The kernel of the problem: Bloqade circuits from 0 $\rightarrow$ Q

Neutral-atom quantum computing is advancing at an incredible pace. The era of analog devices is giving place to gate-based computers and, in particular, error correction protocols. Much of this is due to the amazing compilation design space of this platform, born out of the flexibility of entangling arbitrary pairs of qubits, a high degree of gate parallelization, as well as a wide variety of native gates.

To support this uncanny speed of change, bespoke tools are necessary. Bloqade has been born in 2022 as the go-to language for simulating and programming neutral-atom quantum hardware. At that time, Bloqade's resources supported analog workflows, the only type of quantum computing available with neutral atoms then.

We welcome the tides of 2025 with a new version of Bloqade, built on QuEra's also new compiling tool-chain Kirin designed for kernel functions and composable representations. With these advances, Bloqade's infrastructure has been enhanced to be able to represent gate-based quantum computing workflows with a focus on neutral-atom quantum hardware. This includes extending QASM programming to accept `annotations`, `for` loops, `if` statements and, overall, all the tooling needed to efficiently represent parallelizable and global gate structures in quantum circuits.

This challenge is an invitation for you to engage with the new Bloqade, which remains an open-source package, and rekindle your passion for the idiosyncrasies of neutral-atom quantum computing. You will work as an architect and, by the end of this challenge, we expect you to develop expertise on the unique design rules for circuits with neutral-atom devices, on how to formulate your favorite circuits as efficiently as possible with Bloqade,and on showcasing how powerful kernel structures can be to co-design circuits for hardware awareness.

Challenge statement

- **Develop a tutorial on the deployment of your favorite quantum algorithm, subroutine, or circuit in a neutral-atom quantum hardware using the kernel features of Bloqade.**
  - example ideas: QAOA, QFT, multipliers, Trotterized evolution, error correction codes, state injection.
- **Display how specific design rules of neutral-atom systems influence your circuit layout, and how its performance can be affected by the architect's choice (testing against our pre-defined noise model).**
- **Extra points if you develop new components or dialects on Bloqade that allow users to more directly reach their applications of interest, visualize results and processes, etc.**

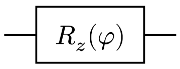
Design Rules

We have a list of references that can help you understand some details of neutral-atom gate-based devices [1], [2], [3], as well as some neat tricks for circuit rewrites [4]. You can access these for free on the open-access repository arXiv. Use these sparingly, as the time is tight for the challenge.

Here goes the summarized set of design rules we will use for our challenge. When in doubt, all options and possibilities defer to the constraints defined here.

Native Gate Set

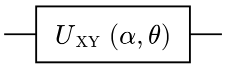
Neutral-atom quantum computers have some flexibility with regards to their native gate set, and a given choice depends on the specific hardware implementation. We will consider the processor to be functionalized in two areas: (i) a storage zone and (ii) an entangling zone (details below). Building on that, we will consider the following scenario:



Z-rotation

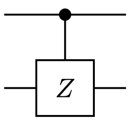
Z-rotations can be done globally (i.e. on all qubits for the same angle) or locally (i.e. on a sub-selection of qubits, with the same angle for all qubits in this subset). The choice of global or local rotations impart different errors in the qubits, with global rotations being strongly favored. In fact, global Z-rotations are substituted for simple delays in phases of subsequent gates, so they are free of errors and cost effectively zero time.

Global **and** local 1-qubit gates can be done without moving the atoms, and we will assume that can affect them both in the storage and entangling zones. Read more about our zones and geometric constraints below.



XY-rotation

Similar to the Z rotations above, we can do rotations around any axis in the XY-plane both globally or locally. These rotations are defined by an arbitrary angle α on the plane, setting the rotation axis at an angle α in reference to the x-axis, with the rotation angle defined by θ. Once more, global operations are favored over local ones.



The standard entangling gate for us will be the controlled-Z gate. To apply the entangling gate, atoms have to be put in proximity of each other. This is why the gate zone (below) is structured with clustered pairs of sites. The ability to shuttle atoms means you can pick any two atoms in the storage zone regardless of their distance apart and move them to the proper pair of sites in the gate zone. Furthermore, you can pick up multiple atoms and put them in the gate zone *in parallel* as long as you respect our shuttling constrain rules below.

Importantly, we will assume here that every time a set of parallel CZ operations are applied, the corresponding atoms will be moved to the entangling zone, and then moved back to their respective original positions. *No shuffling of register address will be allowed.*

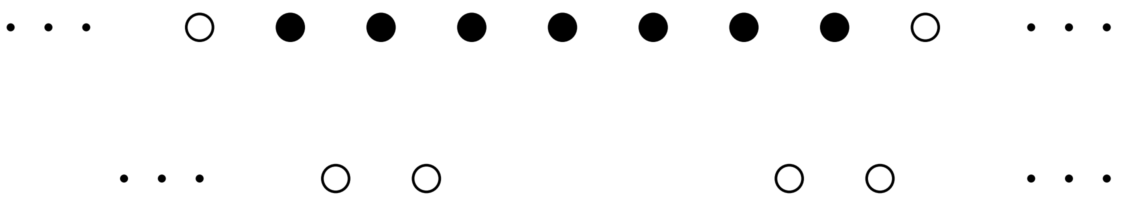
Native gates and Bloqade

The set of gates above can be summarized via general U3$(\theta, \varphi, \lambda)$ - parallelizable - gates, and our trusty CZ. This gate set can be accessed on Bloqade via `parallel.u` and `parallel.cz`, primitives that are able to incorporate lists of targets to be deployed jointly as needed. Note that Bloqade also has an automated rewrite capable of converting any circuit to this gate-set.

That can be accessed by `from bloqade.qasm2.rewrite.native_gates import RydbergGateSetRewriteRule`; an example of using it for automated transpilation is provided in our exemplary tutorial materials based on GHZ state preparation. Look for it on the challenge folders.

Register Geometry
To keep things simple, in this challenge we will consider a simplified version of a neutral-atom quantum processor. It will be defined by two zones: a **storage zone** with atoms about $4\mu m$ apart, and a **gate zone** with pairs of $20\mu m$ apart. The distances are not terribly important for this challenge. Both of them are chosen to be 1-dimensional straight lines.

Here is an illustration to help you make the description above concrete:

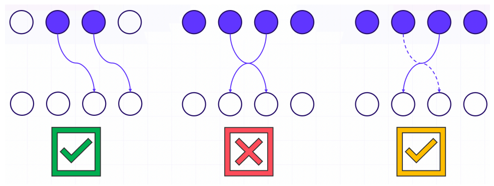


The first row with black dots correspond to the storage zone and contains your atoms. Once you define the number of qubits in your register, a line segment of host spots *will* be filled with qubits; atoms in this zone can go through 1-qubit rotations, either local of global. The second row with pairs of white spots correspond to the gate zone. Atoms from the storage zone have to be moved to these white spots when you want to entangle them. After that, they move back to the storage zone. Once registers have been defined, we will assume (unrealistically) infinite extensions of the zones to either sides of the original register. This will keep the spreading of moves between zones simple for automatic compilation and error modeling.

Shuttling Dynamics
Shuttling will happen whenever a CZ gate is called. Atoms will move to meet their pairs in the entangling zone and after the gate has been applied, *they are assumed to be moved back to their original positions.* In this challenge, shuffling qubit labels, or their storage positions, will not be allowed. Yet, entangling arbitrarily distant qubits is allowed.

Ensuring that the circuit will minimize the number of moves when entanglement is called is a job for the hackers. Here are the two simple rules which must be respected and will be constraining your atom-moving strategies:

1. A single move operation may pick up any subset of atoms from the storage zone and then place those atoms in another row in the entangling zone under the constraint that the atoms are not reordered during the move. Shortly, the spots that pick up and drag atoms cannot cross paths. Crossings can be done in series though, but that costs more moves.



2. Atoms are not allowed to collide. A spot with one atom cannot get a second one.

Error budget
In this challenge, you will not be judged directly by the performance of your final circuits, as much as by the quality of your tutorial exposition. Still, showcasing the features of Bloqade and of good, hardware-aware, circuit design implies being able to compare performances of different deployments.

For our purposes, Bloqade already contains channels for Pauli errors and atom losses. These can be applied randomly to a circuit and their effect can be tracked by sampling iterations of operations. From those, fidelities or KL divergences can be computed.

We will work with a simple hierarchy of errors:

$$E_{\text{CZ}} > E_{1-\text{qubit},L} > E_{1-\text{qubit},G}$$

That is, CZ errors - which include pick/drop errors from atom moves, which induce losses,etc - are worse than local 1-qubit gates, which are worse than global 1-qubit gates. Furthermore, decoherence is taken into account and an incentive exists to parallelizing your circuit as much as possible, reducing its total run time.

Judging criteria

Performance will be judged based largely on write-ups, and then your presentations. Because of that, you have a deadline of **8am Sunday** deadline for a pre-write-up (so judges can get acquainted with your work accordingly).

Some other items that we will be keeping an eye on include:
- How you demonstrate the value of parallelization, for example scoring circuits against Bloqade's heuristic error model
- How you explain the use of high-level features of Bloqade - recursion, for loops, if statements, maps - to simplify circuit building
- Bonus points for bug reports
  - Bonus^2 if you fix the bug!

Sub-modules

Bloqade contains several sub-modules, domain-specific languages, that are made to improve the experience of users when abstracting specific workflows in quantum algorithm design and implementation.

In this hackathon, we will focus on the initial packages that support workflow for gate-based quantum computing. These include:

`bloqade.qasm2` - this module allows the generation of QASM files and contains two dialects: `core` and `extended`. The former fixes the output QASM to the standard version which can be ingested by any other quantum programming language. The latter, however, contains new features that enable your program to be parsed by a neutral-atom quantum computer to take advantage of parallelization and other features

`bloqade.pyqrack` - this module contains a state vector simulator used for analysis of circuit performance

`bloqade.noise` - finally, this is the module you will want to use if you want to study the effects of a heuristic noise model on your circuit. This model will evolve as our technology evolves!

Example tutorial

As alluded through the text, we have prepared a tutorial demo to serve as a baseline for you. It showcases some Bloqade features and uses them to explain how to build GHZ state preparation circuits in log-depth, and simulate them. Use this as a baseline for your own tutorials, to contextualize on Bloqade's unique functionalities for neutral atoms, and start developing a feeling for which are your favorites!

Bibliography

[1] D. Bluvstein *et al.*, "A quantum processor based on coherent transport of entangled atom arrays," *Nature*, vol. 604, no. 7906, pp. 451–456, Apr. 2022, doi: 10.1038/s41586-022-04592-6.

[2] D. Bluvstein *et al.*, "Logical quantum processor based on reconfigurable atom arrays," *Nature*, vol. 626, no. 7997, pp. 58–65, Dec. 2023, doi: 10.1038/s41586-023-06927-3.

[3] P. S. Rodriguez *et al.*, "Experimental Demonstration of Logical Magic State Distillation." [Online]. Available: https://arxiv.org/abs/2412.15165

[4] W. Schober, "Extended quantum circuit diagrams." [Online]. Available: https://arxiv.org/abs/2410.02946