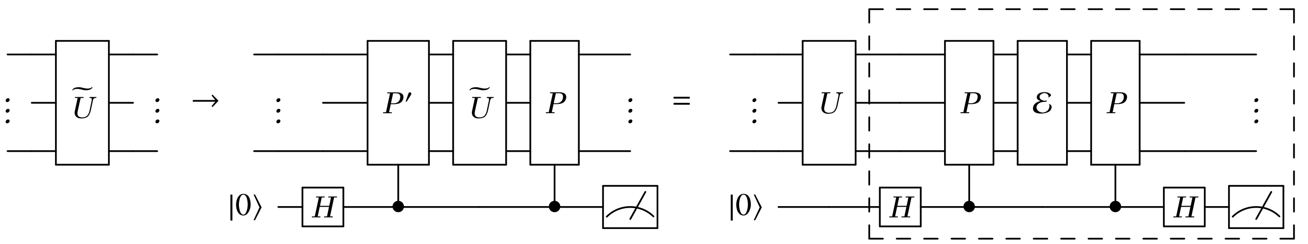
Hack the North 2022: Quantum Error Mitigation
Presented by Zapata Computing **September 17, 2022**

Welcome to Hack the North!

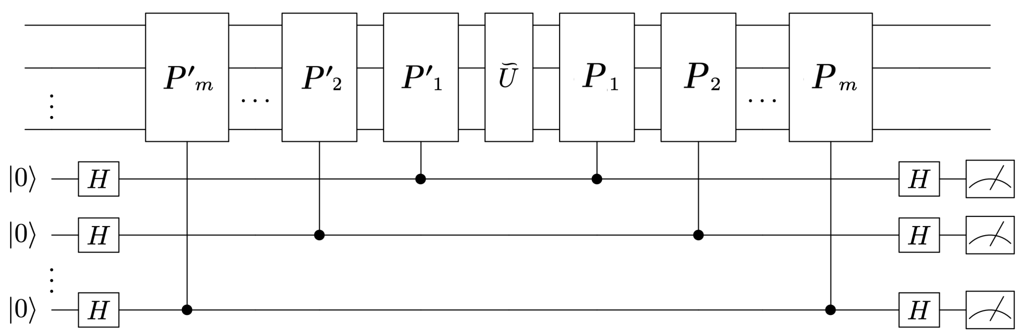
The main limitation quantum computers face is that qubits are often too noisy to perform meaningful computation. In response scientists developed a number of methods to decrease noise in near-term quantum computers, collectively called **Error Mitigation Techniques.**

In this challenge, you will implement a recently developed error mitigation technique by Gonzalez et al. called **Pauli Sandwiching** (arXiv doi: 2206.00215). Operations on a quantum computer is represented by a matricies. The basic idea of Pauli Sandwiching is to catch errors on an operation U using a set of Pauli operators. We then repeat the quantum circuit until we don't catch an error. The procedure takes in a noisy operation U and a Pauli gate P and catches all errors on U which do not commute with P. By repeating the procedure with several different Ps we can catch all possible errors on U.

To be precise, assume that we can write $\widetilde{U} = U\mathcal{E}$ for some noise model $\mathcal{E}$. Furthermore, assume that $P' = UPU^{-1}$ for some other Pauli P' (i.e. assume U is a Clifford gate). The Pauli Sandwiching protocol finds each instance of U in the circuit and "sandwiches" U between two paulis which are controlled by an ancilla (an introduction to controlled gates can be found here).



With a little linear algebra, one can show that the ancilla qubit will be 0 if $\mathcal{E}$ commutes with P and 1 otherwise. This can be done by calculating the output of the ancilla As an example, you can pick U to be a Hadamard gate, $\mathcal{E} = X$, $P = Z$, and $P' = X$ to show that the ancilla will always be 1. We can **repeatedly sandwich** U between Paulis to catch more errors.



One can show sandwiching U between all of the X's and Z's is sufficient to catch any error on U.

For this challenge you will implement a Zapata `QuantumBackend` called `PauliSandwichBackend` to perform Pauli sandwiching on a given gate in a circuit. Zapata's `Circuit` objects contain an `operations` property which contains a list of `GateOperations` which the quantum computer runs. Your backend will iterate through a given `Circuit` and replace any instance of U in it's `operations` with the appropriate controlled operations "sandwiching" U. It will then run this new circuit with a given backend.

Here is a general idea of what your code should look like:

```
class PauliSandwichBackend(QuantumBackend):
    def __init__(self, U, bread_gates, inner_backend):
        # define attributes for U, bread_gates, and inner_backend

    def run_circuit_and_measure(self, circuit, n_samples):
        data_qubit_indices = tuple(range(circuit.n_qubits))
        new_circuit = Circuit([])
        n_sandwiches = 0

        # create and run sandwiched circuit
        for operation in circuit.operations:
            if operation.gate is self.U:
                for P in self.bread_gates:
                    n_sandwiches += 1
                    op_indices = operation.qubit_indices
                    control_qubit_index = circuit.num_qubits + n_sandwiches
                    controlled_P_qubits = (control_qubit_index,) + data_qubit_indices
                    # sandwich U between controlled operations
                    Pprime = U(*op_indices) * P * U.gate.dagger(*op_indices)
                    new_circuit += Pprime.gate.controlled(1)(*controlled_P_qubits)
                    new_circuit += operation
                    new_circuit += P.gate.controlled(1)(*controlled_P_qubits)
            else:
                new_circuit += operation
        raw_meas = self.inner_backend.run_circuit_and_measure(new_circuit, n_samples)

        # eliminate runs in which an error was caught
        raw_counts = raw_meas.get_counts() # get dictionary of outputs
        sandwiched_counts = {}
        for key in raw_counts.keys():
            if "1" not in key[circuit.n_qubits:]:
                sandwiched_counts[key[:circuit.n_qubits]] = raw_counts[key]
        return Measurements.from_counts(sandwiched_counts)
```

To demonstrate your new `PauliSandWichBackend` and factory, you can use a `qiskit` noise model given in this repository. Showing that your backend can decrease the errors on the noisy Clifford gates will indicate a successful implementation!

Deliverables:

- A `PauliSandwichBackend` which implements the Pauli Sandwiching technique.
- Bonus! Add typing to your function.
- Super Bonus! Update our documentation, provide tests for your `PauliSandwichBackend`, and have your code follow our style conventions. See our contributing guide for details.
- Ultra Bonus! If U is a Clifford, make `run_circuit_and_measure` in the complexity class **FP**.

To submit your work, fork `orquestra-quantum` and send a copy of your solution to my email.

Happy Hacking!

Zapata Quantum Software

For a full list of opportunities at Zapata, visit our careers page!