Using Compiler Frameworks for the Evaluation of Hardware Design Choices 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-ion quantum computers hold significant promise as platforms for high-quality qubits and reliable quantum computation. The *Quantum Charge Coupled Device (QCCD)* architecture shows potential in enabling scalable trapped-ion quantum computers through its modular design. Within QCCD devices, ions can be shuttled throughout the trap, which is divided into different dedicated zones, e.g. a memory zone for storage and a processing zone for actual computation. While the modular approach shows promise, designing a large-scale QCCD device remains challenging due to the numerous possibilities involved in designing the precise architecture of the hardware. In this work, we propose leveraging existing compilers to evaluate design choices in the development of QCCD architectures. Rather than compiling specifically for a single device, we use the compiler to explore and evaluate the execution of representative quantum circuits on various architectures. This allows for a fast assessment of different design choices, considering individual requirements and physical limitations. To validate the feasibility of this idea, we conducted a study focusing on key design decisions, including system size, junction density, processing zone positioning, and the benefits of SWAP operations within the memory zone. Based on these results, we are able to provide first recommendations and insights for the design of future QCCD architectures.**

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

I. Introduction

Quantum computers are built to exploit quantum phenomena such as *superposition* (where a quantum state can be composed of multiple basis states) and *entanglement* (a type of correlation that is not present in classical mechanics) to perform calculations beyond the capabilities of today's classical computers. Classical examples of such calculations include algorithms like Shor's algorithm for integer factorization [1], Grover's search for unstructured data search [2], and simulations in quantum chemistry [3]. Accelerating the progress in this field, companies such as IBM, Google, Microsoft, Rigetti, AQT, Infineon Technologies, Quantinuum, IonQ, IQM, and more are investing actively in this technology.

Quantum computers have been realized on several physical platforms, such as superconducting quantum computers [4], neutral atom quantum computers [5], [6], or optical quantum computers [7]. Among those, trapped-ion quantum computers are a strong contender to demonstrate quantum advantage in the foreseeable future [8].

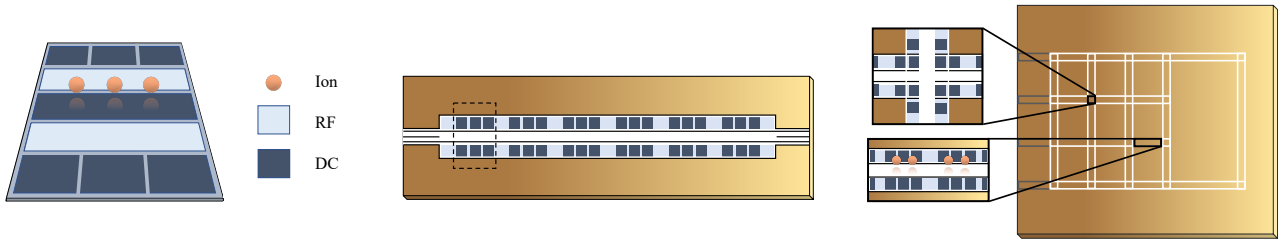
Despite their promise, the development of trapped-ion quantum computers as scalable platforms still proves to be challenging. However, the modularity of the QCCD architecture allows for a large degree of freedom in the design of such devices. Moreover, state-of-the-art hardware implementations change rapidly, which opens up even more possibilities. This leads to a situation, where a lot of optimization in the design of new architectures would be possible, but the hardware community lacks tools to evaluate their design choices and make use of this potential.

In this work, we propose to use existing software tools to evaluate key design choices in state-of-the-art and future QCCD architectures. More precisely, we discuss the use of shuttling compilers as a way of simulating the execution of quantum algorithms on shuttling-based QCCD devices. For this matter, we conduct a study of four design choices that cover critical aspects of the design process, ranging from the actual design of the hardware to the evaluation of new technical advancements. We present first results for all four design choices and provide precise insights for grid-type QCCD architectures.

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 general idea of the proposed approach. Section IV discusses the specifics of the four considered design choices. Section V summarizes the obtained results. Finally, Section VI concludes the paper.

II. Background

In this section, we provide an overview of trapped-ion quantum computing and the challenges involved in designing state-of-the-art and future devices. It introduces the *Quantum Charge Coupled Device* (QCCD) architecture, which promises to enable scalable trapped-ion quantum computing. For a more detailed description, further references are provided.



(a) Surface trap (b) 1D Linear QCCD architecture (c) 2D QCCD architecture

Fig. 1: An illustration of a possible ion trap realization as a surface trap module, which can be connected to a linear and a grid-type QCCD architecture. A correct combination of radio-frequency (RF) and quasi-static (DC) electric fields produced by control electronics creates a potential that confines the ions.

A. Trapped-Ion Quantum Computing

Ions can be confined in an ion trap. The so called Paul traps isolate and control the position of their ions with electric fields. A combination of radio-frequency and quasi-static electric fields creates an electric potential that confines multiple ions arranged in a chain-like configuration. Importantly for quantum computing, the internal states of the ions can be manipulated via electromagnetic interactions, either in the optical or microwave domain. [9]–[11]

Example 1. *For example, the so Paul trap is a commonly used ion trap that can be also realized as a two-dimensional (2D) surface trap. Figure 1a illustrates the resulting configuration of control elements (radio-frequency, light-blue and quasi-static, dark-blue). Individual ions are illustrated as orange spheres. These ion chains have been coined* ion registers*, because they may be used similarly to registers in classical computers. We refer to this kind of a surface trap module as a trap site.*

However, scaling up the number of ions N in a single trap decreases the gate speed approximately as $R_{\text{gate}} \sim \frac{1}{\sqrt{N}}$. This makes the scaling of single chains challenging for practical quantum algorithms that require more qubits.

B. Quantum Charge Coupled Device Architecture

The *Quantum Charge Coupled Device* (QCCD) architecture was introduced in [12] to tackle this challenge. Single linear trap modules can be connected to form a larger trap, which holds one ion chain at each site. Such architectures can exploit another advantage of ion traps: shuttling. Moving the ions within these modular architectures allows all-to-all connectivity of all qubits. To exploit this even further, the QCCD architecture proposes to use different parts of its system for different purposes. For example, a zone (constructed from one or multiple individual sites) can be dedicated as a *processing zone* that is specifically designed to perform efficient quantum gates. Another zone can be used as a *memory zone*, which is protected from potential sources of decoherence. Additional zones may include regions for measuring qubits (*measurement zone*) or loading new ions into the system (*loading zone*).

Example 2. *Consider the linear QCCD architecture illustrated in Figure 1b. We refer to this kind of a system as a linear trap. The device is designed as multiple trap sites connected in a straight line. One individual site is marked in Figure 1b. Within these linear devices, ion chains can be shuttled to a neighboring site.*

Linear traps lack the ability to move ions past each other. Resolving a blockage would require slow interactions like chain reordering and reconfiguration. To circumvent this, QCCD architectures can be extended with junctions, that connect linear regions to form two-dimensional systems.

Example 3. *Figure 1c illustrates a QCCD architecture in a grid-type configuration of linear traps and "X"-junctions.*

Junctions can be built to connect linear regions in various ways. Different types of junctions (i.e., termed "T"-, "Y"- or "X"-junctions, where the capital letter is referring to their shape), can completely change the layout of devices.

III. Motivation

Quantum computers using the QCCD architecture have already been demonstrated in [11], [13], and recently as a first two-dimensional architecture in [14]. However, designing these devices is challenging, and there is a need for enhanced tool support to evaluate design choices for future devices.

A. General Design Process

Central to the functionality of QCCD architectures is the ability to shuttle ions between various functional zones effectively. Design choices can heavily affect these shuttling capabilities, e.g. the choice of an architecture significantly influences how ion chains can be moved around and, hence, how fast a given quantum circuit can be executed. Since trap sites and junctions in trapped-ion quantum computers can be freely connected, there is a significant degree of freedom involved in the development of QCCD architectures. Junctions can be designed to fit connections in every direction, which allows the design of new devices in various shapes. Additionally, control electronics and optics have to be scaled to fit demands and integrated into the device. Moreover, continual technological advancements and new hardware implementations are reshaping the methods by which ions are

controlled and manipulated, further enhancing the flexibility and capability, but also increasing the complexity of these quantum computing systems. Unfortunately, there exist little to no means to simulate the effect of these design choices and, hence, their impact on the overall performance of new devices.

B. Existing Compiler Frameworks

While the community lacks support for the evaluation of actual ion trap hardware, there already exist multiple compiler frameworks developed for various QCCD architectures. These compiler tools provide a large playing ground to experiment with and allow developers to predict system behavior and performance across various hypothetical architectures without the need of actual physical hardware. In [15] general architectures can be constructed by connecting trap sites and junctions to form a graph structure. For small, near-term devices, [16]–[18] provide compilation support and even running full quantum circuits on large machines is supported by [19], [20].

C. General Idea

In this work, we propose an innovative application of these compiler frameworks to not only compile a quantum circuit but also to simulate the execution across different architectures. This allows us to assess various metrics, aiding in the design of more efficient quantum devices and expanding the utility of existing tool support. To guide the design process, we define and discuss key design choices of QCCD architectures. These choices revolve around optimizing the system's physical layout and shuttling mechanics to minimize the execution time of quantum circuits. Using representative quantum circuits, we systematically evaluate these design choices. The outcomes from these evaluations will not only provide new insights but also generate first practical recommendations for grid-type QCCD architectures.

IV. Exploration of Key Design Choices

Determining the optimal architecture of a shuttling-based quantum computing device is crucial for efficient qubit processing and storage. This section explores four design choices in the general context of QCCD devices, that are aimed toward helping in the decision making process of hardware experts. Respective evaluations are presented in the subsequent section.

A fundamental aspect to consider is the size of a shuttling-based device. Specifically, determining the most suitable size of the ion trap to effectively process and store a given number of qubits required by different quantum algorithms. Obviously, the goal of current research is to scale quantum systems and build large-scale devices. For a given amount of ions, more space to move freely reduces the complexity of the shuttling operations and, thus, decreases execution time and potential decoherence. On the other hand, constructing larger devices introduces significant engineering challenges, increased costs, and the need for a larger and more complex control system. That is, larger architectures should only be considered if it indeed is helpful and actually improves the execution of the considered quantum circuits. This leads us to the first crucial design question:

Design Choice 1: *What is a good size of the trap, given a fixed number of qubits that must be processed and stored?*

Example 4. *Consider the illustration of a QCCD grid-like device in Figure 2a. Both the memory zone and processing zone are indicated. The figure also depicts a closer look at a linear region in between two junctions, that holds up to two individual ion chains. Ranging from* 3×3 *to* 5×5*, architectures with an increasing size of the memory zone grid are shown in the top-left box of Figure 2b.*

As a second design choice, we look into the effect of junctions on the shuttling efficiency. Integrating more and more junctions into the architecture is a logical next step in the design of large-scale devices. More junctions increase the connectivity of the individual linear regions in the memory zone and, thus, prevent bottlenecks and may allow smoother, more efficient ion transportation. On the other hand, junctions inherently slow operational speeds, since, as of right now, ions can not be moved through junctions at the same speed as through linear regions. Additionally, they also increase the complexity of new devices and lead to additional engineering challenges. This presents a critical trade-off: is the potential increase in shuttling efficiency worth the added complexity and reduced operational speed? Summarized as our second design question, this reads:

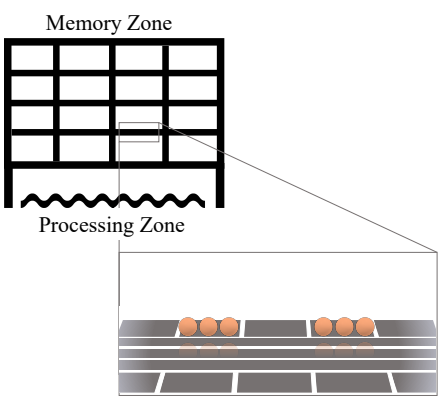
Design Choice 2: *How does the number of junctions in a QCCD architecture — many versus few — affect overall performance?*

Example 5. *The two architectures in Figure 2b illustrate a* 3×3 *(left) and a* 6×6 *(right) grid-like QCCD device. To match the capacity of the larger grid, the linear regions between the junctions of the* 3×3 *grid can hold more ion chains than the smaller linear regions within the* 6×6 *grid. In contrast to that, the right grid employs more junctions than the one on the left.*

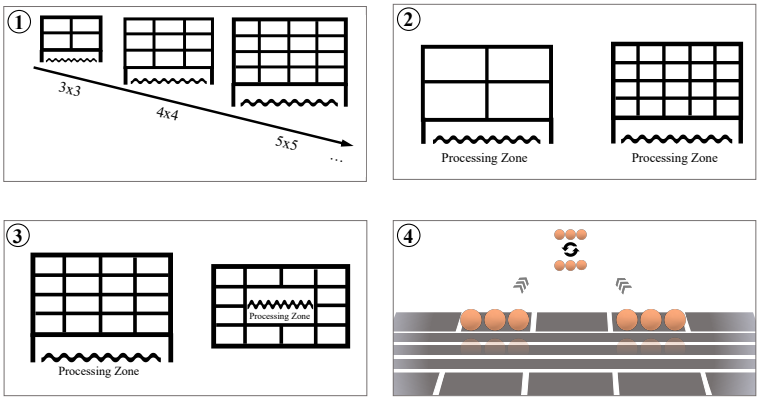
Another crucial role in the design process plays the positioning of the processing zones within these architectures and the effect on the system's performance. Architectures vary widely—some incorporate multiple smaller processing zones dispersed throughout the trap, while others utilize a single processing zone linked directly to the memory zone. In either case, it is not yet clear how much the positions of these zones impact the shuttling efficiency of the devices. This constitutes the third design choice:

Design Choice 3: *How should the processing zone be positioned in relation to the memory zone to maximize shuttling efficiency and minimize circuit execution times?*

Example 6. *Similar to the example of design choice 2, the lower-left box in Figure 2b conceptualizes two different device realizations: one device with a grid-like memory zone which is separated from a processing zone on its outer edge, and one grid-like device that incorporates the processing zone into the the memory zone.*



(a) Illustration of a grid-type QCCD architecture with a memory and processing zone



(b) Design Choices: (1) System size; (2) Number of junctions; (3) Position of the processing zone; (4) Physical swapping

Fig. 2: Illustration of the considered design choices

As mentioned earlier, new research continues to introduce and improve methods for improved shuttling and manipulation of ions. Even though some of these improvements are still in early development and may not even be feasible in the near-term future, we can already evaluate the impact they might have. As an example, consider the physical swap of two ions [21] or even the swapping of two individual ion chains, for which only the rotation of one chain has been studied yet [22]. Physical swapping can be a direct solution to resolve blockages in the memory zone and potentially reduce the complexity of the shuttling operation, but might introduce new risks of errors, especially in densely populated traps. On the other hand, utilizing compiler algorithms to re-route ions around blockages might increase the overall coherence time of the qubits by minimizing disturbances and allows the memory zone to be optimized for shuttling only. With the help of a suitable compiler, we can compare the effectiveness of physically swapping ions or ion chains with the results of a compiler managing blocking ions by moving them out of the way. This constitutes a fourth design choice:

Design Choice 4: *How much improvement is achieved by physically swapping ions or ion chains in the memory zone compared to algorithmic relocation of blocking ions through shuttling operations?*

Example 7. *Similar to the illustration in Figure 2a, a linear region between two junctions is exemplified in the lower-right box of Figure 2b. Ion chains can be rotated in space using the control electronics of the device. In future devices, it may be also possible to move chains around one another and consequently swap the positions of two chains, which is indicated in the figure.*

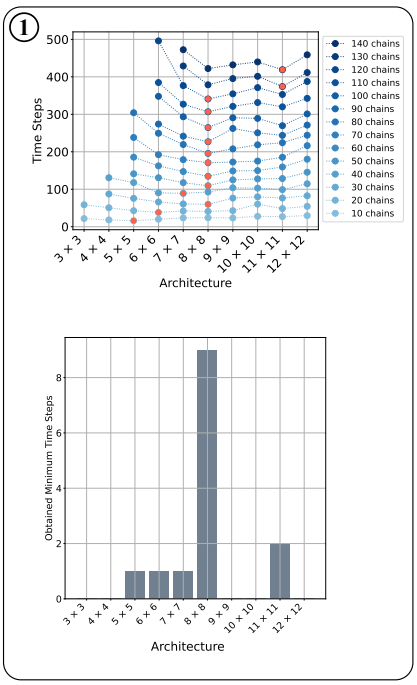
Thus far, all design choices discussed above have hardly been evaluated using automatic methods—usually the designer decided upon those mainly by "gut feeling" or expertise. Using the idea proposed in Section III allows to evaluate what choices lead to an improvement in the performance of QCCD devices and, hence, are worth additional efforts in implementing those. To illustrate that, all four design choices mentioned above have been considered individually and evaluated on a set of representative quantum circuits. Correspondingly obtained results are summarized in the following section. Based on the results, designer can decide whether these choices are worth their trade-offs.

V. Evaluation

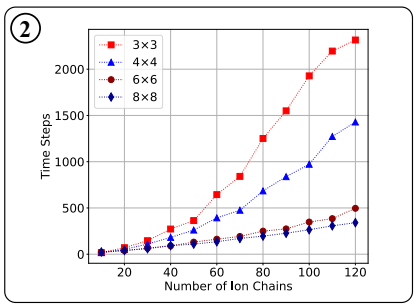
To assess the impact of the discussed design choices, we employ an existing quantum compiler to simulate various scenarios. These simulations help to evaluate how certain design choices would affect the efficiency and performance of QCCD architectures as well as the consequences of these trade-offs. By this, we provide insight on whether corresponding physical challenges in the design of new devices are worth the effort.

A. Detailed Setup

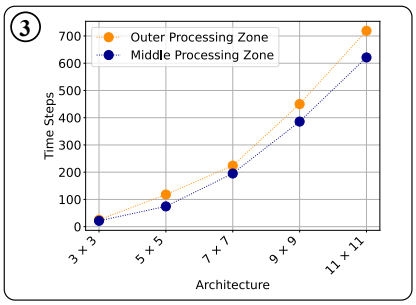
For all our evaluations, we utilized the shuttling compiler described in [20], which is publicly available as part of the open-source Munich Quantum Toolkit (MQT, [23]) at https://github.com/cda-tum/mqt-ion-shuttler. This compiler is specifically designed for grid-type QCCD architectures, which feature a dedicated memory grid connected to a distinct processing zone. The compiler orchestrates the movement of ions from the memory zone to the processing zone—this is crucial for our evaluation purposes, as the majority of ion movements occur between these two zones. The configuration with one dedicated processing zone puts the focus on efficient ion movement and organization within the memory zone. The compiler operates by moving all ions required for computation on their shortest possible path to the processing zone. In instances where blockages arise during shuttling, the compiler uses a cycle-based algorithm to resolve these, i.e. the compiler creates cycles along the paths of the ions and collectively moves all ions on these cycles. This allows the ions to continue on their shortest paths while also moving blocking ions out of the way, without directly swapping positions of ion chains.



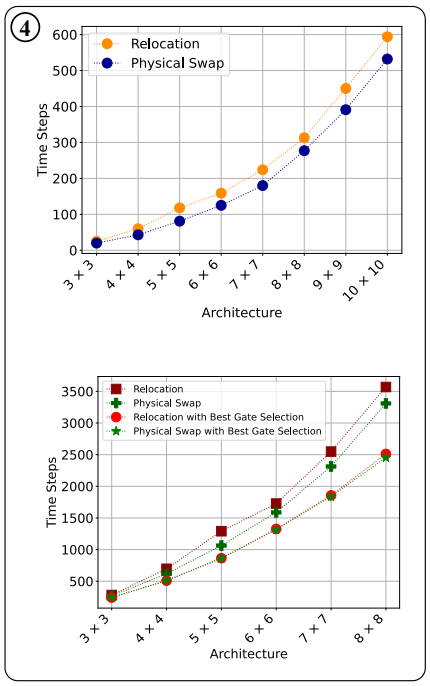
(a) Different system sizes executing "register access" with the minimum result highlighted and plotted below.



(b) Impact of the number of junctions ("register access")



(c) Positioning of the processing zone ("register access")



(d) Advantage of physical swapping executing "register access" (top) and the GHZ state circuit (bottom)

Fig. 3: Evaluation of Key Design Choices

Each architecture follows the same pattern: a grid-type memory zone described by $m \times n$, where m defines the number of linear regions vertically and n the number of linear regions horizontally. The processing zone is a one-way connection to the memory zone and is able to perform single-qubit and two-qubit gates on up to two ion chains.

Since over time, ions collect phonons [24], which lead to decoherence of the qubits' quantum information, ions are usually cooled to or close to the ground state. Especially for large systems, it still remains challenging to integrate the required optical control elements to achieve these conditions. Accordingly, the compiler aims to minimize the execution time of quantum circuits. As a consequence, we use circuit execution time (measured as time steps) as the comparable metric for all following evaluations.

B. Evaluation of the considered Design Choices

The discussed compiler was used to study the four design choices reviewed in Section IV. For each design choice, we evaluate a precise objective, lay out the used implementation and discuss the obtained results (which are summarized in Figure 3). The experiments start with random initial positions of the ion chains within the system and each result is averaged over five runs with different initial configurations. The high-level circuits are provided by [25].

*1) **System Size for Grid-type Architectures**:*

Objective: Determine the optimal size of the ion trap given a fixed number of ion chains. The size of the trap balances the advantages of reduced traffic in larger traps and increased accessibility of ion chains and compactness of smaller devices.

Implementation: For this experiment, we use the compiler to generate shuttling schedules for grid-type architectures with varying trap sizes. In the memory zone, all linear regions within two junctions can hold up to two chains. The processing zone is not part of the grid and placed outside of the memory zone. It can be accessed via a one-way connection as illustrated in Figure 2. The performance of each grid size is evaluated by moving all ion chains of the system into and out of the processing zone randomly, i.e. for a given amount of ion chains, we try to find the grid size that processes all chains in the least amount of time. This is equivalent to applying a gate to each ion chain in the system. This simulates the accessibility of all ions in the device, which provides a simple, but general metric for the performance of a memory zone. In the following, this particular circuit is referred to as "register access".

Results: The plot in the top part of Figure 3a displays the execution time for different grid sizes measured in time steps of the shuttling algorithm (lower is better). The experiment was performed with ion chains ranging from 10 to 140. For each run, the best performing grid size was recorded and the results were converted to a bar diagram shown in the bottom part. Notably, the 8×8 grid achieved the shortest execution time in 9 out of 14 cases. This is also shown as a bar diagram in the bottom part of Figure 3a.

*2) **Impact of the Number of Junctions**:*

Objective: Examine the trade-off between employing many versus few junctions in a QCCD grid architecture. This evaluation seeks to determine how the number of junctions affects execution speed and, with that, the accessibility of ion chains in the memory zone.

Implementation: With the help of the compiler, we evaluated quantum circuits on architectures with varying numbers of junctions. We focused on two scenarios with two distinct architectures each: a 8×8 (6×6) grid with two ion chain positions between two junctions as the representative of a densely packed architecture against a sparser architecture built from a 4×4 (3×3) grid with 10 ion chain positions in between. In both scenarios, the two architectures (8×8 versus 4×4 and 6×6 versus 3×3) can hold roughly the same amount of ion chains. For each configuration, we tracked the execution time of the "register access" circuit on 10 to 120 ion chains.

Results: The results, depicted in Figure 3b, illustrate the execution times for all four architectures. The two architectures with a higher number of junctions significantly outperform the two sparser grids. The higher density of junctions decreases the number of blockages that occur in the shuttling operations and, thus, ion chains are able to more frequently avoid traffic.

*3) **Positioning of the Processing Zone**:*

Objective: Compare the effects of positioning the processing zone outside versus in the middle of the memory zone.

Implementation: The compiler was used to simulate two distinct setups: one with the processing zone centrally located within the memory zone, and one with the processing zone positioned outside of the memory zone. To perform this experiment, we extended the framework of the compiler to support the positioning of the processing zone inside the grid of the memory zone. To make room for the processing zone, we removed the linear regions in the middle of the grid. Integrating the processing zone symmetrically into the memory zone is only possible for odd grid sizes, which is why Figure Figure 3c only includes those. Instead of conducting runs with varying numbers of ion chains in the system, half of the available ion chain positions were filled for all runs. Similar to the earlier experiments, we used the "register access" circuit to investigate the accessibility of the ion chains.

Results: The only difference of the two runs is the position of the processing zone in relation to the memory zone. Figure 3c shows the results of the experiment: for all evaluated grid sizes, the architecture with the processing zone placed into the mid-part outperformed a processing zone at the border of the device with respect to the execution time.

*4) **Physical Swapping of Ion Chains**:*

Objective: Evaluate the effectiveness of physical ion chain swaps versus ion relocation in resolving blockages in the memory zone.

Implementation: In the last experiment, we use the compiler to compare two strategies: direct physical swaps of ion chains and compiler algorithms that reroute ions to circumvent blockages. Since the compiler removes blockages only by shuttling ion chains along cycles, we extended the framework to also support physical swaps. The algorithm then works in the same way as described above: the required ion chains are shuttled on their shortest path to the processing zone until blockages occur. Instead of creating the mentioned cycles, the considered ion chains swap positions and continue on their path.

Results: We again increase the grid sizes and fill half of the system's chain positions. Figure 3d shows the resulting execution times for two different circuits. We found differences between physical swapping and non-swapping for the "register access" circuit (top) and the GHZ state circuit (bottom), with the physical swapping showing an advantage in both cases.

The cycle-based compiler also includes an additional compilation step, that, after each gate execution, picks the best next gate in relation to the distance of the ions from the processing zone. In a second series of evaluations, we measured the impact of this step on the circuit of the GHZ state. The results are included in the plot in the bottom part. It shows that the advantage of physical swapping decreases significantly compared to the runs without the additional compilation step. This suggests that, in this case, improving the efficiency of the shuttling compiler can compensate the advantages of physical swapping ion chains in grid-type architectures.

VI. Conclusion

QCCD architectures promise to enable scalable quantum computing with trapped ions, yet the development of these devices demands software tools to optimize the design of the hardware. In this paper, we proposed the idea of extending the application of existing compiler frameworks to not only compile but also evaluate various QCCD architectures. We discussed four crucial design choices to illustrate potential use cases and conducted evaluations to discuss their impact. While this study is only a first approach, it reveals early trends in execution times for different grid architectures, which are critical for optimizing performance. These initial results already provide first recommendations and insights for future QCCD architectures. This showcases the potential of using existing compiler frameworks to evaluate different choices in hardware design of QCCD architectures. Motivated by this, future work includes studying a broader selection of architectures and exploring the benefits of using multiple memory and processing zones, as well as evaluating different performance metrics, such as phonon acquisition.

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 (FFG 896217) 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*, 1994, pp. 124–134.

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

[3] R. Babbush *et al.*, "Low-depth quantum simulation of materials," *Phys. Rev. X*, vol. 8, p. 011 044, 1 2018.

[4] M. Kjaergaard *et al.*, "Superconducting qubits: Current state of play," *Annual Review of Condensed Matter Physics*, vol. 11, no. 1, pp. 369–395, 2020.

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

[6] D. Bluvstein *et al.*, "Logical quantum processor based on reconfigurable atom arrays," *Nature*, vol. 626, no. 7997, pp. 58–65, 2023.

[7] S. Slussarenko and G. J. Pryde, "Photonic quantum information processing: A concise review," *Applied Physics Reviews*, vol. 6, no. 4, 2019.

[8] M. P. da Silva *et al.*, *Demonstration of logical qubits and repeated error correction with better-than-physical error rates*, 2024. arXiv: 2404.02280 [quant-ph].

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

[10] T. P. Harty *et al.*, "High-fidelity trapped-ion quantum logic using near-field microwaves," *Physical Review Letters*, vol. 117, no. 14, 2016.

[11] S. Debnath *et al.*, "Demonstration of a small programmable quantum computer with atomic qubits," *Nature*, vol. 536, no. 7614, pp. 63–66, 2016.

[12] 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.

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

[14] S. A. Moses *et al.*, "A race-track trapped-ion quantum processor," *Phys. Rev. X*, vol. 13, p. 041 052, 4 2023.

[15] 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.

[16] T. Schmale *et al.*, "Backend compiler phases for trapped-ion quantum computers," in *IEEE International Conference on Quantum Software (QSW)*, 2022.

[17] O. Keszöcze, N. Mohammadzadeh, and R. Wille, "Exact physical design of quantum circuits for ion-trap-based quantum architectures," *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 344–349, 2021.

[18] D. Schoenberger, S. Hillmich, M. Brandl, and R. Wille, "Using Boolean satisfiability for exact shuttling in trapped-ion quantum computers," in *Asia and South-Pacific Design Automation Conf.*, 2024.

[19] F. Kreppel *et al.*, "Quantum circuit compiler for a shuttling-based trapped-ion quantum computer," *Quantum*, vol. 7, p. 1176, 2023.

[20] D. Schoenberger, S. Hillmich, M. Brandl, and R. Wille, *Shuttling for scalable trapped-ion quantum computers*, 2024. arXiv: 2402.14065 [quant-ph].

[21] H. Kaufmann *et al.*, "Fast ion swapping for quantum-information processing," *Phys. Rev. A*, vol. 95, p. 052 319, 5 2017.

[22] M. W. van Mourik *et al.*, "Coherent rotations of qubits within a surface ion-trap quantum computer," *Phys. Rev. A*, vol. 102, p. 022 611, 2 2020.

[23] R. Wille *et al.*, "The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing," in *IEEE International Conference on Quantum Software (QSW)*, 2024. arXiv: 2405.17543, A live version of this document is available at https://mqt.readthedocs.io.

[24] M. F. Brandl, *A quantum von neumann architecture for large-scale quantum computing*, 2017. arXiv: 1702.02583 [quant-ph].

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