Beating Casey with NATZ: the Neutral Atom Transpiler for Zones

Quinn Manning[1], Dmitrii Khitrin[1], Adrian Harkness[1], Mihir Talati[1], and David Nizovsky[1]

[1]Quantum Consortium

February 2025

Abstract

Compiling circuits on monolithic neutral atom computing architectures often assumes the ability to execute arbitrary single-qubit gates. In this iquHACK challenge, single-qubit gates could not be neglected from compilation because atom shuttling to gate zones was required for these rotations. In this work, we build upon a state of the art neutral atom compilation toolkit for zoned architectures, ZAC [1], enabling it to track and optimize single qubit rotations during compilation. We then benchmark our work on a selection of circuits.

1 Introduction

Given a native gate set of R_z, R_{xy} and CZ and constraints on atom shuttling movements forbidding crossing paths in any given time step, our goal is to develop a compiler for neutral atom quantum computing architectures that optimizes the following cost function:

$$E \equiv \alpha\big(N_{G,z} + N_{G,xy}\big) + \beta\big(N_{L,z} + N_{L,xy}\big) + \gamma\, N_{\mathrm{CZ}} + \delta\left(\frac{T}{T_0}\right) + \eta\, N_{\mathrm{pick}}$$

With parameter values:

$$\alpha = 0.02, \quad \beta = 0.5, \quad \gamma = 1.0, \quad \delta = 0.6, \quad \eta = 0.8.$$

Here,

- $N_{G,z}$, $N_{G,xy}$ are the number of *global* 1-qubit gates (rotations around Z and an axis on the XY-plane).
- $N_{L,z}$, $N_{L,xy}$ are the number of *local* 1-qubit gates (rotations around Z and an axis on the XY-plane).

- N_{CZ} is the number of control-Z gates.
- N_{pick} is the number of times atoms are picked or dropped.
- T is the aggregate time of the circuit, normalized by $T_0 = 100\,\mu s$.

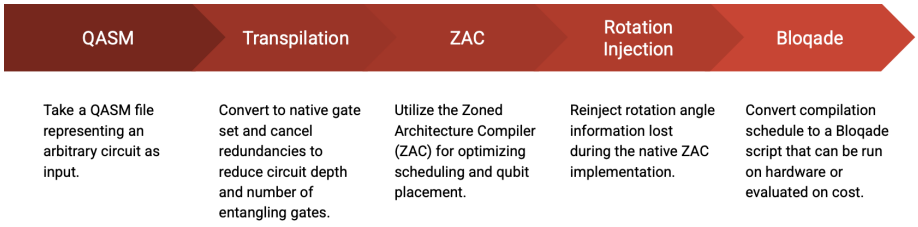


Figure 1: Our automated workflow for optimized compilation of arbitrary circuits to a zonal neutral atom architecture.

Our approach to this challenge was to develop an automated workflow for optimizing compilation for arbitrary input circuits, as outlined in figure 2. This connects circuits expressed in a QASM format to an optimized Bloqade program compiled to the hardware architecture.

We will first describe the intuition we gained from working through simple examples by hand.

2 Setting the Stage

Our manual solution to the first circuit in the first challenge is illustrative of the steps we took to build our final compiler. A CZ gate is performed on qubits 1 and 2 and then a CX is performed on qubits 2 and 3. Our first pass at compiling this circuit was the following procedure:

1. Shuttle qubits 1 and 2 to the same interaction site in the gate zone.
2. Perform a global Rydberg pulse to execute a global CZ.
3. Pick up qubit 3 and shuttle it to a separate interaction site.
4. Pick up qubit 2 with AOD and shuttle it to the interaction site containing qubit 3.
5. Perform a local Hadamard with R_z and R_{xy} on qubit 2.
6. Perform a global Rydberg pulse.
7. Perform a local Hadamard on qubit 2 again.

This is suboptimal and yielded a score of 11.5. We made note of two important points: (1) neighboring gates can be transformed to reduce pulse count and (2) qubits can be reused in the gate zone as they await their next interaction.

First, notice that the Hadamard obeys the equations $R_y(-\pi/2)Z = H = ZR_y(\pi/2)$. If we surround a CZ gate at the target with two Hadamards on either side, since Z commutes with itself and $Z^2 = I$,

$$HZH = R_y(-\pi/2)ZR_y(\pi/2)$$

which reduces the total pulse count by 2 and our cost function by 1. We use Cirq to transform our input QASM circuits in order to take advantage of these commutation relations and decrease the number of pulses.

Second, notice that qubits can be held in the gate zone as long as they do not conflict with necessary CZ operations. In this case, holding qubits results in a 1.5x decrease in aggregate circuit execution time when using the following strategy:

1. Shuttle qubits 1 and 2 to an interaction site and, simulatneously, qubit 3 to a separate interaction site.
2. Perform a global Rydberg pulse to execute a global CZ.
3. Pick up qubit 2 and shuttle it to the interaction site with qubit 3.
4. Perform a local $R_{xy}(-\pi/2, \pi/2) = R_y(-\pi/2)$ on qubit 2.
5. Perform a global Rydberg pulse.
6. Perform a local $R_{xy}(\pi/2, \pi/2) = R_y(\pi/2)$ on qubit 2 again.

This yields a score of 7.27. ZAC [1] refers to holding atoms in the gate zone as qubit reuse and we build on their approach after realizing the power in optimizing this parameter.

3 Existing Solvers

When researching existing DPQA compilers we found Enola and ZAC to be of particular interest. both utilize similar stages of placement, scheduling and routing. Both use similar approaches to routing and scheduling. ZAC builds on Enola's approach of using optimal edge coloring algorithms for scheduling, as well as using simulated annealing (with routing formulated as a two-pin problem familiar to VLSI) to optimize routing. ZAC provides a distinct advantage for our usage in that it allows us to construct the zoned architecture formulated in the problem statement, in addition to employing polynomial time algorithms for the initial placement of qubits in the storage array that can prove to be critical to speed.

4 Single-Qubit Gates

The biggest issue faced when employing the solvers mentioned previously was their lack of support for compiling single qubit gates. Enola entirely dropped all single-qubit gates from an input circuit and compiled only the remaining 2-qubit gates, while ZAC compiled both sets of gates but entirely dropped all information about any rotation angles in its output json file. Any single-qubit gate in ZAC's output is simply refered to as a general U3 gate, which can refer either to R_z or to R_{xy} rotation axes. This meant that we needed to reinject rotation axis and angle information to the ZAC output before we could convert it to a Bloqade program and evaluate the cost function. To achieve compilation we modified ZAC to output U3 gate parameters as a part of the json file. We then compiled U3 gates down into rz and xy gates by first decomposing them into rz,ry,rz triplets and then folding sequential rz and ry gates on the same qubit lines together. Unfortunately this decomposition and recomposition step is where we encounter many of the bugs and errors that prevent the pipeline from completing fully automatic as it stands.

5 Circuit Transpilation

5.1 Circuit Transformation Using Cirq and Qiskit

We first use Cirq to transform the input QASM circuit before passing it to ZAC's Qiskit transpiler.

To begin we first use cirq for initial optimization and staging. We do this through the use of a custom transformer pipeline that mixes, expands,transpiles and collapses the unitaries. We begin by first merging all connected components of unitary operations acting on 2 or less qubits to fold in all single and two qubit gates in the circuits. We then transpile and optimize the collapsed circuit for an auxiliary to our target gateset using the native CZTargetGateset. This collapses the gates to an efficient series of cz and single qubit gates which is the required input for later stages of the compilation routine. The penultimate step before we begin optimizing and calculating routing and placement is to stratify the circuit such that all similar gates are grouped together. THis allows later stages to keep qubits at the gate zone for sequential pulses of similar gates, minimizing excess "touches" and movement.

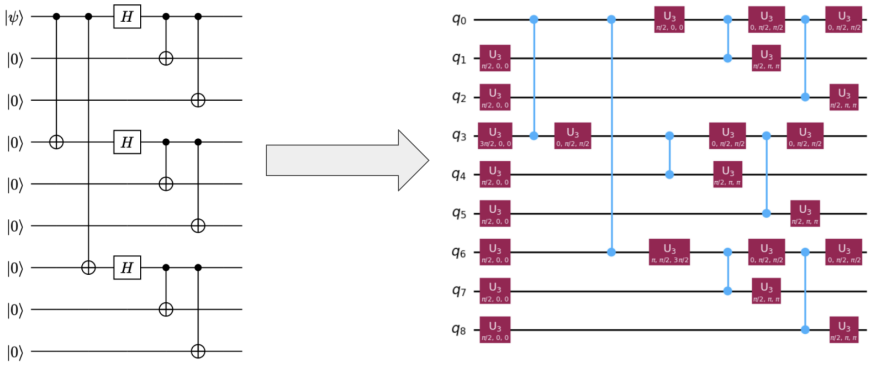
To deconstruct a generic quantum circuit, we first constrain the gates to a set, $[R_y(\phi), R_{xy}(\alpha, \theta), CZ]$, which can be implemented on Aquila via the Qiskit transpiler package. The plugin allows us to use the Qiskit decomposition and optimizer, which optimizes any circuit around our native neutral atom gates. It first takes any non-native gates, such as $CNOT$, and decomposes them in terms of the native gates, before then optimizing around extra rotations. We are open to both a generic backend option, as well as custom backends the users can make for their personal systems.

Once we have a new circuit in terms of native gates, we seperate the single

qubit and two-qubit gates, before sending the 2 qubit gates into the SMT solver, and we re-insert the single qubit gates afterwards. We note that a strong advantage of neutral atom computers is the ability to perform controlled z gates on more than 2 qubits at a time, however we omitted this feature to use the SMT solver in this project.

Rz is also omitted due to not being easily implemented via Raman pulses. Instead of pursuing a separate method of generating a native R_z, we, for now, stick purely to decomposing R_z in terms of $[R_x, R_y]$

We have included an example decomposition of a 3 qubit W state circuit:



6 ZAC Configuration, Single-Qubit Gates, and Translators

To utilize ZAC's zoned compilation, we define three sets of SLMs: the storage zone SLM_0 and the gate zone SLM_1 and SLM_2. Each SLM consists of one row and several columns to obey the requirements of the challenge. SLM_1 and SLM_2 are offset slightly to create pairs of SLM sites each separated by a distance $d < r_b$ where r_b is the characteristic Rydberg blockade radius. These pairs are separated by $d > 2.5r_b$ to limit undesired interactions and enable qubit reuse in the gate zone. AODs are then defined for each SLM row to enable the compiler to execute atom movements.

Within our pipeline, ZAC is fed the Cirq transformed circuits as QASM circuits and a JSON is generated of compiled atom movements and two qubit gates.

One pitfall we noticed while implementing ZAC was the erasure of single-qubit rotation information which was not documented or tracked during Qiskit circuit transformation nor compilation. We extracted optimized single-qubit gate parameters from the input QASM circuit and followed their evolution during compilation. At the end, we inserted these single-qubit gates into the final compilation for execution on Bloqade. We call this slightly enhanced ZAC: NATZ.

With our addition of single-qubit gates, we successfully derived solutions

for all five challenges optimizing for aggregate circuit time and pulse number. The circuits and animations of compiled atom movements can be found in the GitHub.

To automate the translation of ZAC's compilation to Bloqade, we execute a pipeline to ingest the returned JSON which encodes atom movements, two qubit gates, and, with our implementation, single-qubit gate parameters.

7 Performance

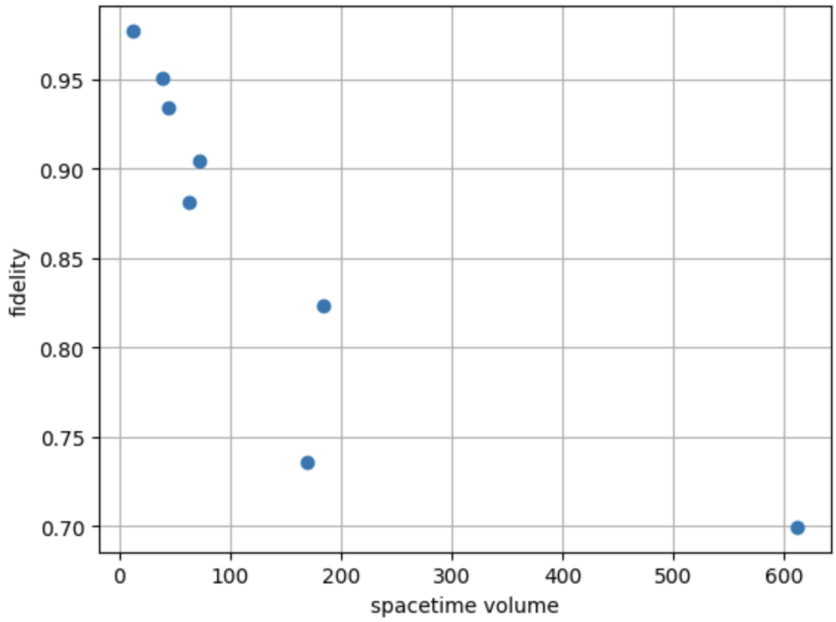


Figure 2: Fidelity of our compiled circuit over the number of qubits times the depth of the circuit.

Figure 2 illustrates our compiler's performance over the number of qubits times of the depth of the circuit (the circuit spacetime volume). Fidelity was calculated using

$$f = (f_1)^{g_1} \cdot (f_2)^{g_2} \cdot (f_{exc})^{|Q|S-2g_2} \cdot (f_{trans})^{N_{trans}} \cdot D,$$

where $f_1, g_1, f_2, Q, S, g_2 f_{trans}, N_{trans}$ and D are the fidelity of the single qubit gates, the number of single qubit gates, the fidelity of two qubit gates, the number of qubits, the number of circuit stages, the number of two-qubit gates, the fidelity of atom shuttling, the number of times atoms are shuttled, and the decoherence. The $\log f$ is roughly proportional to the cost function and gives us an approximation of how the cost function might scale with circuit depth.

8 Conclusions

Our initial approach to solving the first circuit led to our focus on reducing the pulse number and decreasing aggregate circuit time through qubit reuse. We identified ZAC as a state-of-the-art compiler for zoned architecture and encoded our two-row hardware constraint to effectively generate solutions to all of the circuits in the challenge. We developed a pipeline to translate these solutions to Bloqade implementations. It will be interesting to benchmark these solutions against manually derived compilations and against compilations that are optimized for either pulse number or aggregate circuit time.

References

[1] Wan-Hsuan Lin, Daniel Bochen Tan, and Jason Cong. *Reuse-Aware Compilation for Zoned Quantum Architectures Based on Neutral Atoms*. 2024. arXiv: 2411.11784 [quant-ph]. URL: https://arxiv.org/abs/2411.11784.