Project Report

Department of Computer Science
Indian Institute of Technology, Hyderabad

Name:	Kartheek Tammana, Kushagra Gupta, Rishit Devalacheruvu
Student Number:	cs21btech11028, cs21btech11033, cs21btech11053
Course:	CS5863: Introduction to Program Analysis & Optimization
Project Title:	Conditional Quantum Program Optimizations
Primary Supervisor:	Dr Jyothi Vedurada

1 Summary

This project aims to explore the potential and viability of compiler optimization on classically-controlled quantum operations in a general quantum program. The project began with a deep analysis into the `qiskit` [Javadi-Abhari et al. (2024)] framework, its existing transpiler passes, and its shortcomings in capturing compiler optimization on classically-controlled quantum circuits. The second phase aimed at understanding existing compiler techniques and new passes proposed in recent literature to improve the aforementioned class of circuits. The final phase of the project involved in the development and implementation of novel passes to improve circuit complexity.

2 Introduction & Motivation

Quantum programs serve as a gateway to realize quantum algorithms alongside classical interpretations. In most programs, there is considerable interleaving of classical and quantum operations which are highly dependent on each other. One may summarize the flow of quantum programs as follows:

- Fetch quantum circuit from program and optimize appropriately.
- Load the circuit on the quantum device and execute it.
- Fetch measurement results from the device and run classical post-processing.
- Update the quantum circuit with the classical results and repeat.

These models are especially useful for variational quantum algorithms, quantum neural networks, quantum error-correction etc.

2.1 Conditional Quantum Operations

Conditional quantum operations are a class of quantum operations that depend on the outcome of a classical operation. These operations are crucial for implementing quantum algorithms that require feedback from classical computations. For example, in a variational quantum algorithm, the outcome of a measurement may determine the next set of parameters to be used in the quantum circuit.

Deferred Measurement: Deferred Measurement [1] is a quantum mechanical principle that allows us to postpone measurement until a later stage in the quantum circuit. This cannot work in situations where the quantum circuit re-evaluates measured registers as measurements *do not necessarily commute* with quantum gates.

Motivation: Most of the functionalities in quantum programming languages including `qiskit` and `cuda-quantum` ignore or support only minimal compiler optimizations for classically-controlled quantum operations. One may note that many such passes can avoided with good code and understanding of the quantum circuit. However, our proposed passes can improve scope of optimization by breaking the ”measurement barrier”. [Chen (2025)]

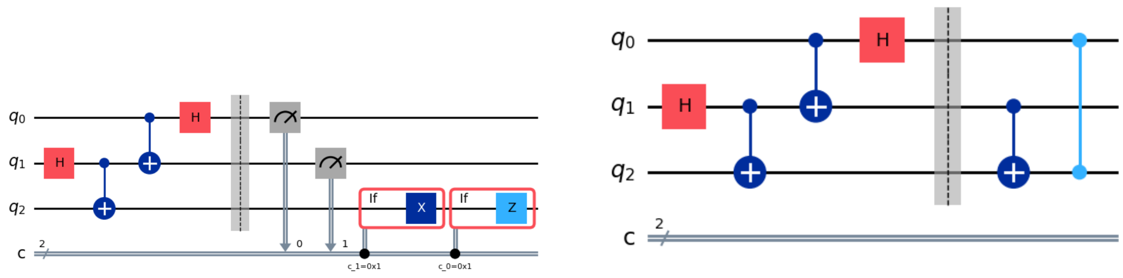

Figure 1: Deferred Measurement

2.2 Problem Statement

The objectives of the project are as follows:

- To study the existing transpiler passes in `qiskit` and their limitations in optimizing classically-controlled quantum operations.
- Given a quantum circuit with mid-circuit measurements and associated branches, we aim to optimize the circuit by integrating existing classical and quantum techniques. To this end, we explore three passes:
  - Branch Merging & Simplification
  - If-Else Splitting & Recombination
  - Hoare Optimizations across Measurements
- We aim to investigate the viability of such passes and the compile time required to perform such passes.

3 Branch Merging & Simplification

A very common optimization on classical computations involves the merging of branches with similar predicates. For instance, consider the following code:

```
if (x == 0) {
    A;    //Statement 1
}
if (x == 0) {
    B;    //Statement 2
}
```

which can be simplified to:

```
if (x == 0) {
    A;    //Statement 1
    B;    //Statement 2
}
```

Similarly, we may choose to ignore branches when they have complementary predicates with same operations. This naturally extends to nested branches as well. Implementations and examples of these can be found in the code linked at the end of the document. The same idea can implemented to quantum circuits with mid-circuit measurements and associated branches. This may result in simplification/merging of certain branches, leading to more possible optimizations.

4 If-Else Splitting & Recombination

The idea behind this pass follows directly from the classical branch prediction mechanisms. We would like to note the following differences:

- All quantum gates (excluding measurements) are *reversible*. In other words, one may easily revert operations on a qubit to its original state when necessary.

- Measurements are irreversible and generally do not commute with quantum gates. Hence, we cannot revert operations after measurements on measured qubits. This limits the scope of optimizations but can help with certain operations on unrelated qubits. We may choose to alleviate some of these constraints through other optimizations (later discussed).
- The optimization extent is dependent on several factors inclusing circuit area, nesting level of branching, window depth for optimization etc.
- Generalization is hard when each branch re-evaluates measured qubits. In such cases, one may be forced to split branches into dependent and non-dependent branches.

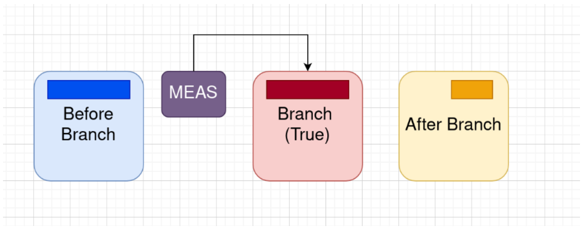

Figure 2: If-Else Circuit

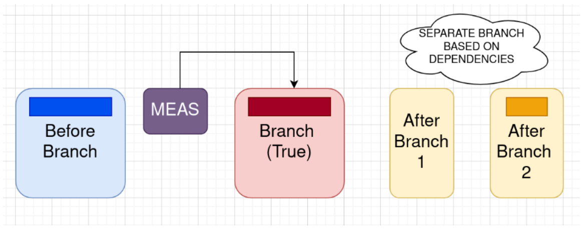

Figure 3: Split Circuit

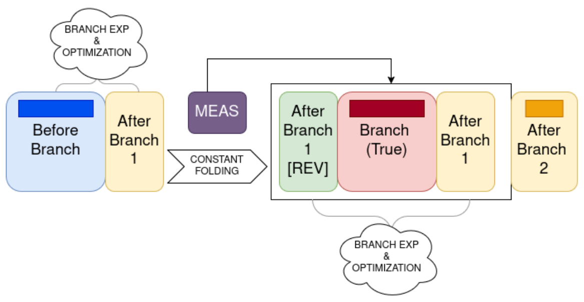

Figure 4: Recombined Circuit

4.1 Phases

The pass consists of the following phases with the circuit initially in a state like [2]:

- **Splitting**: Spilt the circuit after the branch into dependent and independent branches. We only work with the independent branches for optimization. [3]
- **Recombination**: Assuming the branch fails, we recombine before-branch with operations from after-branch (the indepedent branches). The `if-branch` is then replaced with a appropriate reversing and recombination. [4]

4.2 Example & Analysis

Through manual implementations, we were able to see that for toy-examples, there is considerable reduction in gate-complexity. Note that we assume our basis-gate set `RZ, X, SX, CX` where `RZ` is a single-qubit rotation gate, `X` is a Pauli-X gate, `SX` is a square-root of X gate and `CX` is a controlled-X gate. The following example illustrates the optimization: [5]

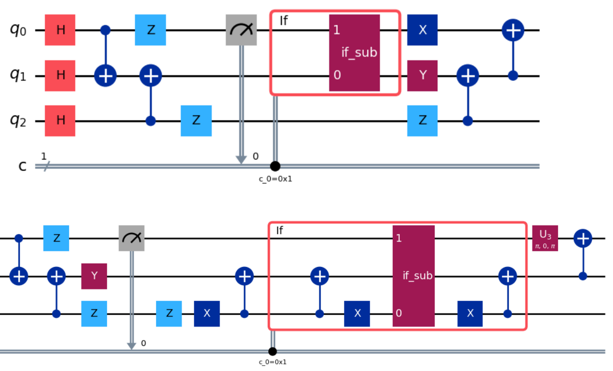

Figure 5: Toy Example for If-Else Splitting & Recombination

For utilitarian examples like quantum bit-flip error correcting codes, we observe very little improvement. Note that all these optimizations are highly dependent on other subcircuit structures.

5 Hoare Optimizations across Measurements

Hoare logic is a formal system used to reason about the correctness of computer programs. It provides a way to specify and prove properties of programs using preconditions and postconditions. Simply put, Hoare optimizations are a generalization of constant-folding. [Häner et al. (2020)]

In the context of quantum circuits, we can use Hoare logic to reason about the behavior of quantum operations and their interactions with classical control flow. Again these may be dependent on whether the initial state of qubits are known or not.

5.1 Example

A simple example of Hoare optimization is as follows [6]. Consider the circuit with the following operations:

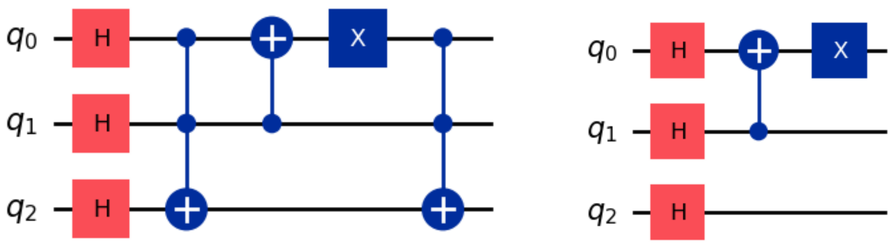

Figure 6: Hoare Optimization Example

Consider the circuit [put label here] operating on three qubits q_0, q_1, q_2. After the Hadamard operations equally mix the states q_0 and q_1, we the `CX` operation transforms $(q_0, q_1) \xrightarrow{CX} (q_0 \oplus q_1, q_1)$. After another application of `X` on q_0, we have $(q_0 \oplus q_1, q_1) \xrightarrow{X} (q_0 \oplus q_1 \oplus 1, q_1)$. The two `CCX` operations are only activated when $q_0 q_1 = 1$. After the transformation, we get $(q_0 \oplus q_1 \oplus 1)(q_1) = q_0 q_1 \oplus q_1 \oplus q_1 = q_0 q_1$. Thus, both `CCX` happen to work simultaneously and cancel out regardless.

5.2 Types of Possible Optimizations

- **Across Measurements**: `qiskit` cannot optimize using Hoare logic when there are measurement-dependent operations sandwiched between target gates. One may simply check dependencies and appropriate ignore these operations. [7] uses the same example quoted previously in [6] but with an additional branch sandwiched in.

- **On Measurements**: Similar to constant-folding, when we know the value of the measurement that determines other operations with certainty, we may simply remove the measurement and the branch. This will lead to standard optimizations. [8] is an example where we can determine with certainty that the first qubit is in state $|1\rangle$ and upon measurement, the branch is executed all the time. We simply remove the measurement and the branch and cancel out circuits.

- **Internal Optimizations**: One may internally apply Hoare logic on branch subcircuits. This can be done with the knowledge of measured qubits as qubits take definite values after measurement. [9] shows an example where we can implement the *If-Else Splitting & Recombination* pass in addition to the above Hoare optimization.

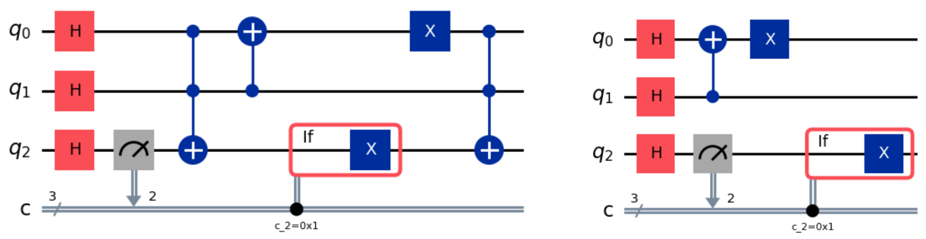

Figure 7: Across Measurements

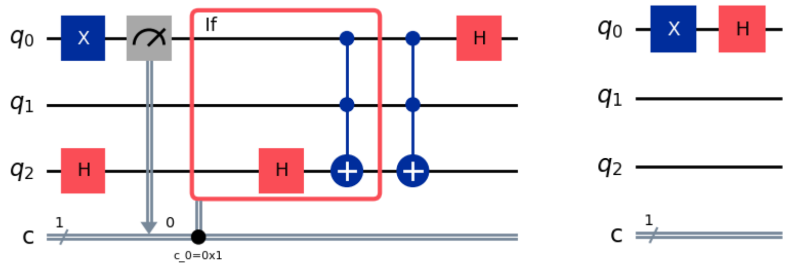

Figure 8: On Measurements

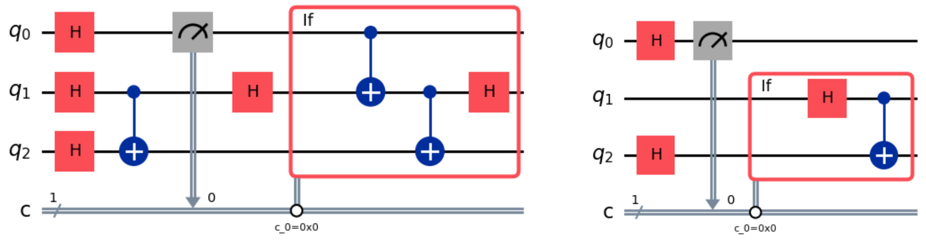

Figure 9: Internal Optimizations

6 Conclusion & Future Work

We have included the proposed passes in [GitHub Link]. Our future work will aim to generalize these passes and analyze performances on actual quantum devices. We also aim to explore the possibility of using these passes in other quantum programming languages like `cuda-quantum` [The CUDA-Q development team (2025)] which exploit GPUs and TPUs to simulate quantum circuits.

References

Chen, Y. 2025, Unleashing Optimizations in Dynamic Circuits through Branch Expansion. https://arxiv.org/abs/2504.09234

Häner, T., Hoefler, T., & Troyer, M. 2020, Proceedings of the ACM on Programming Languages, 4, 1–20, doi: 10.1145/3428201

Javadi-Abhari, A., Treinish, M., Krsulich, K., et al. 2024, Quantum computing with Qiskit, doi: 10.48550/arXiv.2405.08810

The CUDA-Q development team. 2025, CUDA-Q. https://github.com/NVIDIA/cuda-quantum