Visualizing Shuttling in Trapped-Ion Quantum Computers for Compiler Optimizations

Daniel Schoenberger
Technical University of Munich
daniel.schoenberger@tum.de

Robert Wille
Technical University of Munich
robert.wille@tum.de

***Abstract*—Shuttling-based trapped-ion quantum computers rely on compiler support to coordinate ion transport and minimize potential decoherence. Existing compilation approaches optimize either the execution time or the total number of ion moves required for a quantum circuit, but simultaneously minimizing both remains challenging. As a result, schedules may contain redundant or suboptimal trajectories that unnecessarily increase ion motion and phonon accumulation. We introduce a visualization tool that exposes complete shuttling schedules and makes routing behavior directly observable within multi-zone QCCD architectures. By inspecting the animated ion trajectories, we identified three recurring inefficiencies that previously remained hidden. Guided by these insights, we propose three routing logic optimizations: (O1) a dynamic partitioning adjustment, (O2) a priority-preserving selection strategy for movement cycles in the memory zone, and (O3) stricter coordination of movements between zones. Evaluations on two representative benchmark circuits show that these improvements reduce the total number of ion moves by 20–79% while preserving execution time, illustrating the value of visualization in advancing a more balanced compilation for shuttling-based trapped-ion systems.**

I. Introduction and Background

Shuttling-based trapped-ion quantum computers rely on compiler support to coordinate quantum gate execution and ion transport across the device. Current compilation approaches [1–9] focus on either minimizing the number of time steps or the amount of moves necessary to execute a quantum circuit. However, minimizing both time and movement at the same time remains difficult. Crucially, existing compilers offer no visibility into the actual shuttling trajectories they produce. Without visualization, redundant movement may remain hidden, routing instabilities appear only as minor variations in execution time, and inefficiencies moving between zones are difficult to detect. In this work, we present an interactive visualization tool that reveals the full shuttling schedule generated by the compiler and makes routing behavior directly observable. The resulting analysis uncovers three key inefficiencies that were previously undetected in the evaluation of [10]. Based on these insights, we refine the routing logic to significantly reduce unnecessary ion motion.

II. Visualization Tool

We integrated an interactive visualization module into the open-source *MQT IonShuttler* [11] to provide a detailed view of the shuttling schedules produced by the compiler. The tool displays both the physical architecture and all ion trajectories over time, allowing a targeted inspection of scheduling behavior inside and between zones.

Each schedule is represented as a sequence of per-timestep snapshots of ion positions. The device architecture is modeled as a graph following the abstractions in [10], representing a Quantum Charge Coupled Device (QCCD) architecture [12] with distinct zones. Ions occupy the edges of the graph, move between *processing zones* (PZs) and *memory zones* (MZs), while junctions are represented as nodes. Because movement between successive snapshots can be interpolated directly, the tool animates the transitions between timesteps, providing a continuous visualization of ion dynamics. Using only the ion-position data from the schedule, the tool can generate complete animations or videos of the corresponding shuttling processes.

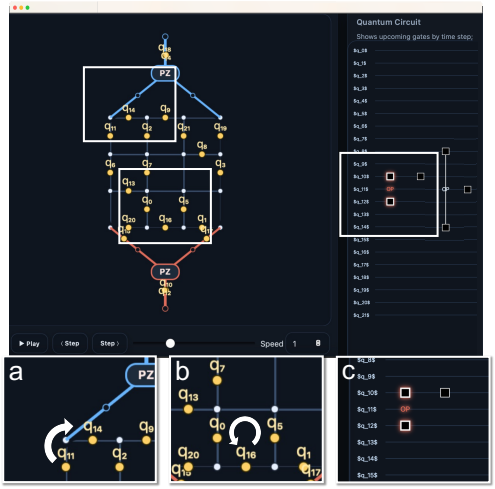

Fig. 1: Screenshot of the Visualization tool with examples:
(a) q_{11} could already move to the PZ in preperation for its future gate, yet q_{14} and q_8 are not in already in the PZ and their movement is blocked in this time step. **O1:** Since q_{14} and q_8 are needed earlier, q_{11} is also blocked.
(b) q_0 could move with all ions on its cycle, which would also move q_5 away from its path to the PZ. **O2:** Block cycle of q_0.
(c) q_{10} and q_{12} are executing a two-qubit gate in the red PZ, followed by a one-qubit gate on q_{10}. Since q_{10} is statically mapped to the blue PZ, it would have to move through the device again. **O3:** q_{10} is dynamically remapped.

To provide additional context on the efficiency of the compiled schedule, the tool also visualizes the circuit progress and corresponding gate execution times (Fig. 1). As the circuit advances in the animation alongside the architecture, the PZ edges executing each gate are highlighted, allowing users to see exactly when and where the compiler deploys gate operations and how ion movement aligns with gate timings.

The animation of the ions' paths makes it possible to directly identify inefficient or redundant movement patterns. With gate executions and circuit progress highlighted alongside the ion dynamics, the tool also reveals delays and unnecessary movements between zones. The following examples and evaluations explore this on devices with a grid-type MZ and two PZs outside the grid. More generally, the visualization tool supports architectures with multiple PZs, whether located outside [10] or embedded into the MZ [13].

III. Insights From the Visualization

Using the visualization tool, we identified three recurring sources of inefficiency in the shuttling schedules produced by the baseline compiler. For detailed examples, see Fig. 1.

TABLE I: Evaluation Results. Comparison of Baseline, individual optimizations (O1–O3), and all optimization effects combined. (M: Moves, t: Time Steps).

Grid	Ions	Baseline		Opt 1		Opt 2		Opt 3		Combined	
		M	t	M	t	M	t	M	t	M	t
Quantum Volume											
3×3	12	6845	2774	4181	2327	6055	2836	5094	2784	1835	2300
3×3	36	389,027	31,948	257,297	22,450	339,987	32,561	378,036	31,554	157,201	22,634
4×4	24	90,341	13,477	52,618	9749	67,438	13,524	81,039	13,457	20,427	9646
4×4	72	4,158,009	163,428	1,246,573	97,210	3,176,802	164,879	2,256,980	97,154	1,240,323	97,120
5×5	40	390,355	39,768	234,041	27,301	258,447	39,833	362,259	39,660	79,935	27,301
GHZ State											
3×3	12	520	214	528	220	392	223	298	214	313	231
3×3	36	9019	784	8917	779	7207	785	8972	784	6975	784
4×4	24	2308	488	2294	482	1506	482	2120	493	1335	482
4×4	72	31,429	1624	30,778	1621	25,183	1646	30,589	1616	24,946	1647
5×5	40	5426	768	5610	820	3999	841	5354	775	2969	836

1) **Unnecessary Movement between Zones**
   Although strict ordering was enforced within the MZ, ions were permitted to enter a PZ immediately upon reaching its entry edge to maximize throughput. The visualization showed that this often lets lower-priority ions enter prematurely and block those needed for the next gate, see Fig. 1a.
2) **Inefficient Cycle Selection**
   The compiler moves cycles of multiple ions to remove blockages. This cycle-based approach occasionally displaces ions that were required for upcoming operations, generating movement churn inside the MZ without contributing to circuit progress, see Fig. 1b.
3) **Inefficient Partitioning**
   The static partitioning approach of the compiler maps each qubit to a specific PZ to execute corresponding one-qubit gates. To also leverage benefits of a more dynamic approach, two-qubit gates are dynamically mapped to available PZs. The visualization revealed that conflicts between these two approaches frequently caused ions to shuttle back and forth between PZs unnecessarily, see Fig. 1c.

These insights motivated the compiler optimizations presented in the following section.

IV. Compiler Opimtizations

The insights obtained from the visualization directly guided three targeted optimizations to the routing logic, each resolving one of the inefficiencies identified in the previous section.

O1: Stricter Movement Logic Between Zones
To mitigate congestion at PZ boundaries, the strict ordering already used in the MZ is extended to the entry of each PZ. Ions may enter a PZ only after all higher-priority ions have already entered. While this may slightly reduce throughput, the resulting decrease in ion movement outweighs the small impact on the time step count.

O2: Robust Cycle Selection
To reduce movement churn in the MZ, cycles are restricted from unintentionally moving higher-priority ions even if this would not violate other constraints. This prevents redundant cycles, further preserving the order of the ions.

O3: Improved Partitioning
To address the repeated back-and-forth movement caused by conflicting static and dynamic partitioning decisions, ions involved in a two-qubit gate are now remapped to the selected PZ of the two-qubit gate. This refinement aligns the static mapping with the dynamic gate assignments and prevents unnecessary transfers between PZs.

These optimizations collectively reduce redundant motion across both the MZ and PZs while maintaining compatibility with the original scheduling framework.

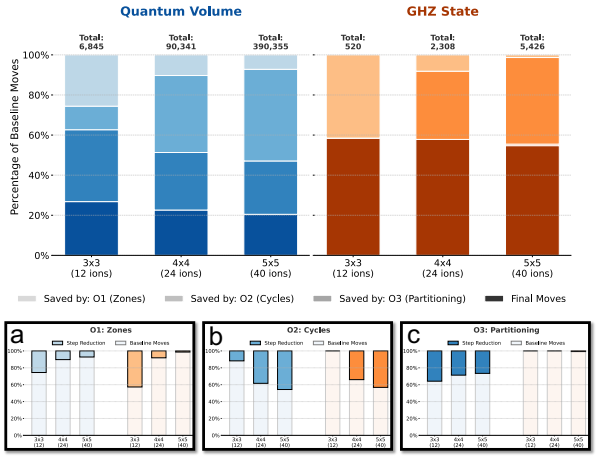

Fig. 2: Reduction of Ion Movement. Top: Cumulative impact of the routing optimizations on the total move count. Bottom: (a)–(c) Result of each individual optimization step.

V. Evaluation

We evaluated the optimizations on Quantum Volume (QV) [14, 15] and Greenberger–Horne–Zeilinger (GHZ) circuits (see Table 1 and Fig. 2). The combined refinements reduced ion movement by 45–79% for QV and 20–45% for GHZ across various grid sizes. While GHZ execution times remained stable (partly due to a baseline "best gate selection" pass) the QV benchmarks showed a simultaneous 17–47% reduction in time steps, indicating that the removed shuttling operations were actively congesting the architecture.

VI. Conclusion

We introduced a visualization tool for trapped-ion shuttling schedules to uncover hidden compilation inefficiencies. These insights guided three compiler optimizations that reduced ion movement by 20–79% across benchmarks while preserving execution time. Beyond these specific improvements, our results demonstrate that integrating visualization-driven feedback into future compiler pipelines enables balanced optimization of time and movement.

Acknowledgments

This work was supported by the EU's Horizon 2020 programme (DA QC, Grant No. 101001318; MILLENION, Grant No. 101114305), the State of Upper Austria via the COMET programme, the QuantumReady project under Quantum Austria (FFG), and the Munich Quantum Valley, funded by the Bavarian government through Hightech Agenda Bayern Plus.

References

[1] P. Murali et al., Architecting noisy intermediate-scale trapped ion quantum computers, *ISCA*, 2020.
[2] O. Keszöcze et al., Exact physical design of quantum circuits for ion-trap-based quantum architectures, *DATE*, 2021.
[3] A. A. Saki et al., Muzzle the shuttle: Efficient compilation for multi-trap trapped-ion quantum computers, *DATE*, 2022.
[4] T. Schmale et al., Backend compiler phases for trapped-ion quantum computers, *QSW*, 2022.
[5] F. Kreppel et al., Quantum circuit compiler for a shuttling-based trapped-ion quantum computer, *Quantum*, 2023.
[6] C.-M. Chang et al., Quantum circuit compilation for trapped-ion processors with the drive-through architecture, *IEEE Transactions on Quantum Engineering*, 2025.
[7] D. Schoenberger et al., Using Boolean satisfiability for exact shuttling in trapped-ion quantum computers, *ASP-DAC*, 2024.
[8] D. Schoenberger et al., Towards Cycle-based Shuttling for Trapped-Ion Quantum Computers, *DATE*, 2024.
[9] D. Schoenberger et al., Shuttling for scalable trapped-ion quantum computers, 2024. arXiv: 2402.14065.
[10] D. Schoenberger et al., Orchestrating multi-zone shuttling in trapped-ion quantum computers, 2025. arXiv: 2505.07928.
[11] R. Wille et al., The MQT Handbook: A Summary of Design Automation Tools and Software for Quantum Computing, *QSW*, 2024. arXiv: 2405.17543, A live version of this document is available at https://mqt.readthedocs.io.
[12] D. Kielpinski et al., Architecture for a large-scale ion-trap quantum computer, *Nature*, no. 6890, 2002.
[13] D. Schoenberger et al., Shuttling for trapped-ion quantum computers with embedded processing zones, *QSW*, 2025.
[14] A. W. Cross et al., Validating quantum computers using randomized model circuits, *Physical Review A*, no. 3, 2019.
[15] C. H. Baldwin et al., Re-examining the quantum volume test: Ideal distributions, compiler optimizations, confidence intervals, and scalable resource estimations, *Quantum*, 2022.