Architecting Scalable Trapped Ion Quantum Computers using Surface Codes

Scott Jones, University of Cambridge. sj665@cam.ac.uk
Dr. Prakash Murali, University of Cambridge. pm830@cam.ac.uk

Abstract

Trapped ion (TI) qubits are a leading quantum computing platform. Current TI systems have less than 60 qubits, but a modular architecture known as the Quantum Charge-Coupled Device (QCCD) is a promising path to scale up devices. There is a large gap between the error rates of near-term systems (10^{-3} to 10^{-4}) and the requirements of practical applications (below 10^{-9}). To bridge this gap, we require Quantum Error Correction (QEC) to build *logical qubits* that are composed of multiple physical qubits. While logical qubits have been demonstrated on TI qubits, these demonstrations are restricted to small codes and systems. There is no clarity on how QCCD systems should be designed to implement practical-scale QEC. This paper studies how surface codes, a standard QEC scheme, can be implemented efficiently on QCCD-based systems. To examine how architectural parameters of a QCCD system can be tuned for surface codes, we develop a near-optimal topology-aware compilation method that outperforms existing QCCD compilers by an average of 3.8X in terms of logical clock speed. We use this compiler to examine how hardware trap capacity, connectivity and electrode wiring choices can be optimised for surface code implementation. In particular, we demonstrate that small traps of two ions are surprisingly ideal from both a performance-optimal and hardware-efficiency standpoint. This result runs counter to prior intuition that larger traps (20-30 ions) would be preferable, and has the potential to inform design choices for upcoming systems.

1 Introduction

Trapped ions (TI) qubits are an important platform for realising scalable quantum computers. Over a hundred academic groups are pursuing this technology [27], and production systems have been demonstrated by IonQ, Quantinuum, Oxford Ionics and other vendors [15, 16, 29]. Small TI systems use a monolithic architecture where all qubits are housed in the same physical trap. This design is not scalable due to control challenges and poor gate fidelities (quality of gate operations), especially beyond 30 qubits [21, 26]. Instead, a modular design where ions are distributed across many small traps is seen as a path towards scalable systems. This architecture, termed the Quantum Charge-Coupled Device (QCCD) was first proposed in 2002 [17] and has been demonstrated in practice by Quantinuum [24]. Figure 1 shows an example QCCD system with four traps.

To achieve a practical quantum advantage over classical computing, we require $\approx 100 - 1000$ algorithmic qubits with an error rate of at least 10^{-9} [1], which is well beyond the limits of all known qubit technologies [2, 14]. Therefore, we require quantum error correction (QEC). Similarly to classical error correction, where bits are redundantly encoded, QEC encodes a *logical qubit* across multiple physical qubits, detecting and correcting errors. The surface code [10] is among the most promising candidates for QEC codes due to its compatibility with planar architectures. In this paper, we study how surface code-based logical qubits can be efficiently implemented on QCCD hardware. Although our work focuses on the surface code, our techniques and framework are more broadly applicable.

For two reasons, implementing scalable surface code logical qubits in a QCCD architecture is non-trivial. First, QCCD systems offer a rich architectural design space with a range of trap capacity (number of ions per trap), communication topology (wiring between traps) and control wiring (hardware responsible for orchestrating ion movement) choices. The performance of the surface code logical qubit and its logical error rate depend heavily on the underlying device architecture. *How should device architects navigate these choices for logical qubit implementation?* Second, the performance of the surface code also depends on its mapping to the hardware and the routing techniques that are used to orchestrate the movement of ions. *How can we optimise these mappings across various architectures and surface code parameters?* Previous work and industry roadmaps either focus on noisy intermediate-scale quantum (NISQ) workloads [26] or use manual mappings [20] or only pick out a few architectural choices without rigorous architectural exploration [33].

Our work performs the first systematic design space exploration for logical qubit implementation on QCCD devices. We require an efficient compilation of surface code parity-check circuits (Figure 3) onto diverse QCCD architectures to enable architecture evaluation. Only a few compilers [26, 30, 32] support QCCD, but they are designed for NISQ applications on small QCCD hardware [25]. We developed a novel QEC and device topology-aware mapping scheme that exploits the parallelism and structure inherent in the parity check operations in the surface code to find good mapping solutions. Our compiler maps logical qubits to hardware and then implements logical qubit instructions using low-level QCCD primitives while adhering to QCCD hardware constraints. Using this compiler, we develop a toolflow for design space

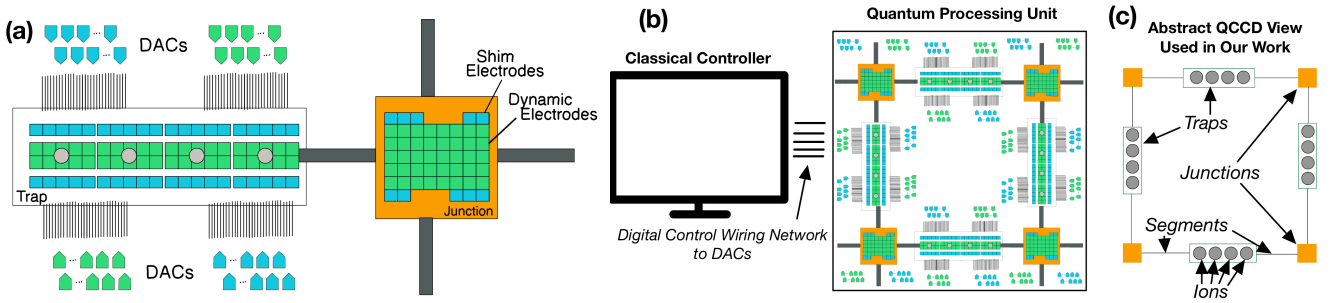

Figure 1. Quantum Charge-Coupled Device (QCCD) system. A detailed view of the QCCD hardware, where ions (grey circles) serve as qubits and are confined within an electromagnetic field known as a trap. **(a)** The trap is structured with different types of electrodes to position ions, including dynamic electrodes (green) for time-varying signals and shim electrodes (blue) for static potentials. Transport segments (black) and junctions (orange) allow ions to move between traps. **(b)** The QCCD device is controlled by a classical system interfacing with Digital-to-Analog Converters (DACs), each responsible for individual electrode voltages, enabling precise ion control [20]. **(c)** We use an abstract QCCD view for this paper.

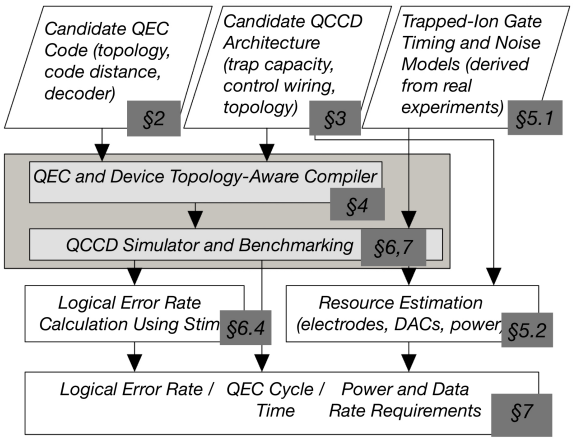

Figure 2. Framework for evaluating the suitability of a candidate QCCD-based TI system for error correction. Taking a candidate architecture and a candidate QEC code as input, the tool flow computes error correction metrics such as logical error rate, QEC round time and power dissipation requirements by using a QEC and device topology-aware compiler, QCCD simulator, and realistic models for performance and resource estimation.

exploration, shown in Figure 2. This toolflow accepts a QEC code and QCCD architecture as input and then arrives at an efficient mapping, which is used alongside architectural models and logical qubit simulations to determine metrics such as cycle time, logical error rate and data rate. We use the tool to sweep the architectural design space and select optimal designs. **Our contributions are as follows:**

- We identify important architectural parameters for the implementation of surface codes in QCCD systems. Unlike previous works[26], we identify that a trap capacity of two ions is surprisingly ideal even though it maximises communication operations. When paired with grid connectivity and direct wiring of electrodes to DACs, we can achieve near-optimal cycle times and low logical error rates across both small and large surface code implementations, compared to higher trap capacity configurations.
- Comparing WISE[22], a state-of-the-art wiring method with the standard QCCD architecture, we identify a power vs. cycle time scaling bottleneck. Existing wiring methods either offer high power with fast logical clock speeds or low power but very slow speeds. For near-term demonstrations, these techniques are sufficient. However, to scale up to hundreds of logical qubits, we require a fundamental re-design of the wiring architecture considering power consumption as part of hardware-software co-design.
- Our QEC and device topology-aware compiler offers near-optimal QEC round times, outperforming existing compilers by 3.8X [26, 30]. Unlike existing QEC compilers for QCCD systems, our compiler can handle large surface code implementations and scale to large trap capacities.

2 Background

Trapped ions: In a TI system, the ions act as qubits. For example, a popular choice is a Calcium ion. To hold ions in place, an electromagnetic field is used. This field is generated using DC electrodes. As a result of this control mechanism, the ions are arranged as a linear chain. Single-qubit gates are implemented using a laser to excite a specific ion, while two-qubit gates involve multiple lasers that excite the internal states and shared vibrational motion of ions within the same trap.

Surface codes: Figure 3 illustrates a surface code qubit. Surface codes are a family of QEC codes that encode a logical qubit into a planar $d \times d$ grid of physical qubits, called data qubits, where d is the *code distance*. QEC is effective only

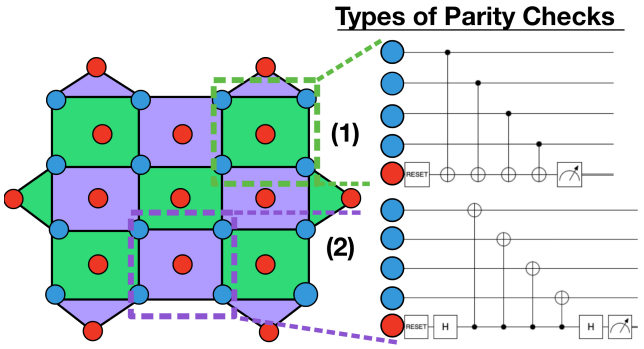

Figure 3. The topology of the distance four surface code. The blue circles represent physical data qubits, and the red circles represent physical ancilla qubits. Data qubits form the vertices of the cells that make up the shaded surface, and there is exactly one ancilla qubit in the centre of every cell. The cells are shaded purple or green to disambiguate the two types of parity checks, with each type of circuit given on the right.

when the physical error rate of the qubits in the hardware is below the *code threshold*. Below the threshold, a larger code distance offers exponentially lower logical error rates at the expense of more physical qubits per logical qubit (scaling as $O(d^2)$).

We focus on the surface code due to its high code threshold and ease of hardware implementation. This is because most quantum circuits for the surface code are a regular set of parity checks, where every ancilla (red) qubit is initialised, then the ancilla has a two-qubit entanglement gate with only each of its 4 neighbouring data (blue) qubits, and finally the ancilla qubit is measured (shown on the right of Figure 3). It is a well-accepted choice for TI systems [20].

Primitive QCCD Operations:

We use a set of primitive operations that provide the quantum gates necessary to maintain a logical qubit [13]. The entangling gate is a two-qubit Mølmer–Sørenson (MS) gate (t1); the implementation details are not relevant for this paper. Single-qubit gates are rotations around the x, y, and z axis on a single isolated ion (t2-t4). In addition, there are (t5) measurements of trapped-ion qubits and (t6) qubit reset. QCCD movement techniques include (t7) shuttling (moving) an ion across a transport segment connecting one trap or junction to another, (t8) splitting (moving an ion from a trap into a segment) and (t9) merging (moving an ion from a segment into a trap). An ion must be at the end of a trap in order to split (t8), which can be done by swapping ions within a trap (via 3 two-qubit gates (t1)). The final primitives are (t10) junction crossing entry and (t11) exit, whereby ions move across junctions that connect different segments. We assume that only a single ion could reside in a junction and that only a single ion could reside in a single segment at any moment [5, 6, 35].

3 QCCD Logical Qubit Design Trade-offs

3.1 Trap Capacity

A key architectural choice for QCCD systems is trap capacity, defined as the maximum number of qubits per trap. For example, Figure 1 shows a trap with capacity 4. There are three aspects to the choice of trap capacity. First, with high capacity, inter-ion spacing reduces and makes it difficult to address individual ions in the trap with laser controllers [26], leading to poor gate fidelity. Second, with high capacity, the need for communication operations is reduced. This can improve overall circuit fidelity due to a shorter depth and the reduction in the number of noisy operations. Third, in typical trapped-ion QC implementations, the gates within the same trap are executed serially. Although parallel two-qubit gates have been demonstrated [9], these gate times are 6X worse than the sequential gate times we assume and the gates have been challenging to realise beyond small scales [26]. To our knowledge, current QCCD platforms (IonQ, Quantinuum) do not offer parallel two-qubit gates within a trap for this reason [7]. Therefore, QCCD systems with multiple small traps can execute more gates in parallel, reducing the overall execution time.

While prior works have explored the choice of trap capacity for NISQ workloads [26], **the optimal trap capacity for logical qubit implementation with QCCD systems is unknown**. For surface code logical qubits, there are intuitive choices for this parameter. For example, each qubit can be mapped to a separate trap. This offers the highest possible two-qubit gate fidelity at the expense of many communication operations. Similarly, non-adjacent parity checks, shown on the right of Figure 3, can each be mapped to a trap with capacity 5. This reduces communication compared to the former case. As an extreme choice, the entire logical qubit in Figure 3 can be mapped to a single trap with capacity 31 (IonQ's systems adopt this approach [7]). As discussed, this serialises operations and kills the inherent parallelism available.

3.2 Communication Topology

To determine the optimal trap capacity, it is crucial to consider the communication topology of the QCCD device. The choice of topology determines the number of ion transport operations (t7-t11 §2) that will be required. Ions have all-to-all connectivity within a trap, while ions in different traps are connected by shuttling paths, which are implemented using segments and junctions in hardware (Figure 1). Unlike general NISQ workloads with widely varying communication requirements [26], surface code parity-check circuits have a regular local structure. As a result, ion movement operations can remain local if the communication topology between ions preserves the structure of the surface code. For example, a grid topology, where traps are interconnected by a grid network of shuttling paths and junctions (Figure 1), closely

aligns with the structure of the surface code when trap capacity is minimal [20]. However, **the performance of the grid topology is unclear when large trap capacities are used**. Further, we consider two more topologies as optimistic and pessimistic cases: an all-to-all switch topology where traps are connected using an n-way junction and a linear topology where all traps are connected to their nearest neighbour on a line. The optimistic case loosely resembles the MUSIQC architecture proposed for trapped ions [23], and the pessimistic case resembles the architecture of Quantinuum's current H-series devices [24].

3.3 Control System Wiring Choices

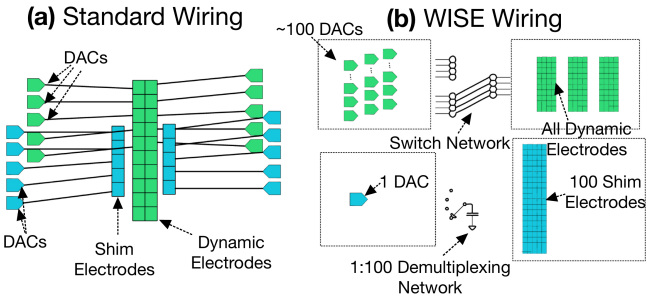

Figure 4. (a) Each electrode is connected to a dedicated DAC in the standard architecture, resulting in a direct but highly resource-intensive wiring scheme. **(b)** The WISE architecture integrates an ion trap with a switch-based demultiplexing network, significantly altering the scaling of control electronics. All dynamic electrodes (green) are controlled with $\approx$ 100 DACs irrespective of system size by using a switch network, but this comes at the cost that only primitive QCCD operations of the same type (t1-t11 §2) can execute simultaneously. One DAC can set $\approx$ 100 shim electrodes (blue).

Another key aspect of scaling trapped-ion QCCD systems for fault-tolerant quantum computation is managing control electronics. *How should electrodes (used to position and move ions) be wired to the digital-to-analog converters (DACs) which control trap voltages?* Traditional QCCD architectures employ one DAC per electrode (Figure 4). Each ion qubit requires tens of electrodes, and therefore, the number of control signals needed for implementing large surface code qubits becomes impractical. For instance, a distance 7 surface code (with 49 physical qubits) requires 5500 DACs per logical qubit, which is equivalent to $\approx$ 275GBit/s controller-to-QPU bandwidth (§5.2).

One leading alternative is the Wiring using Integrated Switching Electronics (WISE) architecture [22], which integrates a *switch-based demultiplexing network* (bottom of Figure 4). By sharing a smaller set of DACs across many electrodes, WISE scales more favourably regarding control complexity and power consumption. However, this benefit comes with a critical trade-off: only one type (t7-t11 §2) of ion movement primitive can co-occur, restricting parallelism in ion routing.

Given a QCCD architecture, the logical error rate of the surface code implementation and its cycle time are the two most important metrics that guide system design. Therefore, we ask *"What is the optimal trap capacity to achieve practical logical error rates for realistic surface code distances and logical clock speeds? Does the grid topology offer good code performance across a range of trap capacities? What is the best current wiring method? Does the reduced hardware overhead in WISE justify the longer logical clock speeds, or is the standard scheme more practical for achieving logical error rates less than* 10^{-9}*?"*

4 Topology-Aware QEC-to-QCCD Compiler

We require a resource-efficient mapping of the surface code onto QCCD systems with different architectures to answer the design questions. Although several tool flows have been developed to map NISQ workloads on QCCD hardware, they incur large communication overheads and do not scale to high code distances. In this section, we develop a surface code compiler shown in Figure 5. The compiler takes a surface code and QCCD configuration as inputs. Then, the surface code parity-check circuit is translated into native gates (§4.1). Each surface code qubit is then assigned to a physical qubit in the hardware (§4.2) and reconfiguration operations are inserted into the circuit to ensure that all two-qubit gates occur within the same trap (§4.3). Finally, the circuit is converted into an execution schedule (§4.4).

4.1 Mapping QEC Instructions to QCCD Instructions

Surface code parity-check circuits are expressed in terms of Hadamard, CNOT and measurement operations. These operations are converted into sequences of MS operations (t1) and single-qubit rotations (t2-t4) from the QCCD toolbox (§2) using known gate identities [8]. This is a straightforward intermediate-representation transformation.

4.2 Mapping Qubits to Ions

The second pass in Figure 5 assigns each qubit in the surface code to a unique physical qubit in the hardware. To determine the mapping, 1) we cluster the qubits into balanced partitions and 2) map the clusters to traps using a matching algorithm. Since there is all-to-all connectivity within a trap, the mapping of individual qubits in the cluster to trap qubits makes almost no difference in the overall execution schedule.

Mappings that fill traps well below capacity will increase the number of ion movements. Similarly, filling traps to maximum capacity is generally inefficient , as incoming ion movement would require displacing another ion. We adopt a design where the traps are filled to $capacity - 1$, leaving one ion position free for communication.

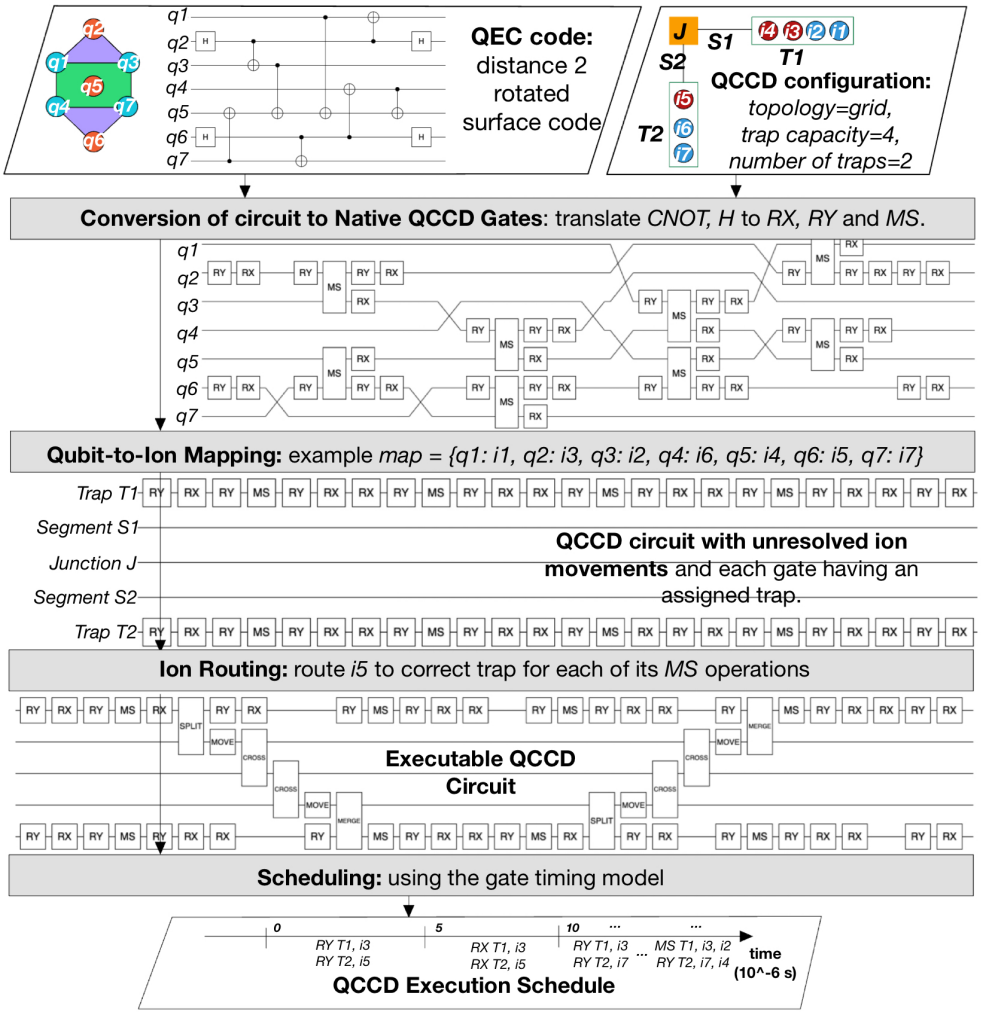

Figure 5. QCCD compilation flow: from a distance 2 surface code (syndrome extraction) circuit (top-left) and QCCD device configuration (top-right) to a scheduled, executable QCCD program. Steps include translation to native gates, qubit-to-ion mapping, ion routing using the movement primitives from the QCCD toolbox (§2), and scheduling using the operation timings in Table 1.

1. Clustering of qubits: To partition qubits (into clusters of size $capacity - 1$), we can solve a balanced graph partitioning problem. Given a graph $G = (V, E)$, where nodes V represent qubits and edges E represent pairs of data and ancilla qubits undergoing entanglement operations, with edge weight proportional to the order of operations in the circuit (early operations have high weight), the objective is to divide V into equal-sized clusters $C_1, \ldots, C_k$ such that the total weight of cut edges is minimised. Here, k will equal the number of traps used by the logical qubit in the QCCD hardware. The number of ion movement operations is minimised by minimising the number of high-priority entanglement operations cut. Note that balancing improves execution time due to fewer ion reconfigurations, which decreases the logical error rate when qubits are noisy. Balancing does not affect correctness: all partitions result in correct sequences of operations for the surface codes if the underlying qubits are perfect.

In general, the balanced graph partitioning problem is NP-complete [11] and has no finite factor polynomial-time approximation when partitions must be exactly equal [3]. Therefore, other compilers [26, 30, 32] that are designed for general quantum circuits are not able to efficiently cluster qubits for large code distances. Whereas, for regular grid-like graphs typical of surface codes, our compiler can approximate a balanced partition well. We use a top-down regular partitioning of the surface code topology, as depicted in Figure 6. This minimises ancilla movement between traps because qubit neighbourhoods are preserved in the map, and the surface code only contains entanglement operations between neighbouring qubits. Minor imbalances (by 1–2 qubits) can occur due to code boundary effects.

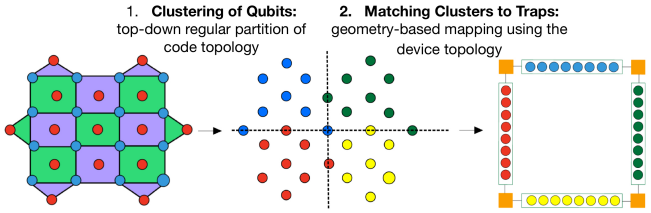

Figure 6. Mapping qubits to ions. Given a distance 4 surface code (left) and a QCCD device with trap capacity 9, we first partition into $ceil(N_{qubits}/(capacity-1)) = ceil(31/8) = 4$ clusters of qubits by top-down regular partitioning of the code topology (recursively bisecting the code's qubit layout). The surface code's regular structure means neighbouring qubits that share entanglement operations are likely grouped into the same cluster, reducing inter-trap communication. Clusters are then mapped to traps using a geometry-based matching that preserves local neighbourhoods, ensuring that qubits in different clusters but adjacent in the code are placed in neighbouring traps, minimising ion movement overhead.

2. Mapping of clusters to traps: Clusters are then mapped to traps by solving a minimum edge-weight, maximum cardinality matching problem, which results in a geometry-based mapping, as depicted in Figure 6, ensuring that the neighbours of each qubit that belong to different clusters still reside in neighbouring traps.

In the matching problem, the edges between clusters and traps are weighted by the distance between the centre of qubit clusters in the code topology and the trap positions in the hardware topology. The problem is solved by considering all subsets of traps with cardinality equal to the number of clusters, where, for each subset, we use the Hungarian algorithm [18] to compute the minimum perfect matching in polynomial time, and the subset with the lowest total cost is selected. For general quantum circuits, the search space can be reduced to an exponential number of trap subsets by considering only contiguous subsets (no holes) whose centres lie near the centre of all traps in the hardware. To achieve polynomial-time compilation, we further prune subsets using patterns in the boundary of the surface code topology. The compiler generalises to other scalable QEC codes, since they are expected to adhere to grid-like structures compatible with the grid QCCD communication topology, making the compiler suitable for expected real-world applications.

4.3 Ion Routing Algorithm

To be able to execute an entanglement operation between ions located in different traps, the compiler must determine the appropriate sequence of ion movement operations to ensure that both ions co-exist in the same trap. The physical state of the QCCD architecture during ion routing is modelled as a directed graph where nodes, representing traps and junctions, track the position of each ion, while edges in the graph track the sequence of movement primitives required to transfer an ion between nodes. For each ancilla qubit, the shortest path from the source to the destination trap is determined in the directed graph, and then edge labels along this route are concatenated for the sequence of primitives needed to move the qubit.

The ion routing algorithm computes a shortest path for each ancilla qubit to reach its corresponding data qubit's trap while satisfying QCCD hardware constraints:

- **Trap capacity:** Each trap has a fixed maximum ion count at any time [21, 26].
- **Junction exclusivity:** Only one ion may occupy a junction at any time [6].
- **Segment exclusivity:** Only one ion may occupy a shuttling segment at any time [5, 35].

Once the QCCD graph is constructed, the routing algorithm processes the sequence in multiple passes, moving the primitive operations into the output schedule until none remain. At the start and end of each pass, each trap is at most one ion below its capacity, and no junction nor segment contains an ion. These invariants ensure that the trap capacity constraint is met during execution. Each pass of the algorithm is described in Figure 7.

Operation	Duration	Infidelity
(t1) Two-qubit MS gate	$40\mu s$	(Refer to 5.1)
(t2-t4) Ion Rotation	$5\mu s$	(Refer to 5.1)
(t5) Measurement	$400\mu s$	1×10^{-3}
(t6) Qubit reset	$50\mu s$	5×10^{-3}
(t7) Ion shuttling	$5\mu s$	$\bar{n} < 0.1$
(t8-t9) Ion split and merge	$80\mu s$	$\bar{n} < 6$
(t10-t11) Junction entry/exit	$100\mu s$	$\bar{n} < 3$

Table 1. Operating parameters for QCCD systems derived from [13]. The reconfiguration steps (t7–t11) do not directly cause gate infidelity; however, they introduce idling noise and increase subsequent gate error rates due to heating, quantified using the mean vibrational energy $\bar{n}$. For our analysis, we pessimistically use the upper bound values.

4.4 Scheduling

During routing, a happens-before relation is constructed between operations. The scheduling is then performed after the routing and follows a dependency-preserving transformation that uses the operation times from Table 1.

5 Modelling Logical Qubits in QCCD

This section uses the compiled parity-check circuit to determine its hardware performance, logical error rate, and physical resource requirement. *The focus here is device modelling. It is essential for technical correctness of our work, but can be skipped by a classical computer-science reader.*

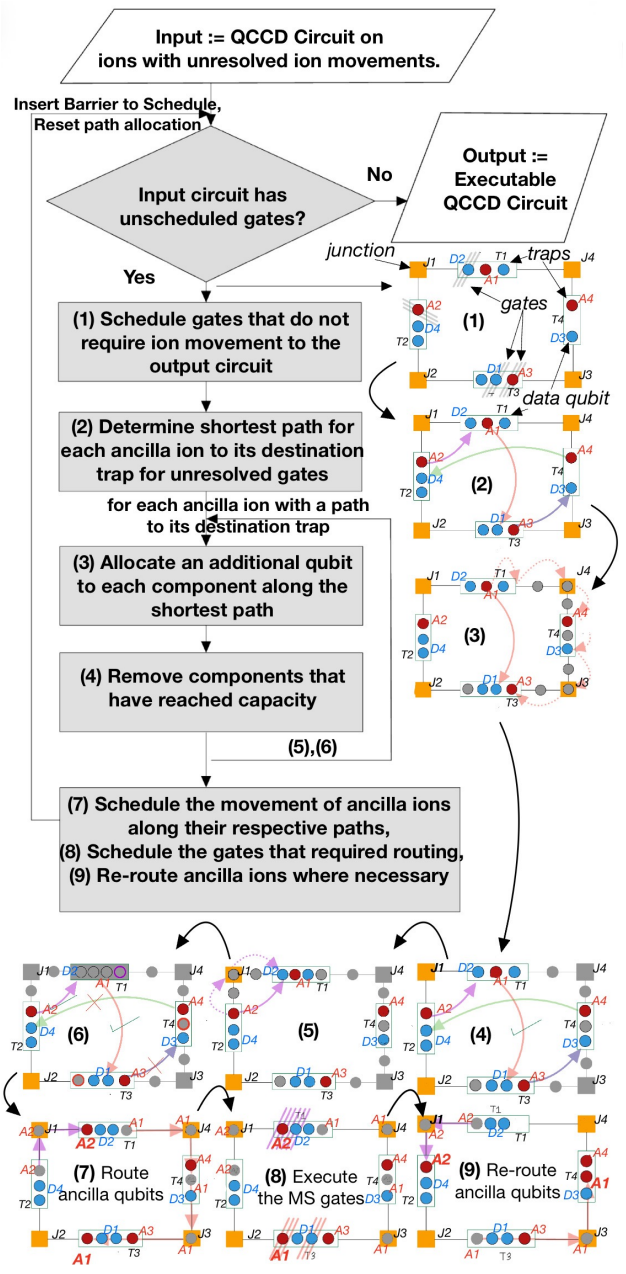

Figure 7. Ion Routing. (1) Gates that are not blocked by other unscheduled gates and do not need routing are scheduled. (2) The destination traps for each ancilla qubit are determined based on their next operation. Routing paths are allocated sequentially to ancilla qubits, prioritising those needed earlier in the input gate sequence. (3) Finds a path for ancilla A1, with each component along the path (except the source) being allocated a qubit. (4) Grey components (J4 and J3) have reached capacity, so they are removed (along with T4) from the QCCD graph. (5-6) repeat process (3-4) for ancilla A2. In (6), neither A3 nor A4 can be routed, so (7) proceeds to schedule the routing of A1 and A2 along their allocated paths. (8) Schedules the gates that require routing for A1 and A2. (9) Re-routes A1 and A2 to T4 and T2, respectively, to ensure traps are atleast 1 below capacity.

5.1 Performance and Noise Models

To determine the performance of a QCCD system for surface codes, we use a realistic performance and noise model for each primitive operation based on prior work, shown in Table 1. The runtime of the compiled circuit is calculated using the schedule of operations and the duration of each operation in Table 1.

Determining the logical error rate of the code is more involved and requires a noise simulation. We use Stim simulations for this purpose [12]. The input to Stim is a hardware noise model, which in our case is a realistic error-model for QCCD systems based on modelling of the relevant noise sources in trapped-ion hardware, as described in [13]. In addition, the model has been modified to account for the dependence of qubit gate error rates on the vibrational energy of ions, the number of ions, and the gate duration, as outlined in [26].

In QEC, physical errors can be decomposed into one of three Pauli channels: X for bit flip, Z for phase flip, or $XZ = Y$ for bit and phase flip. Our error model incorporates five independent noise parameters to account for the leading experimental imperfections, with different stochastic channels of Pauli errors for various operations:

1. **Dephasing e_1:** During ion chain-reconfiguration operations or when qubits are idle, Pauli Z errors occur with a probability $p(e_1)$ to account for collective qubit dephasing:
$$p(e_1) = \frac{1 - \exp(-t/T_2)}{2},$$
where t is the duration of the operation and $T_2 = 2.2$ seconds is the coherence time for the trapped-ion qubit, obtained from real experiments that demonstrated its accuracy [13].
2. **Depolarising errors after single-qubit gates e_2:** After single-qubit rotations, Pauli errors (X, Y, or Z) occur with equal probability $p(e_2)/3$.
3. **Depolarising errors after two-qubit gates e_3:** two-qubit Pauli errors (e.g. two bit flips (XX) or bit flip and phase flip (XZ)) occur with equal probability $p(e_3)/15$.
4. **Imperfect qubit reset e_4:** This is modelled as bit-flip (X) error occurring after qubit reset to the $|0\rangle$ state, with probability $p(e_4) = 5 \times 10^{-3}$.
5. **Imperfect qubit measurement e_5:** This is modelled as bit-flip (X) errors occurring during measurement with probability $p(e_5) = 1 \times 10^{-3}$.

Errors from ion movement are incorporated into the fidelity model, obtained from [26], which influences the probabilities of errors e_2 and e_3. The fidelity of the qubit gate is influenced by two primary factors: background heating from the trap's electromagnetic field and thermal motion from higher vibrational energy of the ion chain. The fidelity $p(e_2), p(e_3)$ is

expressed as:

$$p(e_2), p(e_3) = 1 - \Gamma\tau - A(2\bar{n} + 1),$$

Where Γ is the background heating rate of the trap, τ is the gate duration, $A \propto \frac{\ln(N)}{N}$ is a scaling factor representing thermal instabilities of the laser beams perpendicular to the ion chain, where N is the number of ions in the chain, and $\bar{n}$ is the vibrational energy of the ion chain, quantified in motional quanta (average energy state occupied). The term $\Gamma\tau$ accounts for fidelity loss due to background heating, which increases with the gate duration. The term $A(2\bar{n} + 1)$ captures the effects of thermal motion, which are exacerbated by shuttling operations that increase the vibrational energy of the ion chain.

We have validated our parameters against hardware data sheets from Quantinuum and IonQ. We also consider a range of gate improvements (1X to 10X) in our experiments to account for future improvements. A 5X improvement in our setup corresponds to $\approx 10^{-3}$ depolarising error rates per qubit gate, which is comparable to the best-known devices from Quantinuum and IonQ [7, 24].

Cooling Model: Cooling ions before qubit gates decrease physical error rates at the expense of increased execution times. To model the effect of cooling in the WISE wiring method, the noise model in Table 1 is modified, setting the baseline two-qubit gate error to 2×10^{-3} and the one-qubit gate error to 3×10^{-3}, while ignoring heating effects by adding an extra 850 μs to the two-qubit gate time [28].

5.2 Resource Estimation Model

The total number of electrodes N_e for a trap capacity k, number of junctions N_j, and number of traps N_t is given by: $N_e = N_{de} + N_{se} = N_{de/lz} \times N_{lz} + N_{de/jz} \times N_{jz} + N_{se/z} \times (N_{lz} + N_{jz})$ where:

- The number of linear zones: $N_{lz} = N_t \times k$,
- The number of junction zones: $N_{jz} = N_j$,
- The number of dynamic/shim electrodes per zone: $N_{lz/de} = 10$, $N_{jz/de} = 20$, and $N_{se/z} = 10$ [22].

Decreasing the trap capacity increases the number of electrodes for a fixed qubit count. This is because the ratio of junction zones to linear zones, N_{jz}/N_{lz} increases, so lower trap capacities require more electrodes (since junction zones require more electrodes than linear zones) [22].

The controller-to-QPU data rate (Figure 1) and power dissipation required are calculated using the number of electrodes for the standard QCCD architecture. The data rate between the QPU and its controller is $\approx 50\,\text{Mbit/s} \times N_e$ while the corresponding power dissipation is $\approx 30\,\text{mW} \times N_e$, where N_e denotes the number of electrodes.

In the WISE architecture, the data rate is $\approx 50\,\text{Mbit/s} \times N_{\text{DACs}}$, where the number of DACs is $N_{\text{DACs}} \approx 100 + \frac{N_{se}}{100}$, while N_{se} denotes the number of shim electrodes. As a result, the WISE architecture scales two orders of magnitude more favourably in terms of data rate compared to the standard architecture, significantly reducing the burden on control electronics [22].

6 Experimental Setup

Our experiments benchmark the performance of different combinations of QEC codes and QCCD configurations to answer the architectural questions posed in (§3).

6.1 QEC benchmarks

We use three benchmarks for our compiler: 1) repetition code and 2) unrotated surface code are two simple QEC schemes that serve only as baselines for compiler validation, while 3) rotated surface code (Figure 3) is a more efficient QEC scheme that serves as the primary workload for architectural experiments. We consider code distances d in the range 2 to 20. With increasing code distance, the surface code exponentially reduces errors, but uses a quadratically higher number of qubits (scaling as $2d^2 - 1$ physical qubits per logical qubit) and communication requirements. Our simulations consider the operation of logical identity in the surface code (essentially d rounds of parity-check measurements). This operation is selected because maintaining a logical qubit with an error rate lower than the physical error rate during idling is one of the most challenging aspects of quantum error correction. Other logical operations, implemented using transversal gates or lattice surgery, also rely on rounds of parity-checking, so the logical identity serves as a representative workload.

6.2 Architecture configurations

We explore trap capacities, ranging from 2 to 30, along with the grid, switch and linear connectivities. We also explore the standard choice for control system wiring, where each DAC is connected to one electrode, and the WISE architecture [22]. Since our study aims to understand the design choices for future systems with potentially improved hardware, we scale the physical error rates by a factor called *gate improvement*. For example, a 10X gate improvement corresponds to every gate having a 10X lower physical error rate and the dephasing physical error rate on idling qubits being 10X less. The gate improvement in our experiment varies from 1X to 10X.

We compile the parity-check circuit for each surface code distance d and architecture configuration combination. Then, we determine architectural and hardware parameters using models from the previous section and use Stim simulations to assess the logical error rate.

6.3 Metrics

Elapsed / QEC Round Time: The elapsed time is the **time required to run one round of surface code parity checks when considering gate times and communication times**.

Lower elapsed times are better. Prolonged rounds of parity-checking can exacerbate the effects of idling noise, becoming a bottleneck for error correction. Since every logical operation in a fault-tolerant algorithm contains d rounds of parity-checking to avoid the propagation of errors, the round time directly influences the logical clock speed.

Logical Error Rate: The logical error rate quantifies the primary objective of QEC: suppressing quantum errors to levels that enable fault-tolerant computation. The experiment looks to identify configurations capable of achieving a 10^{-9} logical error rate, which is a minimum requirement for large-scale algorithms [1].

Number of Movement / Routing Operations: The number of primitive ion reconfigurations, including split, move, merge, junction entry, exit (t7-t11), plus the number of gate swaps (with each gate swap being 3 two-qubit MS gates (§2)).

Theoretical Minimum Elapsed Time: To verify our compiler's performance, we manually compute the best possible elapsed time for specific QEC code and QCCD device combinations. For example, with a trap capacity of 2 a repetition code's structure can be exactly mapped to QCCD. However, since this metric is based on intuitive QEC-device mappings, there may be slight suboptimality in some cases.

Data Rate and Power: The data rate is the controller-to-QPU bandwidth required per logical qubit in GBit/s, and the power is the rate of energy dissipation of the QPU per logical qubit, calculated using the resource model in (§5.2).

6.4 Logical Error Rate Calculation Using Stim

The logical error rate calculation is performed by interfacing the physical noise model and the execution schedule of the compiled circuit into a noisy quantum circuit in Stim [12]. We use Stim version 1.13.0.

6.5 Baselines

Our QEC compiler (implemented in Python 3.11) is benchmarked against two other trapped-ion QCCD compilers: QCCDSim [26] and Muzzle The Shuttle [30] in terms of ion movement time and number of movement operations.

7 Results

7.1 Accurate and Scalable QEC Compiler

Table 2 compares the elapsed time for different QEC code and QCCD device model pairs with the theoretical minimum elapsed time. In 10 out of 16 cases, our compiler achieves the theoretical minimum time; in the remaining cases, it is away from the optimum by an average of 1.09X, worst case 1.11X. In addition, we test the routing tool in isolation by comparing the theoretical optimal number of routing operations in a schedule to the measured number of routing operations. On average, our compiler is within 1.04X of the theoretical minimum.

QEC Code	QCCD Topology	Theoretical Minimum Elapsed Time (μs)	Measured Elapsed Time (μs)	Number of Routing Operations (Theoretic / Measured)
Repetition Code Distance = 3	Linear Trap Capacity = 2	1535	1535	18 / 24
	Linear Trap Capacity = 3	1270	1390	6 / 10
	Linear Trap Capacity = 4	1385	1505	6 / 7
	Single Ion-Chain	2190	2190	0 / 0
Repetition Code Distance = 6	Linear Trap Capacity = 2	1535	1535	60 / 60
	Linear Trap Capacity = 3	2060	2300	27 / 29
	Linear Trap Capacity = 4	2425	2785	18 / 21
	Single Ion-Chain	5400	5400	0 / 0
2D Rotated Surface Code Distance = 2	Grid Trap Capacity = 2	4055	4055	48 / 48
	Linear Two Ion Chains	3225	3305	9 / 10
2D Unrotated Surface Code Distance = 2	Grid Trap Capacity = 3	4085	4325	56 / 60
2D Rotated Surface Code Distance = 3	Grid Trap Capacity = 2	4085	4085	288 / 288
	Linear Two Ion Chains	8605	8605	19 / 19
	Switch Trap Capacity = 2	5325	5325	432 / 432
2D Rotated Surface Code Distance = 6	Grid Trap Capacity = 2	4085	4085	1440 / 1440
2D Rotated Surface Code Distance = 12	Grid Trap Capacity = 2	4085	4085	6336 / 6336

Table 2. Comparison of our QEC compiler against theoretically optimal compilation. Our compiler is near-optimal regarding elapsed time and number of routing operations.

Table 3 compares the performance our QEC compiler QCCDSim [26] and Muzzle The Shuttle [30]. We benchmarked five rounds of error correction to account for any changes in qubit layout across rounds. For all baselines, the time required to execute gates is the same. Therefore, we focus on movement time (time required for ion reconfigurations) and the number of movement operations. Our QEC compiler achieves an average 3.85X reduction in movement time and an average 1.91X reduction in movement operations compared to the best of the two compilers in each test case. In the best case, the improvement is up to 6.03X. For the 2D rotated surface code, the QEC compiler successfully compiles five rounds of error correction across a wide range of trap capacities and code distances. In contrast, QCCDSim and MuzzleTheShuttle either produce suboptimal schedules or

	Movement Time For 5 Rounds			Number of Movement Operations		
	QEC Com.	QCCD Sim	Muzzle Shuttle	QEC Com.	QCCD Sim	Muzzle Shuttle
R,3,2,L	3300	8851	6365	40	219	173
R,5,2,L	3300	12521	31893	80	436	880
R,7,2,L	3300	20054	64194	120	713	1715
R,3,3,L	3135	3160	1666	58	71	35
R,5,3,L	3960	4178	4178	127	163	164
R7,3,L	4945	4178	4178	199	217	218
R,3,5,L	0	0	0	0	0	0
R5,5,L	1650	1663	1663	31	31	31
R,7,5,L	3300	1663	2323	61	58	58
S,2,2,G	10800	19083	NaN	240	327	NaN
S,3,2,G	13500	94738	NaN	720	2102	NaN
S,4,2,G	13500	NaN	NaN	1440	NaN	NaN
S,5,2,G	13500	NaN	NaN	2400	NaN	NaN
S,2,3,G	15980	9881	NaN	241	192	NaN
S,3,3,G	19410	59110	NaN	627	240	NaN
S,4,3,G	29610	NaN	NaN	1378	NaN	NaN
S,5,3,G	47920	NaN	NaN	2465	NaN	NaN
S,2,5,G	10260	5054	5076	116	57	67
S,3,5,G	22560	24777	122996	461	472	1740
S,4,5,G	30300	NaN	NaN	868	NaN	NaN
S,5,5,G	40460	NaN	NaN	1740	NaN	NaN

Table 3. Benchmark test of our compiler outlined with other QCCD compilers, namely QCCDSim and MuzzleTheShuttle. Each test determines the movement time and number of movement operations in the compiled schedules for a particular software-hardware configuration. A 4-tuple specifies each configuration: QEC code (R = repetition code, S = 2D Rotated Surface Code), Code Distance, Trap Capacity, and QCCD Communication Topology (L = linear, G = grid). In some cases, a QCCD constraint (§4.3) was violated, or the compilation failed, in which cases' NaN' is reported. For each test (row), the compilers are shaded green (best), amber or red (worst).

fail to compile entirely, especially at higher code distances. These results show that our compiler is well-suited for architectural evaluations.

7.2 Choice of Communication Topology

Figure 8(a) compares QEC round time as a function of code distance for the linear, grid, and all-to-all switch communication topologies. We show the results for capacities of 2, 5 and 12, but the trends are similar for other capacities. We make three observations. First, the linear topology exhibits high

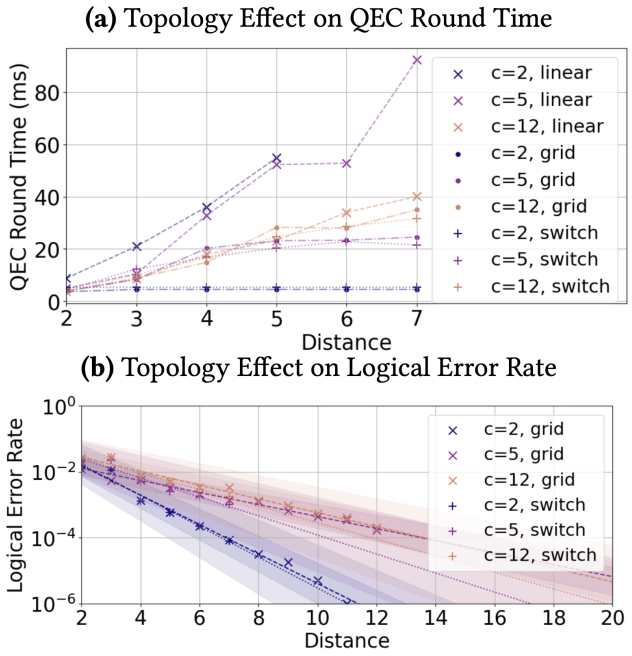

Figure 8. (a) Elapsed time per QEC round (y-axis) as a function of code distance (x-axis) for trap capacities 2, 5, and 12, under linear, grid, and all-to-all switch communication topologies. **(b)** Logical Error Rate as a function of code distance for trap capacities 2, 5, and 12 under the grid and all-to-all switch.

elapsed times across capacities due to routing congestion. For instance, $d = 5, C = 2$ requires over $\approx$ 275ms per logical identity operation for the linear topology, which is $\approx$ 12x greater than the switch and grid topologies. This is expected since a linear topology does not match the surface code's requirements. Second, the switch and grid topologies have approximately the same elapsed time. While this is expected for minimal trap capacity where the grid closely matches the surface code's needs, we may expect a switch topology to have a significant advantage for large capacities. This is not the case because operations within a trap get serialised, making it difficult to use the rich connectivity at high trap capacity. Third, only a trap capacity of two with grid or switch topology offers a constant elapsed time, independent of code distance. We discuss this aspect in the following subsection.

Figure 8(b) compares the logical error rate versus the code distance and the trap capacity for the grid and switch topologies. Although theoretically, the switch should outperform the grid due to lower contention across routing paths, the difference in logical error rate between the grid and the switch is minor and statistically inconclusive (overlapping error bars).

Our work validates that across trap capacities, the grid topology matches very closely the all-to-all switch both in terms of QEC round time and logical error rate,

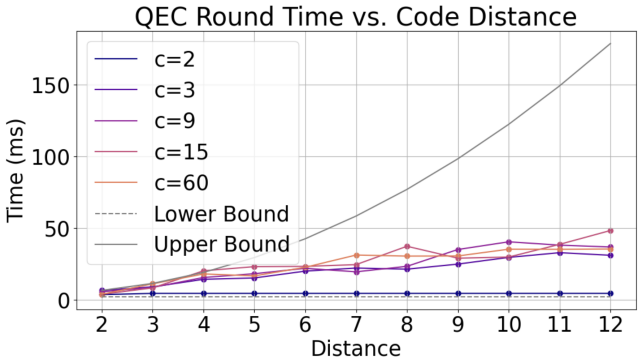

Figure 9. QEC shot time (y-axis) as a function of trap capacity (marked by the legend) and code distance (x-axis). The lower bound (grey dotted) corresponds to the minimal time required (2.5ms) for a single round of surface code parity-check operations when there are no ion reconfigurations, and there is complete parallelism. The upper bound represents the elapsed time when all ions are in the same trap, causing complete serialisation.

making it an ideal choice for hardware implementation. In the following experiments, we use the grid topology.

7.3 Choice of Trap Capacity

Impact on elapsed time: Figure 9 shows the elapsed time for different trap capacities and code distances. A trap capacity of two offers lower elapsed times than higher capacities. These elapsed times are also close to the theoretical lower bound. This is surprising because a capacity of two incurs the maximum number of communication operations; a larger trap capacity reduces the need for reconfiguring ions, as ancilla qubits are more likely to be located with their data qubits. However, using a capacity of two maximises the number of gates that can be executed in parallel; a larger capacity serialises more operations within a trap. Our work shows that maximising parallelism is more important for efficiently mapping surface codes onto QCCD systems and offering the best runtimes for large-scale applications that may use millions of QEC rounds.

Further, a trap capacity of two also offers constant cycle time irrespective of code distance, whereas higher capacities see cycle times grow with code distance. Although this was not a design goal, constant cycle time is an elegant architectural design point that mirrors the fixed cycle time of classical processors. Having this parameter independent of the error correction parameters and application demands will benefit abstraction and predictable system performance in the long term. Importantly, a trap capacity of two does not trade performance for consistency; it also achieves the lowest logical error rates (Figure 10).

Impact on logical error rate: Figure 10 evaluates the effect of trap capacity on the logical error rate of the surface

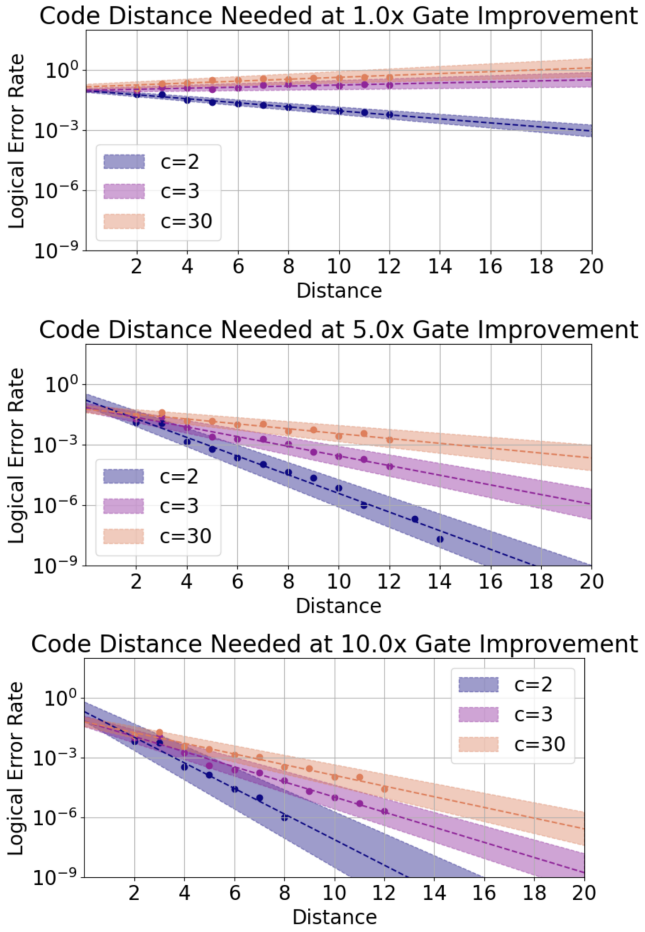

Figure 10. Projections of logical error rate versus code distance for the surface code on a QCCD grid topology at different levels of gate improvement. The target logical error rate of 10^{-9} is used to assess practical feasibility, with the x-axis intercept indicating the code distance required to achieve this target. The three axes show projections for 1X, 5X and 10X gate improvements, respectively.

code. We use three physical gate improvement scenarios, with 1X corresponding to pessimistic scaling of current systems, 5X corresponding to optimistic scaling of current systems, and 10X corresponding to a future improved system. Across gate improvement scenarios, a trap capacity of two outperforms higher capacities by one to two orders of magnitude in logical error rate. This is because a parallel system with very small traps can better localise error propagation and keep gate error rates well below the code threshold (§2), enabling the exponential logical error rate suppression. Even with future improvements in physical gates, a trap capacity of two remains an excellent choice for logical qubit design on QCCD systems.

Further, early scientific applications are expected to require at least a logical error rate of 10^{-9} to offer advantages

over classical computing. From Figure 10, it is clear that to achieve a low logical error rate we can either implement high code distances (increasing the number of physical qubits per logical qubit) or improve the physical gates. Trap capacity of two paired with a code distance of 13, with a 10X improvement in physical gate quality, is a feasible design point for quantum advantage experiments. If a 10X physical improvement proves infeasible in the coming years, increasing the code distance to 18 would offer the same logical qubit quality.

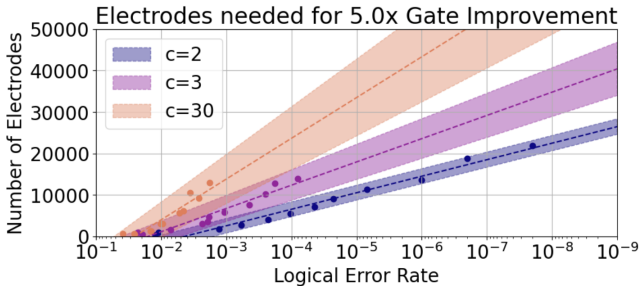

Figure 11. Projected number of electrodes required to achieve a target logical error rate under a 5x gate improvement scenario for different trap capacities.

Impact on hardware footprint: Figure 11 shows the number of electrodes required to implement a QCCD device across different trap capacities. The number of electrodes is an important indicator of the hardware cost (§5.2). Our results show that all trap capacities are expensive from a hardware perspective under the standard control wiring scheme, but **trap capacity two is the most hardware-efficient design point**, reducing the electrode counts needed to achieve a given logical error rate by several orders of magnitude compared to higher trap capacities. This is surprising because junctions in a QCCD system require 2X electrodes compared to traps. Therefore, as the trap capacity increases, the number of junctions needed in the design decreases. A design with a higher capacity is expected to offer lower electrode counts when viewed purely from a hardware perspective. However, when viewed from the standpoint of implementing logical qubits, increasing the trap capacity leads to worse logical error rates (Figure 10). In turn, a given logical error rate requirement necessitates the use of logical qubits with higher code distances, which increases the overall physical qubit count and the number of junctions and traps and, therefore, requires large electrode counts.

Unlike prior NISQ studies, which recommend the use of traps with capacity in the range of 20-30 ions [26], we advocate the use of a trap capacity of two to obtain logical qubits with hardware efficiency, low error rates, and a constant runtime regardless of code distance.

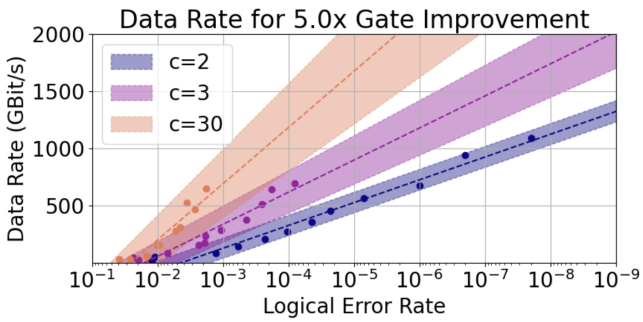

Figure 12. Hardware requirements for achieving a target logical error rate under a 5x gate improvement scenario across different trap capacities (c). The axis shows the required data rate between the QPU and the controller. A trap capacity of $c = 2$ minimises both power dissipation and data rate demands at a logical error rate of 10^{-9}. However, even in this optimal case, achieving 10^{-9} necessitates an impractical 1.3 Tbit/s communication link and $\approx$ 780 W of power dissipation.

7.4 Choice of wiring method

At a trap capacity of two, with every $\approx$ 5,000 additional electrodes, we obtain an $\approx$ 10X decrease in logical error rate. Although this represents the best scaling observed, it remains far from practical. Figure 12 confirms that the data rate and power requirements for a standard QCCD architecture quickly reach impractical levels as the system scales. In particular, a single logical qubit with an error rate of 10^{-9} demands a power consumption of more than 780 Watts. A system with a few thousand logical qubits and much lesser logical error rates is required for practical quantum applications and may lead to trapped-ion systems requiring tens to hundreds of megawatts of power per system.

A key power bottleneck in the standard architecture is that each electrode is wired to a separate DAC. WISE [22] overcomes this with a more intelligent wiring mechanism, trading off execution time for reduced power consumption. *Which mechanism is the most suitable for logical qubit implementation?* Figure 13(a) compares the data required for WISE and the standard wiring mechanism. For the standard mechanism, we only use trap capacity 2. Whereas, for WISE, we examine trap capacities ranging from 2 to 30 but only show the curves for three capacities, since the trends are similar at other capacities. Compared to the standard architecture, WISE achieves an improvement of more than two orders of magnitude in data rate (and, therefore, in power consumption).

WISE requires cooling support from the hardware to reduce physical noise (our simulations with no cooling for WISE indicated that it could not scale beyond a logical error rate of 10^{-4} without). As a result, contrary to the standard architecture, trap capacity two is not more hardware-efficient

than other trap capacities in the WISE architecture. However, smaller traps still achieve the lowest QEC round times while maintaining modest data rate requirements. **In both control systems, designing traps to be as small as possible remains optimal for surface code implementation.**

Figure 13(b) compares the elapsed time at different logical error rates. For the WISE architecture, the elapsed time scales in proportion to the desired logical error rate. For every 10X improvement desired in the logical error rate, the elapsed time increases by 1.17X. WISE suffers from limited transport flexibility, allowing only one transport operation at a time. Under an odd-even sort global reconfiguration scheme [22], this limitation results in logical clock speeds that are up to 25X longer than those of standard QPUs, for logical error rates near 10^{-9}. This runtime increase is acceptable for near-term fault-tolerant applications such as quantum dynamics [34]. However, for large applications such as factoring, which already require month-long computations on trapped ion systems[20], such a runtime increase will lead to impractical executions that run over a year.

Therefore, **we observe a power vs. cycle time trade-off in current wiring mechanisms for QCCD trapped ion systems.** Multiplexed wiring mechanisms lead to low power but very long execution times, while direct wiring of DACs to electrodes offers low execution times with high power consumption. For scaling trapped ions to the regime of several hundred logical qubits, we need to go beyond existing control system designs. We require novel architectures that offer high-performance executions with low power needs.

8 Related Work

This work builds on previous advancements in QCCD system architecture and QEC optimisation. For instance, Gutiérrez et al. [13] inspire the test infrastructure to validate executable QCCD circuits. The relevance of compiler-driven architectural co-design for QCCD systems is demonstrated by Murali et al. [26], which examines the influence of micro-architectural choices on the performance of NISQ algorithms. Similarly, Wu et al. [36] address the challenges in bridging quantum hardware and QEC codes by proposing a framework for efficient implementation and optimisation of surface codes for superconducting architectures. This study extends these concepts by tailoring a QEC compiler to the specific demands of QCCD-based systems, aiming to provide a systematic approach to co-designing hardware and software for fault-tolerant quantum computing.

While there exist QCCD compilers for QEC other than the two benchmarked in (§7.1), such as the MQTIonShuttler [31], we do not benchmark our compiler against these, since they assume distinct memory and processing zones in their QCCD architecture, which is not suitable for surface code implementation. TISCC [19] fixes the trap capacity as two and the standard grid topology [20], then compiles and simulates

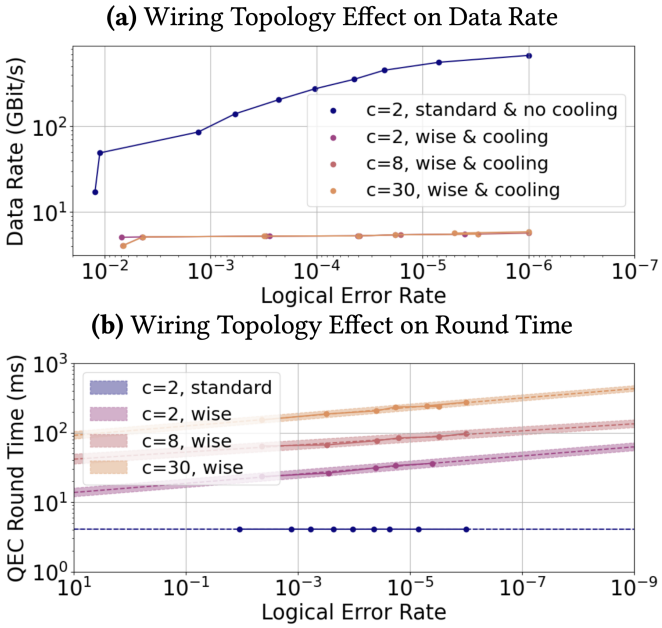

Figure 13. (a) Data rate comparison between the standard architecture without cooling and WISE architecture with cooling, under a 5X gate improvement. Cooling improves data rate scaling across all trap capacities for the WISE architecture, allowing low logical error rates at modest data rate requirements compared to standard capacity-2 systems. **(b)** Elapsed QEC shot time versus target logical error rate under a 5X gate improvement. In the WISE architecture with cooling, logical scale quadratically with code distance, leading to a logical clock speed of $\approx 10^{-1}$ operations per second for a 10^{-9} target error rate. In contrast, the standard, no cooling, trap capacity two architecture exhibits linear scaling of cycle times with increasing code distance.

high-level logical circuits into a quantum circuit on physical qubits using the surface code. The compiler does not map to primitive QCCD directly but uses the performance models of these primitives for resource estimation.

Consideration of Limiting Factors: In contrast to superconducting platforms, decoder runtimes are not the limiting factor for ion-trap systems since their cycle time is considerably longer. Specialist hardware is already available for the fast decoding of surface codes up to a distance of 8 [4].

We recognise that there are other architectural challenges not addressed: integrating many logical qubits in monolithic QCCD systems (since such scaling will require networking between multiple ion-trap systems), general noise inhomogeneity across the ion chain, and universal gate set implementation. However, if lattice surgery is used to perform entanglement between logical qubits, only boundary qubits of the two logical qubits will need to participate in such circuits, leaving the bulk of the surface code intact. Since the quantum circuits from lattice surgery are very similar in

structure to the circuits within one surface code qubit, we expect our results to hold.

9 Conclusion

TI qubit technology is at the threshold of supporting systems with several logical qubits. Current demonstrations of logical qubits are limited to small systems of less than 60 physical qubits. To scale up to systems with several hundred physical qubits (tens of logical qubits), we need to understand what the right trap capacities and topologies are and how control systems must be designed to support QEC workloads. The TI community has been exploring these choices for several years, with 1) monolithic, large trap capacity devices (e.g., IonQ Forte) 2) QCCD devices with small trap capacities (e.g., Quantinuum H2) 3) architecture research showing the value of QCCD systems with 15-25 ions per trap [26] and 4) other manual design efforts [20, 22, 33].

We conduct a systematic architectural design exploration for implementing logical qubits on TI systems. Unlike prior studies, our work shows the value of using a trap capacity of two to obtain high-performance, hardware-efficient, low error rate logical qubits with a constant runtime irrespective of QEC code distance. Our work also shows the importance of co-designing control architectures with QEC needs.

To scale TI systems to the sizes required for practical quantum advantage, our architectural guidance and toolflow are likely to be very important.

References

[1] Rajeev Acharya, Igor Aleiner, Richard Allen, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Juan Atalaya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Joao Basso, Andreas Bengtsson, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Dripto M. Debroy, Alexander Del Toro Barba, Sean Demura, Andrew Dunsworth, Daniel Eppens, Catherine Erickson, Lara Faoro, Edward Farhi, Reza Fatemi, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, William Giang, Craig Gidney, Dar Gilboa, Marissa Giustina, Alejandro Grajales Dau, Jonathan A. Gross, Steve Habegger, Michael C. Hamilton, Matthew P. Harrigan, Sean D. Harrington, Oscar Higgott, Jeremy Hilton, Markus Hoffmann, Sabrina Hong, Trent Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Pavol Juhas, Dvir Kafri, Kostyantyn Kechedzhi, Julian Kelly, Tanuj Khattar, Mostafa Khezri, Mária Kieferová, Seon Kim, Alexei Kitaev, Paul V. Klimov, Andrey R. Klots, Alexander N. Korotkov, Fedor Kostritsa, John Mark Kreikebaum, David Landhuis, Pavel Laptev, Kim-Ming Lau, Lily Laws, Joonho Lee, Kenny Lee, Brian J. Lester, Alexander Lill, Wayne Liu, Aditya Locharla, Erik Lucero, Fionn D. Malone, Jeffrey Marshall, Orion Martin, Jarrod R. McClean, Trevor McCourt, Matt McEwen, Anthony Megrant, Bernardo Meurer Costa, Xiao Mi, Kevin C. Miao, Masoud Mohseni, Shirin Montazeri, Alexis Morvan, Emily Mount, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Murphy Yuezhen Niu, Thomas E. O'Brien, Alex Opremcak, John Platt, Andre Petukhov, Rebecca Potter, Leonid P. Pryadko, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Michael J. Shearn, Aaron Shorter, Vladimir Shvarts, Jindra Skruzny, Vadim Smelyanskiy, W. Clarke Smith, George Sterling, Doug Strain, Marco Szalay, Alfredo Torres, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Theodore White, Cheng Xing, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Grayson Young, Adam Zalcman, Yaxing Zhang, Ningfeng Zhu, and Google Quantum AI. 2023. Suppressing quantum errors by scaling a surface code logical qubit. *Nature* 614, 7949 (2023), 676–681.

[2] Google Quantum AI. 2025. Google Quantum AI Roadmap. https://quantumai.google/roadmap

[3] Konstantin Andreev and Harald Räcke. 2006. Balanced graph partitioning. *Theory of Computing Systems* 39, 6 (Nov. 2006), 929–939. https://doi.org/10.1007/s00224-006-1350-7 Funding Information: This work was supported by the NSF under Grants CCR-0085982 and CCR-0122581*..

[4] Ben Barber, Kenton M. Barnes, Tomasz Bialas, Okan Buğdaycı, Earl T. Campbell, Neil I. Gillespie, Kauser Johar, Ram Rajan, Adam W. Richardson, Luka Skoric, Canberk Topal, Mark L. Turner, and Abbas B. Ziad. 2025. A real-time, scalable, fast and resource-efficient decoder for a quantum computer. *Nature Electronics* 8, 1 (Jan. 2025), 84–91. https://doi.org/10.1038/s41928-024-01319-5

[5] R. Bowler, J. Gaebler, Y. Lin, T. R. Tan, D. Hanneke, J. D. Jost, J. P. Home, D. Leibfried, and D. J. Wineland. 2012. Coherent Diabatic Ion Transport and Separation in a Multizone Trap Array. *Phys. Rev. Lett.* 109 (Aug 2012), 080502. Issue 8. https://doi.org/10.1103/PhysRevLett.109.080502

[6] William Cody Burton, Brian Estey, Ian M. Hoffman, Abigail R. Perry, Curtis Volin, and Gabriel Price. 2023. Transport of Multispecies Ion Crystals through a Junction in a Radio-Frequency Paul Trap. *Physical Review Letters* 130, 17 (April 2023). https://doi.org/10.1103/physrevlett.130.173202

[7] Jwo-Sy Chen, Erik Nielsen, Matthew Ebert, Volkan Inlek, Kenneth Wright, Vandiver Chaplin, Andrii Maksymov, Eduardo Páez, Amrit Poudel, Peter Maunz, and John Gamble. 2024. Benchmarking a trapped-ion quantum computer with 30 qubits. *Quantum* 8 (Nov. 2024), 1516. https://doi.org/10.22331/q-2024-11-07-1516

[8] Caroline Figgatt. 2018. Building and Programming a Universal Ion Trap Quantum Computer. (2018).

[9] C. Figgatt, A. Ostrander, N. M. Linke, K. A. Landsman, D. Zhu, D. Maslov, and C. Monroe. 2019. Parallel entangling operations on a universal ion-trap quantum computer. *Nature* 572, 7769 (July 2019), 368–372. https://doi.org/10.1038/s41586-019-1427-5

[10] Austin G. Fowler and Craig Gidney. 2018. Low overhead quantum computation using lattice surgery. *arXiv: Quantum Physics* (2018). https://api.semanticscholar.org/CorpusID:119447706

[11] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W.H. Freeman and Co., San Francisco, CA.

[12] Craig Gidney. 2021. Stim: a fast stabilizer circuit simulator. *Quantum* 5 (July 2021), 497. https://doi.org/10.22331/q-2021-07-06-497

[13] M. Gutiérrez, M. Müller, and A. Bermúdez. 2019. Transversality and lattice surgery: Exploring realistic routes toward coupled logical qubits with trapped-ion quantum processors. *Phys. Rev. A* 99 (Feb 2019), 022330. Issue 2. https://doi.org/10.1103/PhysRevA.99.022330

[14] IBM. 2025. IBM Quantum Roadmap. https://www.ibm.com/roadmaps/quantum/

[15] Oxford Ionics. 2025. Oxford Ionics | High Performance Quantum Computing. https://www.oxionics.com/

[16] IonQ. 2025. IonQ | Trapped Ion Quantum Computing. https://ionq.com/

[17] D. Kielpinski, C. Monroe, and D. J. Wineland. 2002. Architecture for a large-scale ion-trap quantum computer. *Nature* 417, 6890 (01 Jun

2002), 709–711. https://doi.org/10.1038/nature00784

[18] Harold W. Kuhn. 1955. The Hungarian Method for the Assignment Problem. *Naval Research Logistics Quarterly* 2, 1–2 (March 1955), 83–97. https://doi.org/10.1002/nav.3800020109

[19] Tyler Leblond, Ryan S. Bennink, Justin G. Lietz, and Christopher M. Seck. 2023. TISCC: A Surface Code Compiler and Resource Estimator for Trapped-Ion Processors. In *Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023)*. ACM, 1426–1435. https://doi.org/10.1145/3624062.3624214

[20] Bjoern Lekitsch, Sebastian Weidt, Austin G. Fowler, Klaus Mølmer, Simon J. Devitt, Christof Wunderlich, and Winfried K. Hensinger. 2017. Blueprint for a microwave trapped ion quantum computer. *Science Advances* 3, 2 (Feb. 2017). https://doi.org/10.1126/sciadv.1601540

[21] Pak Hong Leung and Kenneth R. Brown. 2018. Entangling an arbitrary pair of qubits in a long ion crystal. *Phys. Rev. A* 98 (Sep 2018), 032318. Issue 3. https://doi.org/10.1103/PhysRevA.98.032318

[22] M. Malinowski, D.T.C. Allcock, and C.J. Ballance. 2023. How to Wire a 1000-Qubit Trapped-Ion Quantum Computer. *PRX Quantum* 4 (Oct 2023), 040313. Issue 4. https://doi.org/10.1103/PRXQuantum.4.040313

[23] C. Monroe, R. Raussendorf, A. Ruthven, K. R. Brown, P. Maunz, L.-M. Duan, and J. Kim. 2014. Large-scale modular quantum-computer architecture with atomic memory and photonic interconnects. *Phys. Rev. A* 89 (Feb 2014), 022317. Issue 2. https://doi.org/10.1103/PhysRevA.89.022317

[24] S. A. Moses, C. H. Baldwin, M. S. Allman, R. Ancona, L. Ascarrunz, C. Barnes, J. Bartolotta, B. Bjork, P. Blanchard, M. Bohn, J. G. Bohnet, N. C. Brown, N. Q. Burdick, W. C. Burton, S. L. Campbell, J. P. Campora, C. Carron, J. Chambers, J. W. Chan, Y. H. Chen, A. Chernoguzov, E. Chertkov, J. Colina, J. P. Curtis, R. Daniel, M. DeCross, D. Deen, C. Delaney, J. M. Dreiling, C. T. Ertsgaard, J. Esposito, B. Estey, M. Fabrikant, C. Figgatt, C. Foltz, M. Foss-Feig, D. Francois, J. P. Gaebler, T. M. Gatterman, C. N. Gilbreth, J. Giles, E. Glynn, A. Hall, A. M. Hankin, A. Hansen, D. Hayes, B. Higashi, I. M. Hoffman, B. Horning, J. J. Hout, R. Jacobs, J. Johansen, L. Jones, J. Karcz, T. Klein, P. Lauria, P. Lee, D. Liefer, S. T. Lu, D. Lucchetti, C. Lytle, A. Malm, M. Matheny, B. Mathewson, K. Mayer, D. B. Miller, M. Mills, B. Neyenhuis, L. Nugent, S. Olson, J. Parks, G. N. Price, Z. Price, M. Pugh, A. Ransford, A. P. Reed, C. Roman, M. Rowe, C. Ryan-Anderson, S. Sanders, J. Sedlacek, P. Shevchuk, P. Siegfried, T. Skripka, B. Spaun, R. T. Sprenkle, R. P. Stutz, M. Swallows, R. I. Tobey, A. Tran, T. Tran, E. Vogt, C. Volin, J. Walker, A. M. Zolot, and J. M. Pino. 2023. A Race-Track Trapped-Ion Quantum Processor. *Phys. Rev. X* 13 (Dec 2023), 041052. Issue 4. https://doi.org/10.1103/PhysRevX.13.041052

[25] S. A. Moses, C. H. Baldwin, M. S. Allman, R. Ancona, L. Ascarrunz, C. Barnes, J. Bartolotta, B. Bjork, P. Blanchard, M. Bohn, J. G. Bohnet, N. C. Brown, N. Q. Burdick, W. C. Burton, S. L. Campbell, J. P. Campora, C. Carron, J. Chambers, J. W. Chan, Y. H. Chen, A. Chernoguzov, E. Chertkov, J. Colina, J. P. Curtis, R. Daniel, M. DeCross, D. Deen, C. Delaney, J. M. Dreiling, C. T. Ertsgaard, J. Esposito, B. Estey, M. Fabrikant, C. Figgatt, C. Foltz, M. Foss-Feig, D. Francois, J. P. Gaebler, T. M. Gatterman, C. N. Gilbreth, J. Giles, E. Glynn, A. Hall, A. M. Hankin, A. Hansen, D. Hayes, B. Higashi, I. M. Hoffman, B. Horning, J. J. Hout, R. Jacobs, J. Johansen, L. Jones, J. Karcz, T. Klein, P. Lauria, P. Lee, D. Liefer, S. T. Lu, D. Lucchetti, C. Lytle, A. Malm, M. Matheny, B. Mathewson, K. Mayer, D. B. Miller, M. Mills, B. Neyenhuis, L. Nugent, S. Olson, J. Parks, G. N. Price, Z. Price, M. Pugh, A. Ransford, A. P. Reed, C. Roman, M. Rowe, C. Ryan-Anderson, S. Sanders, J. Sedlacek, P. Shevchuk, P. Siegfried, T. Skripka, B. Spaun, R. T. Sprenkle, R. P. Stutz, M. Swallows, R. I. Tobey, A. Tran, T. Tran, E. Vogt, C. Volin, J. Walker, A. M. Zolot, and J. M. Pino. 2023. A Race-Track Trapped-Ion Quantum Processor. *Physical Review X* 13, 4 (Dec. 2023). https://doi.org/10.1103/physrevx.13.041052

[26] Prakash Murali, Dripto M. Debroy, Kenneth R. Brown, and Margaret Martonosi. 2020. Architecting Noisy Intermediate-Scale Trapped Ion Quantum Computers. arXiv:2004.04706 [quant-ph] https://arxiv.org/abs/2004.04706

[27] Quantum Optics and Spectroscopy Institut für Experimentalphysik Universität Innsbruck. 2023. World list of QIP ion trapping groups. https://www.quantumoptics.at/images/miscellaneous/IonTrappers.pdf

[28] J. M. Pino, J. M. Dreiling, C. Figgatt, J. P. Gaebler, S. A. Moses, M. S. Allman, C. H. Baldwin, M. Foss-Feig, D. Hayes, K. Mayer, C. Ryan-Anderson, and B. Neyenhuis. 2021. Demonstration of the trapped-ion quantum CCD computer architecture. *Nature* 592, 7853 (April 2021), 209–213. https://doi.org/10.1038/s41586-021-03318-4

[29] Quantinuum. 2025. Quantinuum. https://www.quantinuum.com/

[30] Abdullah Ash Saki, Rasit Onur Topaloglu, and Swaroop Ghosh. 2022. Muzzle the Shuttle: Efficient Compilation for Multi-Trap Trapped-Ion Quantum Computers. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. 322–327. https://doi.org/10.23919/DATE54114.2022.9774619

[31] Daniel Schoenberger, Stefan Hillmich, Matthias Brandl, and Robert Wille. 2024. Shuttling for Scalable Trapped-Ion Quantum Computers. arXiv:2402.14065 [quant-ph] https://arxiv.org/abs/2402.14065

[32] Seyon Sivarajah, Silas Dilkes, Alexander Cowtan, Will Simmons, Alec Edgington, and Ross Duncan. 2020. t|ket⟩: a retargetable compiler for NISQ devices. *Quantum Science and Technology* 6, 1 (Nov. 2020), 014003. https://doi.org/10.1088/2058-9565/ab8e92

[33] Marco Valentini, Martin W. van Mourik, Friederike Butt, Jakob Wahl, Matthias Dietl, Michael Pfeifer, Fabian Anmasser, Yves Colombe, Clemens Rössler, Philip Holz, Rainer Blatt, Markus Müller, Thomas Monz, and Philipp Schindler. 2024. Demonstration of two-dimensional connectivity for a scalable error-corrected ion-trap quantum processor architecture. arXiv:2406.02406 [quant-ph] https://arxiv.org/abs/2406.02406

[34] Wim van Dam, Mariia Mykhailova, and Mathias Soeken. 2024. Using Azure Quantum Resource Estimator for Assessing Performance of Fault Tolerant Quantum Computation. arXiv:2311.05801 [quant-ph] https://arxiv.org/abs/2311.05801

[35] A. Walther, F. Ziesel, T. Ruster, S. T. Dawkins, K. Ott, M. Hettrich, K. Singer, F. Schmidt-Kaler, and U. Poschinger. 2012. Controlling Fast Transport of Cold Trapped Ions. *Phys. Rev. Lett.* 109 (Aug 2012), 080501. Issue 8. https://doi.org/10.1103/PhysRevLett.109.080501

[36] Anbang Wu, Gushu Li, Hezi Zhang, Gian Giacomo Guerreschi, Yufei Ding, and Yuan Xie. 2022. A synthesis framework for stitching surface code with superconducting quantum devices. In *Proceedings of the 49th Annual International Symposium on Computer Architecture* (New York, New York) *(ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 337–350. https://doi.org/10.1145/3470496.3527381