GPU-based Quantum Circuit Simulation Transpilation Optimizations

Scott Sikorski
University of Virginia
nqj5ak@virigina.edu

***Abstract*—Recent developments in quantum computing have illustrated the immense benefit that these computers and algorithms can bring to classically hard problems. While hardware is not publicly available, simulations are increasingly more important to enable research in new quantum computing methods and optimizations. These simulations can be can provide a lot of use; however, they suffer from computation and memory costs which exponentially scale to the size of the quantum computer. GPUs have been used to increase throughput relieving computational pressure but memory access latency can lag the system. Thus, a vital part in creating capable simulators is the transpiling phase, which like its classical compiling counterpart, breaks down the high level circuit to primitives. A mixture of circuit optimizations can then be applied that exploit information about the system.**

In this paper, I propose GPU-based quantum computer transpilation optimizations within a quantum circuit simulator. Quantum Peephole Optimization (QPhO) performs gate cancellation and qubit gate clustering to reduce the number of necessary gates and reduce costly CPU-GPU data exchanges. It leverages single qubit basis state information that can be statically determined at transpile time. Using this information, two qubit $CNOT$ operations can be eliminated or replaced with less expensive gates due to the control qubit basis state not activating the target qubit. Passes are conducted over the circuit until convergence is reached. Gate cancellation reduces the number of qubits unnecessarily being transferred to the GPU and can delay involvement. This approach is also extended with cluster circuit reordering techniques. To reduce the number of data transfers, single non-involved qubit gates can be delayed according to the latest used definition. This uses a line by line basic block representation of QASM2.0 code. Experimentation is done on a variety of 28+ qubit circuits that are the core of many popular algorithms. Transpilation time was not shown to dramatically increase while benefitting from an average of 10.6% (up to 23.5%) reduction in number of gates and 91% (up to 300%) decrease in execution time over the baseline non-optimized circuit.

I. Introduction

Quantum computing has become an increasingly more popular solution over classical computers in many high application domains. These quantum computers have been shown to have asymptotic speedups in large data search algorithms, chemistry and physics simulations, cryptography, and more [20]. Recently, IBM has released quantum computers with one chip containing 1,121 qubits and chiplets containing 133 qubits each [6]. The biggest problem facing quantum computers is the error-prone probabilistic nature of quantum mechanics. Typically, it takes anywhere from 10 to 100 physical qubits to encode one fault-tolerant quit. So while these new developments in qubit amount are close to containing enough qubits to build fault-tolerant qubits, we must accommodate noisy computation to further develop quantum computing development. These machines are deemed Noisy Intermediate Scale Quantum (NISQ) [16] computers due to the error associated with computation. This may include decoherence errors (transition from a high to low energy state), bit flip errors from environmental noise, or operational errors.

While quantum hardware has become more realized, there is still not a large number that are publicly available to researchers and developers. Thus, a huge aspect of researching and developing new algorithms involves building noisy simulators that can simulate the quantum process. These simulations use a sampling based approach to determine the output of a circuit. The circuit applies a variety of gates onto qubits, similar to assembly instructions. These gates alter the probability amplitudes of the qubits. The qubit can then be measured out which will collapse the probability wave function to output the result. As a result, the correct output can be shrouded by the incorrect options on a single run.

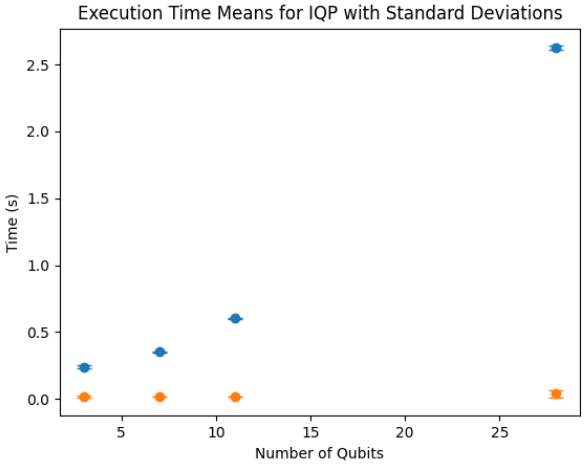

Fig. 1: Exponential Execution Times for IQP Circuit

One major drawback of these simulations is that they are compute-intensive and memory intensive. This is due to the requirement to fully and accurately compute and store the information of the entire evolving quantum system state. Similar requirements are necessary when applying gates as this requires traversing all the qubits and their probability amplitudes. These costs are exponential to the number of qubits

resulting in poor simulation times when the same circuit needs to be rerun thousands of times to compute a correct output. Figure 1 illustrates this execution time using the Instantaneous Quantum Polynomial (IQP) circuit. In an effort to speedup simulations, GPUs have been tasked with delivering high throughput in the underlying quantum simulation computation. They have shown a lot of promise as the amplitudes can be split up and calculated in parallel. Current Qiskit [17] frameworks support GPU acceleration through their specific backend simulators, such as Qiskit-Aer.

GPUs are extremely useful in increasing computational throughput but greatly suffer from current GPU device memory constraints. This means that data needs to be allocated on the main CPU memory then transferred over to the GPU for computation to take place on that data. This data exchange latency is extremely high and can cause bottlenecks in computation and lead to underutilization of the GPU parallelization capabilities.

The proposed transpilation optimizations help alleviate the compute and data exchange costs that can hinder GPUs and underutilize the full parallelization abilities. They are built on the fact that the qubit basis states can be inferred and tracked throughout the circuit at transpile time. Basis state equivalences can then be used to update the system information. These equivalences are particularly useful when two qubit gates such as $CNOT$ are activated based on the basis state of one qubit which acts as the control. If the control exists in a determined state which does not cause an operation on the target qubit, the gate can be effectively eliminated. The same idea can be applied for tracking the basis state of the target qubit, although in a different predetermined basis state. Therefore, this information can guide the transpiler in eliminating unnecessary gates and replacing two qubit gates with a single qubit gate. This simplification and optimization can prove to be particularly useful for GPUs which can suffer from long memory exchanges to effectively compute the same result. This can also prevent involvement of a qubit which allows the GPUs to focus on computing the already stored qubits.

Additional optimizations are proposed to further increase the utilization of gate cancellation and decrease GPU data exchanges. Circuit reordering methods are developed on the back of basis state evaluation to delay gate operations until there is an affect on the system. As a motivating example, imagine the GPU is constrained to 2 qubits in memory at a time and our circuit allocates and executes over 3 qubits. We start by performing computation on $q0$ and $q1$. A $CNOT$ operation involving $q0$ as the control and $q2$ as the target is read in. Usually, this would mean we evict $q1$ from memory and load in $q2$ causing a latency period before computation can resume. However, our basis state tracking has determined during transpile time that $q0$'s basis state would not cause a change in $q2$ basis state. We can eliminate this gate from the circuit, preventing $q1$ from needing to be evicted before further work is done on it.

I build on previous static iterative dataflow analysis compiler techniques that utilize a definition of earliest and latest used. This essentially makes multiple forward and backward passes through the code (circuit in context of quantum programs) to determine the earliest and latest lines or blocks that an operation can be placed at. These definitions can then be extended with knowledge of the basis state to indicate the range of motion that an operation has to delay qubit involvement.

Overall I propose the following contributions

1) Integration of gate cancellation into the Qiskit-Aer framework to reduce the overall number of gates in a given quantum circuit. This exploits qubit basis state equivalences to eliminate unnecessary gates.
2) Development of circuit clustering to mitigate costly CPU-GPU data exchange times by pushing back qubit involvement on a earliest-used to have a controlled effect on other qubits. This prioritizes dedicated GPU execution without continuously storing and reloading data.
3) Experimentation on a variety of benchmarks used in today's quantum algorithms showcase that the number of gates reduces, quantum circuit execution time decreases, and GPU memory transfer times are decreased.

II. Background

A. Quantum Computing

Quantum computing is centered on the quantum bit or qubit. Classically a bit is represented as being in a 1 or 0 state. A qubit is a unit of quantum information that is defined by two computational basis states $|0\rangle$ and $|1\rangle$. It can be thought of as a point on the bloch sphere (Figure 2) where operations shift and rotate where this point is located. The quantum state

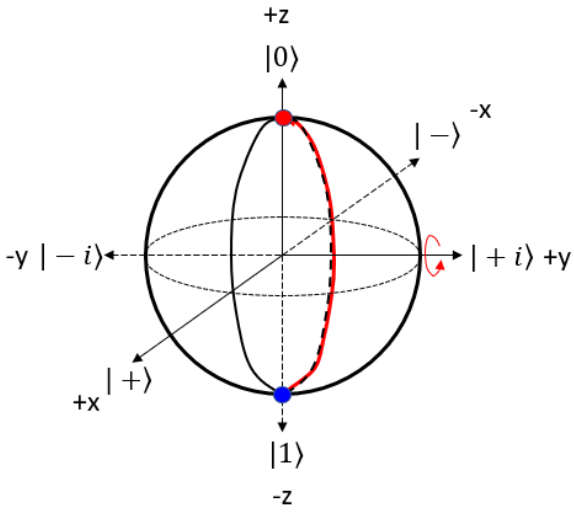

Fig. 2: Qubit Bloch Sphere With Basis States

is typically expressed as a linear combination of these two states.

$$|\psi\rangle = a_0|0\rangle + a_1|1\rangle \quad (1)$$

a_0 and a_1 are complex numbers which describe the probability amplitudes of the basis state. The squares of these complex numbers equal 1 as $|a_0|^2 + |a_1|^2 = 1$. This means that on measurement of any qubit, the result will equal the basis state with the described probability. Another way of expressing the state of a qubit is through state vectors

$$a_0 = \begin{bmatrix}1\\0\end{bmatrix} \quad a_1 = \begin{bmatrix}0\\1\end{bmatrix} \quad (2)$$

For any n qubit system, we express $|\psi\rangle$ as a linear combination

$$|\psi\rangle = a_{0..00}|0..00\rangle + a_{0..01}|0..01\rangle + ... + a_{1..11}|1..11\rangle \quad (3)$$

This means that there are 2^n state amplitudes that must be tracked during computation.

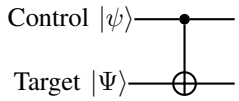

Fig. 3: Controlled Not Gate Circuit

A quantum circuit is essentially made up of a sequence of gates which act as instructions operating on qubits. There are a variety of single qubit gates such as the identity gate I, Hadamard gate H, Phase gates S, and the Pauli gates X, Y, Z. These are accompanied by double gates like Controlled-NOT $CNOT$, illustrated in Figure 3, and the controlled counterparts of the Pauli gates CX, CY, CZ. These use the control qubit $|\psi\rangle$ state to determine if the target qubit Ψ is flipped, like a traditional XOR instruction. All of these are uniquely defined by a 2^n x 2^n unitary matrix which is multiplied using the tensor product with the complex amplitudes of the n qubits is operating on.

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix} \quad (4)$$

The goal is to decompose algorithms of larger sequences of gates into these IBM supported gate primitives. This is done with a series of compilation and transpilation passes that can benefit from optimizations.

B. Qiskit-Aer Framework

Qiskit-Aer [17] is a high performance simulator backend framework for Qiskit circuits that takes advantage of GPU accelerators. The framework is primarily developed in C++ to enhance the simulator methods provided which can optimize the tensor products of the qubit probability amplitudes and gate matrices. Aer is able to transpile and run ideal or noisy simulations of any given circuit. This transpilation stage involves 3 main IBM developed optimizations, cache block pass [4], gate fusion [1], and optimized gates through sparse matrix representation.

Cache block pass aims to reorganize the circuit with sometimes an increased number of gates to make effective use of the cache. Guided by an execution cost-based heuristic, it achieves this by inserting $SWAPs$ from a cached qubit that has no instructions on it to a qubit with new instructions in a certain range of time. The $SWAP$ is done again after execution of what $q2$ would have done without needing to evict and load new memory in. An example is illustrated in Figure 4. The other featured optimization is the fusion of gates which simply identifies sequences of single qubit gates that can be combined and produce one equivalent gate. This only focuses on gates that have no 2 qubit gate intermediate steps.

C. Related Work

Prior works have been proposed to optimize circuits at the gate level. Peephole optimizations have been proposed for classically computers [13] that targets a window of code and deduces the pattern. Similarly, circuit equivalences have been discovered at the algorithmic level [19] and the quantum compiler peephole optimization [15]. Liu et al. [12] developed an analysis of the qubit basis states to statically infer the results of the gates. Sections of gates in a specified window can be eliminated as the unitary matrix of the input equals that of the output. Other techniques have proposed circuit reordering and remapping to aid noisy characteristic optimizations [18].

Qiskit [17] has a variety of transpiler optimizations available on their generic framework. This includes Hoare logic [9] that uses a set of defined in and out gate constraints on the circuit. This is implemented with a Z3 solver which incurs a large overhead. Qiskit also introduces passes to identify repetitive gates. Commutative Cancellation, Block Consolidation, and CX Cancellation attempt to find sequences of single and two qubit gates to be decomposed into higher throughput gates.

There are other works that use GPUs as the main execution engine of the quantum circuit. Most of which face issues in storing all the data in the GPU. Li et al. [11] tried multi-GPU clusters to store the entire system solely on the GPU memory to prevent any data exchanges. This method was limited to 14 qubits due to the aggregated memory not being enough. Doi et al. [5] initially proposed that GPUs can still be used even if we do not store all the data on the GPU. A data exchange through CPU secondary storage would allow us to compute on both the CPU and GPU. The computation benefits were reduced due to these data exchanges introducing overhead.

Distributed systems have shown promise in creating fast quantum circuit simulations. Compiler passes devloped by Guerrischi, Gian [8] manipulated the order in which the state amplitudes are stored to maximize the distributed abilities. Zhao et al. [23] use proactive state amplitude transfers, data compression, $|0\rangle$ state pruning, and delayed qubit involvement to enhance the effects of pruning. Zhao et al. [22] similarly proposed an extension that utilizes secondary storage and data transfer management to store all the qubits in memory and prevent costly exchanges. Pednault et al. [14] claimed

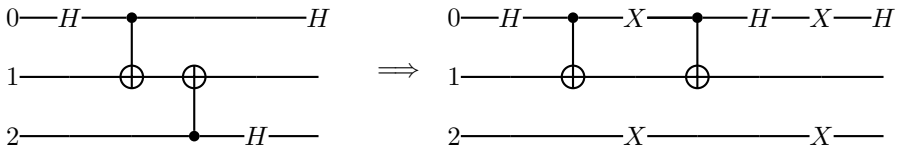

Fig. 4: Cache Block Pass

to simulate 49 qubits. They achieved this with subcircuit partitioning to handle parts of the circuit at times so that only a partition was stored in the GPU at once. On top of that, Zhang et al. [21] found that the most optimal simulation method differs for different parts of a circuit based on the pattern. They developed a hybrid framework which partitioned the circuit and simulated using the optimal method.

D. Benchmarks

Listed below, there are various representative quantum circuits which I aim to target. These are the core of popular algorithms such as Shor's algorithm. They explore different properties such as circuit depth and width, qubit involvement, and gate patterns.

1) Instantaneous Quantum Polynomial (IQP) [3] helps complex quantum computation simulation when sampling the output probability distribution.
2) Quantum Fourier Transform (QFT) [10] is the quantum equivalent to the inverse discrete Fourier Transform. This circuit is a main component of Shor's algorithm which is used for prime factorization.
3) Hidden Linear Function (HLF) [2] solves the 2D hidden linear function problem that uses a shallow circuit. The goal is to determine if a function or oracle is linear by querying the function in parallel at multiple inputs. Grover's algorithm is typically used to help this problem by flipping the states $|x\rangle$ which $f(x) = 1$
4) Grover's Search (GS) [7] is a technique used to search an unsorted database for a value. This typically has a $O(N)$ runtime but this shows that the probability of finding if a value exists scale with $O(\sqrt{N})$ number of iterations.

The specific characterization in terms of number of qubits, single qubit gates, and two qubit gates are described in Table I.

	Qubits	1-Qubit Gates	2-Qubit Gates	Total Gates
IQP	28	59	319	377
QFT	28	32	388	420
HLF	28	58	25	83
GS	28	115	27	142

TABLE I: Benchmarks

III. Gate Cancellation

Gate cancellation is centered around the idea that certain basis states result in a statically determined operation. For example, our $CNOT$ gate takes the control $|\psi\rangle$ and flips the basis state of the target $|\Psi\rangle$ if $|\psi\rangle = |1\rangle$. However, if $|\psi\rangle = |0\rangle$, then $|\Psi\rangle$ remains in the position on the bloch sphere as before. No computation is necessary, but the GPU will still anticipate

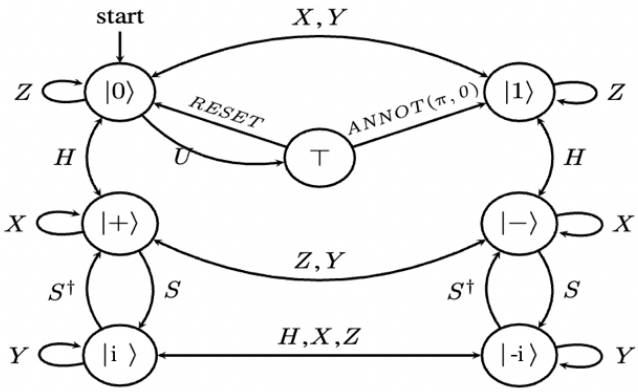

Fig. 5: Single Qubit Basis State Analysis

the data exchange of both qubits. From here, we can simply remove the gate and the need to load in that data and compute over all the system's probability amplitudes.

This technique can be useful and can be extended to a variety of basis states. Like how $|0\rangle$ and $|1\rangle$ point along the $+z$ and $-z$ direction, respectively, other basis states are determined by which axis the qubit is pointing along. We introduce 4 new basis states, $|+\rangle$, $|-\rangle$, $|+i\rangle$, $|-i\rangle$ as seen in Figure 2. These basis states can be represented as a unique unitary matrix applied to qubit's amplitudes.

Therefore, the Pauli gates can be thought of as rotations among the bloch sphere to different basis states based on the operation. For example, the X gate performs a rotation about the x-axis which transitions $|0\rangle$ and $|1\rangle$ between each other. Liu et al. [12] provide a comprehensive automata for single qubit basis state analysis as seen in Figure 5. This analysis lays the groundwork for cancelling gates.

A. Optimizing Single Qubit Gates

We can begin by simply eliminating single qubit gates that cause the input basis state to equal the output basis state. Applying Z to a qubit in $|0\rangle$ will simply result in $|0\rangle$, allowing us to eliminate the Z gate. This is possible because the input $|0\rangle$ state is the eigenstate of the Z matrix with an eigenvalue of 1. In general, for any U function, we can eliminate U if $|\psi\rangle = U\ |\psi\rangle$. This case rarely occurs but can prove useful for multi qubit optimization as we begin to eliminate $CNOTs$ from the circuit.

TABLE II: $CNOT$ Equivalences

$\|\Psi\rangle$ \ $\|\psi\rangle$	$\|0\rangle$	$\|1\rangle$	$\|+\rangle$	$\|-\rangle$	$\top$
$\|0\rangle$	Remove	Replace X on $\|\Psi\rangle$	Keep $CNOT$	Keep $CNOT$	Keep $CNOT$
$\|1\rangle$	Remove	Replace X on $\|\Psi\rangle$	Keep $CNOT$	Keep $CNOT$	Keep $CNOT$
$\|+\rangle$	Remove	Remove	Remove	Remove	Remove
$\|-\rangle$	Remove	Remove	Replace Z on $\|\psi\rangle$	Replace Z on $\|\psi\rangle$	Replace Z on $\|\psi\rangle$
$\top$	Remove	Replace X on $\|\Psi\rangle$	Keep $CNOT$	Keep $CNOT$	Keep $CNOT$

B. Optimizing CNOT

Most circuits are not made up of just these single gate rotations. The benefit of quantum computers comes from their ability to entangle the system state. This achieved by two qubit gates that make qubits outcome depend on each other. The main operation is the $CNOT$ gate, which mentioned before, entangles qubits so two qubits' outcomes correlate with each other. This effectively means that any additional gates that are applied to one of the entangled qubits must be applied to the other qubit as well. Thus, the proposed technique seeks to remove as many of the $CNOT$ gates as possible or replace them with less expensive and involved gates. Liu et al. [12] provide a table that documents the possible circuit changes that are possible with a given pair of control and target qubit basis states. Table II describes the operations that can take place based on the basis state at each step.

```
Initialize all qubits'
basis state = |0⟩
do
    for op in c do
        q = op.qubitsOn;
        if |q| == 1 then
            Update_Basis_State(q, op);
        end
        else if |q| == 2 then
            Remove(op) if removable;
             Converged = True;
            Replace(op) if replaceable with single
             qubit op; Converged = True;
            Do nothing;
        end
        else
            ∀q_i ∈ q,  q_i = ⊤
        end
    end
while Not Converged;
```

Algorithm 1: Gate Cancellation Pass Overview

The general overview is outlined in Algorithm 1. To begin our gate cancellation pass, we initialize all our qubits to have a basis state of $|0\rangle$. A first forward pass is conducted, iterating through all the operations. For each operation, if the gate operates on a single qubit, the basis state of that qubit is updated. This basis state is updated according to Figure 5. All Pauli gates and phase rotations are supported that can shift the basis state of the qubit. The basis state for that qubit is now stored and updated then propagated down into future uses.

Once we reach a two qubit gate U[1], we will attempt to infer the result of applying U to the input unitary matrix. The most applicable inference is the $CNOT$ equivalence. From the propagated basis state values for both the control and target qubit, we follow the equivalences guide. If we can remove the gate, the gate is deleted from the circuit. If we can replace the $CNOT$, the dictated control or target qubit is given an additional gate to execute before the target qubit receives its phase gate. This additional gate is typically less expensive than $CNOT$ and can prevent another qubit from being transferred if the replacement gate is on $|\Psi\rangle$. However, there are 3 three cases where the control qubit is given a new gate and must be involved. And so if we have deleted or replaced any gates in the circuit, we will conduct another pass until convergence is achieved. This convergence allows a consistent propagation of values such that the number of gates can greatly decrease in circuits that have a lot of 2 qubit operations.

However, gate cancellation can cause fragmentation in the structure of the circuit that results in more data exchanges. Some circuits will prepare qubits by applying a single qubit gate, typically a Hadamard gate, to each qubit at the start of the process. Eventually, gates operating on some qubits will be eliminated. But those single qubit gates still linger at the top which will cause the circuit to load the data in, compute, and store for later use. To alleviate this, qubit cluster circuit reordering is proposed.

IV. Circuit Clustering

Gate cancellation can greatly aid in preventing unnecessary operations. However, to alleviate the fragmentation that it may cause and to take advantage of the full benefits, gate clustering is proposed to group operations based on the qubits together. Say our circuit declared N number of qubits then applied a single gate operation to each at the beginning of the process. Two qubit gates are applied sequentially to various qubits. I propose a circuit reordering technique that will identify group these operations together and cluster them based on the latest used definition.

To achieve a clustering of qubits, the latest used mapping must first be defined. **Latest Used** maps qubits to the latest possible basic block that a qubit can exist in before having two qubit gates applied to. This basic block is set

[1]This U gate stands for any controlled gate that is supported. Qiskit offers support for high level controlled rotations that activate rotations based on the input to the control qubit. By using basis state propagation, the phase may be eliminated. Thus, this approach can be generalized to any grouping of $CNOT$ + $U^\dagger$ gate, where $U^\dagger$ gate represents any single qubit rotation or shift operation on the target qubit.

H q0		H q0
H q1		H q1
H q1	$\Longrightarrow$	H q1
H q2		cx q0, q1
cx q0, q1		H q2

Example Clustered Circuit

to the lines that the QASM code exists at. I chose this basic block discretization for two main reasons. *1)* Qiskit represents the circuits as directed acyclical graphs (DAG) *2)* This discretization represents the closest over-approximation to the real solution. Combined with reason 1, we can guarantee the correctness of the solution even after optimizations are applied. The second part of this definition is based on once two qubits are entangled, their states are correlated. This means that we can only postpone the single qubit gates to the latest used line where the first two qubit gate is located. This is because we cannot alter the system behavior when qubits begin to entangle. Thus, this approach keeps the correctness while grouping the gates which act on the same qubits.

Input: Circuit c, Qubits q
1 `Perform Gate Cancellation`
2 **do**
3 **for** *op in c* **do**
4 q = op.qubitsOn;
5 **if** $|q|$ == *2* **then**
6 latest[if q_1 == null, if q_2 == null] = opIdx;
7 **end**
8 **if** $|q|$ == *1* **then**
9 **while** *latest[q] (!= opIdx+1 || == null) && nextOp.qubitOn != q && latest[q] != nextOp.qubitOn* **do**
10 eraseAt(op, originalIdx);
11 insert(op, opIdx);
12 latest[q] = opIdx + 1;
13 Converged = True;
14 **end**
15 **end**
16 **end**
17 **while** *Not Converged*;

Algorithm 2: Circuit Reordering For Latest Use Clustering

From here, the overview of the clustering is outlined in Algorithm 2 and follows. We perform any initial gate cancellation which can eliminate unnecessary gates while fragmenting the circuit. From here, we take the new circuit and initialize an empty **latest** map. The first pass through the circuit is focused on finding the first two qubit gate operations. Seen in line 5 and 6, the latest map is updated with the 2 qubit gate at which we first find the qubit. If either of the qubits have already been found earlier in the pass, the non found qubit will be the only that gets the later gate line. This is key to respect the earlier latest used value of the 2 qubit gate.

Next we focus on the single qubit gates which will be the main operations that we move and reorder the circuit with. This is because we wish to delay these involvement operations until we are forced to. I achieved this by iterating through the circuit until we reached the latest or last line or that the next operation would operate on q. The operation could then be deleted from its prior location and inserted at the line before it is used or its latest possible line. The latest is now updated in the case that the next gate operates on q. Since we have changed the circuit, another forward pass is conducted over the circuit, this time with newly updated latest map.

During development, two main conflicts arose, same latest line and lack of 2 qubit gate for a qubit. The first of which was solved by the third condition in line 9. Simply if two single qubit gates shared the same latest line, no reordering took place. Secondly, if the circuit includes qubits that never entangle, then we group operations on the same qubit together preferably at the end of the algorithm. This should almost never occur since entanglement is the key property behind quantum algorithms working.

V. Results

Both of the described passes were implemented directly into the Qiskit-Aer circuit executor. These were done using version 0.14.1 of Qiskit-Aer with version 1.0.1 of Qiskit. The optimizations were simply included after fusion and cache block pass as additional passes modifying the circuit structure. The transpile optimization time was set to 0. This means that no non-Qiskit-Aer optimizations were included before or after QPhO. I aimed to analyze just Aer optimizations as some regular Qiskit optimizations can slow down the Aer simulator. The regular Qiskit simulator will organize partitions of the circuit as unitary functions which can be applied as a gate and cannot be run on the GPU. I seek to analyze how the circuit can be optimized at the gate level within Aer.

A. Configuration

All experiments were conducted on the University of Virginia Computer Science GPU servers. Each circuit was run for 1024 shots and then conducted 10 times total to retrieve the mean and standard deviation of the execution times. These times varied as a result of server times.

The benchmarks were split up into four separate groups, baseline, baseline with QPhO, IBM optimized, and IBM optimized with QPhO.

- Baseline is described with no optimizations circuit level optimizations applied. This includes any found in the Qiskit or Qiskit-Aer framework.
- QPhO pass takes place over the unoptimized baseline circuits. This provides just the gate cancellation and clustering methods.
- IBM optimized group was given the gate fusion and cache block passes that are described in section II-B.
- QPhO applied on top of the IBM optimizations. The QPhO pass is done after fusion and cache blocking to fully analyze how it can integrate into the IBM system.

TABLE III: HLF Circuit simulation (shots = 1024)

	Baseline	Baseline + QPhO	IBM Optimized	IBM Optimized + QPhO
Total Execution Time	1.1856 ± 0.0174	0.5592 ± 0.00928	1.1162 ± 0.0102	0.5055 ± .0076
Transpile Time	0.0314 ± 0.0457	0.0379 ± 0.06340	0.0153 ± 0.00395	.0269 ± .01947
Minimum Number of Gates	84	68	83	68

TABLE IV: GS Circuit simulation (shots = 1024)

	Baseline	Baseline + QPhO	IBM Optimized	IBM Optimized + QPhO
Total Execution Time	2.2009 ± .0095	.5924 ± .0232	0.999 ± .0125	.5120 ± .0130
Transpile Time	.0210 ± .0123	.0274 ± .0120	.0242 ± .0149	.0237 ± .0183
Minimum Number of Gates	142	115	142	115

TABLE V: IQP Circuit simulation (shots = 1024)

	Baseline	Baseline + QPhO	IBM Optimized	IBM Optimized + QPhO
Total Execution Time	2.712 ±0.0268	2.509 ± 0.0187	2.680 ± 0.0146	2.4602 ± .0172
Transpile Time	0.0310 ± 0.0132	0.0267 ± 0.0132	0.0290 ± 0.0162	0.0301 ± .0195
Minimum Number of Gates	377	375	355	355

TABLE VI: QFT Circuit simulation (shots = 1024)

	Baseline	Baseline + QPhO	IBM Optimized	IBM Optimized + QPhO
Total Execution Time	2.477 ± 0.0156	2.542 ± 0.0103	2.404 ± 0.0161	2.3852 ± .0190
Transpile Time	0.0296 ± 0.0122	0.0295 ± 0.01423	0.0299 ± 0.0117	.0297 ± .01293
Minimum Number of Gates	420	420	420	420

B. Hidden Linear Function

Hidden linear function was the smallest of the benchmarks that I chose with 83 total gates for 28 qubits. It also had the least potential for gate cancellation. Of the 83 total gates, only 25 of which were 2 qubit gates, all of which were controlled-Z. However, the simplistic nature of the circuit benefited from QPhO a lot. A possible reduction of 23.5% of gates could be eliminated making it the profitable from gate cancellation. This reduction is relative to increased proportion of two qubit to single qubit gates when compared to GS. HLF puts all qubits into superposition then entangles pairs of them then brings them out of superposition. This effectively makes N queries which is number of qubits the circuit has. Because HLF only looks for when the hidden function returns $|1\rangle$ from the resulting gates, the problem can be simplified greatly. The gate cancellation is seen from any inputs that map to $|0\rangle$ eliminating the need for their $CNOTs$. Then the circuit clustering takes full benefit in reordering. This is because a singular $CNOT$ is applied to just one pair of qubits. Thus, we effectively cluster the before H gates with their $CNOT$. Then the after H gates can be fully parallelized by the GPU. There was shown to be a 134% decrease in execution time for this circuit more than halving that execution time. This execution time was second only to GS.

C. Grover's Search Algorithm

Grover's Search is a variation of HLF which attempts to find the index based on multiple queries. So similarly, GS showed a lot of benefits from QPhO. GS applies more gate rotations after the initial H gates, but these are generally Pauli rotations which allowed gate cancellation to track the basis state. In turn, we can eliminate any gates connected to qubits that do not correspond to the $|1\rangle$ output state. This resulted in a 19% possible reduction in the number of gates.

As well, GS most greatly benefitted from the circuit clustering method. The circuit applies single qubit gates with a $CNOT$ for each qubit then more single qubit gates. This enabled the approach to fully cluster each individual qubit to its unique gates which are not associated with other gates. As seen the execution time for the GS circuit was halved by the IBM optimizations and then halved again with the QPhO. This circuit was the only one that the current Aer optimizations seemed to have a noticeable difference on. This circuit offers a lot of propagation and peephole optimizations from the multitude of single qubit gates.

D. Instantaneous Quantum Polynomial

IQP benefitted slightly from the proposed QPhO methods. Only 2 gates of a possible 319 were cancelled. This is because IQP used controlled phase gates and not the Pauli gates so only 2 qubits in the $|0\rangle$ or $|+\rangle$ were identified. However, the general structure of IQP allowed for clustering to have an effect on reducing the execution time. IQP is organized to iteratively introduce new qubits into superposition then do controlled phase shifts over a various number of those qubits with the newly superposed qubits as the target qubit. However, each iteration does not include controlled gates on every introduced qubit. This allows the clustering method to benefit from reorganizing some operations to focus on the computed qubits. An 8% decrease was observed when using QPhO over the unoptimized circuit. IBM's optimizations were able to fuse some gates together, but there were not replaced or given opportunity to speedup execution. Combined with the reduced gate number and clustering, there was about a 10% decrease using all optimizations in execution time over

the pure baseline approach with minimal increase in transpile time.

E. Quantum Fourier Transform

QPhO offered the least amount of benefit to the QFT circuit in terms of execution time and number of gates cancelled. No gates were cancelled, and execution time remained the same as neither method seemed to be useful for this circuit. This benchmark is an important aspect of Shor's algorithm, but the structure showcases QPhO weaknesses. The circuit is made up of iteratively putting one qubit into superposition with the Hadamard gate then applying a custom phase shift with the superposed qubit as the control. The initial superposition application removes any $CNOT$ equivalences. However, this showcases that phase rotation could be analyzed more continuously across the bloch sphere to determine optimizations.

F. Overview

Overall, the QPhO passes were generally able to decrease the number of gates in the circuit when given $CNOT$ or controlled-Z gates, as seen in HLF and GS. These methods were shown to eliminate any $CNOT$ and rotations on qubits that don't produce a $|1\rangle$ output basis state. This can theoretically reduce our best case complexity to $\theta(\sqrt{M})$ where M is the number of coefficients with 1 in the hidden function. This is because we will eliminate all the gates that produce a 0 meaning we don't operate on those qubits, effectively enhancing our θ complexity while keeping the worst case complexity the same $O(\sqrt{N})$.

IQP and QFT did not benefit from this pass due to their circuit structures and only including λ phase rotations which were not considered in how they affected the basis state. This rendered all the operations to make the target qubit exist in $\top$. For all benchmarks, the transpilation time did not change dramatically and was able to stay consistent for the number of qubits. This transpile time varied greatly for each circuit which means that it took about the same time and this additional pass did not incur more overhead. In turn, the total execution time was decreased with the application of QPhO on a totally unoptimized circuit and when integrated with the Qiskit-Aer system. This is beneficial to our noisy simulations where we need to run thousands of shots to infer the correct answer. QPhO offers a lot of benefit to the multiple query parallelized circuits.

An intersting point is that the optimizations were good enough on their own with the other optimizations offered by the Aer transpiler. For all of the circuits, a similar execution time was achieved when only QPhO was applied and then when it was applied on top of the fusion and cache block pass. I believe this is because both passes optimize away and consolidate unnecessary gates.

VI. Limitations and Further Work

One major limitation to this work is that the gate cancellation is reserved for specific rotation gates. Only the Pauli gates were considered in the analysis and how they would change the basis state of the qubits that they were operating on. Realistically, this is not the case for a majority of quantum algorithms as they contain rx, ry, rz gates and other user defined phase rotation gates. This leads us to further develop analysis that track the positioning of the bloch sphere. We can look to optimizations in the continuity of superposition that exploit the positioning. Reordering can take place that better tracks the position and can use an exact phase angle for potential optimizations. As well, if we are working on a noisy backend, we can perhaps exploit the noisy characteristics. We can include more approximation into our optimizations, especially with very small phase rotations. This is because our approximation will mix in with the quantum noise. Additionally, this noise could be filtered and corrected once fault-tolerant quantum computers are achieved.

Along the limited single qubit gates, only two qubits gates are supported for optimizations in both the gate cancellation and cluster reordering. Qiskit's QASM includes gates that can operate on 3 or more qubits at a time, usually with one control qubit and then $N-1$ target qubits. Possible future work could be to extend the equivalences for multi qubit gates. This would require more analysis in the basis states and how they are affected by these other high-level operations. Another solution is to fully decompose these multi-controlled gates into only single and two qubit gates. This in turn can be fed to the transpiler and optimized while ensuring that the correctness properties are saved.

A beneficial addition to this work would be to support QASM3.0 code. All the benchmarks were written in QASM2.0 which does not support any branching or function definitions. An extension to the parsing and compiling of branching and functions would allow for classical static iterative dataflow analysis techniques to be applied to quantum computers. Constant propagation, a close classical gate cancellation counterpart, can be used with branching-split basic blocks where the join and transfer rules are defined. This work would be hinged on Qiskit integrating support for QASM3.0 into Qiskit-Aer and integrating classical registers into the proposed techniques.

Future work in this cluster approach includes more manually integrating which qubits can be accessed along each GPU thread. This work focused primarily on the higher transpiler level to perform intermediate optimizations on the circuit. I wasn't able to access or find anything lower level, so the optimizations were mainly constrained by Qiskit-Aer and their codebase for this reason.

A. Note

The data in the results is different from that presented in the project presentation. I found a bug in my gate cancellation pass that effectively caused a majority of gates to be eliminated if the target qubit was in the $|+\rangle$ basis state. This diminishes some of the benefits that I initially saw and that the circuit choices seemed to be poor (those being the IQP and QFT circuits).

VII. Conclusion

I proposed Quantum Peephole Optimizations (QPhO) which aimed to cancel gates through the use of statically inferred computational basis states of qubits. The gate cancellation pass is effectively able to iterate through all the operations, updating the basis state on single qubit gates and determining a more cost-effective gate for two qubit gates. This method can help reduce redundant operations and delay involvement of qubits in the circuit. Experimentation on hidden linear function and grover's search illustrated the possible gate cancellations that can take place depending on initial basis state.

On top of this pass, a circuit clustering forward pass was implemented to reorganize the qubits into computational clusters. This clustering technique prevented CPU-GPU data exchanges to increase throughput and decrease circuit execution time. Using techniques from classical compilers, a latest possible block is identified and used to group the qubits together. This method preserves the system behavior and correctness of entangled qubits while benefitting circuits that have scattered qubits. This can be improved further within Qiskit's unitary function definitions that create defined U functions that map input to output. Effectively, these can be defined by the qubits that they are operating on while combining those that fit in the GPU memory. All of these optimizations were implemented directly into the Qiskit-Aer framework such that they applied to that backend.

VIII. Supplementary Material

The artifact for this work can be found at https://github.com / sgsikorski / aer-peephole. The main code is coupled with the official Qiskit-Aer release to enable the packaging of the implementation into a Python library. This is found at https : / / github . com / sgsikorski / qiskit-aer, more specifically https://github.com/sgsikorski/qiskit-aer/tree/main/src/transpile/peephole.hpp.

References

[1] S. Bartolucci, P. Birchall, H. Bombin, H. Cable, C. Dawson, M. Gimeno-Segovia, E. Johnston, K. Kieling, N. Nickerson, M. Pant *et al.*, "Fusion-based quantum computation," *Nature Communications*, vol. 14, no. 1, p. 912, 2023.

[2] S. Bravyi, D. Gosset, and R. König, "Quantum advantage with shallow circuits," *Science*, vol. 362, no. 6412, p. 308–311, Oct. 2018. [Online]. Available: http://dx.doi.org/10.1126/science.aar3106

[3] M. J. Bremner, R. Jozsa, and D. J. Shepherd, "Classical simulation of commuting quantum computations implies collapse of the polynomial hierarchy," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 467, no. 2126, 2010. [Online]. Available: http://dx.doi.org/10.1098/rspa.2010.0301

[4] J. Doi and H. Horii, "Cache blocking technique to large scale quantum computing simulation on supercomputers," in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, Oct. 2020. [Online]. Available: http://dx.doi.org/10.1109/QCE49297.2020.00035

[5] J. Doi, H. Takahashi, R. Raymond, T. Imamichi, and H. Horii, "Quantum computing simulator on a heterogenous hpc system," in *Proceedings of the 16th ACM International Conference on Computing Frontiers*, 2019, pp. 85–93.

[6] J. Gambetta, "Ibm's roadmap for scaling quantum technology," *IBM Research Blog (December 2023)*, 2023. [Online]. Available: https://www.ibm.com/quantum/blog/quantum-roadmap-2033

[7] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[8] G. G. Guerreschi, "Fast simulation of quantum algorithms using circuit optimization," *Quantum*, vol. 6, p. 706, 2022.

[9] T. Häner, T. Hoefler, and M. Troyer, "Assertion-based optimization of quantum programs," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–20, 2020.

[10] A. JavadiAbhari, S. Patil, D. Kudrow, J. Heckey, A. Lvov, F. T. Chong, and M. Martonosi, "Scaffcc: Scalable compilation and analysis of quantum programs," *Parallel Computing*, vol. 45, p. 2–17, Jun. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.parco.2014.12.001

[11] A. Li, O. Subasi, X. Yang, and S. Krishnamoorthy, "Density matrix quantum circuit simulation via the bsp machine on modern gpu clusters," in *Sc20: international conference for high performance computing, networking, storage and analysis*. IEEE, 2020, pp. 1–15.

[12] J. Liu, L. Bello, and H. Zhou, "Relaxed peephole optimization: A novel compiler optimization for quantum circuits," in *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. IEEE, 2021, pp. 301–314.

[13] W. M. McKeeman, "Peephole optimization," *Communications of the ACM*, vol. 8, no. 7, pp. 443–444, 1965.

[14] E. Pednault, J. A. Gunnels, G. Nannicini, L. Horesh, T. Magerlein, E. Solomonik, and R. Wisnieff, "Breaking the 49-qubit barrier in the simulation of quantum circuits," *arXiv preprint arXiv:1710.05867*, vol. 15, 2017.

[15] A. K. Prasad, V. V. Shende, I. L. Markov, J. P. Hayes, and K. N. Patel, "Data structures and algorithms for simplifying reversible circuits," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 2, no. 4, pp. 277–293, 2006.

[16] J. Preskill, "Quantum Computing in the NISQ era and beyond," *Quantum*, vol. 2, p. 79, Aug. 2018. [Online]. Available: https : //doi.org/10.22331/q-2018-08-06-79

[17] Qiskit contributors, "Qiskit: An open-source framework for quantum computing," 2023.

[18] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for nisq-era quantum computers," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 987–999.

[19] G. F. Viamontes, I. L. Markov, and J. P. Hayes, "Checking equivalence of quantum circuits and states," in *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, 2007, pp. 69–74.

[20] S. P. Wang and E. Sakk, "Quantum algorithms: overviews, foundations, and speedups," in *2021 IEEE 5th international conference on cryptography, security and privacy (CSP)*. IEEE, 2021, pp. 17–21.

[21] C. Zhang, Z. Song, H. Wang, K. Rong, and J. Zhai, "Hyquas: hybrid partitioner based quantum circuit simulation system on gpu," in *Proceedings of the ACM International Conference on Supercomputing*, 2021, pp. 443–454.

[22] Y. Zhao, Y. Chen, H. Li, Y. Wang, K. Chang, B. Wang, B. Li, and Y. Han, "Full state quantum circuit simulation beyond memory limit," in *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2023.

[23] Y. Zhao, Y. Guo, Y. Yao, A. Dumi, D. M. Mulvey, S. Upadhyay, Y. Zhang, K. D. Jordan, J. Yang, and X. Tang, "Q-gpu: A recipe of optimizations for quantum circuit simulation using gpus," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 726–740.