Using Boolean Satisfiability for Exact Shuttling in Trapped-Ion Quantum Computers

Daniel Schoenberger[1] Stefan Hillmich[2] Matthias Brandl[3] Robert Wille[1,2]

[1] Chair for Design Automation, Technical University of Munich, Germany
[2] Software Competence Center Hagenberg GmbH, Austria
[3] Infineon Technologies AG, Germany
daniel.schoenberger@tum.de, stefan.hillmich@scch.at, matthias.brandl@infineon.com, robert.wille@tum.de
https://www.cda.cit.tum.de/research/quantum/

***Abstract*—Trapped ions are a promising technology for building scalable quantum computers. Not only can they provide a high qubit quality, but they also enable modular architectures, referred to as *Quantum Charge Coupled Device* (QCCD) architecture. Within these devices, ions can be shuttled (moved) throughout the trap and through different dedicated zones, e.g., a memory zone for storage and a processing zone for the actual computation. However, this movement incurs a cost in terms of required time steps, which increases the probability of decoherence, and, thus, should be minimized. In this paper, we propose a formalization of the possible movements in ion traps via Boolean satisfiability. This formalization allows for determining the *minimal* number of time steps needed for a given quantum algorithm and device architecture, hence reducing the decoherence probability. An empirical evaluation confirms that—using the proposed approach—minimal results (i.e., the lower bound) can be determined for the first time. An open-source implementation of the proposed approach is publicly available at https://github.com/cda-tum/mqt-ion-shuttler.**

***Index Terms*—quantum computing, trapped-ions, shuttling**

I. Introduction

Quantum computing as a new paradigm promises to solve certain problems which are computationally intractable on classical computers. Famous examples include Shor's algorithm to factorize integers [1] and Grover's search for unstructured data [2], but also problems in quantum chemistry can profit from simulation in an actual quantum computer [3]. The advantage of quantum computing lies in the exploitation of quantum mechanical effects [4], such as *superposition*, where a quantum state can assume a combination of basis states, and *entanglement*, where qubits can lose their locality and can no longer be described individually. This potential also led companies to heavily invest in research towards quantum computing, as witnessed by IBM, Alphabet (Google), Microsoft, Rigetti, AQT, Infineon Technologies, and IonQ.

Among the possible physical technologies, such as superconducting quantum computers [5], neutral atom quantum computers [6], [7], and optical quantum computers [8], trapped-ion quantum computers are one of the most promising candidates to show quantum advantage in the foreseeable future [9]. This is due to their ability to scale from smaller modules in an approach termed *Quantum Charge Coupled Device* (QCCD, [10], [11]). In this architecture, the qubits are encoded into ions that are trapped by electromagnetic fields. By manipulating these fields, the ions can be moved on the architecture.

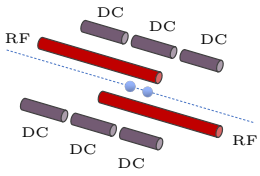
However, the scaling of trapped-ion quantum computers requires corresponding tooling support to exploit the full potential. Without proper support, there is the possibility that powerful trapped-ion quantum computers will be available but there will be no means to use that power. Indeed, this holds true for all quantum computing technologies. For ion traps in particular, efficiently moving (i.e., *shuttling*) the ions on a QCCD architecture is essential because of the potential loss of quantum information over time due to decoherence. This makes determining exact schedules for the movement paramount for efficient computations in trapped-ion quantum computers.

First solutions addressing this problem have been proposed, e.g., in [12], [13]. However, the considered architectures do not cover a large part of possible QCCD architectures and all solutions rely on heuristics, hence, cannot guarantee exactness. Moreover, generating *exact* (i.e., minimal) solutions requires traversing a huge search space and, hence, constitutes a computationally expensive design task (a reason why previously proposed works resorted to heuristics in the first place). Boolean satisfiability, i.e., formulating the problem in a symbolic fashion as a *satisfiability* (SAT) problem and using dedicated solvers afterward, promises to be an efficient approach. In fact, through sophisticated reasoning strategies, modern SAT solvers manage to cope with huge search spaces as demonstrated in numerous (classical) design automation tasks (see e.g., [14]–[22]) and, recently, also got utilized for quantum computing [23]–[26]. In this paper, we are investigating whether this promising approach can also be utilized for determining an exact shuttling in trapped ion quantum computers.

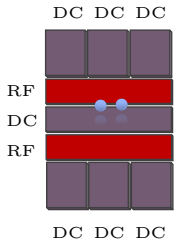
To this end, we propose a symbolic encoding of the shuttling problem that can be used to employ SAT solvers in order to determine exact solutions. More precisely, we first define an abstraction of general QCCD architectures as a graph that captures information about the system state that is necessary for solving the shuttling problem—most importantly the position of ion chains at given times. Based on this abstraction, we subsequently propose a symbolic encoding of the system state with Boolean variables. This encoding is then constrained to ensure (i) that only valid system states can be generated and (ii) that transitions between system states only include changes that are possible on the hardware. By employing a SAT solver afterward, we finally attain the desired exact solution.

Empirical evaluations confirm the efficacy of the proposed approach. For the first time, exact (i.e., minimal) solutions for the shuttling problem for trapped-ion quantum computers are determined. Although this is only possible for small architectures (after all, the problem remains computationally expensive), this provides the first lower bounds for this problem. Moreover, the obtained exact results allow for a more sophisticated analysis and design exploration of architectures for this promising technology. An open-source implementation of the proposed approach is publicly available at https://github.com/cda-tum/mqt-ion-shuttler.

The remainder of this paper is structured as follows: Section II provides background on trapped-ion quantum computers and QCCD architectures. Section III motivates the problem and outlines the proposed solution. Section IV details the symbolic state descriptions and the constraints to ensure only valid states. Section V summarizes the obtained results. Finally, Section VI concludes the paper.



(a) 3D Paul trap



(b) 2D surface trap

Fig. 1: An illustration of two possible linear trap realizations. The combination of radio-frequency (RF) and quasi-static (DC) electric fields produced by control electronics creates a potential that confines the ions (blue).

II. Background

In this section, we briefly review the trapped-ion quantum computing technology and the challenges that have to be addressed to realize scalable devices. This way, we motivate the considered problem that is abstracted and solved in the following sections. For a more detailed description, the interested reader is referred to the provided references.

A. Trapped-Ion Quantum Computing

The main idea of trapped-ion quantum computers [13], [27]–[29] is to use ions as entities for qubits whose states are manipulated by electromagnetic interactions in the optical or the microwave domain. To this end, ions are confined and suspended in free space using electromagnetic fields. A trap can hold a chain of multiple ions confined in an electric potential. The potential is created by a combination of radio-frequency and quasi-static electric fields produced by control electronics. The individual ion chains have been coined *ion registers*, because they may be used similarly to registers in classical computers.

Example 1. *Fig. 1 shows two realizations of a trap that holds a single chain. We refer to this as a* single linear trap*. In both realizations, the ions are held in an electric field generated by radio-frequency (RF, red) and quasi-static (DC, purple) elements. In Fig. 1a, the trap is constructed as a* 3D linear Paul trap *with 3D control electronics. The same combination of control electronics can also be combined into a two-dimensional* surface trap*, as depicted in Fig. 1b.*

While trapping all ions in one trap is feasible for small quantum computers, physical limitations prevent this approach from scaling to larger qubit numbers that are needed for practical quantum algorithms [11]. Increasing the number N of ions in a trap leads to slower gate speed R_{gate} which approximates to $R_{\text{gate}} \sim \frac{1}{\sqrt{N}}$. Longer gate times give rise to different types of background errors—limiting the size of a practical single trap. As of now, trapped-ion quantum computers have been realized using up to tens of qubits [30].

B. Quantum Charge Coupled Device Architecture

These limitations can be addressed by building modular systems, which can hold multiple ion chains. A promising modular approach is called the *Quantum Charge Coupled Device* (QCCD) architecture [13]. The QCCD approach proposes to exploit the fact that ion chains can be split, merged, and moved through the system by applying different operating voltages to the control electrodes. The key idea of the QCCD approach is then to assign certain trap regions certain tasks. For example, ion chains that store quantum information may be placed into a zone dedicated to *memory*, where

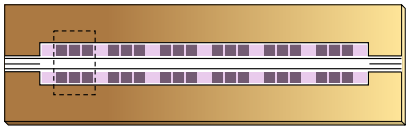
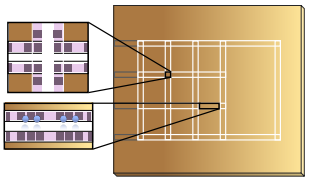
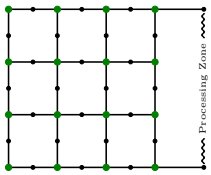


Fig. 2: An illustration of a linear QCCD device. In this example, each site (indicated by a dotted line) consists of six control electrodes (purple). At every site exactly one ion chain can be trapped between the electrodes similar to the surface trap in Fig. 1b.



(a) QCCD architecture



(b) Corresponding Graph

Fig. 3: An illustration of a 2D QCCD architecture and its corresponding graph

ions are shielded from potential sources of decoherence. All quantum operations are then performed in a separated *processing* zone that is specifically tailored toward efficient qubit interactions. A complete QCCD device may also include optimized regions like *measurement* zones for the readout of qubit information and *loading* zones for initializing new ions. In its simplest form, a QCCD device is constructed in a straight line to form a linear system. At every site, the system can trap one chain. A linear QCCD trap has been successfully realized in [11], [29].

Example 2. *Fig. 2 shows a schematic linear QCCD device. Three control electrodes (purple) combined on each side of the horizontal axis represent one site of the linear trap. The ion registers can be held in between the control electronics and shuttle to neighboring sites.*

In a linear trap, ion chains cannot move out of each other's way, thus slow interactions like chain reordering and reconfiguration are needed to address all ions. To tackle this problem, linear regions can be connected via junctions to form two-dimensional (2D) architectures. If an ion chain blocks the way of its neighbor chain, it can use a junction to simply move out of the way. 2D QCCD architectures can be constructed in a variety of ways. One may use only small linear regions and a large number of junctions, or instead, reduce the number of junctions and employ larger linear regions.

Example 3. *Fig. 3a shows one possible layout of a 2D QCCD device. Inner linear regions are connected via "X"-junctions, which produce a system in a grid structure. Each linear region can hold up to two chains. In this architecture, the square grid on the left-hand side is dedicated as a memory zone which is connected to a linear processing zone on the right-hand side.*

To execute a quantum algorithm on a QCCD device, the ions have to be shuttled to the correct zone at the right time. A quantum circuit determines in which order ions have to be moved from the memory zone to the processing zone. After they have been processed in the processing zone, the ions move back to the memory zone to store their acquired quantum information.

C. Shuttling Costs

Conducting the shuttling operations of ion chains in a QCCD architecture comes with certain costs. Depending on the architecture,

different amounts of energy have to be invested into the system to trap and move the ions. The more the ions move through the system, the more they collect energy through the acquisition of phonons [31]. To avoid losing quantum information, ions must be cooled to or close to the ground state—often by a combination of Doppler and side-band cooling (see [9] for more information). Integrating and scaling the required optical control elements still proves to be difficult. It is therefore crucial to find exact shuttling schemes that minimize the time ions have to be moved through the system.

III. Exact Shuttling in QCCD Architectures

As described above, the execution time of a given algorithm is dictated by the time that the respective ion chains need to move from the memory zone to the processing zone and the required movement inside the memory zone to give way for other ion chains. Since the possible movement inside the processing zone is immediately given by the architecture, we put the focus of this paper on the movement inside the memory zone and at the interface to the processing zone. In this section, we formulate these considerations as a discrete problem set, by representing the architecture of a memory zone and the interface to the processing zone in a QCCD device as an undirected graph. Time can then be discretized into time steps, where at every time step, one can describe the location of the ion chains on the graph by means of Boolean variables in a symbolic fashion. Based on this representation, we propose an encoding as a Boolean satisfiability problem to determine an exact, i.e., minimal, solution.

A. Considered Problem

The problem we are considering in this paper is determining an exact way of shuttling in a QCCD architecture given a quantum circuit. As discussed before, the ion chains can not be split up or directly swap places within a memory zone, therefore, each chain is abstracted as one individual entity moving on a graph. Further, since movement outside the memory zone only allows very limited choices, we focus our investigations on the memory zone. In the following, the abstraction of a QCCD architecture is referred to as a *layout*.

Definition 1. *Consider a graph $G = (V, E)$ that represents the architecture of the memory zone and the interface to the processing zone on the QCCD device.*

The set of nodes V contains two different types: major nodes *representing junctions and* minor nodes *separating adjacent traps.*

The set $E = \{e_0, \dots, e_k\}$ denotes the edges of the graph, representing all possible positions of the ion chains. There are two special edges: The (i) outbound edge *for ion chains exiting the memory zone for processing in the processing zone and the (ii)* inbound edge *for ion chains returning to the memory zone. Edges can connect two major nodes (when there is only one site between junctions), a major and a minor node, as well as two minor nodes.*

In addition to the fixed architecture, there is also the set of ion chains $C = \{i_0, \cdots, i_l\}$ which contains all ion chains considered in the given shuttling problem. One ion chain can contain multiple individual ions, where each ion carries the information for one qubit.

Example 4. *Consider the QCCD architecture shown in Fig. 3a. All regions between two junctions in the memory zone can hold up to two individual ion chains. The corresponding graph is illustrated in Fig. 3b where the major nodes (green) form a grid of size 4×4. Minor nodes (black) separate the two individual sites between two junctions. The edge of the processing zone is labeled and connected via one inbound and one outbound edge to the memory zone.*

On this graph-based abstraction, we can describe the state of the memory zone at any time. Discretizing the time into individual *time steps* enables the inclusion of movement in the description without having to include hardware-dependent information. Each time step represents the system at a given time and, between time steps, ion chains can move through junctions. More precisely, per transition between time steps, at most one ion chain may pass through a given junction as long as there is an available site it can move into "after" the junction.

While the underlying architecture constrains the possible ways an ion chain may move, the considered quantum algorithm dictates which ion chains have to be at what position at what time. More precisely, since the considered quantum algorithm defines the sequence of gates and, hence, the sequence of qubits needed in the processing zone, the corresponding ion chains including those qubits have to move to the outbound edge and, after processing, return to the inbound edge and back into the memory zone. This can be represented by a *sequence* S of ion chains derived from the respectively considered quantum algorithm.

Based on all that, the overall problem is defined as follows:

- **Given:** A graph layout G of a memory zone with a one-way connection to a processing zone and a quantum circuit as a sequence S.
- **Goal:** Determine the minimum amount of time steps $\hat{T}$ for shuttling to execute the circuit.

B. General Idea

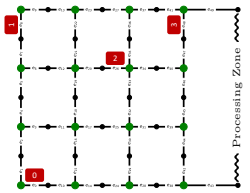
The problem defined above, namely determining an exact shuttling schedule for trapped-ion quantum computers, is an optimization problem that yields significant computational complexity. At the same time, it constitutes a classical combinatorial optimization problem. In the past, SAT solvers have been proven very beneficial and efficient in tackling this complexity. Accordingly, in this work, we propose to exploit this power and, by this, provide an example of how quantum computing can benefit from the potential of SAT solving. To this end, we briefly review the basics of Boolean satisfiability to keep the work self-contained.

Definition 2. *Let Φ be a Boolean function. Then, the* Boolean satisfiability *problem is to determine an assignment to the variables of Φ such that Φ evaluates to 1 or to prove that no such assignment exists.*

Example 5. *Let $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3)$. Then, $x_1 = 1$, $x_2 = 1$, and $x_3 = 1$ is a satisfying assignment for Φ.*

SAT is one of the central NP-complete problems. Despite this complexity, today's SAT solvers incorporate powerful solving strategies [32]. They can handle instances composed of hundreds of thousands of variables and, hence, found applications in automatic test pattern generation [16], [17], [19], logic synthesis [21], [22], verification [15], [18], and more. While they have hardly been exploited in the domain of quantum computing yet, first promising SAT-based solutions recently have been proposed, e.g., in [24], [25]. In this work, we are aiming to solve another important problem from this domain utilizing SAT—working towards establishing SAT-based solutions in quantum computing and, by this, repeating the success SAT had in classical computing for this new domain.

In order to utilize SAT solvers for the considered problem, we first translate the given optimization problem into a sequence of decision problems. More precisely, the question of "What is the minimum amount of time steps $\hat{T}$ for shuttling to execute the circuit?"



(a) Time step 0

ion chain 0	ion chain 1	ion chain 2	ion chain 3
$\mathbf{x^0_{e_0,i_0}=1}$	$x^0_{e_0,i_1}=0$	$x^0_{e_0,i_2}=0$	$x^0_{e_0,i_3}=0$
$x^0_{e_1,i_0}=0$	$\vdots$	$\vdots$	$\vdots$
$\vdots$	$\mathbf{x^0_{e_8,i_1}=1}$	$\mathbf{x^0_{e_{26},i_2}=1}$	$\vdots$
$\vdots$	$\vdots$	$\vdots$	$\mathbf{x^0_{e_{48},i_3}=1}$
$x^0_{e_{49},i_0}=0$	$x^0_{e_{49},i_1}=0$	$x^0_{e_{49},i_2}=0$	$x^0_{e_{49},i_3}=0$

(b) Encoding of the state at time step 0

Fig. 4: Correspondence between the abstraction of a state and its encoding in Boolean variables.

is deconstructed into a sequence of "Can the circuit be executed with a shuttling procedure of T time steps". An exact and, thus, minimal solution is then guaranteed by starting with $T = 1$ and increasing T by 1 whenever it could be shown that no valid solution with T time steps exists. With this approach, the first value of T for which a satisfying solution is found is also the exact solution which requires $T = \hat{T}$ time steps. This can then be formulated as a SAT instance Φ in which all possible solutions are formulated in a symbolic fashion, i.e., through variables representing the problem and constraints enforcing the validity of the solution. Details on this encoding are provided in the next section.

IV. Encoding of the Shuttling Problem as an SAT Instance

In this section, we propose an encoding of the previously introduced graph-based abstraction into a Boolean satisfiability problem. To this end, we first provide a description of the variables which symbolically represent all possible system states; followed by constraints that ensure consistency and only valid transitions between two states.

A. Symbolic State Description

In order to symbolically encode all possible states in a QCCD architecture in the Boolean function Φ, the following variables are introduced:

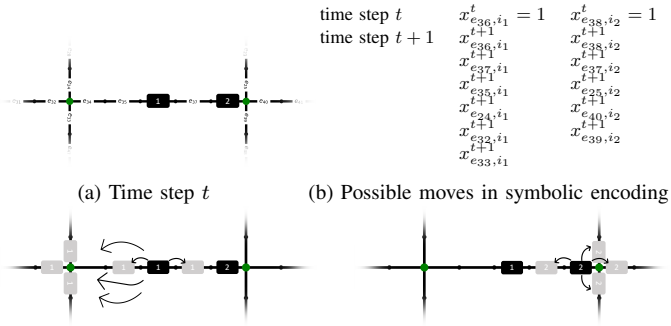
Definition 3. *Consider again Definition 1. The Boolean variables employed to describe the state of the system are denoted $x^t_{e,i} \in \{0,1\}$ with $0 \leq t \leq T$, $i \in C$, and $e \in E$. Each such variable represents whether there is an ion chain i present on site e at time step t. The value 0 ("false") means absent, whereas 1 ("true") means present.*

Example 6. *Consider the 4×4-grid shown in Fig. 4a which represents the initial time step of a possible shuttling problem. The edges e_0, e_8, e_{26}, and e_{51} are occupied by the ion chains i_0, i_1, i_2, and i_3, respectively. All other edges are empty, including the inbound and outbound edge interfacing the processing zone (which are always considered to be empty at the beginning). The resulting variables of this configuration for the time step $t = 0$ are shown in Fig. 4b.*

Having all these variables, the positions of all present ion chains can be described at any time step t. All time steps $0 \leq t \leq T$ combined, this eventually represents a shuttling sequence on the given grid. However, passing those variables without any further constraints to a SAT solver, obviously yields an arbitrary assignment and, hence, an arbitrary (and most likely invalid) shuttling sequence. To prevent that, additional constraints are enforced which are described next.

B. Validity Constraints

Given the symbolic encoding of the system state with the variables $x^t_{e,i}$ from Definition 1, we now provide constraints to ensure that assignments to the variables cannot result in an invalid system



(a) Time step t (b) Possible moves in symbolic encoding

(c) Possible moves of chain i_1 (d) Possible moves of chain i_2

Fig. 5: Visualization and encoding of possible movement of two ion chains i_1 and i_2 in one time step. Current ion chain positions at time step t are indicated as black boxes, possible positions at time step $t+1$ as grey boxes.

state. For each constraint, we provide a textual description to provide an intuition as well as a formal description. To build the function Φ, the individual constraints are combined as a logical conjunction, i.e., the AND operation. Furthermore, we define the following functions to increase the readability of constraints:

Definition 4. *To improve the readability of the constraints, the functions* AtLeast *and* AtMost *are used, which, given a set M of Boolean variables, are defined as:*

$$\textsc{AtLeast}(M) = \sum_{m \in M} (m) \geq 1 \qquad \textsc{AtMost}(M) = \sum_{m \in M} (m) \leq 1$$

Regarding the summation, true *values are treated as* 1 *and* false *values are treated as* 0*. These functions (also termed* cardinality constraints*) are provided by common SAT solvers and transparently translated into basic Boolean operations [33].*

The first set of constraints ensures that the description at a single time step is valid:

Since we consider a trapped-ion quantum computer, each and every ion chain $i \in C$ in the model has to occupy exactly one site (modeled as edge $e \in E$) in each time step t, i.e.,

$$\bigwedge_{0 \leq t \leq T} \bigwedge_{i \in C} \left(\textsc{AtMost}\left(\{x^t_{e,i}\}_{e \in E}\right) \wedge \textsc{AtLeast}\left(\{x^t_{e,i}\}_{e \in E}\right) \right).$$

In correspondence to the previous constraint, the following constraint ensures that each site can only hold zero or one ion chain in any given time step (in the case of the inbound edge e_{in}, this is relaxed to at most two ion chains, which allows us to simulate the presence of two ion chains in the processing zone), i.e.,

$$\bigwedge_{0 \leq t \leq t_T} \bigwedge_{e \in E} \textsc{AtMost}\left(\{x^t_{e,i}\}_{i \in C}\right).$$

After constraining the variables to only assume valid values regarding individual time steps, the following section adds constraints to ensure the SAT solver only assigns valid values with respect to possible transitions (and, hence, movements) between time steps.

C. Movement Constraints

Movement constraints ensure that only valid changes are assigned to the system in the transition from one time step to the next.

To formulate the movement of ion chains, we introduce the notations of *neighbor edges* $N(e)$ of an edge $e \in E$ and the *path edges* $P(e, e^*)$ between two edges e and e^*. More precisely, $N(e)$ denotes edge e and all edges that are directly connected to it. Additionally, $N^*(e)$ also includes all edges, that follow after the two nearest junctions of e. The set of path edges $P(e, e^*)$ contains all edges that are part of the shortest path in the graph between e and e^*. Fig. 5 provides a visualization of $N^*(e)$. During the transition

from one time step t to the next one $t+1$, each ion chain is allowed to stay at its current edge e or move to one of the edges e^* given by $N^*(e)$, if the path $P(e, e^*)$ in between e and e^* is not occupied at time step t by any other ion chain $i' \in C \setminus \{i\}$. Ion chains are always free to move to directly connected edges, e.g., edges of $N(e)$, since there are no edges between them. The corresponding encoding of possible moves for the ion chains in Fig. 5a is given in Fig. 5b. This approach allows every chain to move past one junction, given the path leading to this junction is not blocked, and also preserves the order of chains that are positioned in between two junctions. This leads to

$$\bigwedge_{0\leq t\leq T}\bigwedge_{i\in C}\bigwedge_{e\in E}\left(x^t_{e,i} \Rightarrow \bigvee_{e^*\in N^*(e)}\left(x^{t+1}_{e^*,i} \wedge \bigwedge_{e'\in P(e,e^*)}\bigwedge_{i'\in C\setminus\{i\}} \neg x^t_{e',i'}\right)\right). \quad (1)$$

As described before, it is not possible that ion chains are moved past each other in a memory zone. Furthermore, a junction can only shuttle one ion chain at a time. To enforce this in our encoding, we limit the number of ion chains that are allowed to pass over a node to 1. This includes every node in the layout, meaning all junctions and all minor nodes within two junctions. For this purpose, we introduce the set of edges $Sh(v)$ which *share* a node v, e.g., edges e_1 and e_2 share node v, if $v \in e_1$ and $v \in e_2$. Additionally, we denote the set of *former* edges as $F(e)$. Former edges include all possible edges of an ion chain at $t-1$, given it occupies edge e at t and Equation 1 is fulfilled. Overall, this leads to

$$\bigwedge_{0\leq t\leq T}\bigwedge_{v\in V}\textsc{AtMost}\left(\left\{x^t_{e,i} \wedge \bigvee_{f\in F(e)} x^{t-1}_{f,i}\right\}_{i\in C; e\in Sh(v)}\right).$$

We consider a one-way connection to the processing zone, i.e., ion chains that use the outbound edge e_{out} have to move to the connected processing zone edge e_{in} and back into the memory zone graph to a neighbor edge of the inbound edge $n \in N(e_{in})$. For simplicity, we split the corresponding encoding into three separate constraints:

- at the outbound edge, an ion chain has to move to the inbound edge at the next time step, i.e.,
$$\bigwedge_{0\leq t<T}\bigwedge_{i\in C}(x^t_{e_{out},i} \Rightarrow x^{t+1}_{e_{in},i}),$$
- at the inbound edge, an ion chain was either in the inbound edge or in the outbound edge one time step before, i.e.,
$$\bigwedge_{0<t\leq T}\bigwedge_{i\in C}(x^t_{e_{in},i} \Rightarrow x^{t-1}_{e_{out},i} \vee x^{t-1}_{e_{in},i}), \text{and}$$
- a chain in the inbound edge has to either stay in e_{in} or move to a neighbor edge $n \in N(e_{in})$ of e_{in} (excluding e_{out}), i.e.,
$$\bigwedge_{0\leq t<T}\bigwedge_{i\in C}\left(x^t_{e_{in},i} \Rightarrow \bigvee_{n\in N(e_{in})\setminus\{e_{out}\}} x^{t+1}_{e_{in},i} \vee x^{t+1}_{n,i}\right).$$

Example 7. *Consider the scenario in Fig. 5. Fig. 5a illustrates the system state at time step t and Fig. 5b gives the possible positions of the considered ion chains in time step $t+1$ in terms of variables in the encoding. All possible moves of ion chain i_1 (ion chain i_2) are illustrated in Fig. 5c (Fig. 5d). Both ion chains can either stay at their current edge, move to their neighbor edge or move over a connected junction. Since they block the path of each other, only the moves passing the opposite junction are possible for each ion chain.*

D. Enforcing the Target Sequence

The combination of the discussed validity and movement constraints allows a solving engine to determine valid configurations of ion chain positions that simulate the movement of ion chains in a QCCD device for a given amount of time steps. Since we want to determine if a given sequence of ions can be shuttled in order to the processing zone in T time steps, we introduce further constraints to enforce this goal. As discussed above, a quantum algorithm results in a sequence of qubits that have to be processed in the processing zone. Each qubit is mapped to an individual ion. Each ion is part of one ion chain. For clarity, we only consider ion chains that contain exactly one ion and therefore one qubit, i.e., the sequence of qubits is identical to the sequence of ion chains. The generalization to ion chains with multiple ions is straightforward, since this only changes the resulting sequence. This can be handled with data preprocessing.

Example 8. *The sequence of qubits that have to be processed in the processing zone for a quantum Fourier transform with four qubits reads $[q_0; (q_0, q_1); (q_0, q_2); q_1; (q_1, q_2); q_3]$. For a given architecture with four ion chains, each holding exactly one ion, the resulting sequence of ion chains is given by: $[i_0; (i_0, i_1); (i_0, i_2); i_1; (i_1, i_2); i_3]$.*

In order to formulate a constraint that enforces this sequence, we introduce helper variables that describe the elements of the sequence.

Definition 5. *We represent every step of the sequence as a Boolean variable $s^t_j \in \{0, 1\}$ with $0 \leq t \leq T$ and $j \in C$. Each such variable represents one element of the sequence with index j at time step t. The value 0 ("false") indicates that the corresponding element is not executed, while 1 ("true") means it is executed at this time step.*

Then, to make use of these helper variables, we need to connect them with the corresponding variables $x^t_{e,i}$. For example, if the variable of sequence element j corresponding to ion chain i_k is 1, variable $x^t_{e_{in},i_k}$ must also be 1. This means, that if s^t_j is true, the respective ion chains have to be in the inbound edge e_{in} at t, which confirms that they have been moved to the processing zone. Overall, this yields

$$\bigwedge_{0<t\leq T}\bigwedge_{0\leq j\leq |S|}\left(s^t_j \Rightarrow \bigwedge_{i_k\in C_j} x^t_{e_{in},i_k}\right),$$

where $|S|$ is the number of elements in the given sequence and C_j is the set of ion chains that have to be moved to the processing zone because of element j in the sequence.

In order to formulate the order of the sequence, we process the sequence in a pairwise fashion. More precisely, if variable s^t_j of sequence element j is true at time step t, the variable of the next sequence element $j+1$ has to be true at a later time step $t' > t$. Since every sequence element has a unique index, applying this to every variable enforces the correct order of sequence elements. The corresponding constraint is given by

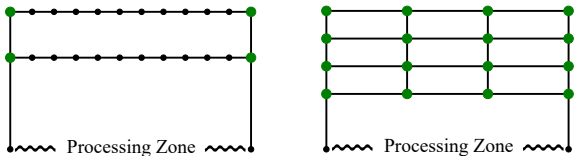
$$\bigwedge_{0\leq j\leq L}\bigvee_{0\leq t<T}\left(s^t_j \wedge \bigvee_{t<t'\leq T} s^{t'}_{j+1}\right).$$

Lastly, every sequence element s^t_j is only true once. This leads to

$$\bigwedge_{0\leq j\leq L}\textsc{AtMost}\left(\{s^t_j\}_{0<t\leq T}\right).$$

Example 9. *Consider again the sequence in Example 8: $[i_0; (i_0, i_1); (i_0, i_2); i_1; (i_1, i_2); i_3]$. Satisfying all discussed constraints, a valid solution given by the solver would start by moving ion chain i_0 to the processing zone. Both i_1 and i_2 then join i_0 one after the other. After they have been processed, both ion chains move back to the memory zone, while ion chain i_1 starts moving to the processing zone. To realize the rest of the sequence, i_2 joins i_1, before i_3 occupies the processing zone alone for at least one time step to finalize the sequence. A valid assignment of all variables achieves these movements within a given amount of time steps T.*

Overall, this results in a symbolic encoding of system states in a QCCD device and corresponding constraints to ensure validity in the position and movement of ion chains. Consider a fixed number of time steps T. If the SAT solver returns "unsat" (i.e., no satisfying assignment exists), then it has been proven that no movement



(a) "Racetrack": $L(2, 2, 1, 11)$ (b) "Lattice": $L(4, 4, 1, 1)$

Fig. 6: Two types of layouts considered in the benchmark.

realizing the given sequence within T time steps exist. If the SAT solver determines a satisfying assignment, then such a movement within T time steps has been obtained. Since T is iteratively increased by 1 starting with $T = 1$ (as described above in Section III-B), this eventually proves this movement to be minimal, i.e., $T = \hat{T}$. This symbolic encoding may also be extended to include new constraints that model the behavior of new types of QCCD devices.

V. Empirical Evaluation

The approach proposed above has been implemented in Python3 utilizing the publicly available SAT solver Z3 (version 4.12.1) [34]. The resulting implementation is available as open-source at https://github.com/cda-tum/mqt-ion-shuttler. To the best of our knowledge, this leads to the first *exact* shuttling approach for trapped-ion quantum computers. In this section, we discuss the efficacy of the resulting tool and summarize the obtained results.

To evaluate the tool, we considered different architectures following a grid structure, i.e., the angles of all junctions are set to $90°$. The corresponding grid-like graphs are denoted $L(m, n, v, h)$ and constructed as follows:

- The graph is laid out as a $m \times n$ grid, with m nodes vertically and n nodes horizontally.
- Each node in this grid represents a junction in the trapped-ion quantum computer.
- In between two major nodes (junctions) at most v ion chains can be trapped vertically.
- In between two major nodes (junctions) at most h ion chains can be trapped horizontally.

The grid is further extended by two additional edges which represent one outbound edge to a processing zone and one inbound edge leading back to the memory grid. Using a random starting configuration of ion chains on these grids, we then used the proposed approach to determine the minimum number of time steps $\hat{T}$ that are sufficient to realize a given quantum circuit, e.g., shuttling sequence. All evaluations were conducted on a machine with an Intel(R) Xeon(R) W-1370P CPU (running at 3.6 GHz) and 128 GiB main memory running Python 3.8.10.

For this evaluation, we consider two categories of layouts. The first category contains layouts with long "streets" of connected edges that are not interrupted by junctions, similar to a *racetrack*. We compared these layouts to *lattice* layouts, in which every edge is enclosed by two junctions. The major difference between these two types of layouts is the amount of junctions in the system. The racetrack configuration consists of the minimum amount of junctions for a grid-form graph, while the lattice type employs the maximum amount. One example of each category is given in Fig. 6, where both layouts can trap up to 24 ion chains at a time.

Within the two different layouts, two different types of sequences were considered for the evaluation:

- *Demonstration of full register access:* In a first series, we investigated the accessibility of all registers in a system, i.e., the minimum number of time steps to move all ion chains in order to the processing zone. Hence, the corresponding sequence is an ascending list of all ion chain indices. This is akin to a quantum circuit, that applies a single-qubit gate successively to each qubit.
- *Demonstration of quantum Fourier transform:* In a second series, we considered the sequence derived from the quantum Fourier transform [4] (the precise instance has been taken from MQT Bench [35]). For better comparability, we set the amount of ions in each ion chain to one. Otherwise, the results would also strongly depend on this additional degree of freedom.

In all experiments, the initial placement of chains was done randomly.

The results are shown in Table I. To this end, the results are split into one part concerning all racetrack layouts and the other all lattice layouts. In each block, the first column defines the variables of graph $L(m, n, v, h)$. Then, the second column provides the number of ion chains $|C|$, the number of edges that constitute the memory zone $|E_M|$ (for the sake of clarity, we only list the edges that vary in the considered layouts, i.e., all edges in the memory zone) and the percentage relationship between these two values. The next three columns list the length of the sequence $|S|$, the minimal amount of time steps $\hat{T}$ obtained over 10 runs, each with a different random starting configuration of ion chains, and the average time t_{CPU} taken to generate these exact results. The abbreviation "t.o." indicates that no satisfying assignment has been determined for all 10 runs within a timeout of 5 000 CPU seconds (in these cases, the value on the $\hat{T}$-column provides the largest number of time steps for which a non-existence of a shuttling has been proven—providing a lower bound for $\hat{T}$). If some, but not all runs resulted in a timeout, the averages of $\hat{T}$ and t_{CPU} are given for the successful runs. The last column provides the number of timeouts that occurred. Variable q refers to the number of qubits in the considered quantum Fourier transform.

Overall, the results confirm that the proposed approach is applicable to determine *exact* results for shuttling in trapped-ion quantum computers. Although this is only possible for relatively small layouts (after all, the problem remains computationally expensive), this provides first-time *minimal* solutions and, hence, lower bounds are available for this problem. The discussed approach enables the comparison of different layouts, providing a guideline for future proposals of QCCD architectures. As an example, our first results indicate that, even for small instances, lattice layouts with many junctions provide better access to all registers compared to layouts with few junctions in respect to shuttling time.

VI. Conclusion

Quantum computers based on trapped-ion technology are promising candidates for enabling scalable quantum computations with the QCCD architecture. Inside these architectures, efficient shuttling of ion chains in the memory zone is important to reduce run time and, thus, decrease the likelihood of errors. In this paper, we proposed an abstraction of the physical memory zone in the QCCD architecture and a corresponding description by means of Boolean variables. More precisely, we abstract the physical architecture through a graph, where nodes represent junctions and edges the individual sites where ion chains are held. Based on this graph, we introduce Boolean variables to encode the position of the ion chains at any time step. Added constraints ensure only valid states and transitions between states in a time step are possible. Leveraging the power of modern SAT solver, we utilize the symbolic description to find exact solutions to the shuttling problem. The empirical evaluation confirms the efficacy for determining exact shuttling. Moreover, even in instances which timeout, i.e., do not finish in the given time, the results provide a lower bound for the number of required time steps.

TABLE I: Results of the Empirical Evaluation

Algorithm	Racetrack						Lattice					
	$m\ n\ v\ h$	$\lvert C\rvert/\lvert E_M\rvert$	$\lvert S\rvert$	$\hat{T}$	t_{CPU}	t.o.	$m\ n\ v\ h$	$\lvert C\rvert/\lvert E_M\rvert$	$\lvert S\rvert$	$\hat{T}$	t_{CPU}	t.o.
Full Register Access	2 2 1 5	6/12 (50%)	6	11.0	11.3 s	0	3 3 1 1	6/12 (50%)	6	10.9	7.1 s	0
		12/12 (100%)	12	22.6	1721.8 s	0		12/12 (100%)	12	17.0	36.1 s	0
	2 2 1 11	6/24 (25%)	6	11.0	39.6 s	0	4 4 1 1	6/24 (25%)	6	12.5	17.7 s	0
		12/24 (50%)	12	21.4	1985.1 s	1		12/24 (50%)	12	18.5	80.6 s	0
		18/24 (75%)	18	>28	t.o.	10		18/24 (75%)	18	24.5	237.6 s	0
	2 2 1 19	6/40 (15%)	6	10.9	106.0 s	0	5 5 1 1	6/40 (15%)	6	12.9	31.1 s	0
		12/40 (30%)	12	23.0	2404.7 s	4		12/40 (30%)	12	18.9	139.2 s	0
		18/40 (45%)	18	>28	t.o.	10		18/40 (45%)	18	24.9	386.2 s	0
	2 2 1 29	6/60 (10%)	6	10.0	97 s	0	6 6 1 1	6/60 (10%)	6	14.3	58.3 s	0
		12/60 (20%)	12	20.3	3616.0 s	6		12/60 (20%)	12	20.3	243.9 s	0
		18/60 (30%)	18	>25	t.o.	10		18/60 (30%)	18	26.3	647.3 s	0
Quantum Fourier Transform $q=5$	2 2 1 5	5/12 (42%)	15	22.4	40.0 s	0	3 3 1 1	5/12 (42%)	15	26.9	42.4 s	0
$q=6$		6/12 (50%)	21	30.0	99.9 s	0		6/12 (50%)	21	33.9	90.6 s	0
$q=7$		7/12 (58%)	28	38.6	221.2 s	0		7/12 (58%)	28	41.9	185.6 s	0
$q=8$		8/12 (67%)	36	48.2	463.6 s	0		8/12 (67%)	36	50.9	378.7 s	0

ACKNOWLEDGMENTS

This work was funded under the European Union's Horizon 2020 research and innovation programme (DA QC, grant agreement No. 101001318 and MILLENION, grant agreement No. 101114305), the State of Upper Austria in the frame of the COMET program, the QuantumReady project within Quantum Austria (managed by the FFG), and was part of the Munich Quantum Valley, which is supported by the Bavarian state government with funds from the Hightech Agenda Bayern Plus.

REFERENCES

[1] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Symp. on Foundations of Computer Science*, IEEE Computer Society, 1994, pp. 124–134.

[2] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Symp. on Theory of Computing*, ACM, 1996, pp. 212–219.

[3] R. Babbush, N. Wiebe, J. McClean, J. McClain, H. Neven, and G. K.-L. Chan, "Low-depth quantum simulation of materials," *Phys. Rev. X*, vol. 8, p. 011 044, 1 2018.

[4] M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016.

[5] J. Clarke and F. K. Wilhelm, "Superconducting quantum bits," *Nature*, vol. 453, no. 7198, pp. 1031–1042, 2008.

[6] L. Henriet, L. Beguin, A. Signoles, *et al.*, "Quantum computing with neutral atoms," *Quantum*, vol. 4, p. 327, 2020.

[7] L. Schmid, D. F. Locher, M. Rispler, *et al.*, *Computational capabilities and compiler development for neutral atom quantum processors: Connecting tool developers and hardware experts*, 2023. arXiv: 2309.08656 [quant-ph].

[8] E. Knill, R. Laflamme, and G. J. Milburn, "A scheme for efficient quantum computation with linear optics," *Nature*, vol. 409, no. 6816, pp. 46–52, 2001.

[9] C. D. Bruzewicz, J. Chiaverini, R. McConnell, and J. M. Sage, "Trapped-ion quantum computing: Progress and challenges," *Applied Physics Reviews*, vol. 6, no. 2, p. 021 314, 2019.

[10] D. Kielpinski, C. Monroe, and D. J. Wineland, "Architecture for a large-scale ion-trap quantum computer," *Nature*, vol. 417, no. 6890, pp. 709–711, 2002.

[11] J. M. Pino, J. M. Dreiling, C. Figgatt, *et al.*, "Demonstration of the trapped-ion quantum CCD computer architecture," *Nature*, vol. 592, no. 7853, pp. 209–213, 2021.

[12] J. Durandau, J. Wagner, F. Mailhot, *et al.*, *Automated generation of shuttling sequences for a linear segmented ion trap quantum computer*, 2022. arXiv: 2208.04881 [quant-ph].

[13] P. Murali, D. M. Debroy, K. R. Brown, and M. Martonosi, "Architecting noisy intermediate-scale trapped ion quantum computers," in *International Symposium on Computer Architecture*, 2020, pp. 529–542.

[14] A. Biere and D. Kröning, "SAT-based model checking," in *Handbook of Model Checking*, Springer, 2018, pp. 277–303.

[15] D. Brand, "Verification of large synthesized designs," in *Int'l Conf. on CAD*, IEEE Computer Society / ACM, 1993, pp. 534–537.

[16] S. Eggersglüß, R. Wille, and R. Drechsler, "Improved SAT-based ATPG: more constraints, better compaction," in *Int'l Conf. on CAD*, IEEE, 2013, pp. 85–90.

[17] A. Gebregiorgis and M. B. Tahoori, "Test pattern generation for approximate circuits based on boolean satisfiability," in *Design, Automation and Test in Europe*, IEEE, 2019, pp. 1028–1033.

[18] D. Kaufmann, A. Biere, and M. Kauers, "Verifying large multipliers by combining SAT and computer algebra," in *Int'l Conf. on Formal Methods in CAD*, IEEE, 2019, pp. 28–36.

[19] T. Larrabee, "Test pattern generation using boolean satisfiability," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 11, no. 1, pp. 4–15, 1992.

[20] R. Wille, D. Große, F. Haedicke, and R. Drechsler, "SMT-based stimuli generation in the SystemC verification library," in *Forum on Specification and Design Languages*, IEEE, 2009, pp. 1–6.

[21] W. Haaswijk, M. Soeken, A. Mishchenko, and G. D. Micheli, "SAT-based exact synthesis: Encodings, topology families, and parallelism," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 871–884, 2020.

[22] V. Khomenko, M. Koutny, and A. Yakovlev, "Logic synthesis for asynchronous circuits based on STG unfoldings and incremental SAT," *Fundam. Informaticae*, vol. 70, no. 1-2, pp. 49–73, 2006.

[23] R. Wille, L. Burgholzer, and A. Zulehner, "Mapping quantum circuits to IBM QX architectures using the minimal number of SWAP and H operations," in *Design Automation Conf.*, ACM, 2019.

[24] L. Berent, L. Burgholzer, and R. Wille, "Towards a SAT encoding for quantum circuits: A journey from classical circuits to clifford circuits and beyond," in *Conference on Theory and Applications of Satisfiability Testing*, vol. 236, 2022, 18:1–18:17.

[25] R. Wille and L. Burgholzer, "MQT QMAP: Efficient quantum circuit mapping," in *Int'l Symp. on Physical Design*, Virtual Event, USA, 2023, pp. 198–204.

[26] T. Peham, N. Brandl, R. Kueng, R. Wille, and L. Burgholzer, *Depth-optimal synthesis of clifford circuits with sat solvers*, 2023. arXiv: 2305.01674 [quant-ph].

[27] J. I. Cirac and P. Zoller, "Quantum computations with cold trapped ions," *Phys. Rev. Lett.*, vol. 74, pp. 4091–4094, 20 1995.

[28] T. P. Harty, D. T. C. Allcock, C. J. Ballance, *et al.*, "High-fidelity preparation, gates, memory, and readout of a trapped-ion quantum bit," *Phys. Rev. Lett.*, vol. 113, p. 220 501, 22 2014.

[29] 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*, vol. 536, no. 7614, pp. 63–66, 2016.

[30] K. R. Brown, J. Chiaverini, J. M. Sage, and H. Häffner, "Materials challenges for trapped-ion quantum computers," *Nature Reviews Materials*, vol. 6, no. 10, pp. 892–905, 2021.

[31] M. F. Brandl, *A quantum von neumann architecture for large-scale quantum computing*, 2017.

[32] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability – Second Edition* (Frontiers in Artificial Intelligence and Applications). IOS Press, 2021, vol. 336.

[33] O. Bailleux and Y. Boufkhad, "Efficient cnf encoding of boolean cardinality constraints," in *Principles and Practice of Constraint Programming*, Springer Berlin Heidelberg, 2003, pp. 108–122.

[34] L. de Moura and N. Bjørner, "Z3: An efficient smt solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, Springer Berlin Heidelberg, 2008, pp. 337–340.

[35] N. Quetschlich, L. Burgholzer, and R. Wille, "MQT Bench: Benchmarking software and design automation tools for quantum computing," *Quantum*, vol. 7, p. 1062, 2023.