checking sequence. Following this, we demonstrate how multiple CPC gadgets can be combined to form QEC codes. Finally, we introduce the automated search techniques that will be used in the CPC code design process.

2.1. Traditional quantum error correction

Before beginning our presentation of the CPC framework, we briefly outline the key concepts and shortcomings of conventional stabilizer QEC codes. This will provide a point-of-reference with which to compare CPC codes.

The circuit in figure 1 shows the basic structure of a traditional stabilizer code. A register of data qubits, $|\psi\rangle_D$, is entangled with a number of blank redundancy qubits, $|0\rangle_R$, via an encoding operation to create a logical qubit $|\psi\rangle_L$. At this stage, the data previously stored solely in $|\psi\rangle_D$ is distributed across the combined Hilbert space of data and redundancy qubits [22].

Once the quantum information has been encoded as a logical qubit, errors can be detected by making parity measurements. In practice, this is achieved via the construction shown to the right of the circuit in figure 1. A parity check $\mathcal{P}$ is applied to the logical qubit, and the result copied to an auxillary qubit A, which is prepared in the conjugate basis by Hadamard gates H. Note that a parity check $\mathcal{P}$ is a product of Pauli operators and has eigenvalues ± 1 (for the definition of the Pauli group, consult appendix A). The auxillary qubit is then measured to yield a syndrome. For a well chosen parity check, this syndrome measurement provides information about whether the logical qubit has been subject to an error.

It has been shown that QEC codes based on the above construction can achieve arbitrarily low logical error rates, provided certain threshold conditions are met by qubits at the physical level [23]. However, constructing efficient codes with this approach is difficult owing to limitations on the type of parity check that can implemented. In order to ensure that the syndrome measurement of qubit A does not decohere the encoded

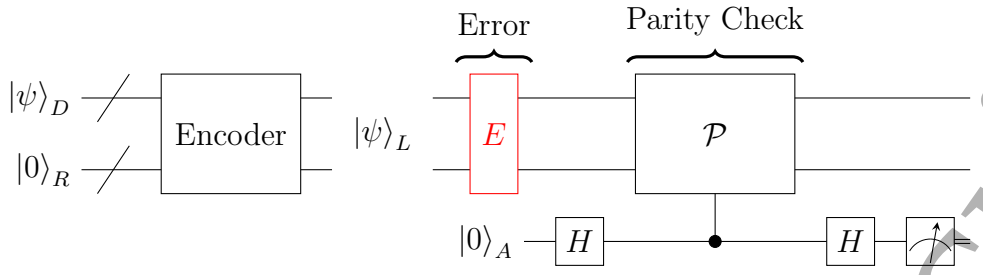


Figure 1: Circuit illustrating the structure of a traditional stabilizer code. A quantum data register $|\psi\rangle_D = |\psi_{d_1}\psi_{d_2}...\psi_{d_n}\rangle$ is entangled with redundancy qubits $|0\rangle_R = |0_{r_1}0_{r_2}...0_{r_m}\rangle$ via an encoding operation to create a logical qubit $|\psi\rangle_L$. After encoding, a parity check $\mathcal{P}$ can be performed on the register to determine whether an error has occurred. The result of this parity check is measured via an auxillary qubit A, which is prepared in the conjugate basis by Hadamard gates H. The slashed wires denote that $|\psi\rangle_D$ and $|0\rangle_R$ are multi-qubit registers. The measurement operator at the end of the wire for qubit A represents a measurement in the computational basis.

quantum information, the parity check must stabilize the logical qubit. Formally [24], we can write this requirement as follows

$$\mathcal{P} \in \mathcal{S}, \tag{1}$$

where the stabilizer $\mathcal{S} = \langle K_i, ..., K_n \rangle$ is a sub-group of the Pauli group $\mathcal{G}$ defined by

$$\mathcal{S} = \left\{ K_i \mid K_i |\psi\rangle_L = (+1)|\psi\rangle_L,\ [K_i, K_j] = 0,\ K_i \neq -\mathbb{1},\ \forall\ (i,j) \right\} \in \mathcal{G}, \tag{2}$$

where $K_{\{i,j\}}$ are the elements of the stabilizer group and $|\psi\rangle_L$ is the logical codeword. The challenges of constructing traditional stabilizer quantum codes are therefore twofold. First, an appropriate encoding operation must be built to create the logical qubit. Second, a compatible set of stabilizer parity checks needs to be discovered so that errors can be checked without compromising the encoded quantum data. As a result of these challenges, the majority of existing QEC codes are limited to the simplest case in which only a single qubit is encoded per logical block. Such $[[n, 1, d]]$ codes can be considered quantum analogues of the most basic classical repetition codes, and incur high overheads in terms of the number of redundancy qubits necessary to achieve the desired error suppression rate.

2.2. The fundamental CPC gadget

The fundamental CPC gadget, shown in figure 2, is the building block upon which all CPC codes are based [3]. The basic premise behind the CPC gadget is that the parity of the quantum register is never explicitly measured. Instead, parity information is stored coherently as quantum data and compared over time. This is made possible by the gadget's symmetric *encode-error-decode* structure.

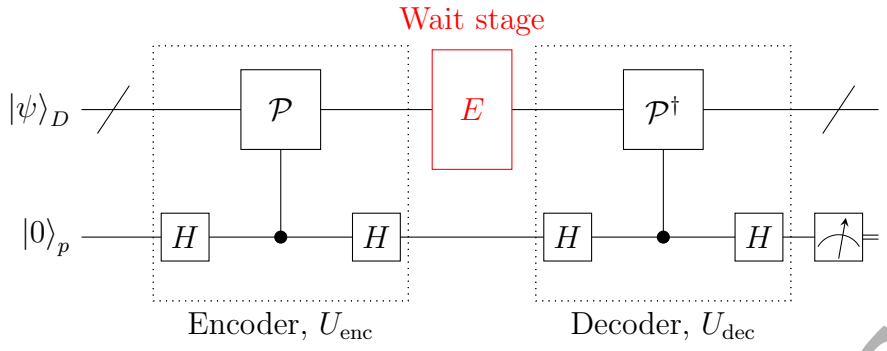


Figure 2: The fundamental CPC gadget illustrating the symmetric *encode-error-decode* structure. The parity qubit p is prepared in the conjugate basis by Hadamard gates, H. Encode stage: a parity check $\mathcal{P}$, controlled by the parity qubit, is applied to the multi-qubit register $|\psi\rangle_D = |\psi_{d_1}\psi_{d_2}...\psi_{d_n}\rangle$ and the result is copied to the parity qubit. The parity qubit is kept coherent throughout the wait stage, during which an error E can occur on the register. Decode stage: the register is disentangled from the parity qubit via the application of the unitary inverse of the first parity check $\mathcal{P}^\dagger$. The final syndrome measurement of qubit p tells us whether the results of the two parity checks differ. For appropriately chosen parity checks, this information can be used to detect errors. The slashed wire denotes that $|\psi\rangle_D$ is a multi-qubit register.

The CPC gadget takes a multi-qubit register $|\psi\rangle_D$ and a parity qubit p, prepared in the state $|0\rangle_p$, as its input. The action of the encode stage of the gadget, labelled U_{enc} in figure 2, is to apply the parity operator $\mathcal{P}$ to the register and record the outcome in parity qubit p. Rather than measuring the syndrome immediately, the parity qubit is kept coherent during a wait stage in which the register is potentially subject to an error E. Note that we are not yet considering errors that occur on the parity qubit. In section 2.5, we outline how multiple CPC gadgets can be combined to allow for error detection on the combined system of register and parity qubits.

Following the wait stage, the parity qubit is disentangled from the register via a decoder operation, labelled U_{dec} in figure 2, which is the unitary inverse of the encoder. The encoder applies the parity operator $\mathcal{P}$ to the register and the decoder applies its inverse $\mathcal{P}^\dagger$. The final syndrome measurement of parity qubit p tells us whether the results of these two parity checks differ. For an appropriately chosen parity check, this syndrome information can indicate whether an error occurred during the wait stage.

To prove its error detection capabilities, it is convenient to rearrange the circuit for the CPC gadget into the form shown in figure 3. This rewrite is achieved by moving the error operator E through the parity check operator $\mathcal{P}$. Both the error gate and the parity check gate are Pauli group operations. A property of the Pauli group is that its elements either commute or anti-commute with one another. Consequently, the effect of pushing the error operator to the front of the circuit is to introduce a global phase $\Phi(E, \mathcal{P})$ on the register which is controlled by the parity qubit. This global phase is

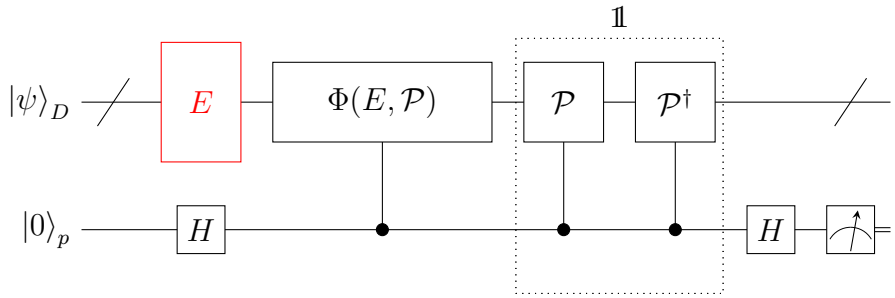


Figure 3: To prove the non-disturbing nature of the fundamental CPC gadget, it is useful to rearrange the circuit by moving the error operator E through the first parity check $\mathcal{P}$. Following this rewrite, the controlled parity check operators are adjacent and cancel. In this form, the CPC gadget can be viewed as a measurement of the $\Phi(E,\mathcal{P}) = \pm\mathbb{1}_D$ operator on the data register. The value of the final syndrome measurement depends only upon whether E commutes with $\mathcal{P}$. The slashed wire denotes that $|\psi\rangle_D$ is a multi-qubit register.

dependent upon both the parity check and the error operator, and is defined as follows,

$$\Phi(E,\mathcal{P}) = \begin{cases} (+1)\mathbb{1}_D, & \text{if } [E,\mathcal{P}] = 0 \\ (-1)\mathbb{1}_D, & \text{if } [E,\mathcal{P}] \neq 0, \end{cases} \tag{3}$$

where $\mathbb{1}_D$ is the identity operator on the data register and the commutator is given by $[E,\mathcal{P}] = E\bullet\mathcal{P} - \mathcal{P}\bullet E$. Note that, after the rewrite, the controlled parity-check operators are adjacent to each other and cancel. The full mathematical action of the CPC circuit U_{CPC}, can now be expressed as follows,

$$U_{\text{CPC}} |\psi\rangle_D |0\rangle_p = (\mathbb{1} + \Phi(E,\mathcal{P}))\, E |\psi\rangle_D |0\rangle_p + (\mathbb{1} - \Phi(E,\mathcal{P}))\, E |\psi\rangle_D |1\rangle_p . \tag{4}$$

Using the definition of the global phase operator $\Phi(E,\mathcal{P})$ given in equation (3), the output of the CPC gadget simplifies to

$$U_{\text{CPC}} |\psi\rangle_D |0\rangle_p = \begin{cases} E |\psi\rangle_D |0\rangle_p , & \text{if } [E,\mathcal{P}] = 0 \\ E |\psi\rangle_D |1\rangle_p , & \text{if } [E,\mathcal{P}] \neq 0. \end{cases} \tag{5}$$

From the above we can see that eventual syndrome measurement of parity qubit p depends only upon whether $\mathcal{P}$ commutes with E. If no error occurs during the wait stage, then $E = \mathbb{1}_D$ and the syndrome is measured deterministically as '0'. Likewise, if an error does occur, but it commutes with the parity operator, $[E,\mathcal{P}] = 0$, then the syndrome is also '0'. Finally, if the error anti-commutes with the parity check, $[E,P] \neq 0$, then the syndrome is measured as '1'. A quantum error detection protocol can therefore be constructed from the CPC gadget by selecting a parity check that anti-commutes with the error to be identified. In the following subsections, we will show that CPC gadgets can be combined to create full QEC codes which can detect and localise multiple error types simultaneously.

The CPC gadget can be thought of as an extended measurement of the $\pm\mathbb{1}_D$ operator on the data register, where the sign depends upon the commutation relation between $\mathcal{P}$ and E. As the $\pm\mathbb{1}_D$ operator is trivially non-disturbing for all quantum

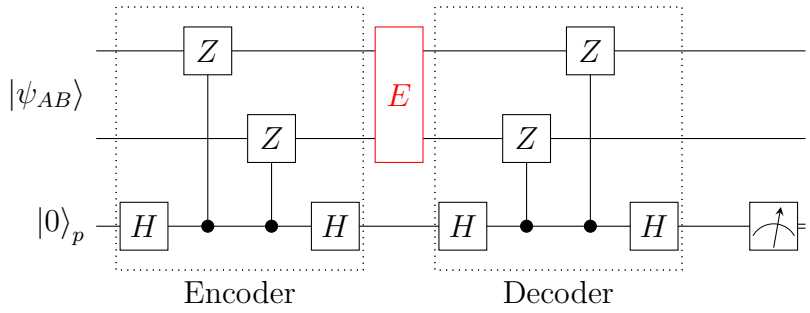


Figure 4: A CPC gadget for detecting single bit-flips on a two-qubit data register $|\psi_{AB}\rangle$. The gadget returns a '0' syndrome measurement if there is no error and a '1' if a bit-flip occurs during the wait stage.

states, there is no need for CPC codes to encode quantum information as logical qubits. Furthermore, it is clear from the output of the CPC gadget in equation (5), that the quantum data register is completely disentangled from the parity qubit prior to syndrome measurement. As a result, the only requirement on the parity checks is that they are Pauli group operators

$$\mathcal{P} \in \mathcal{G}. \tag{6}$$

Recall from equation (1), that for traditional codes, the choice of parity checks is limited to the set of stabilizers of the encoded logical qubits. The CPC framework lifts this restriction.

It should be noted that, as the encoders and decoders consists entirely of Clifford operations, CPC codes form a class of stabilizer codes. A detailed explanation of the correspondence between CPC codes and stabilizer codes can be found in Chancellor et al [3]. The specific strength of the CPC framework lies in the fact that the symmetric *encode-error-decode* structure provides a general method for creating a stabilizer code using any sequence of parity checks.

2.3. A CPC gadget for detecting bit-flips

We now provide specific examples of CPC gadgets to detect bit-flips and phase-flips on a two-qubit data register $|\psi_{AB}\rangle$. Following this, we describe how the two types of CPC gadget can be combined to create a $[[4, 2, 2]]$ detection code.

In order to design a CPC gadget that will detect single bit-flips on the register $|\psi_{AB}\rangle$, we need a parity check that anti-commutes with the errors in the set $\mathcal{E}_X = \{X_A, X_B\}$. Setting $\mathcal{P}_{AB} = Z_A Z_B$ satisfies this requirement to give the bit-flip CPC gadget depicted in figure 4. Note that X and Z are Pauli operators which are defined in appendix A. It is useful to rewrite the circuit in figure 4 in terms of CNOT gates using the gate substitution defined by the following matrix equation

$$\mathrm{CZ}_{q_2}^{q_1} = (\mathbb{1}_{q_1} \otimes H_{q_2}) \cdot \mathrm{CNOT}_{q_2}^{q_1} \cdot (\mathbb{1}_{q_1} \otimes H_{q_2}), \tag{7}$$

where CZ is a controlled-Z gate and q_1 and q_2 are the input qubits. The resultant circuit

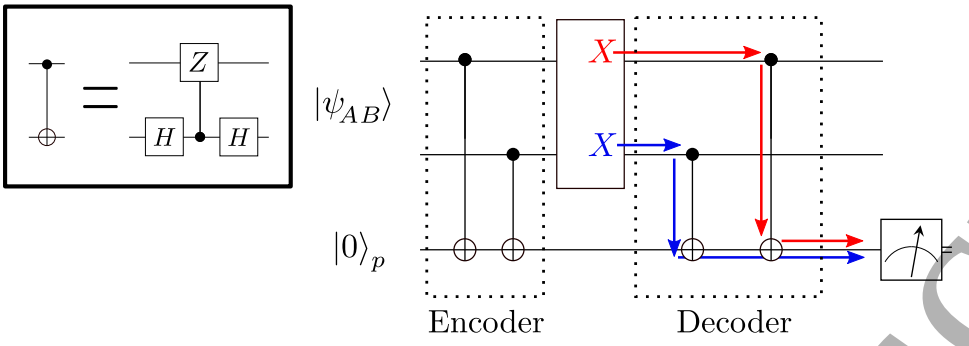


Figure 5: Right: the bit-flip CPC gadget rewritten using the circuit rewrite rule to the left. Expressing the gadget in this form allows for easy visualisation of the propagation of errors from the wait stage to the parity check measurement. The red and blue arrows show the two possible bit-flip propagation pathways.

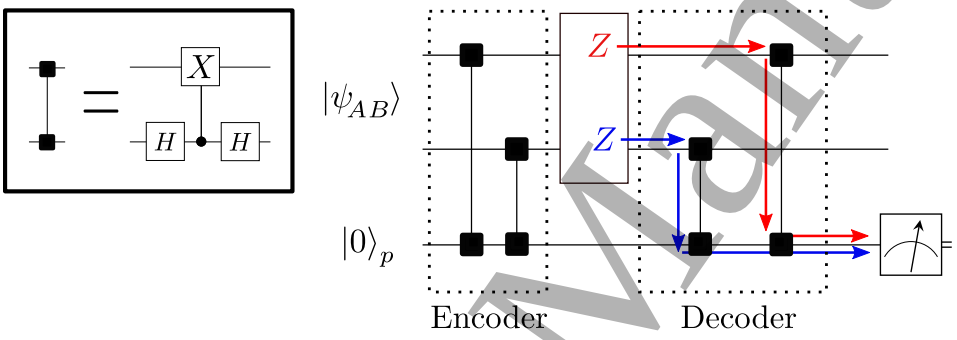


Figure 6: A phase-flip CPC gadget. Phase flips on the register $|\psi_A\psi_B\rangle$ can be detected by setting $\mathcal{P} = X_A X_B$. In the above-right, we have expressed the phase-flip gadget in terms of conjugate-propagator gates, which are defined in the box to the left. The conjugate-propagator gates are symmetric gates which are designed to copy Z errors from one qubit to another. The red and blue arrows depict the possible propagation pathways for Z errors from the wait stage to the parity check measurement.

is shown in figure 5. In this form, the operation of the CPC gadget can easily be visualised by considering the propagation of errors through the decoder. A CNOT gate will propagate a bit-flip error from the control qubit q_1 to the target q_2 as follows,

$$\text{CNOT}_{q_2}^{q_1} \cdot (X_{q_1} \otimes \mathbb{1}_{q_2}) \cdot \text{CNOT}_{q_2}^{q_1} = X_{q_1} \otimes X_{q_2}. \tag{8}$$

Implementing the above propagation rule, the red and blue arrows in figure 5 depict the possible detection pathways for bit-errors from the wait stage to the parity check qubit.

2.4. A CPC gadget for detecting phase-flips

A CPC gadget that detects errors from the set $\mathcal{E}_Z = \{Z_A, Z_B\}$ can be obtained using a parity check of the form $\mathcal{P}_{AB} = X_A X_B$. Figure 6 depicts the phase-flip CPC gadget expressed in terms of the conjugate-propagator gate $\Lambda_{q_2}^{q_1}$ given by

$$\Lambda_{q_2}^{q_1} = (\mathbb{1}_{q_1} \otimes H_{q_2}) \cdot \text{CNOT}_{q_1}^{q_2} \cdot (\mathbb{1}_{q_1} \otimes H_{q_2}). \tag{9}$$

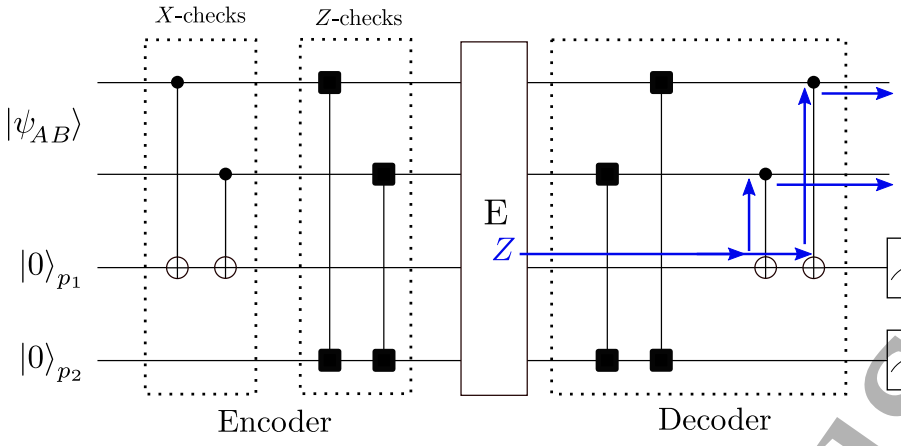


Figure 7: The circuit formed by combining the bit-flip CPC gadget to the phase-flip CPC gadget. This circuit can detect both X and Z errors on qubits A, B and p_2. However, a phase-flip error on qubit p_1 will propagate errors to the register without triggering a syndrome, as shown by the blue arrows. This propagation loophole can be closed through the addition of cross-check operators.

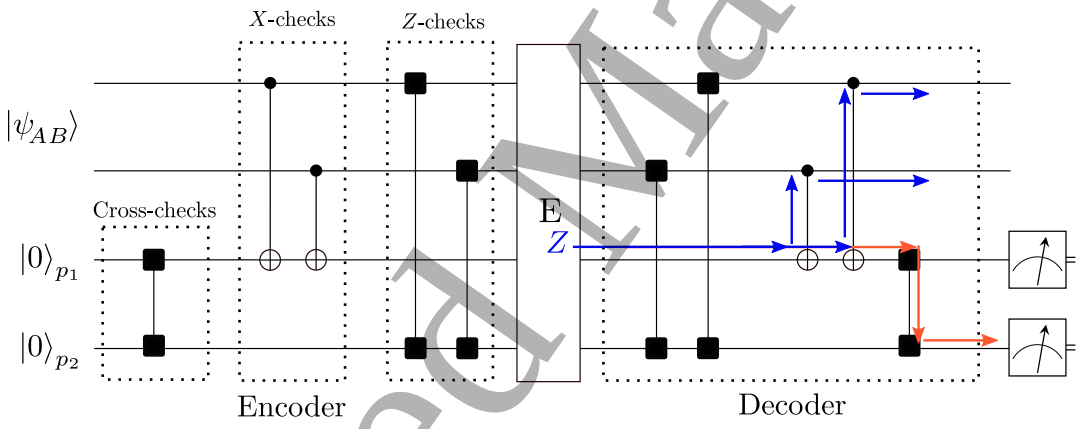


Figure 8: The $[[4, 4, 2]]$ CPC quantum detection code, formed by combining the bit-flip and phase-flip gadgets. The addition of cross-check operators ensures that errors do not propagate from the parity qubits to the register in an undetectable way.

The conjugate-propagator gate is a symmetric two-qubit operator with the following propagation rule for Z-errors

$$\Lambda_{q_2}^{q_1} \cdot (Z_{q_1} \otimes \mathbb{1}_{q_2}) \cdot \Lambda_{q_2}^{q_1} \rightarrow Z_{q_1} \otimes X_{q_2}. \tag{10}$$

Phase-flip errors in the wait stage are copied to the parity qubit via a conjugate-propagator gate which converts the Z-error to an X-error that can be detected in the computational basis. Figure 6 depicts the possible error propagation pathways for errors in the phase-flip CPC gadget.

Error	**Syndrome**
I	$0_{p_1}0_{p_2}$
$X_A, X_B, X_{p_1}, Z_{p_2}$	$1_{p_1}0_{p_2}$
$Z_A, Z_B, Z_{p_1}, X_{p_2}$	$0_{p_1}1_{p_2}$
$Y_A, Y_B, Y_{p_1}, Y_{p_2}$	$1_{p_1}1_{p_2}$

Table 1: The syndrome table for the $[[4, 2, 2]]$ quantum error detection code. If no errors occur, the code returns a '00' syndrome. If a single X, Y or Z error occurs on any of the four qubits, a non-zero syndrome is returned.

2.5. The [[4,2,2]] error detection code

We now show how the bit-flip and phase-flip CPC gadgets can be combined to form a full quantum error detection code. Figure 7 shows the CPC circuit formed by combining the bit-flip gadget with the phase flip-gadget. By considering the error propagation rules outlined in the previous subsections, it can be verified that this circuit will detect errors which occur on the register qubits $|\psi_{AB}\rangle$, but not errors which occur the parity qubits p_1 and p_2. We now show how the code can be modified to enable error detection across all four of the qubits.

The blue arrows in figure 7 show that a phase-flip error on the first parity qubit p_1 will propagate errors to the register in an undetectable way. Fortunately, a detection pathway can be created by applying a conjugate-propagator gate between the parity qubits at the end of the decoder (from now on, we will refer to these additional gates as 'cross-checks'). As shown by the orange arrows in figure 8, this cross-check propagates the phase-error to the parity-check qubit p_2 and converts it to an X-error that can be picked up by a computational basis measurement. With the addition of the cross-check, the circuit becomes a fully functional [[4,2,2]] quantum error detection code. The single-qubit error syndromes are given in table 1, and demonstrate the code can detect the occurrence of X, Y and Z errors on any of the 4 qubits.

As the [[4,2,2]] code is a detection code, the syndromes do not give us enough information to pinpoint which qubit the error occurred on. The construction of full error correcting CPC codes, that can both identify and localise errors, will be outlined in the next section.

2.6. The canonical form of CPC codes

The [[4,2,2]] quantum error detection code illustrates the basic principles behind the operation of a CPC code. The encoder is constructed by combining a bit-flip CPC gadget with a phase-flip CPC gadget. Under this canonical ordering, errors on the parity qubits are identifiable via the addition of the cross-check operators. A compact way of representing CPC codes is in terms of adjacency matrices which describe the connectivity between the register and parity qubits. For example, the adjacency matrices

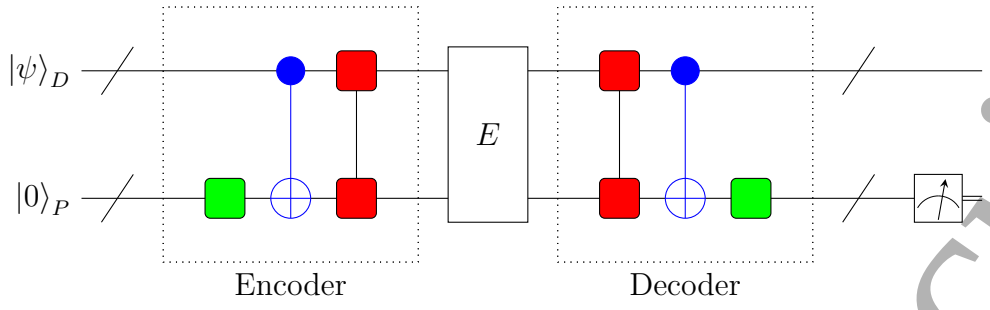


Figure 9: The canonical form of CPC codes, showing the symmetric *encode-error-decode* structure. In an $[[n,k,d]]$ CPC code the qubits are split into two distinct types: k data qubits, $|\psi\rangle_D = |\psi_{D_1}\psi_{D_2}...\psi_{D_k}\rangle$, and $n-k$ parity qubits, $|0\rangle_P = |0_{p_1}0_{p_2}...0_{p_{n-k}}\rangle$. The encoder involves successive rounds of cross-checks (green), bit-checks (blue) and phase-checks (red). The decoder is simply the unitary inverse of the encoder.

for the [[4,2,2]] code are

$$m_b = \begin{matrix} \\ [A] \\ [B] \end{matrix}\begin{matrix} \begin{matrix}[p1] & [p2]\end{matrix} \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \end{matrix}, \quad m_p = \begin{matrix} \\ [A] \\ [B] \end{matrix}\begin{matrix} \begin{matrix}[p1] & [p2]\end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix} \end{matrix}, \quad m_c = \begin{matrix} \\ [p1] \\ [p2] \end{matrix}\begin{matrix} \begin{matrix}[p1] & [p2]\end{matrix} \\ \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \end{matrix}, \tag{11}$$

where m_b represents the bit-checks, m_p the phase-checks and m_c the cross-checks. For the bit-flip and phase-flip adjacency matrices, m_b and m_p, the rows refer to the data qubits and the columns the parity qubits. Looking at the bit-flip matrix, we can see that both register qubits connect to parity qubit p_1 via CNOT gates in accordance with the circuit in figure 8. Likewise, matrix m_P tells us that both register qubits are connected to parity qubit p_2 via conjugate-propagator gates. Finally, from matrix m_c, we see that there is a single cross-check between parity qubits p_1 and p_2. The cross-checks result in a matrix that is always symmetric. We follow [3] in representing this as an upper triangular matrix, so that the number of non-zero entries corresponds to the number of two-qubit gates.

We are now in a position to extend the CPC framework to enable the description of more general codes. The canonical form of an [[n,k,d]] CPC code is shown in figure 9. Such codes have k data qubits, $|\psi\rangle_D = |\psi_{D_1}\psi_{D_2}...\psi_{D_k}\rangle$, and $m = n-k$ parity qubits, $|0\rangle_P = |0_{p_1}0_{p_2}...0_{p_m}\rangle$. As with the detection schemes described previously, the encode stage of a general CPC code involves successive rounds of cross-checks, bit-checks and then phase-checks. The sequence of gates within each stage of the encoder can be compactly described in terms of adjacency matrices of the form

$$m_b = \begin{array}{c} \\ [D_1] \\ [D_2] \\ [...] \\ [D_k] \end{array} \begin{array}{c} \begin{array}{cccc} [p_1] & [p_2] & [...] & [p_m] \end{array} \\ \begin{pmatrix} b_{11} & b_{12} & ... & b_1 \\ b_{21} & b_{22} & ... & b_{2m} \\ ... & ... & ... & ... \\ b_{k1} & b_{k2} & ... & b_{km} \end{pmatrix} \end{array}, \quad \mathrm{m_p} = \begin{array}{c} \\ [D_1] \\ [D_2] \\ [...] \\ [D_k] \end{array} \begin{array}{c} \begin{array}{cccc} [p_1] & [p_2] & [...] & [p_m] \end{array} \\ \begin{pmatrix} h_{11} & h_{12} & ... & h_{1m} \\ h_{21} & h_{22} & ... & h_{2m} \\ ... & ... & ... & ... \\ h_{k1} & h_{k2} & ... & h_{km} \end{pmatrix} \end{array},$$

$$m_c = \begin{array}{c} \\ [p_1] \\ [p_2] \\ [...] \\ [p_{(m-1)}] \\ [p_m] \end{array} \begin{array}{c} \begin{array}{cccc} [p_1] & [p_2] & [...] & [p_m] \end{array} \\ \begin{pmatrix} 0 & c_{12} & ... & c_{1m} \\ 0 & 0 & ... & c_{2m} \\ ... & ... & ... & ... \\ 0 & 0 & ... & c_{(m-1)m} \\ 0 & 0 & ... & 0 \end{pmatrix} \end{array}, \tag{12}$$

where b_{xy}, h_{xy}, c_{xy} are binary values. As mentioned previously, for simplicity, the cross-check matrix m_c is always represented as an upper triangular matrix.

2.7. Numerical CPC code discovery

We have now outlined the canonical structure of CPC codes, and shown how they can be represented in terms of three adjacency matrices. The CPC framework removes the need to start out by redundantly encoding quantum data, and allows QEC protocols to be implemented with any parity check. As such, the CPC framework essentially reduces the task of deriving QEC codes to a classical decoding problem.

A new $[[n, k, d =?]]$ CPC circuit can be generated simply by selecting a random instance of the adjacency matrices for an n-qubit code with k data qubits. The symmetric *encode-error-decode* structure of the CPC code will ensure that the random sequence of parity checks this set of adjacency matrices represents does not decohere the register. The only task necessary to verify whether the circuit represents a working CPC code is to measure the code distance d. This can be done by testing the circuits with all of the errors in the chosen error model. If each error produces a unique syndrome, then the code distance is $d \geq 3$, and the circuit represents a working CPC code.

In this paper we only consider quantum memories. As a result, the codes under consideration are Clifford circuits. The code distance can therefore be efficiently verified for small-distance codes using a stabilizer simulator such as [25, 26]. Alternatively, we have developed an algorithm specifically for calculating the syndromes of CPC codes, which is based on error propagation rules outlined in sections 2.3 and 2.4. This algorithm is described in appendix C, and can be implemented in less than 200 lines of Python code.

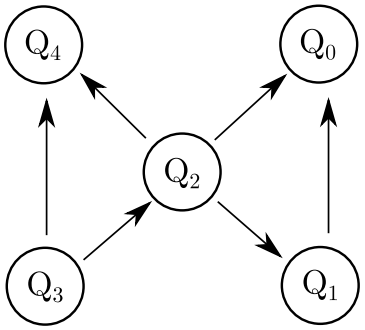


Figure 10: The connectivity map of the IBMQX4 version of the IBM 5Q quantum computer illustrating the 'bowtie' layout. The arrows indicate the allowed CNOT operations and their preferred directions.

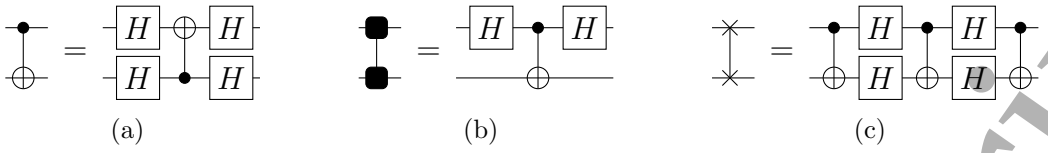
3. Implementation of the $[[4, 2, 2]]$ code on the IBM 5Q device

As a simple first experimental example of a CPC code, we now consider the compilation and execution of a $[[4, 2, 2]]$ quantum detection code on a superconducting qubit device. The IBM 5Q is a small-scale quantum computer, built and maintained by IBM Quantum [2]. The device has five programmable superconducting transmon qubits, and is accessible to the public via the Internet. In [27], the IBM 5Q was shown to allow fault tolerant preparation of codewords for a $[[4, 2, 2]]$ code. It has also been demonstrated, in [28], that certain $[[4, 2, 2]]$ encoded operations on the IBM 5Q have a lower error rate than the equivalent operation on the device's raw qubits. Here, we implement a complete encode-decode cycle of a $[[4, 2, 2]]$ CPC quantum memory using the IBM 5Q. Our aim is to demonstrate that the fidelity of the code's output state can be improved by post-selection.

3.1. Experimental overview and conditions for success

Our experiment on the IBM 5Q encodes a single input state $|\psi_{AB}\rangle = |+_A 0_B\rangle$ using a $[[4, 2, 2]]$ CPC quantum memory of the type described in section 2.5. The $|+_A 0_B\rangle$ state is an easy-to-prepare quantum state that is susceptible to both bit- and phase-flip errors, and therefore provides a suitable test of the $[[4, 2, 2]]$ CPC code as a quantum memory.

Ultimately, the condition for success for a quantum code is to test whether the encoded protocol has a lower logical error rate than the equivalent circuit before encoding. In the case of quantum memory, the circuit that is encoded is simply an extended identity operation. The usefulness of the $[[4, 2, 2]]$ code could therefore be assessed by comparing the fidelity of the encoded output to the equivalent output of an unprotected two-qubit data register. However, the gate error rates on the IBM 5Q hardware are too high for such a comparison to yield a positive result. This problem is compounded by the fact that the IBM hardware limits the experiment to a single encode-decode cycle, meaning certain regions of the $[[4, 2, 2]]$ circuit – before the encoder and after the decoder – are left unprotected. Consequently, the aim of the experiment



(a) (b) (c)

Figure 11: (a) The direction of a CNOT operation on the IBMQX4 can be reversed via the addition of Hadamard gates to the inputs and outputs. (b) The conjugate propagator gate expressed in terms of a CNOT gate. (c) Realisation of a SWAP gate using three CNOT operations.

presented here is restricted to demonstrating that, whilst not suppressing the logical error rate, the $[[4, 2, 2]]$ CPC code does detect errors. We now describe the method by which this is achieved.

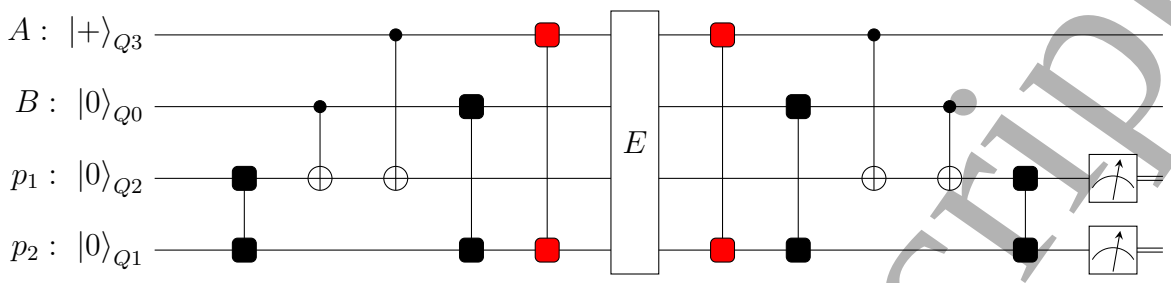
The compiled $[[4, 2, 2]]$ CPC code is run multiple times with the input state $|\psi_{AB}\rangle = |+_A 0_B\rangle$ on the IBM 5Q hardware. At the end of each CPC code cycle, the parity qubits are measured to provide a syndrome designed to indicate if an error has occurred. An approximation to the output state of the register is reconstructed from the experimental data using quantum state tomography. The quality of this output is quantified by calculating its fidelity relative to the input state $|+_A 0_B\rangle$. In this experiment we compare the output fidelity of the $[[4, 2, 2]]$ protocol before and after post-selection. In the former, the syndrome information is ignored, whereas in the latter it is used to determine which experimental runs are discarded during post-selection. The condition for success is that the post-selection should improve the output fidelity. If this is the case, it will demonstrate that the $[[4, 2, 2]]$ CPC code is detecting errors and produces useful syndrome information.

3.2. Compiling a $[[4, 2, 2]]$ *CPC circuit onto the IBM 5Q*

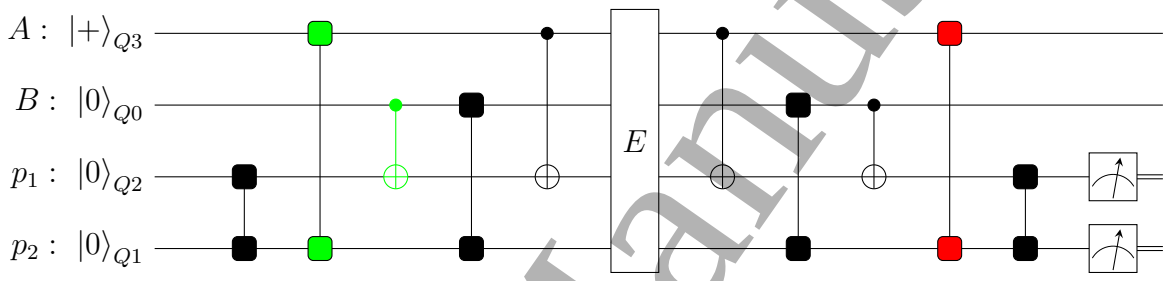
Our experiment is run on the IBMQX4 version of the IBM 5Q, the technical details for which can be found in [29]. Figure 10 depicts the 'bow tie' layout of the chip. The arrows represent the allowed CNOT operations between qubits. The direction of the arrow indicates the preferred CNOT direction, but the operation can be reversed via the circuit transformation shown in figure 11a.

The $[[4, 2, 2]]$ code, as depicted in figure 8, has two data qubits $\{A, B\}$ and two parity qubits $\{p_1, p_2\}$. In this experiment, the code qubits are mapped onto the physical qubits of the IBMQX4 device as follows: $\{A \to Q_3,\ B \to Q_0,\ p_1 \to Q_2,\ p_2 \to Q_1\}$. The input state becomes $|\psi_{AB}\rangle = |+\rangle_{Q_3} \otimes |0\rangle_{Q_0}$, and the resultant circuit is shown in figure 12a. The two conjugate propagator gates marked in red are not possible on the IBMQX4, as there is no connectivity between qubits Q_3 and Q_1 (see figure 10). The $[[4, 2, 2]]$ CPC circuit must therefore be modified to accommodate this hardware constraint.

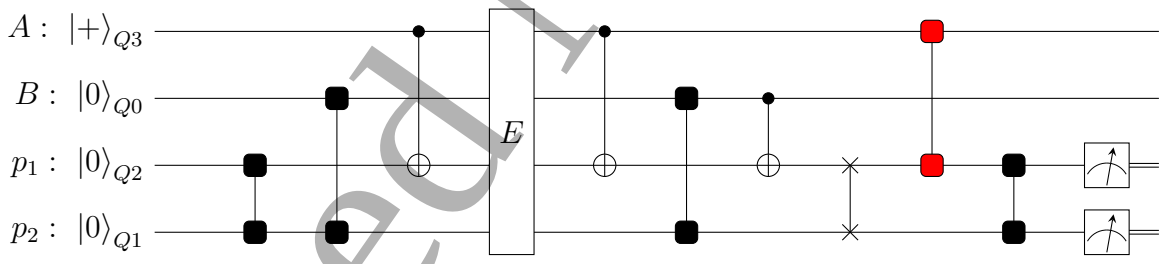
The first step in compiling the $[[4, 2, 2]]$ circuit for the IBMQX4 is to rearrange the gates into the order shown in figure 12b. This is a departure from the canonical



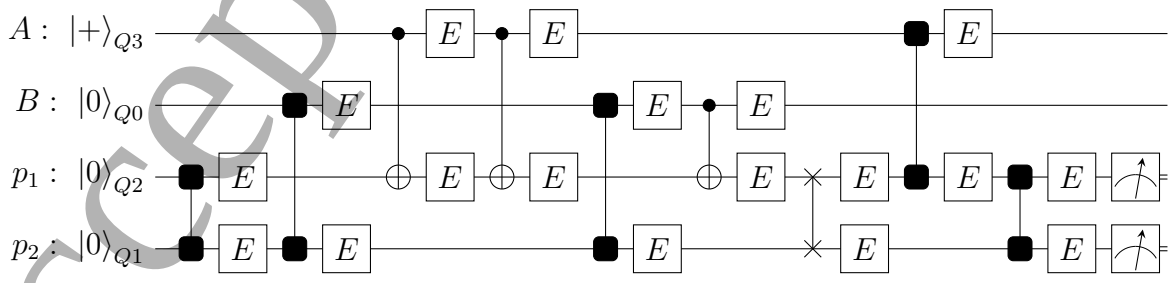
(a) The [[4,2,2]] CPC code with a $|\psi_{AB}\rangle = |+_A 0_B\rangle$ input mapped onto the IMBQX4 chip. The red conjugate propagator gates are not possible according to the connectivity map for the IBMQX4 shown in figure 10.



(b) A modified version of the [[4,2,2]] circuit in which the order of gates in the encoder and decoder has be rearranged. In this new form, the circuit can be simplified by noting that the action of the gates marked in green is the identity.



(c) A SWAP gate can be added to the [[4,2,2]] circuit to exchange the p_1 and p_2 parity qubits. This allows the 'illegal' operation marked in red in the decoder to be performed via a nearest-neighbour interaction.



(d) When running the [[4,2,2]] code on a real device, it can no longer be assumed that errors occur only in the wait-stage between the encoder and decoder. For the [[4,2,2]] circuit in question, a single-qubit fault E after any gate will not propagate a multi-qubit error to the register without triggering a syndrome.

Figure 12: Steps for compiling the [[4,2,2]] CPC quantum memory onto the IBMQX4 chip.

form of CPC codes outlined in section 2.6. However, it can easily be checked that the modified circuit remains a functional $[[4, 2, 2]]$ CPC code capable of detecting single X- and Z-errors on any of the qubits during the wait-stage.

In the rearranged form of the circuit in figure 12b, and when the input state is $|\psi_{AB}\rangle = |+\rangle_{Q_3} \otimes |0\rangle_{Q_0}$, it can be seen that the action of the gates highlighted in green is the identity. The green gates can therefore be omitted from the circuit without affecting the function of the quantum memory. Following this simplification, the only operation that remains prohibited by the IMBQX4's connectivity constraints is the red conjugate propagator gate between Q_1 and Q_3 in the decoder. One way of resolving this problem is to perform a SWAP operation between Q_2 and Q_1, as shown in figure 12c. The SWAP gate exchanges the positions of the p_1 and p_2 parity check qubits, enabling the red conjugate propagator gate to be performed via a nearest-neighbour interaction. A SWAP gate is achieved via the application of three CNOT gates (see figure 11c), and is therefore an expensive operation that should be used sparingly. In section 6, we explore how the CPC code design process can be used to minimise the SWAP gate count when compiling larger codes onto quantum hardware.

3.3. A note on fault tolerance for the $[[4, 2, 2]]$ *circuit*

So far, we have considered a simplified model of CPC code operation in which it is assumed errors only occur during the wait-stage between the encoder and the decoder. However, we have observed that the error rates for CNOT operations and readout on the IBMQX4 are of the order 10^{-2} (daily calibration data can be obtained from the IBM Q website [2]). This realistically means that any quantum code must be designed to detect errors that occur at any point in the circuit. To this end, figure 12d shows the IMBQX4-compiled $[[4, 2, 2]]$ circuit under a more general error model.

Fault tolerant circuit construction ordinarily necessitates the introduction of additional qubits [30, 31, 32]. However, in this particular instance of the $[[4, 2, 2]]$ CPC code with a known $|+\rangle_{Q_3} \otimes |0\rangle_{Q_0}$ input, it can be verified that a single fault at any of the locations marked on figure 12d will not propagate a multi-qubit error to the register without triggering a syndrome. The circuit can therefore be considered to have been hardened against single-qubit errors in the encode and decode stages of the circuit. It should be noted, however, that this does not extend the circuit to full fault tolerance when implemented on the IBMQX4 chip. State preparation and measurement (SPAM) errors are not accounted for, nor is the $[[4, 2, 2]]$ code capable of detecting correlated two-qubit errors that might occur after a CNOT gate. Another issue is that the circuit allows certain single-qubit errors to propagate to the register in an undetectable way. It is not currently possible to measure, then reset a qubit on the IBMQX4 via the public API. As a result, our implementation is restricted to a single encode-decode cycle, meaning the undetected single-qubit errors will reduce the output fidelity. However, as outlined in [3], CPC codes can be expressed in terms of stabilizer codes. Adopting this approach allows CPC codes to be implemented using existing syndrome extraction techniques,

and enables errors to be decoded over multiple cycles. Assuming access to hardware that allows qubit reset, CPC codes implemented in this way would be tolerant of the single-qubit errors that propagate to the register.

3.4. Experimental data reconstruction methods

The IBM Quantum Information Software Kit (QISKIT) [33] was used to prepare the $[[4,2,2]]$ experiment for quantum state tomography on the output qubits Q_0 and Q_3. QISKIT quantum tomography tools were used to create a set of nine circuits from the original $[[4,2,2]]$ circuit (depicted in figure 12c), each of which was designed to measure the output qubits Q_0 and Q_3 in a different measurement basis from the list $\{XX,\ XY,\ XZ,\ YX,\ YY,\ YZ,\ ZX,\ ZY,\ ZZ\}$. These quantum tomography circuits were then run multiple times to create a distribution of results that could be used reconstruct an approximation to the density matrix of the output state. The QISKIT method used for state reconstruction from the experimental data was the fast maximum likelihood method for quantum tomography, a description of which can be found in [34].

The quantum tomography circuits for the $[[4,2,2]]$ memory were run in batches of 8192 shots. After each batch, the QISKIT maximum likelihood method was used to reconstruct the density matrix ρ_{dd} of the directly decoded output before post-selection. The syndrome qubits were then inspected to determine which of the shots in the batch should be discarded during post-selection. State reconstruction was then performed again on the reduced set to obtain a post-selected density matrix ρ_{ps}. The quality of the directly decoded and post-selected output state for each batch was quantified by calculating the fidelity, $F(\rho) = \left(\mathrm{Tr}\left[\sqrt{\rho^{1/2}\sigma\rho^{1/2}}\right]\right)^2$, where σ is the target density matrix. For the chosen input state $|\psi_{AB}\rangle = |+\rangle_{Q_3} \otimes |0\rangle_{Q_0}$, the target density matrix is given by

$$\sigma = \begin{pmatrix} \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \tag{13}$$

The purity of the density matrices, defined by $P(\rho) = \mathrm{Tr}\left[\rho^2\right]$, was also calculated to provide a coherence measure for the output states.

3.5. Experimental results

The $[[4,2,2]]$ CPC quantum memory circuit, depicted in figure 12c, was run on the IBMQX4 device between the 25$^{\text{th}}$ and 27$^{\text{th}}$ November 2017. A summary of the experimental results for state purity, fidelity and yield can be found in table 2. Error bars were calculated as one standard deviation of a single run value consisting of 8192 experimental executions of the quantum tomography circuit set. The standard error of the mean over all 154 runs was too small to be visible on our plots. Calibration data for the device on each of the three days of the experiment can be found in appendix D.

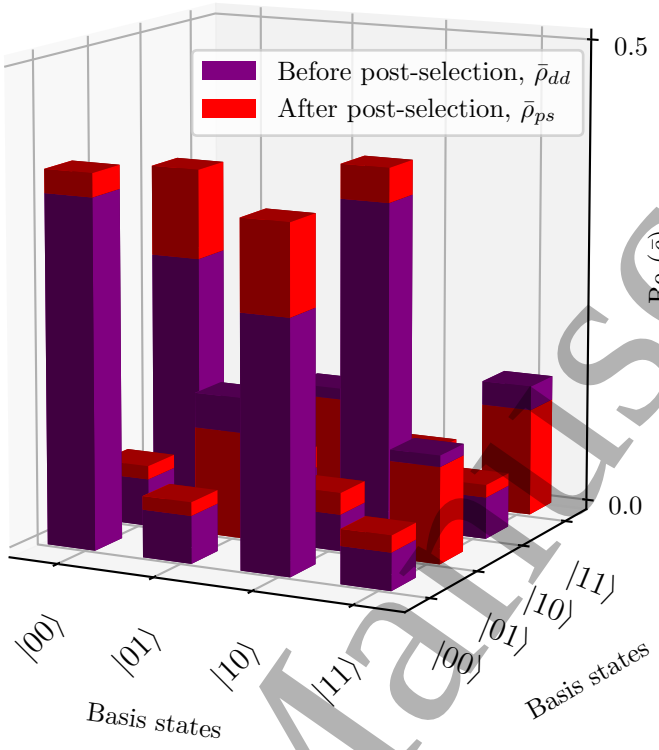


Figure 13: Plot of the real components of the density matrices $\bar{\rho}_{dd}$ and $\bar{\rho}_{ps}$ corresponding to the experimental output state before and after post-selection.

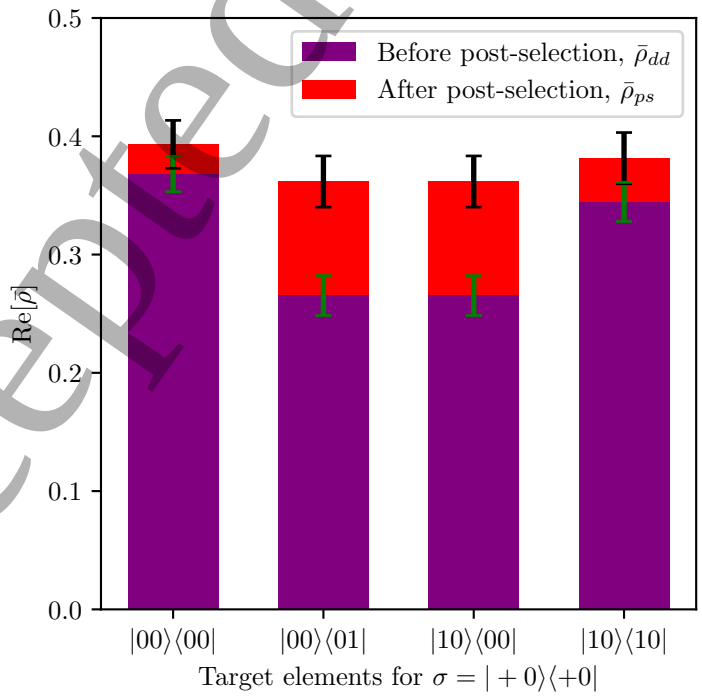


Figure 14: Plot of the target elements for ρ_{dd} and ρ_{ps}. The target elements are the non-zero elements in the density matrix σ given in equation (13).

	Purity, $P(\rho)$	**Fidelity**, $F(\rho)$	**Yield**
Before post-selection, $\bar{\rho}_{dd}$	0.52 ± 0.02	0.62 ± 0.03	100%
After post-selection, $\bar{\rho}_{ps}$	0.74 ± 0.03	0.75 ± 0.04	$(54 \pm 2)\%$
no. runs: 154 batches of 8192 shots			

Table 2: Quality metrics for the reconstructed density matrices before and after post-selection. The fidelity is calculated relative the target density matrix σ which is defined in equation (13). The yield is the proportion of shots per batch that are retained during the post-selection process. The errors are calculated as one standard deviation of a single run value consisting of 8192 experimental shots.

A total of 154 batches of 8192 shots were run over the course of the experiment. Figure 7 shows a plot of the real components of the elements of $\bar{\rho}_{dd}$ and $\bar{\rho}_{ps}$ averaged across the 154 batches. It is immediately clear that the post-selected density matrix $\bar{\rho}_{ps}$ better preserves the four target elements, which we identify as the non-zero elements in the target state σ given by equation (13). The bar-chart in figure 14 shows these target elements in isolation, from which it is apparent that post-selection has the biggest impact in preserving the strength of the off-diagonal coherences. This can also be seen when comparing the purity values, shown in table 2, for $\bar{\rho}_{dd}$ and $\bar{\rho}_{ps}$. The directly-decoded density matrix $\bar{\rho}_{dd}$ has a purity of $P(\bar{\rho}_{dd}) = 0.52 \pm 0.02$, implying it represents a near-fully mixed classical ensemble with a purity of 0.5. In contrast, the post-selected density matrix $\bar{\rho}_{ps}$ has a purity of $P(\bar{\rho}_{ps}) = 0.74 \pm 0.03$, suggesting it has undergone only partial decoherence.

The fidelities of $\bar{\rho}_{dd}$ and $\bar{\rho}_{ps}$ relative to the target state are $F(\bar{\rho}_{dd}) = 0.62 \pm 0.03$ and $F(\bar{\rho}_{ps}) = 0.75 \pm 0.04$ respectively. The fidelity of the post-selected state is therefore greater than the directly-decoded state with a confidence level of three standard deviations. From this we can conclude that the $[[4, 2, 2]]$ quantum memory produces useful syndrome information for protecting a $|\psi_{AB}\rangle = |+_A 0_B\rangle$ state. A consideration, however, is that the average yield (the proportion of results retained after post-selection) was $(54 \pm 2)\%$ averaged over the 154 batches.

3.6. Summary of IBM 5Q experiment

The results of our experiment with the IBMQX4 device show that the syndrome information produced by a $[[4, 2, 2]]$ CPC quantum memory can be used to improve the fidelity and purity of the code output. The $[[4, 2, 2]]$ CPC code is one the simplest quantum memories, and as such, it was possible to compile the circuit for the IBMQX4 device by inspection. In the following sections, we outline a CPC design process that provides automated methods for compiling and optimising more complex CPC codes onto quantum hardware. As an example, we demonstrate the utility of the CPC design process in the compilation of a custom quantum memory for an idealised seven-qubit ion trap device.

4. Overview of ion trap hardware for quantum computing

Ion traps are considered one the leading platforms for quantum computation. Ion-based qubits have long coherence times, and can be read out with near 100% efficiency [35]. It has also been proposed that multiple ion-trap cells could be networked via auxillary qubit systems to create larger hybrid quantum computers [7]. In such a hybrid networked architecture, good QEC codes will be vital to ensure the quantum data in each ion trap is protected.

In this paper, we provide an illustrative example of how the CPC design process can be used to create a bespoke QEC code for a specific ion trap device. We consider a linear ion-trap with seven application qubits. This scheme has been chosen because several existing ion trap experiments have a similar size and layout [11, 12, 13, 14].

We assume that arbitrary single-qubit operations can be performed on any of the ions in the register. It is in principle possible to implement interactions between spatially separated qubits, for example, by exploiting the collective vibrational modes of the ions as a quantum bus [36]. In practice, however, the fidelity of two-qubit interactions decreases with separation [37]. For this reason, in our idealised model ion trap, two-qubit gates are limited to nearest-neighbour interactions.

Under nearest-neighbour constraints, interactions between spatially separated qubits are achieved by performing SWAP operations to move quantum information around the trap. These SWAP operations can be realised either by physically shuttling qubits between zones of the trap [38], or by synthesising SWAP gates from CNOT interactions [37]. In the CPC design process, we show how CPC codes can be compiled with SWAP gates to allow for implementation with only nearest neighbour interactions.

We assume that our idealised ion trap has a two-qubit entangling gate that gives rise to a unitary of the form

$$U = \exp\left(-\mathrm{i}\frac{\pi}{2}\left[Z\otimes Z\right]t\right) = e^{\mathrm{i}\pi t/2}\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & e^{-\mathrm{i}\pi t} & 0 & 0\\ 0 & 0 & e^{-\mathrm{i}\pi t} & 0\\ 0 & 0 & 0 & 1\end{pmatrix}. \tag{14}$$

where t is a tuning parameter. Such interactions can be realised via geometric phase gate procedures [36, 39, 40]. In this paper, we consider the symmetrised phase (SP) gate, which is one of the simplest possible maximally entangling gates that arises from the above ion trap unitary [10]. The SP native gate is realised by setting the tuning parameter in equation (14) to $t_{\mathrm{SP}} = 1/2$. Up to a global phase, the gate can then be described as a matrix, F, of the form

$$F^{q_1}_{q_2} = \begin{pmatrix}1 & 0 & 0 & 0\\ 0 & \mathrm{i} & 0 & 0\\ 0 & 0 & \mathrm{i} & 0\\ 0 & 0 & 0 & 1\end{pmatrix}, \tag{15}$$

where q_1 and q_2 are the input qubits to the gate. In section 5, we explicitly show how a $[[4, 2, 2]]$ detection code can be efficiently compiled with the SP native gate of equation (15). Building on this example, we then demonstrate how efficient compilation is in principle possible for any experimentally realistic maximally entangling native gate.

5. Compiling CPC codes with any realistic maximally entangling Clifford gate

In our discussion of the CPC framework so far, quantum codes have been expressed in terms of CNOT and conjugate-propagator gates. This allows for intuitive visualisation of the propagation of errors through the decoder, and simplifies the calculation of syndrome tables via the techniques described in sections 2.3 and 2.4. However, in practice, the native two-qubit entangling interaction of a given experiment will be of a different form. As a result, when compiling a CPC code, additional operations are required to allow CNOT and conjugate-propagator gates to be synthesised from the native interaction. If the native interaction is maximally entangling, this will involve the addition of single-qubit corrections. In this section, we show that the symmetric *encode-error-decode* structure of the CPC framework enables efficient QEC code compilation with a broad range of native gates.

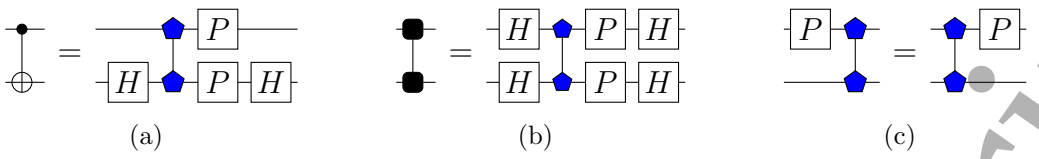
5.1. Compiling the $[[4, 2, 2]]$ *CPC detection with an ion trap native gate*

Here we show that the $[[4, 2, 2]]$ CPC detection code, introduced in section 2.5, can be efficiently compiled with an ion trap native gate. For the purposes of this example, we adopt an ion trap with a SP native gate as introduced in equation (15) in section 4. The SP native gate can be transformed into a CNOT via the application of local unitary operations to its inputs and outputs. A possible mapping, in matrix equation form, is given by

$$\mathrm{CNOT}_{q_2}^{q_1} = (I_{q_1} \otimes H_{q_2}) \cdot (P_{q_1} \otimes P_{q_2}) \cdot F_{q_2}^{q_1} \cdot (I_{q_1} \otimes H_{q_2}), \tag{16}$$

where $F_{q_2}^{q_1}$ is the matrix representation of the SP gate defined in equation (14), and P is a phase gate defined as $P = \mathrm{diag}(1, -\mathrm{i})$. Realising a CNOT gate on ion trap hardware, via the above mapping, requires the application of the native gate combined with four single-qubit gates, as shown in figure 15a. Likewise, figure 15b shows how the conjugate-propagator gate can be constructed from the native gate via the addition of six single-qubit operations. We will see that, when the native gates are compiled into a CPC circuit, constructive simplifications become possible to reduce the total number of single-qubit gates required.

Figure 16 illustrates the steps involved in the compilation and simplification of the $[[4, 2, 2]]$ CPC code with the SP native gate. The un-compiled circuit, expressed in terms of CNOT and conjugate-propagator gates, is shown in figure 16a. The first step of compilation involves substituting the CNOT and conjugate-propagator gates with the



(a) (b) (c)

Figure 15: (a) A CNOT gate expressed in terms of the SP native gate, which is represented by the connected blue pentagons. The matrix form of the SP native gate is given in equation (15). (b) A conjugate-propagator gate expressed in terms of the SP native gate. (c) Both the phase gate P and the SP native gate are represented as diagonal matrices in the computational basis. As a result, the P gate can be moved freely through the SP native gate.

SP native interaction, via the circuit rewrites rules defined in figure 15. The resultant circuit is shown in figure 16b.

Now that the circuit is written in terms of the native gate, circuit simplifications can be applied to reduce the single-qubit gate count. In figure 16b, pairs of H gates that cancel to the identity are labelled in red. In the encoder, the H gates labelled in blue are paired with their counterparts from the decoder. We can now exploit the symmetry of the CPC code to further reduce the gate-count. The effect of the blue H gates around the wait-stage is to transform X errors into Z errors and vice-versa, as described by the following matrix transformations

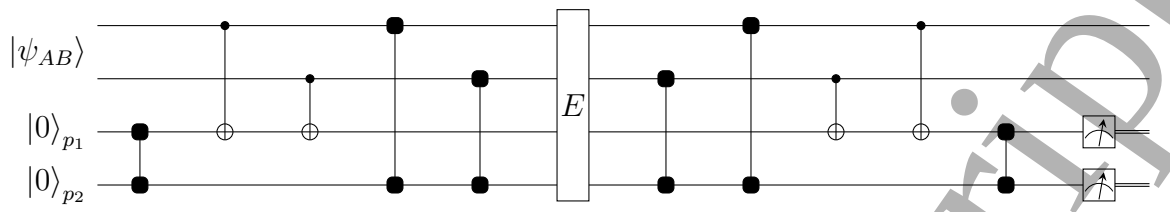
$$\begin{aligned} H \cdot (E = X) \cdot H &= H \cdot X \cdot H = Z, \\ H \cdot (E = Z) \cdot H &= H \cdot Z \cdot H = X, \end{aligned} \tag{17}$$

where E represents the error that occurs in the wait stage. The $[[4, 2, 2]]$ code can detect both X and Z errors, as shown in syndrome table 1 in section 2.5. As a result, the blue H gates do not change the errors into a form that cannot be detected. The blue H gates can therefore be discarded without affecting the operation of the $[[4, 2, 2]]$ code.

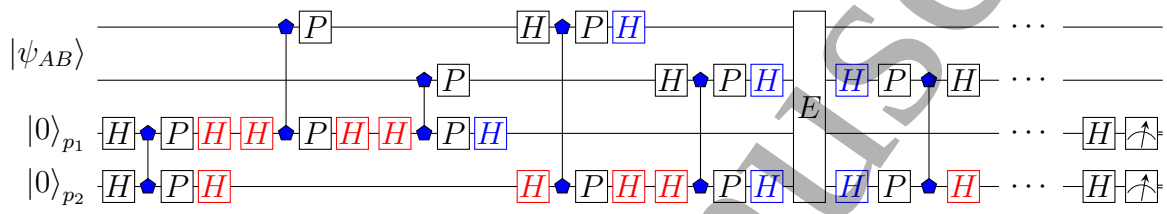
Figure 16c shows the compiled $[[4, 2, 2]]$ code following the removal of the unnecessary H gates. Notice that both the P gate and the SP gate are described by diagonal matrices in the computational basis. As a result, we have the freedom to move P gates through the SP native gate as shown in figure 15c. Two P gates combine to form a Z gate as follows $P \cdot P = Z$. In the circuit in figure 16c, pairs of P gates are highlighted in red. As Z gates are diagonal in the computational basis, they can also be moved through the SP gates.

In the circuit in figure 16d, the Z gates and blue P gates have been pushed to the centre of the circuit. In the event that no error occurs, these P gates combine to form a Z-error via the relation $Z = P \cdot P$. However, the locations of these errors are known, and they can therefore be accounted for in post-processing. If an error does occur, the effect of symmetric P gates about the wait stage, E, is to transform X-errors into Y-errors and vice versa, as described by the following matrix transformation rules

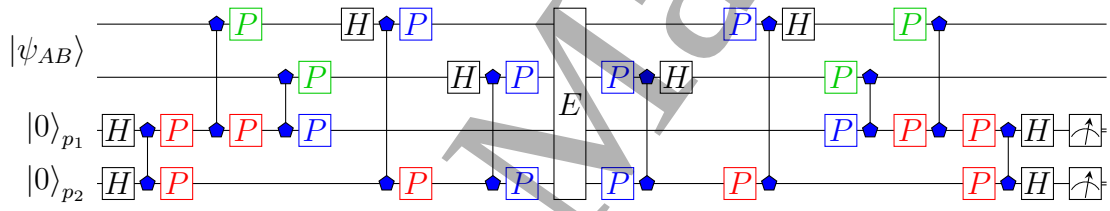
$$\begin{aligned} P \cdot (E = X) \cdot P &= P \cdot X \cdot P = (-\mathrm{i})Y, \\ P \cdot (E = Y) \cdot P &= P \cdot Y \cdot P = (-\mathrm{i})X, \end{aligned} \tag{18}$$



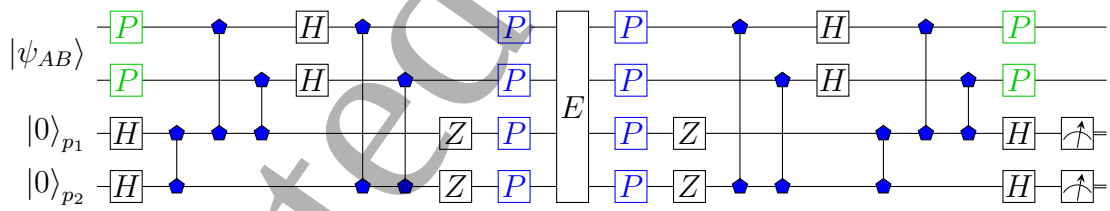
(a) The un-compiled [[4,2,2]] CPC error detection code expressed in terms of CNOT and conjugate propagator gates.



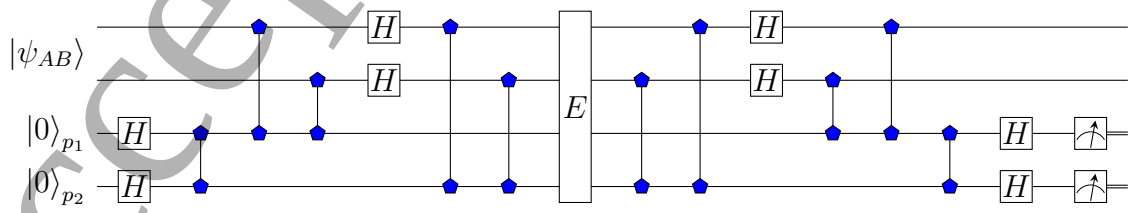
(b) The compiled circuit prior to simplification. Note that parts of the decoder have been hidden to save space. The pairs of Hadamards, labelled red, cancel to the identity. The blue H gates can also be discarded without affecting the operation of the code.



(c) The circuit following removal of the H gates. The pairs of red P gates combine to form Z gates.



(d) The Z gates and blue P gates can be moved freely through the SP gates to the centre of the circuit. Due to their symmetry about the error window, the P and Z gates can be omitted from the code. The green P gates do not affect circuit operation and are also discarded.



(e) The compiled [[4,2,2]] CPC detection code following circuit simplification. Only four single-qubit gates remain in the encoder.

Figure 16: Compiling the $[[4, 2, 2]]$ detection code with the SP native gate.

where the (−i) global phase does not affect the syndrome measurement. These transformations are unproblematic as $[[4,2,2]]$ code can detect both X and Y errors (see syndrome table 1). As the effect of the blue P gates can be described in terms of single-qubit Clifford operations on the output, they can be removed from the circuit and accounted for in post-processing. There are also P gates highlighted in green, located on the register qubits at the beginning and end of each error cycle. These gates occur before the first round of CPC checks, and can therefore be removed from the circuit without affecting the final syndrome readout. Finally, the Z gates located symmetrically about the wait-stage introduce a global phase to the errors. This global phase does not affect the propagation of errors through the circuit, meaning the Z gates can be removed. It should be noted that the above simplifications will result in a modified syndrome table. However, the *no-error* case will remain unique meaning the function of the code is maintained.

The final simplified form of $[[4,2,2]]$ CPC code compiled with the SP native gate is shown in figure 16e. The single-qubit gate count in the encoder has been lowered from 26 gates in the original compiled circuit (figure 16b), to 4 gates in final circuit (figure 16e).

5.2. Requirements for CPC gates

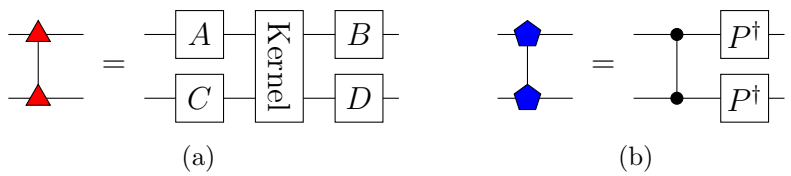
We have now shown that the $[[4,2,2]]$ code can be efficiently compiled with the SP native gate. Most of the single-qubit corrections can be eliminated, either by direct cancellation between adjacent Hadamards, or by moving P gates through the circuit. We now show that efficient CPC code translation, from the idealised CNOT version to the hardware-compiled version, is possible for a range of native gate types. We begin by outlining the general requirements for two-qubit gates in a CPC circuit.

In a CPC code, the role of two-qubit interaction gates is to distribute error information from the register to the parity qubits. For example, CNOT gates propagate bit-errors from their control to target via the rule in equation (8). More generally, we require that the two-qubit CPC gate, $\Omega^{q_1}_{q_2}$, has the ability to change the weight of an error operator, $E^i_{q_1} \otimes \mathbb{1}_{q_2}$, such that

$$\Omega^{q_1}_{q_2} \cdot \left(E^i_{q_1} \otimes \mathbb{1}_{q_2}\right) \cdot \left(\Omega^{q_1}_{q_2}\right)^\dagger = \left(E^j_{q_1} \otimes E^k_{q_2}\right), \tag{19}$$

where q_1 and q_2 are the control and target qubits respectively, and $E^{i,j,k}$ are non-identity elements of the single-qubit Pauli group. As both $E^i_{q_1} \otimes \mathbb{1}_{q_2}$ and $E^j_{q_1} \otimes E^k_{q_2}$ are Pauli group operators, we see that $\Omega^{q_1}_{q_2}$ must be a Clifford gate (for an overview of the Clifford group see appendix B). CPC quantum memories can be described entirely in terms of Clifford gates, as their operations are restricted to manipulating stabilizer states. This allows for efficient classical simulation. For an example of such a simulation, see the CPC syndrome calculation algorithm we outline in appendix C.

Another way of thinking about the CPC interaction gates is in terms of entanglement. In equation (19), it can be seen that the general CPC gate de-localises



(a) (b)

Figure 17: Left: A general maximally entangling Clifford native gate (red triangles) can be expressed in terms of a kernel supplemented by local corrections on its inputs and outputs. The kernel will always be of the form CZ or CZ-SWAP . The local corrections, $\{A, B, C, D\}$, are single-qubit Clifford gates and can expressed as products of H and P gates. Right: The SP native gate expressed in terms of its CZ kernel.

error information from the control to the target, suggesting the operation has the potential to entangle states. Furthermore, we know that elements of the two-qubit stabilizer states are either maximally entangled or separable. Any Clifford entangling gate that maps between these states, and therefore any CPC interaction, is a maximally entangling operation.

We have now established that CPC gates must be maximally entangling Clifford operations. However, many experiments will have native gates that do not satisfy these requirements. For example, several qubit technologies have a native interaction of the form $\sqrt{\text{SWAP}}$ [41], which is only partially entangling. In these circumstances, multiple applications of the native gate, in addition to local operations, are required to synthesise the desired maximally entangling behaviour. It is typically the case that quantum computing experiments will have different error rates for single-qubit and two-qubit gates [42]. Circuit compilation strategies should therefore aim to minimise the gate-type with the highest error rate. In the case of ion traps, for example, the two-qubit gates have lower fidelities than single-qubit gates [11, 43].

5.3. Circuit simplification with any maximally entangling Clifford gate

We will now outline general CPC circuit simplification procedures for maximally entangling Clifford gates. It can be shown that all Clifford entangling gates are local Clifford equivalent to either the CZ or the CZ-SWAP interaction. With this knowledge, we can write all maximally entangling Clifford gates in terms of a central kernel, containing either a CZ or CZ-SWAP interaction, supplemented by local Clifford gates (see figure 17a).

The single-qubit Clifford group is generated by P and H gates. Any native gate can therefore be constructed from its by kernel via the addition of local gates generated from combinations of P and H. The P gates can be trivially pushed through the CZ kernel. Likewise, it is possible to push P gates through the CZ -SWAP kernels, although the effect of the SWAP gate must be taken into account.

In section 6.3 we demonstrated the compilation of a CPC code using the SP native gate, which is local Clifford equivalent to a CZ gate. The exact transformation from CZ kernel to SP gate is shown in figure 17b. As the local operations in this case consist

of $P^\dagger$ gates, we have the freedom move P gates through the SP native gate. Hadamard gates H, however, restrict movement, but in many cases will cancel when the native gate is compiled into a CPC circuit.

The general procedure for compiling a CPC code with a given native gate can now be written as follows. First, eliminate any unnecessary H gates by identifying cancellations between adjacent CPC gates. Second, determine the behaviour when P gates are pushed through the native gate. As all maximally entangling Clifford gates have either a CZ or CZ-SWAP kernel, it is often possible to trivially move P gates through each block of the encoder. Once these simplifications rules have been established, they can be applied systematically to substantially reduce the CPC circuit gate count.

6. The CPC code design process

The first generation of quantum computers will be limited in size to no more than a couple of hundred qubits [1, 2]. In this section, we outline a design process for constructing hardware-optimised quantum codes with the CPC framework. By maximising the encoding density, such bespoke CPC codes will help early quantum computers realise their full potential.

To illustrate our design process, the quantum device we consider is the seven-qubit linear ion trap which was introduced in section 4. Our CPC design process is split into three stages. 1) **CPC code discovery**: numerical search techniques are used to find CPC codes that maximise the quantum encoding density for a seven qubit register. 2) **Hardware optimisation**: the best CPC codes from the discovered set are identified by analysing which ones have the lowest two-qubit count when implemented on a linear nearest-neighbour architecture. 3) **Native gate compilation**: further optimisations are made by identifying CPC circuits with efficient translations from the CNOT version of the code to the native gate version, using the circuit simplification strategies outlined in section 5.

6.1. Stage 1: CPC code discovery

The idealised ion trap we are considering has seven application qubits. We assume that during the wait stage the ion trap qubits are subject to a biased depolarizing noise channel of the form

$$\mathcal{E}[\rho] = (1 - p_x - p_z - p_x p_z)\rho + p_x X\rho X + 2p_x p_z Y\rho Y + p_z Z\rho Z, \tag{20}$$

where ρ is the single-qubit density matrix, and p_x and p_z are the probabilities of X- and Z-errors respectively. This error model assumes the ion trap has independent error mechanisms for X- and Z-errors, but Y-errors occur only as a result of successive single-qubit errors of the form XZ and ZX †. Similar error models have recently been

† Note that the ion trap we consider is an idealised proof-of-concept model, and does not correspond to a specific ion trap experiment.

considered in [44, 45]. For the purposes of our ion trap model, we assume that the error probabilities p_x and p_z are low enough that the probability of Y-errors becomes negligibly small. The effective error model can then be written as

$$\mathcal{E}[\rho] \approx (1 - p_x - p_z)\rho + p_x X\rho X + p_z Z\rho Z. \tag{21}$$

Under the above error model for the idle ion trap qubits, the CPC quantum memory only needs to correct X- and Z-errors. We choose this noise model for our proof-of-concept outline of the CPC design process, as it corresponds to the simplest possible non-classical error model. Our aim is to discover non-degenerate quantum codes which produce a unique syndrome for single X- and Z-errors on any of the seven qubits in the trap.

The maximum possible encoding density of a non-degenerate quantum error correction code is constrained by the quantum Hamming bound, which states that an $[[n, k, d]]$ code must satisfy the following inequality

$$\sum_{j=0}^{(d-1)/2} \binom{n}{j} |\mathcal{E}|^j \; 2^k \leq 2^n, \tag{22}$$

where $|\mathcal{E}|$ is the size of the single-qubit error set [9]. As we are considering only X- and Z-errors in the generic ion trap under the error model described by equation (21), the size of the error set is $|\mathcal{E}| = 2$. Under this error model, the quantum Hamming bound tells us that the maximum number of data qubits that can be encoded in 7 physical qubits is $k_{\max} = 3$. The optimal 7 qubit CPC code will therefore be of the form $[[n = 7, k = 3, d = 3]]$. Note that the code distance is $d = 3$, indicating that these $[[7, 3, 3]]$ codes will be able to correct one error per CPC cycle.

An advantage of the CPC framework lies in the fact that new instances of such optimal codes can be discovered numerically, either using brute-force or more sophisticated optimisation techniques [3]. We now demonstrate these strategies in practice, by showing how optimal $[[7, 3, 3]]$ CPC codes can be discovered via exhaustive search.

A $[[7, 3, 3]]$ CPC code has $k = 3$ data qubits and $n - k = 4$ parity qubits. The adjacency matrices therefore have the form

$$m_b = \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} \\ b_{21} & b_{22} & b_{23} & b_{24} \\ b_{31} & b_{32} & b_{33} & b_{34} \end{pmatrix}, \quad \mathrm{m_p} = \begin{pmatrix} h_{11} & h_{12} & h_{13} & h_{14} \\ h_{21} & h_{22} & h_{23} & h_{24} \\ h_{31} & h_{32} & h_{33} & h_{34} \end{pmatrix},$$
$$m_c = \begin{pmatrix} 0 & c_{12} & c_{13} & c_{14} \\ 0 & 0 & c_{23} & c_{24} \\ 0 & 0 & 0 & c_{34} \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{23}$$

where b_{xy}, h_{xy}, c_{xy} are binary values. New CPC circuits can be made by generating different instances of these matrices. The single-qubit error syndrome table for each code

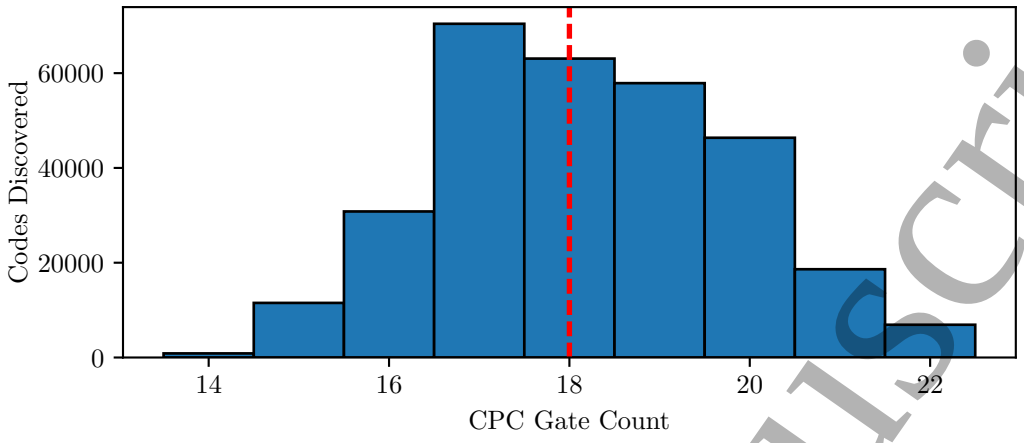


Figure 18: A histogram showing all of the $[[7,3,3]]$ CPC codes discovered in stage 1 of the CPC design process, binned by encoder length. The CPC gate count is the combined total of CNOT and conjugate propagator gates in the encoder. The median length, marked in red, is 18. In comparison, the shortest $[[7,3,3]]$ circuits have a CPC gate count of 14.

Encoder gate length (number of CPC gates)		
Mininum	**Median**	**Depth reduction**
14 (864 discovered)	18	22%
Size of $[[7,3,d=?]]$ search space: 1.07×10^9 circuits		
Number of $[[7,3,3]]$ codes discovered: $306,480$ (0.03% of search space)		
Number of symmetry-inequivalent $[[7,3,3]]$ codes: 2190		

Table 3: Summary of the exhaustive search for $[[7,3,3]]$ CPC codes. The number of CPC gates is defined as the combined total of CNOT + conjugate propagator gates in the encoder. The depth reduction is calculated as the percentage decrease in the encoder length of the smallest circuit relative to the median.

can be calculated efficiently using a stabilizer simulator or the algorithm we describe in appendix C. If the set of syndromes is unique, the respective matrices represent a valid $[[7,3,3]]$ code.

The number of possible combinations of the adjacency matrices for CPC circuits of type $[[7,3,d=?]]$ is 2^{30}. By an exhaustive search, we have discovered that $306,480$ of these permutations (0.03% of the search space) are working $[[7,3,3]]$ codes. These codes have distance $d=3$, and produce unique syndromes for all single-qubit X and Z errors across the seven qubits. Of the discovered set, there are 2190 symmetry-inequivalent codes that cannot be transformed from one to another by rearranging the qubit order. However, some symmetry-equivalent code permutations are more amenable to circuit optimisation than others. We will therefore continue to consider the entire set of $306,480$ codes in the CPC design process.

Now that a set of $[[7,3,3]]$ circuits has been found, the next stages in the CPC design process involves analysis to determine which one of the $306,480$ codes is best

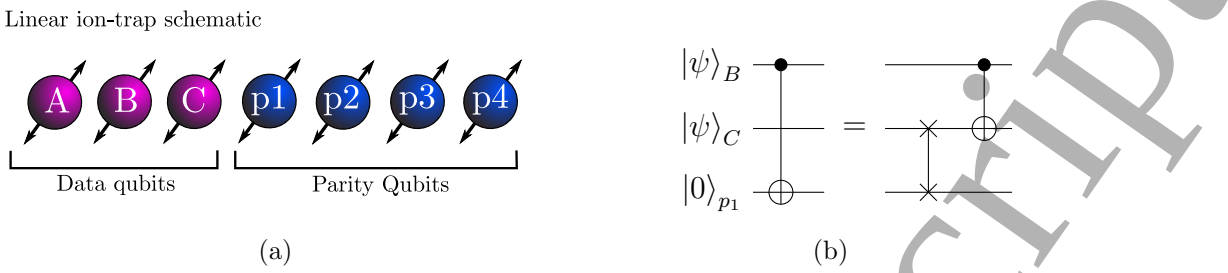


Figure 19: (a) A schematic of the seven-qubit linear ion trap. Three of the qubits have been labelled as data qubits and four as parity qubits as required by the $[[7, 3, 3]]$ code. It is assumed that entangling gates can only be performed between nearest-neighbour qubits. (b) A CNOT gate between spatially separated qubits can be implemented using only nearest-neighbour interactions through the addition of SWAP gates.

suited for implementation on the ion trap device. Figure 18 shows a histogram of the discovered $[[7, 3, 3]]$ codes, binned by the combined number of CNOT gates and conjugate propagator gates in their encoder. This quantity will be referred to as the CPC gate count, and can be determined by counting the number of non-zero entries across the three adjacency matrices.

In ion trap hardware, inter-qubit operations are typically the most expensive gate-type in terms of their potential to introduce errors [11, 43]. As a result, the CPC circuits with the lowest CPC gate count are most desirable. In the set of $[[7, 3, 3]]$ codes, the shortest circuit encoders have 14 CPC gates. This is a 22% reduction in circuit depth compared to the median gate count of 18. The number of $[[7, 3, 3]]$ circuits with the minimum encoder depth of 14 is 864 of which 245 are symmetry inequivalent. Further work is therefore necessary to narrow down the code set, and find the optimum quantum memory for the ion trap device.

The encoder length statistics for the $[[7, 3, 3]]$ CPC codes are summarised in table 3. Note that in this simple first analysis, we have not accounted for any of the constraints imposed by the ion-trap's nearest-neighbour requirement for two-qubit operations. In the next section, we outline how the $[[7, 3, 3]]$ codes can be compiled in such a setting through the introduction of additional SWAP gates.

The results in this section demonstrate that the CPC framework provides constructive tools for discovery of optimal $[[7, 3, 3]]$ codes that saturate the quantum Hamming bound. Furthermore, the search was performed using a simple brute-force technique that requires only a basic knowledge of the CPC code structure to implement. The Python script used to perform the code search is approximately 200 lines long, and required approximately 4 days to run on a CPU clocked at $3.2GHz$ with $8Gb$ of RAM.

6.2. Stage 2: Hardware optimisation

The second stage of the CPC design process involves selecting codes to meet the demands of the chosen quantum hardware and its qubit layout. Figure 19a shows the

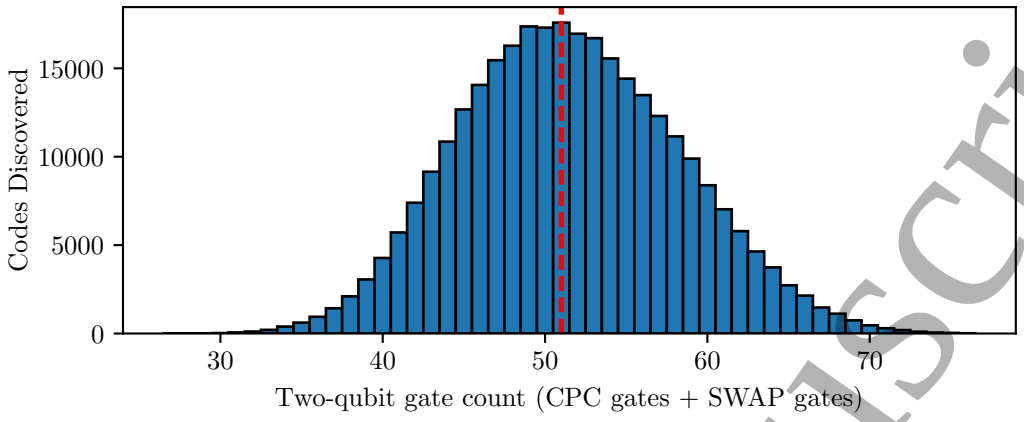


Figure 20: The distribution of $[[7,3,3]]$ CPC codes binned by two-qubit gate count after compilation onto a nearest-neighbour architecture. The total gate count is defined as the number of CPC gates + SWAP gates in the encode stage of the circuit.

Encoder gate length (number of two-qubit gates)		
Mininum	**Median**	**Depth reduction**
27 (1 discovered)	51	47%
Optimum code: Encoder length=27 gates; no. CPC gates=14; no. SWAP gates=13		

Table 4: Summary of the gate-count statistics for the set of $[[7,3,3]]$ codes following the SWAP gate compilation. The two-qubit gate count is defined as the number of CPC gates + SWAP gates. The depth reduction is the percentage decrease in gate-count of the smallest circuit relative to the median.

idealised model ion trap under consideration, labelled with 3 data qubits and 4 parity qubits as required by the $[[7,3,3]]$ code. Under the restriction of nearest-neighbour connectivity, interactions between spatially separated qubits can still be realised by performing SWAP operations. For example, interacting qubit B with p_1 would first require a SWAP gate between qubits B and C, or qubits C and p_1. Circuits with fewer long range interactions will require fewer SWAP gates, and will therefore have a reduced two-qubit gate count.

There are a number of strategies for calculating the sequences of SWAP operations required to compile a CPC circuit on a nearest-neighbour architecture. Here we adopt a simple approach in which qubits are always swapped in the upwards direction. As an example of this, in figure 19b, qubit p_1 is swapped upwards, instead of qubit B being swapped downwards. More advanced SWAP compilation strategies, that combine upwards and downwards moves, can yield circuits with lower SWAP counts. However, such analysis is computationally expensive, and can impose a bottleneck in the CPC design process. By restricting our approach to upwards SWAP moves only, an exhaustive search of the $[[7,3,3]]$ CPC codes remains possible.

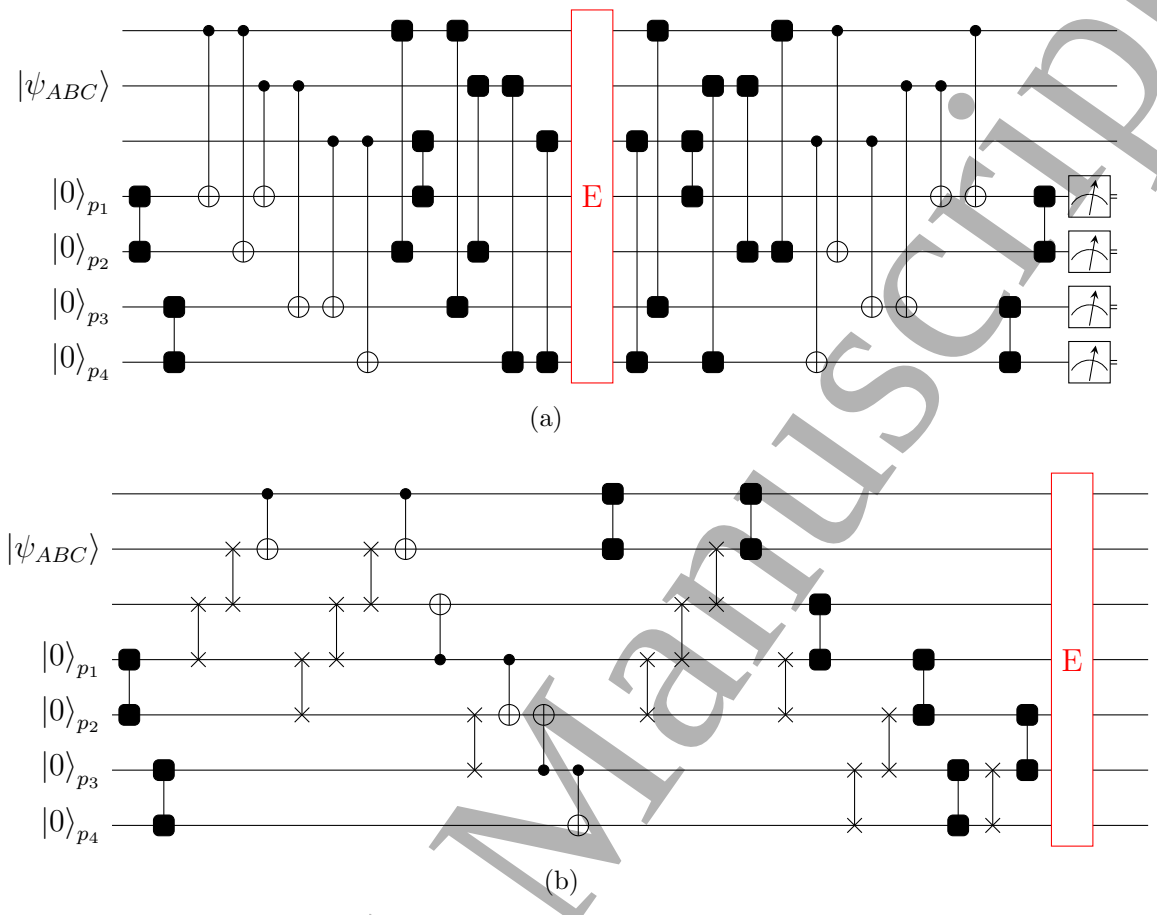


Figure 21: Circuit diagrams demonstrating SWAP gate compilation for a nearest-neighbour architecture. (a) The $[[7, 3, 3]]$ code with the smallest two-qubit gate count prior to the addition of SWAP gates. (b) The encoder for the same circuit, with SWAP gates included.

Figure 20 shows the histogram of the SWAP compiled $[[7, 3, 3]]$ codes distributed by the total two-qubit gate count (CPC gates + SWAP gates). The optimum $[[7, 3, 3]]$ CPC code with the shortest encoder is shown in figure 21b. The encoder for this circuit includes 14 CPC gates, and requires an additional 13 SWAP operations to be implemented on a linear, nearest-neighbour architecture. The depth of the encoder, in terms of the number of two qubit gates, is therefore 27. For comparison, the un-compilied version of this $[[7, 3, 3]]$ code is shown in figure 21a.

The results of two-qubit gate count analysis for the $[[7, 3, 3]]$ codes, following compilation onto the nearest-neighbour hardware, are summarised in table 4. The optimum circuit has an encoder length of 27, compared to the median of 51, a 47% reduction in circuit gate count. Only one CPC code was discovered with the minimum encoder length. The CPC circuit optimisation, with regards to qubit layout, can therefore be considered complete.

	Local gate count (number of single-qubit gates)	
	Mininum	**Median**
Before simplification	72 (1 discovered)	92
After simplification	7 (1 discovered)	10
% change	90%	89%

Table 5: Summary of the local gate-count for the $[[7, 3, 3]]$ following compilation with the SP native gate.

6.3. Stage 3: Native gate compilation

The ion trap under consideration has a native gate that resembles the symmetrised phase (SP) gate introduced in section 4. The final stage of the CPC design process involves systematically applying the SP simplification procedures described in section 5.1 to each of the 306,480 discovered CPC codes. The compilation efficiency of a given code can be quantified by counting the number of local gates that remain in the simplified circuit. The optimal code for the ion trap device is then identified as the circuit with the shortest total encoder length, defined by

$$\mathcal{L}_{\text{CPC}} = |\text{CPC}| + |\text{SWAP}| + |\text{LOCAL}|, \tag{24}$$

where $|\text{CPC}|$ is the CPC gate count, $|\text{SWAP}|$ is the SWAP gate count and $|\text{LOCAL}|$ is the local gate count.

Table 5 summarises the simplification statistics for the local gate counts when the $[[7, 3, 3]]$ CPC codes are compiled with the SP native gate. Without applying any simplifications, the median local gate count is 92. After applying the simplification routine, the median is 10, an 89% reduction in gate count.

The compiled $[[7, 3, 3]]$ CPC circuit with the lowest local gate count after simplification is shown in figure 22. This circuit is compiled from the CPC code with the lowest two-qubit gate count, as discovered in the last subsection and depicted in figure 21. We can therefore identify the compiled $[[7, 3, 3]]$ code in figure 22 as the optimum code for our device with a total gate count of $\mathcal{L}_{\text{CPC}} = 34$. For comparison, the median total gate count across all 306,480 CPC codes was $\mathcal{L}_{\text{CPC}} = 61$. The total reduction in circuit depth for the optimised circuit relative to the median is therefore 44%. Note that it will not always be the case that the circuit with the lowest two-qubit gate count will also be the circuit that compiles most efficiently. For this reason, the entire discovered set of $[[7, 3, 3]]$ CPC codes were considered in stage 3 of the design process, rather than restricting the analysis to the single code identified in stage 2.

7. Outlook and conclusion

In this work, we assessed the real-world functionality of CPC codes by implementing full encode-decode cycles of a $[[4, 2, 2]]$ quantum error detection code on the IBM 5Q

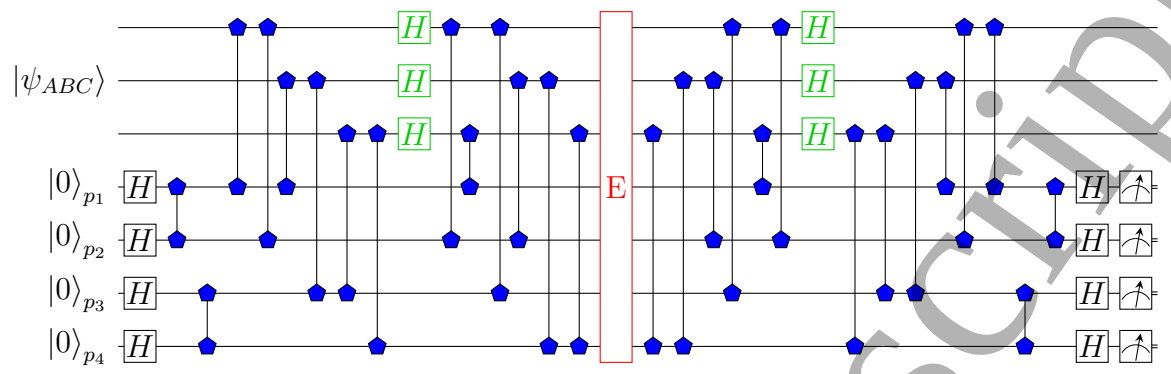


Figure 22: The native gate compiled form of the $[[7, 3, 3]]$ CPC with the lowest total gate count. Note that the SWAP operations have been omitted to save space.

quantum computer. We then explored the utility of the CPC framework in deriving larger quantum codes. In particular, we illustrated a design process for the automated discovery and optimisation of CPC codes by applying it to a seven qubit ion trap device. In the first stage of the design process, exhaustive code-search methods were used to find $[[7, 3, 3]]$ CPC codes that saturate the quantum Hamming bound for seven qubits. These circuits were then modified through the addition of SWAP gates to allow them to be implemented on a nearest-neighbour architecture. Finally, the circuits were compiled with a SP native gate. At the end of the design process, the optimum hardware-ready code with the lowest gate count of $\mathcal{L}_{\text{CPC}} = 34$ was identified.

The design process outlined for ion traps will be adaptable to other qubit technologies. In section 5 we demonstrated that the symmetric *encode-error-decode* structure of CPC codes allows for efficient compilation with any realistic maximally entangling Clifford gate. This result means that simplification routines, similar to those seen with the ion trap SP gate, will be possible for a broad range of native gates from different quantum experiments.

The final circuit in the outline of the CPC design process, drawn in figure 22, shows the best CPC code in terms of total gate count. Here it was assumed, however, that each gate type – CPC, SWAP and local – are equal in terms of the overhead they impose on the code implementation. In practice, however, some types of operations will be more expensive than others. For example, in an ion trap setting, it is typically the case that two-qubit interactions have a lower fidelity than single-qubit operations [11, 43]. When implementing the CPC design process, such considerations should be taken into account for choosing the optimum code for the given device. For example, each CPC code could be assigned a weighted total gate count, $\mathcal{R}_{\text{CPC}}$, given by

$$\mathcal{R}_{\text{CPC}} = \gamma_1|\text{CPC}| + \gamma_2|\text{SWAP }| + \gamma_3|\text{LOCAL}|, \tag{25}$$

where $|\text{CPC}|$, $|\text{SWAP}|$ and $|\text{LOCAL}|$ are the counts for CPC gates, SWAP gates and local gates respectively. The count for each gate-type is weighted by a penalty strength γ which is based on the gate count.

In the code discovery stage of the CPC design process for the ion trap device, the aim was to find working $[[7, 3, 3]]$ codes that saturate the quantum Hamming bound for seven qubits. This involved calculating the code distance for all possible permutations of $[[7, 3, d =?]]$ CPC codes, a total of 2^{30} circuits. Using the syndrome calculation algorithm outlined in appendix C, it was possible to exhaustively analyse all the circuits in less than a week on a desktop computer. In total, the search yielded $306,480$ working $[[7, 3, 3]]$ codes (0.03% of the search space).

For a CPC code with 4 data qubits, the quantum Hamming bound tells us that the optimal CPC code is of type $[[9, 4, 3]]$. However, there are 2^{50} permutations of this circuits of the form $[[9, 4, d =?]]$, which is an impractical search space for exhaustive methods. In the original CPC paper, it was shown that $[[9, 4, 3]]$ codes can be discovered simply by randomised search [3]. In future work, more sophisticated techniques, such as simulated annealing or parallel tempering, could be employed to more efficiently search for CPC codes.

When searching for large CPC codes, the number of circuits in the search space could be reduced by considering hardware constraints in advance. For example, for a nearest-neighbour device, each circuit permutation could be assigned a score on the basis of how many long-range interactions it contains. The code distance would then only be measured for the circuits with fewer long range interactions. Another optimisation parameter that could be considered is the weight of the code's stabilizers, a parameter that is useful to minimise when constructing fault tolerant circuits. Exhaustive and random search strategies for quantum code discovery have also been studied in [46, 18]. The particular strength of the CPC framework is that the symmetric *encode-error-decode* structure ensures the search is constrained to a space of non-disturbing codes. Investigating whether optimised CPC search strategies provide a higher density of good codes compared to other code search techniques would be an interesting area for future research.

Another feature of the CPC framework is that any classical code can be re-purposed for the bit and phase checking stages of the code. If such an approach is adopted, only the space of cross-checks needs to be searched in order to obtain a CPC code with fixed distance. Owing to the demands of modern high-density communication networks, classical error correction protocols such as low density parity check and turbo codes have been extensively optimised [6, 47]. At large scales, these codes can be decoded in real time at close to the theoretical maximum rate for information transfer along a noisy channel given by the Shannon limit [48]. The tools of the CPC framework could help construct quantum versions of low density parity check and turbo codes. A presentation of the CPC framework in terms of classical factor graph notation can be found in [49].

An important direction for future work is to investigate ways of making CPC circuits fault tolerant. For most quantum computing architectures, it is not realistic to assume that the encode and decode stages will be fault-free, or that errors will only occur within a specified wait-stage. In section 3 it was shown that a specific implementation of a $[[4, 2, 2]]$ CPC detection code can be specially hardened against single-qubit errors

occuring after any multi-qubit gate in the encoder or decoder. However, further work is required to develop methods for extending general CPC codes to full fault tolerance. Of particular interest are recent studies into fault tolerant computing using flag checks, which have a similar construction to CPC parity checks [32, 50].

The CPC framework lifts many of the restrictions that have hindered the development of traditional QEC codes. In particular, CPC codes have a canonical structure that allows any sequence of parity checks to be performed on a quantum register without risk of decohering the encoded information. The process of deriving CPC codes is therefore reduced to a classical decoding problem, allowing for code discovery via numerical search. This opens up the possibility of constructing custom QEC protocols to meet the hardware and layout demands of a specific quantum computing experiment.

Acknowledgements

Joschka Roffe was supported by a Durham Doctoral Studentship (Faculty of Science). Nicholas Chancellor, Dominic Horsman and Viv Kendon were supported by EPSRC (grant ref:grant ref: EP/L022303/1). We acknowledge use of the IBM Quantum Experience for this work. The views expressed are those of the authors and do not reflect the official policy or position of IBM or the IBM Quantum Experience team. The quantum circuits in this paper were drawn using the QPIC package by Thomas Draper and Samuel Kutin [51].

References

[1] Networked Quantum Information Technologies (NQIT) project 2017. *https://nqit.ox.ac.uk/*, Accessed: 17/12/2017.

[2] IBM Quantum Experience 2017. *https://quantumexperience.ng.bluemix.net/qx/community*, Accessed: 17/12/2017.

[3] Nicholas Chancellor, Aleks Kissinger, Joschka Roffe, Stefan Zohren, and Dominic Horsman. Coherent parity check construction for quantum error correction. *ArXiv:1611.08012*, 2016.

[4] Bob Coecke and Ross Duncan. Interacting quantum observables: categorical algebra and diagrammatics. *New Journal of Physics*, 13(4):043016, 2011.

[5] Bob Coecke and Aleks Kissinger. *Picturing Quantum Processes: A First Course in Quantum Theory and Diagrammatic Reasoning*. Cambridge University Press, 2017.

[6] D.J.C. MacKay and R.M. Neal. Near Shannon limit performance of low density parity check codes. *Electronics Letters*, 32(18):1645, 1996.

[7] Naomi Nickerson, Ying Li, and Simon Benjamin. Topological quantum computing with a very noisy network and local error rates approaching one percent. *Nature Communications*, 4:1756, 2013.

[8] Timothy Proctor and Viv Kendon. Hybrid quantum computing with ancillas. *Contemporary Physics*, 57(4):459, 2016.

[9] Daniel Gottesman. Class of quantum error-correcting codes saturating the quantum Hamming bound. *Physical Review A*, 54(3):1862, 1996.

[10] Christopher Ballance. High-fidelity quantum logic in Ca^+. *Oxford University*, PhD thesis, 2014.

[11] C. J. Ballance, T. P. Harty, N. M. Linke, M. A. Sepiol, and D. M. Lucas. High-fidelity quantum logic gates using trapped-ion hyperfine qubits. *Physical Review Letters*, 117(6), 2016.

[12] J. Randall, S. Weidt, E. D. Standing, K. Lake, S. C. Webster, D. F. Murgia, T. Navickas, K. Roth, and W. K. Hensinger. Efficient preparation and detection of microwave dressed-state qubits and qutrits with trapped ions. *Physical Review A*, 91(1), 2015.

[13] S. Debnath, N. M. Linke, C. Figgatt, K. A. Landsman, K. Wright, and C. Monroe. Demonstration of a small programmable quantum computer with atomic qubits. *Nature*, 536(7614):63, 2016.

[14] M. F. Brandl, M. W. van Mourik, L. Postler, A. Nolf, K. Lakhmanskiy, R. R. Paiva, S. Möller, N. Daniilidis, H. Häffner, V. Kaushal, T. Ruster, C. Warschburger, H. Kaufmann, U. G. Poschinger, F. Schmidt-Kaler, P. Schindler, T. Monz, and R. Blatt. Cryogenic setup for trapped ion quantum computing. *Review of Scientific Instruments*, 87(11):113103, 2016.

[15] C. Horsman, A. Fowler, S. Devitt, and R. Van Meter. Surface code quantum computing by lattice surgery. *New Journal of Physics*, 14(12):123011, 2012.

[16] Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3), 2012.

[17] Jean-Pierre Tillich and Gilles Zemor. Quantum LDPC codes with positive rate and minimum distance proportional to the square root of the blocklength. *IEEE Transactions on Information Theory*, 60(2):1193, 2014.

[18] Winton Brown and Omar Fawzi. Short random circuits define good quantum error correcting codes. In *2013 IEEE International Symposium on Information Theory*. IEEE, 2013.

[19] Sergey Bravyi and Matthew B. Hastings. Homological product codes. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 273, New York, NY, USA, 2014. ACM.

[20] Nikolas P. Breuckmann and Barbara M. Terhal. Constructions and noise threshold of hyperbolic surface codes. *IEEE Transactions on Information Theory*, 62(6):3731, 2016.

[21] Benjamin Audoux and Alain Couvreur. On tensor products of CSS codes. *arXiv:1512.07081*, 2015.

[22] Peter Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52(4):R2493, 1995.

[23] Daniel Gottesman. Stabilizer codes and quantum error correction. *arXiv:quant-ph/9705052*, 1997.

[24] Simon J Devitt, William J Munro, and Kae Nemoto. Quantum error correction for beginners. *Reports on Progress in Physics*, 76(7):076001, 2013.

[25] Scott Aaronson and Daniel Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5), 2004.

[26] Simon Anders and Hans Briegel. Fast simulation of stabilizer circuits using a graph-state representation. *Physical Review A*, 73(2), 2006.

[27] Maika Takita, Andrew W. Cross, A. D. Córcoles, Jerry M. Chow, and Jay M. Gambetta. Experimental demonstration of fault-tolerant state preparation with superconducting qubits. *Physical Review Letters*, 119(18), 2017.

[28] Christophe Vuillot. Error detection is already helpful on the IBM 5Q chip. *arXiv:1705.08957*, 2017.

[29] IBM. IBMQX4 technical specifications. *https://github.com/QISKit/ibmqx-backend-information/blob/master/backends/ibmqx4/README.md*, Accessed: 17/12/2017.

[30] Peter Shor. Fault-tolerant quantum computation. *IEEE Comput. Soc. Press*, Proceedings of 37th Conference on Foundations of Computer Science.

[31] A. M. Steane. Active stabilization, quantum computation, and quantum state synthesis. *Physical Review Letters*, 78(11):2252, 1997.

[32] Rui Chao and Ben W. Reichardt. Quantum error correction with only two extra qubits. *arXiv:1705.02329*, 2017.

[33] IBM. Quantum Information Softare Kit. *www.qiskit.org*, Accessed: 17/12/2017.

[34] John A. Smolin, Jay M. Gambetta, and Graeme Smith. Efficient method for computing the maximum-likelihood quantum state from measurements with additive Gaussian noise. *Physical Review Letters*, 108(7), 2012.

[35] M Acton, K.-A. Brickman, P. C. Haljan, P. J. Lee, L. Deslauriers, and C. Monroe. Near-perfect simultaneous measurement of a qubit register. *Quantum Information and Computation*, 6:465, 2006.

[36] J. I. Cirac and P. Zoller. Quantum computations with cold trapped ions. *Physical Review Letters*, 74(20):4091, 1995.

[37] T. R. Tan, J. P. Gaebler, Y. Lin, Y. Wan, R. Bowler, D. Leibfried, and D. J. Wineland. Multi-element logic gates for trapped-ion qubits. *Nature*, 528(7582):380, 2015.

[38] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Physical Review A*, 89(2), 2014.

[39] D. Leibfried, B. DeMarco, V. Meyer, D. Lucas, M. Barrett, J. Britton, W. M. Itano, B. Jelenković, C. Langer, T. Rosenband, and D. J. Wineland. Experimental demonstration of a robust, high-fidelity geometric two ion-qubit phase gate. *Nature*, 422(6930):412, 2003.

[40] Anders Sørensen and Klaus Mølmer. Quantum computation with ions in thermal motion. *Physical Review Letters*, 82(9):1971, 1999.

[41] Daniel Loss and David P. DiVincenzo. Quantum computation with quantum dots. *Physical Review A*, 57(1):120, 1998.

[42] Norbert M. Linke, Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A. Landsman, Kenneth Wright, and Christopher Monroe. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305, 2017.

[43] T. P. Harty, D. T. C. Allcock, C. J. Ballance, L. Guidoni, H. A. Janacek, N. M. Linke, D. N. Stacey, and D. M. Lucas. High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit. *Physical Review Letters*, 113(22), 2014.

[44] Alan Robertson, Christopher Granade, Stephen D. Bartlett, and Steven T. Flammia. Tailored codes for small quantum memories. *Physical Review Applied*, 8(6), 2017.

[45] David K. Tuckett, Stephen D. Bartlett, and Steven T. Flammia. Ultrahigh error threshold for surface codes with biased noise. *Physical Review Letters*, 120(5), 2018.

[46] Markus Grassl. *Searching for linear codes with large minimum distance*, volume 19 of *Algorithms and Computation in Mathematics*. Springer, Heidelberg, 2006.

[47] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: turbo-codes. *IEEE Transactions on Communications*, 44(10):1261, 1996.

[48] David MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge University Press, Cambridge, United Kingdom, 2003.

[49] Joschka Roffe, Stefan Zohren, Dominic Horsman, and Nicholas Chancellor. Quantum codes from classical graphical models. *arXiv:1804.07653*, 2018.

[50] Christopher Chamberland and Michael E. Beverland. Flag fault-tolerant error correction with arbitrary distance codes. *Quantum*, 2:53, 2018.

[51] Thomas Draper and Samuel Kutin. QPIC: Quantum circuit diagrams in latex. *https://github.com/qpic/qpic*, Accessed: 17/12/2017.

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

[53] Daniel Gottesman. Theory of fault-tolerant quantum computation. *Physical Review A*, 57(1):127, 1998.

[54] Daniel Gottesman. The Heisenberg representation of quantum computers. *Group22: Proceedings of the XXII International Colloquium on Group Theoretical Methods in Physics, pp. 32-43, Cambridge, MA, International Press*, 1999.

Appendix

A. The Pauli group

The Pauli group on a single-qubit, $\mathcal{G}_1$, is defined as the set of Pauli operators

$$\mathcal{G}_1 = \{\pm\mathbb{1}, \pm\mathrm{i}\mathbb{1}, \pm X, \pm\mathrm{i}X, \pm Y, \pm\mathrm{i}Y, \pm Z, \pm\mathrm{i}Z\}, \tag{A1}$$

where the ± 1 and $\pm\mathrm{i}$ terms are included to ensure $\mathcal{G}_1$ is closed under multiplication and thus forms a legitimate group [52]. In matrix form, the four Pauli operators are given by

$$\mathbb{1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad Y = \begin{pmatrix} 0 & -\mathrm{i} \\ \mathrm{i} & 0 \end{pmatrix}, \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}. \tag{A2}$$

The general Pauli group, $\mathcal{G}$, consists of the set of all operators that are formed from tensor products of the matrices in $\mathcal{G}_1$. For example, the operator

$$\mathbb{1} \otimes X \otimes \mathbb{1} \otimes Y \in \mathcal{G} \tag{A3}$$

is an element of the four-qubit Pauli group. Note that for simplicity, in the context of quantum computing, the above operator would usually be expressed as X_2Y_4. The identity operators are omitted, and the remaining elements are subscripted with the label of the qubit they act on.

The elements of the Pauli group have eigenvalues $\{\pm 1, \pm\mathrm{i}\}$. Another useful property of the Pauli group is that its elements either commute or anti-commute with one another.

B. The Clifford group and stabilizer states

The Clifford group $\mathcal{C}$ is defined as the set of operators that normalise the Pauli group such that

$$U_\mathrm{C} \bullet P_i \bullet U_\mathrm{C}^\dagger = P_j, \quad U_\mathrm{C} \in \mathcal{C}, \quad \{P_i, P_j\} \in \mathcal{G} \;\forall\; \{i, j\}, \tag{B4}$$

where $U_\mathrm{C} \in \mathcal{C}$ is a Clifford operator and P_k are elements of the Pauli group. Clifford gates, $\mathcal{C}$, are generated by the set of three gates $\langle \mathrm{CNOT}, H, P\rangle$, such that $\mathcal{C} = \langle \mathrm{CNOT}, H, P\rangle$ [53]. Likewise, single-qubit Clifford gates, $\mathcal{C}_1$, are generated by the set $\langle H, P\rangle$, such that $\mathcal{C}_1 = \langle H, P\rangle$.

The stabilizer states are all the quantum states that can be reached from a blank register, $|0\rangle^{\otimes N}$, via the application of Clifford gates and computational basis measurements. Quantum circuits consisting only of Clifford gates acting on stabilizer states can be efficiently classically simulated. The proof of this is given by the Gottesman-Knill theorem [54]. Although the Clifford group is not a universal quantum gate set, it is sufficient for simulating many QEC circuits and all the quantum memories described in this paper.

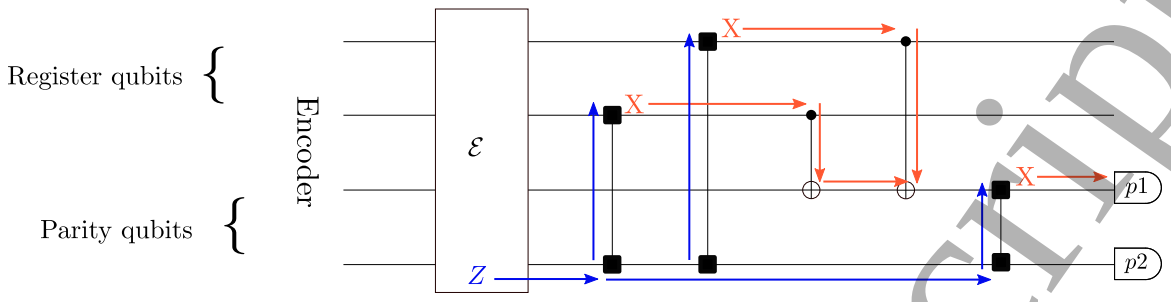


Figure C1: The [[4,2,2]] code decoder depicting the different propagation pathways for Z errors on the second parity qubit.

C. Efficient calculation of CPC code syndrome table

In addition to providing a compact way to describe CPC codes, the adjacency matrix representation can be leveraged to create a simple algorithm for calculating syndrome tables, bypassing the need to perform a full stabilizer simulation. We will begin our presentation of this algorithm by considering errors on the data qubits, which are represented in terms of the row vectors $E_{d,x}$ and $E_{d,z}$. For example, in the [[4,2,2]] detection code, depicted in figure 8, a bit-flip error on qubit A would have the form $E_{d,x} = (1, 0)$. Likewise, a phase-flip on qubit B would be given by $E_{d,x} = (0, 1)$.

In a CPC code X and Z errors on the data qubits are propagated to the parity qubits via gate sequences described by the adjacency matrices m_b and m_p respectively. The syndromes resulting from this propagation can be calculated by multiplying the error vector by its corresponding adjacency matrix modulo 2. For example, the syndrome for a bit-flip error on qubit A of the [[4,2,2]] code is given by

$$S_{d,x} = E_{d,x} \cdot m_b = \begin{pmatrix} 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{C5}$$

The bit-flip error information is propagated to the parity qubit by a CNOT, and the column vector on the right gives the subsequent measurement outcomes of the parity qubits $p1$ and $p2$. Our expression therefore tells us that error on the data qubit A produces the syndrome '10', a result in agreement with the values given in table 1 in section 2.5. Similarly, the syndromes for phase-flip errors on the data qubits can be computed with the expression $S_{d,z} = E_{d,z} \cdot m_p$.

We now need a method for calculating the syndromes for errors occurring on the parity qubits. Again, we represent X and Z errors in terms of two row vectors $E_{p,x}$ and $E_{p,z}$. In the case of bit-flip errors, the syndrome is simply given by $S_{p,x} = E_{p,x}$. This is the case as the bit-flip errors commute through the conjugate propagator gates and the CNOT targets, and will therefore propagate directly to the end of the circuit. The final error type to consider are phase-flips on the parity qubits.

Figure C1 depicts the propagation of such an error through the decoder of the $[[4, 2, 2]]$ code. To calculate the syndromes, there are two propagation pathways to be considered. Figure C1 shows that Z errors can be propagated to the register by the

phase-check conjugate propagator gates, after which they can be considered as bit-errors. These bit-flip errors are then propagated to the register, as illustrated by the orange arrows in figure C1. This propagation pathway can be represented mathematically by the expression $E_{p,z} \cdot m_p^T \cdot m_b$. Note that we have taken the transpose of the phase-check matrix as we are propagating information from the parity bits to register. The second pathway to be considered for phase-flip errors on the parity qubits, is the propagation due to the cross-check operators. As the cross-check operators can act both ways, this pathway is described by the expression $E_{p,z} \cdot \left(m_c + m_c^T\right)$. Combining both error propagation pathways, the syndrome expression for phase-flip errors on the parity qubits is $S_{p,z} = E_{p,z} \cdot m_p^T \cdot m_b \ + \ E_{p,z} \cdot \left(m_c + m_c^T\right)$, where all addition and multiplication is performed modulo 2. The full syndrome equation can now be written by summing the contributions $S_{d,x}$, $S_{d,z}$, $S_{p,x}$ and $S_{p,z}$ to give

$$S = \left(E_{d,x} \cdot m_b + E_{d,z} \cdot m_p + E_{p,x} + E_{p,z} \cdot m_p^T \cdot m_b + E_{p,z} \cdot \left(m_c + m_c^T\right)\right) \bmod 2. \quad \text{(C6)}$$

The above equation allows the syndromes for a given error circuit to be calculated in time $O(n^2)$. It would be interesting to investigate how this algorithm relates to other efficient stabilizer simulators such as [25] and [26].

D. IMBQX4 calibration data

The experiment on the IBMQX4 outlined in section 3 was run over three days on 25$^{\text{th}}$ November 2017, 26$^{\text{th}}$ November and 27$^{\text{th}}$ November 2017. The calibration data for each of these days can be found below:

```
{Date: 11-25-2017,
Single-qubit error rates (10^-3):
{Q0: 0.94, Q1: 0.60, Q2: 1.12, Q3: 1.37, Q4: 1.80},
Readout error rates (10^-2):
{Q0: 4.10, Q1: 5.70, Q2: 4.00, Q3: 3.30, Q4: 5.10},
Two-qubit error rates (10^-2):
{CX1_0: 1.88, CX2_0: 2.09, CX3_2: 1.97, CX2_1: 4.28, CX3_4: 2.15, CX2_4: 3.89}}

{Date: 11-26-2017,
 Single-qubit error rates (10^-3):
 {Q0: 0.86, Q1: 0.69, Q2: 1.12, Q3: 1.89, Q4: 2.06},
 Readout error rates (10^-2):
 {Q0: 4.10, Q1: 4.10, Q2: 4.30, Q3: 5.30, Q4: 7.10},
 Two-qubit error rates (10^-2):
 {CX1_0: 2.31, CX2_0: 2.22, CX3_2: 2.18, CX2_1: 4.80, CX3_4: 2.43, CX2_4: 3.93}}

{Date: 11-27-2017,
Single-qubit error rates (10^-3):
{Q0: 0.77, Q1: 0.43, Q2: 1.20, Q3: 1.72, Q4: 1.89},
```

```
Readout error rates (10^-2):
{Q0: 3.70, Q1: 4.90, Q2: 4.20, Q3: 5.30, Q4: 5.80},
Two-qubit error rates (10^-2):
{CX1_0: 2.01, CX2_0: 1.93, CX3_2: 2.75, CX2_1: 4.47, CX3_4: 2.28, CX2_4: 4.13}}
```