UNIVERSITY OF CALIFORNIA, SAN DIEGO

Evaluating IBM's Quantum Compiler and Quantum Computer Architectures As They Pertain to Quantum Walk Simulation Algorithms

BY CLARA WOODS

Honors Thesis, Department of Mathematics, 2019

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

My greatest thanks to Professor David Meyer from the UC San Diego Department of Mathematics for advising me this past year and providing the requisite background knowledge of quantum computing to complete this project. I would also like to thank Asif Shakeel for contributing to this project by creating the uncompiled circuits for both the first and second quantized versions of the quantum walk problem.

Evaluating IBM's Quantum Compiler and Quantum Computer Architectures As They Pertain to Quantum Walk Simulation Algorithms

BY CLARA WOODS

Honor's Thesis, Department of Mathematics

University of California, San Diego, 2019

ABSTRACT

Quantum computers, with their ability to place qubits into a superposition of states, promise more computational power per qubit-timestep than classical computers can deliver per bit-timestep. IBM is delivering some of the world's first quantum computers, known as noisy, intermediate-scale quantum computers, or NISQs. We worked with two of these devices on the cloud, one with five qubits and one with fourteen qubits. This project studies two data structures used to simulate a quantum walk. The one-dimensional quantum walk problem consists of N sites arranged in a circle, with a particle hopping among them. The first quantized version represents the particle's position using $\log_2 N$ qubits and uses an additional qubit to store the velocity. The second quantized version uses $2N$ qubits, with each pair of qubits representing a single site, and the particle's position within the pair indicating its velocity. I hand-compiled these two algorithms onto two architectures offered by IBM to determine which architecture works best for each algorithm. I found that the five qubit architecture results in a shorter circuit for the first quantized version with two sites whereas the fourteen qubit architecture results in a shorter circuit for the second quantized version with two sites. Overall, the first quantized version requires fewer qubits, gates, and timesteps than the second quantized version does for both $N = 2$ and $N = 4$. I also compared my hand-compiled circuits to those IBM's compiler produces. By carefully determining the initial assignment of physical qubits, I produced circuits with smaller size and less depth than IBM's compiler did. I then ran all the compiled circuits on

IBM's quantum computers to see how close the distribution of measured states was to the expected distribution of states. The first quantized version produced better results than the second quantized version because it had smaller circuits to run, so less noise was introduced. IBM's circuits produced better distributions than mine for the first quantized version with four sites, but my circuits produced better distributions than IBM's for the second quantized version with two sites.

1 Introduction

The theory behind quantum computing has been around for decades, but physical quantum computing devices have only recently become available. IBM is at the forefront of building these early quantum computers. The ones publicly available have either five or fourteen qubits[1] and a limited set of hardware-supported gate operations[2]. As the name noisy, intermediate-scale quantum implies, these devices are also incredibly noisy. Circuits that are too long or have too many gates output distributions of almost pure noise. This motivates the optimization of circuits in both depth and size, where depth refers to the number of timesteps required to run the circuit and size refers to the number of gates included in the circuit.

IBM's software package qiskit[3] allows users to create circuits using an expanded set of gates. When users execute their circuits on a certain architecture, qiskit first compiles the circuit to OpenQASM[4], the assembly-level code. This step compiles the high-level circuit into a circuit composed solely of gates supported at the hardware level. This makes creating circuits easy as one does not have to consider the underlying hardware; however, it hides the true complexity of the circuit physically running on the quantum computer. Often, what appears to be a simple circuit on the user-end gets compiled into a longer circuit with additional gates. Some of this lengthening is unavoidable given the hardware restrictions of IBM's quantum computers. However, the circuits IBM's compiler produces are not always optimal for the underlying hardware. Given a circuit and an architecture, hand-compiling the circuit onto the architecture allows one to choose which physical qubits to use, which often leads to savings in both depth and size compared to IBM's compiler.

Ideally, running optimized circuits will reduce the amount of noise in the output and lead to more accurate results. I quantify the accuracy of results by measuring the ℓ^1 distance between the distribution of measured states and the expected distribution of states. Then I compare the ℓ^1 distances from running IBM's circuits against the ℓ^1 distances from running my hand-compiled, optimized, circuits. I also consider the ℓ^1 distance between the uniform distribution of states and the expected distribution of states to set a threshold above which the results cannot be considered anything more significant than noise.

The remainder of this paper is structured as follows: Chapter 2 gives a brief background of quantum computing theory intended to cover the basic concepts needed to understand the rest of the paper. Chapter 3 introduces the quantum walk problem and two different data structures we will use to implement it. Chapter 4 outlines the two quantum computer architectures we used and discusses how to compile gates not supported at the hardware level. Chapters 5 and 6 investigate how the first and second quantized versions of the quantum walk compile onto the two architectures. Chapter 7 discusses the results of running the compiled circuits and compares the distributions of measured states to the expected distributions of states. Chapter 8 recaps the results of compiling and running circuits for the first and second quantized versions.

2 Background

Quantum computers are fundamentally different from classical computers. In a quantum computer, qubits store information. At any given time, a single qubit is in a state

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle , \tag{2.1}$$

where $|\alpha|^2 + |\beta|^2 = 1$ and $\alpha, \beta \in \mathbb{C}$. The states $|0\rangle$ and $|1\rangle$ are the basis states, analogous to bit values of 0 and 1 in classical computers. The state of a qubit can also be represented in vector notation as

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} . \tag{2.2}$$

Quantum gates, represented by unitary matrices, manipulate qubits and are reversible. That is, given the output and the gate, we can reconstruct the input. We perform gate operations by multiplying a qubit state vector by a unitary gate matrix. Given these tools, we can construct quantum circuits as a series of quantum gates acting on a set of qubits. The gains in computing power of quantum computers come from the ability to entangle qubits and place them in superpositions, but a finer understanding of the physics is not necessary for the scope of this paper.

We will need the following gates for this paper: X, CNOT, SWAP, Hadamard, the general unitary, the controlled general unitary, R_Y,R_Z, and, while technically an operation, not a gate, the measurement operation. Table 2.1 shows how the X, CNOT, and SWAP gates act on basis qubit states. X is a one-qubit gate that flips the value of the qubit. CNOT is a two-qubit gate that only flips the value of the target qubit if the control qubit is in the state $|1\rangle$. The SWAP gate is a two-qubit gate that swaps the values of the two qubits.

Basis State	$\boldsymbol{X}$	**CNOT**	**SWAP**
$|(0)0\rangle$	$|1\rangle$	$|00\rangle$	$|00\rangle$
$|(0)1\rangle$	$|0\rangle$	$|01\rangle$	$|10\rangle$
$|10\rangle$		$|11\rangle$	$|01\rangle$
$|11\rangle$		$|10\rangle$	$|11\rangle$

Table 2.1: Truth Table for applying X, CNOT, and SWAP gates to basis states. Note that X is a one-qubit gate and only acts on the basis states $|0\rangle$ and $|1\rangle$ while the CNOT and SWAP gates are two-qubit gates and act on the basis states $|00\rangle$, $|01\rangle$, $|10\rangle$, and $|11\rangle$. In the case of the CNOT gate, the leftmost qubit is the control qubit, and the rightmost qubit is the target qubit.

The Hadamard gate is a one-qubit gate used to place a qubit in a superposition of states and is represented by the following matrix

$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}. \tag{2.3}$$

The one-qubit general unitary gate takes three arguments, θ, ϕ and λ, and is represented by the following matrix

$$U(\theta, \phi, \lambda) = \begin{pmatrix} \cos(\frac{\theta}{2}) & -e^{i\lambda}\sin(\frac{\theta}{2}) \\ e^{i\phi}\sin(\frac{\theta}{2}) & -e^{i\phi+i\lambda}\cos(\frac{\theta}{2}) \end{pmatrix}. \tag{2.4}$$

The two-qubit controlled general unitary gate acts on two qubits in a similar way to CNOT, with the general unitary only acting on the target qubit if the control qubit is in the state $|1\rangle$. The R_Y and R_Z one-qubit gates rotate the qubit's state around the $Y-$ and $Z-$axes, respectively. The measurement operation performs a measurement on the qubit on which it acts, collapsing it down to a classical bit with a value of either 0 or 1.

Each of these gates has a representation in circuit diagrams. Figure 2.1 shows each of these gates in a quantum circuit. q_0 and q_1 are qubits and are initialized in the $|0\rangle$ state. c_0 and c_1 are classical bits and are initialized to 0. Outlined in red, the X gate is represented by an 'X'. This changes the state of q_0 from $|0\rangle$ to $|1\rangle$. Outlined in orange are two CNOT gates. The first one goes from q_0 to q_1, so q_1 will get flipped only if q_0 is in the state $|1\rangle$. The X gate flipped q_0 from $|0\rangle$ to $|1\rangle$, so the first CNOT gate flips q_1 to the state $|1\rangle$. The second CNOT gate goes from q_1 to q_0. q_1 is in the state $|1\rangle$, so q_0 gets flipped back to the state $|0\rangle$. The SWAP gate is outlined in yellow. It swaps the states of q_0 and q_1, so q_0 gets set to the state $|1\rangle$ and q_1 gets set to the state $|0\rangle$. Outlined in green are two Hadamard gates. The one acting on q_0 is represented by an 'H' while the one acting on q_1 is represented as a U_2 gate with parameters 0 and π. A U_2 gate is equivalent to the general unitary gate from Equation 2.4, with $\theta = \frac{\pi}{2}$, the first parameter setting the value of ϕ, and the second parameter setting the value of λ. When IBM compiles circuits, it represents Hadamard gates this way. In circuits I compile, I represent Hadamard gates with an 'H' for easier readability. The general unitary gate, outlined in cyan, is denoted as a U_3 gate in qiskit with the same parameters taken by the general unitary gate defined in Equation 2.4. A controlled general unitary gate, outlined in dark blue, goes from q_1 to q_0, so the general unitary will only be applied to q_0 if q_1 is in the state $|1\rangle$. Two rotation gates, R_Y and R_Z, each take one parameter, the angle of rotation, and are outlined in purple. Finally, the measurement gates are outlined in pink. They collapse the qubits to classical bits, which can then be read out at the end of the computation.

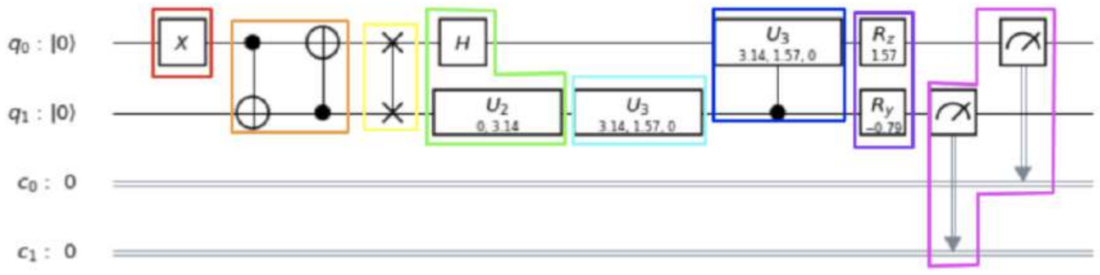


Figure 2.1: A quantum circuit diagram showcasing the gates we will use throughout the paper.

3 The Quantum Walk Problem

This paper focuses specifically on circuits that simulate a finite, discrete quantum walk. The quantum walk is the quantum analog to the classical random walk. Quantum walks, which can be viewed as a discretization of the Dirac equation, are prevalent in many quantum algorithms. For example, they provide a polynomial speedup over classical algorithms that solve the element distinctness problem[5], the triangle finding problem[6], and evaluating NAND trees[7].

A quantum walk can be visualized as N sites, arranged in a circle. The particle hops among these sites with a position and a velocity at each timestep. The velocity indicates the direction the particle will hop at the next timestep. After each hop, the velocity is scattered. Thus, if a particle has position x and velocity v, at the next timestep it will hop to position $x + v$, and its velocity will scatter according to the matrix

$$S = U(\frac{\pi}{2}, \frac{\pi}{2}, -\frac{\pi}{2}) = \begin{pmatrix} \cos(\frac{\pi}{4}) & -e^{i\frac{\pi}{2}} \sin(\frac{\pi}{4}) \\ e^{i\frac{\pi}{2}} \sin(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) \end{pmatrix}. \tag{3.1}$$

The $\frac{\pi}{4}$ argument to the cosines and sines quantifies the amount of scattering; *i.e.*, how much right-moving velocity switches to left-moving and *vice versa*. The amount of coupling between left-moving and right-moving velocities is analogous to the mass of the particle in the Dirac equation. Figure 3.1 shows one timestep of evolution of a quantum walk with eight sites.

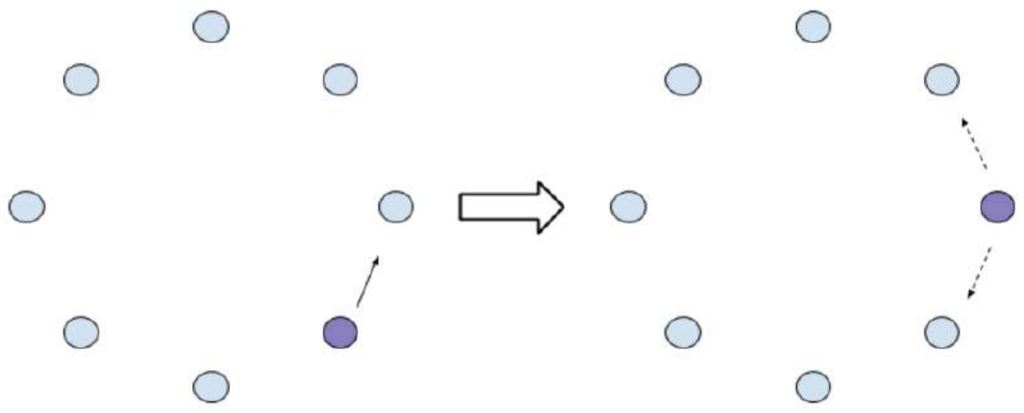


Figure 3.1: One timestep of evolution of a quantum walk with eight sites. The purple site indicates the particle's position, and the solid black arrow in the left circle indicates its velocity. On the right circle, the particle has hopped in the direction of its velocity, and the dashed black arrows indicate the scattering of the velocity after its hop.

3.1 The First Quantized Version

We consider two different data structures to simulate a quantum walk. The first quantized version uses $\log_2 N$ qubits to represent the particle's position, where N is the number of sites in the walk. For example, if $N = 4$, we would use two qubits, and the basis states $|00\rangle, |01\rangle, |10\rangle$, and $|11\rangle$ would indicate the particle at the first, second, third, and fourth site, respectively. The first quantized version uses an additional qubit to store the velocity, or direction, of the particle. The state $|0xx\rangle$ indicates the particle moving to the right, and the state $|1xx\rangle$ indicates the particle moving to the left. Overall, the first quantized version requires $\log_2 N + 1$ qubits. Figure 3.2 visualizes the role of the three qubits when $N = 4$.

Figure 3.3 shows the circuit for one timestep of evolution when $N = 4$. The most important part of the circuit is outlined in purple. This sequence of CNOT from q_0 to q_1 followed by an X on q_0 adds one to the state $|q_1 q_0\rangle$. The parts of the circuit outlined in green flip both q_0 and q_1 before and after the critical purple section when q_2 is in the state $|1\rangle$. This results in subtracting one from the state $|q_1 q_0\rangle$. Thus, the particle hops forward (+1) when the velocity is in the $|0\rangle$ state and hops

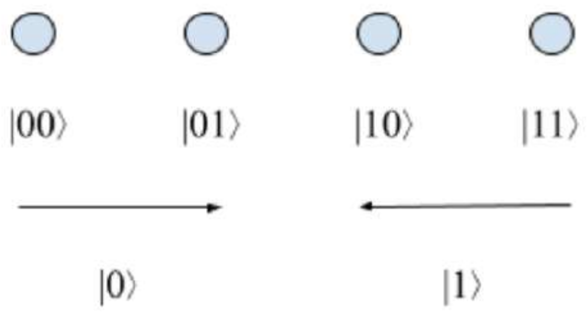


Figure 3.2: A visualization of how the first quantized version utilizes qubits. Four sites are shown as circles and labelled using two qubits. The velocity is shown as two arrows, with the right-pointing arrow labelled by the $|0\rangle$ state and corresponding to a $+1$ velocity, and the left-pointing arrow labelled by the $|1\rangle$ state and corresponding to a -1 velocity.

backwards (-1) when the velocity is in the $|1\rangle$ state. The final part of the circuit, outlined in blue, performs a scattering operation on the velocity.

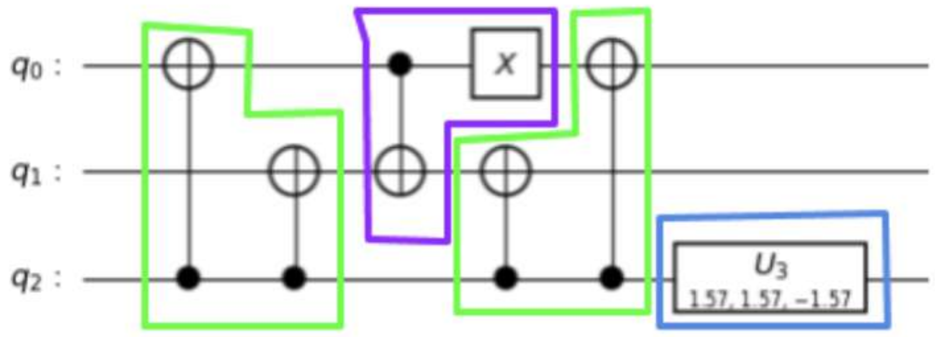


Figure 3.3: Circuit for one timestep of the first-quantized version with four sites. q_0 and q_1 are the two qubits representing position, and q_2 represents the velocity.

3.2 The Second Quantized Version

The second quantized version of the quantum walk uses $2N$ qubits, organized into pairs. Each pair represents a single site, and the particle's position within the pair indicates its direction. Thus, for $N = 2$, the state $|1000\rangle$ represents the particle at the first position hopping left while the state $|0100\rangle$ represents the particle at the first position hopping right. Similarly, the state $|0010\rangle$ represents the particle at the second position hopping left while the state $|0001\rangle$ represents the particle at the second position hopping right. Figure 3.4 visualizes the role of the four qubits when $N = 2$, and Figure 3.5 shows the circuit for one timestep of evolution. In Figure 3.5, the swaps at the beginning outlined in pink are the inter-cell swaps to move the particle to the next site. This swap does not preserve the particle's velocity, however, so a second intra-cell swap, outlined in purple for one site, is needed. Finally, outlined in blue, the scattering of velocity takes place before the second swap ends.

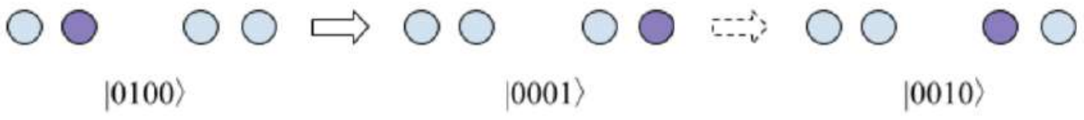


Figure 3.4: A visualization of how the second quantized version utilizes qubits. The purple coloring in the far left representation of two sites indicates the state $|0100\rangle$. Thus, the particle jumps right to state $|0001\rangle$ at the next timestep, preserving its velocity. The dashed arrow indicates the possible scattering of the velocity to move to state $|0010\rangle$, changing the particle's direction to move left.

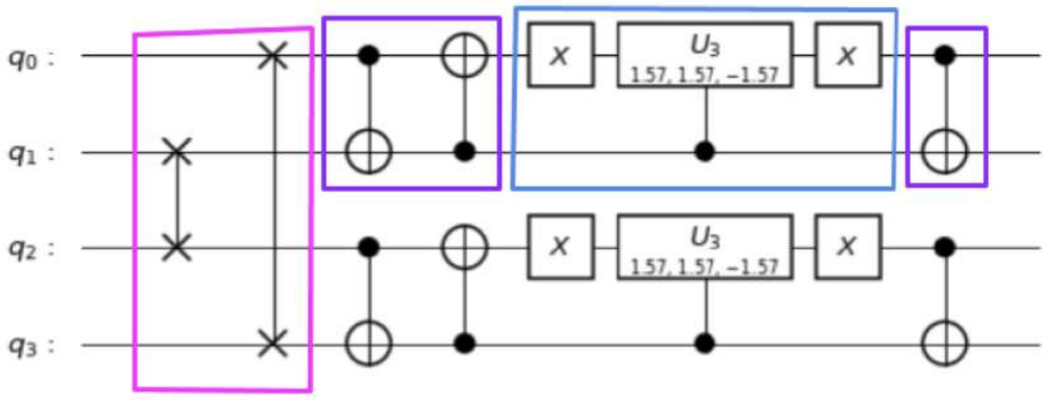
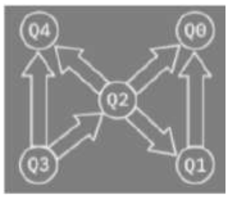


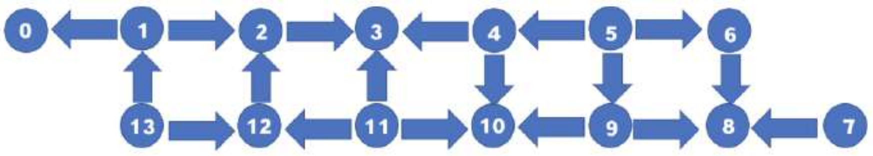
Figure 3.5: Circuit for one timestep of evolution of the second quantized version with two sites. q_0 and q_1 make up the first site, or cell, and q_2 and q_3 make up the second site, or cell.

4 IBM's Quantum Computers

I compiled the first and second quantized versions of the quantum walk onto two different architectures provided by IBM. For each architecture, IBM documents the layout of the physical qubits and the allowed connections among them. The only supported multi-qubit gate at the hardware level is the controlled-not, or CNOT gate. Figure 4.1 shows the layouts of the IBM Q 5 Tenerife, a five qubit machine, and the IBM Q 16 Melbourne, a fourteen qubit machine.



(a) IBM Q 5 Tenerife[8].



(b) IBM Q 16 Melbourne[9].

Figure 4.1: The layouts of qubits and connections among them.

From here on out I will refer to them by their backend names in qiskit: ibmqx4[10] and ibmq 16 melbourne[11], respectively. The arrows in the diagrams indicate which qubits are connected, with the tail of the arrow indicating the control qubit and the head of the arrow indicating the target qubit. Note that the ibmqx4 has a bowtie structure of two connected triangles, and the ibmq 16 melbourne is laid out into squares of connected qubits, with an extra qubit placed at either end. These structural differences will change how circuits compile onto each architecture. We will see that choosing the correct architecture can influence the size and depth of the compiled circuit.

4.1 Directly Compiling a Simple Circuit

As an example, we can compile the circuit for the first quantized version with two sites. We know that this will require $\log_2(2) + 1 = 2$ qubits, so it will fit onto both architectures introduced above. Figure 4.2 shows the circuit we want to compile.

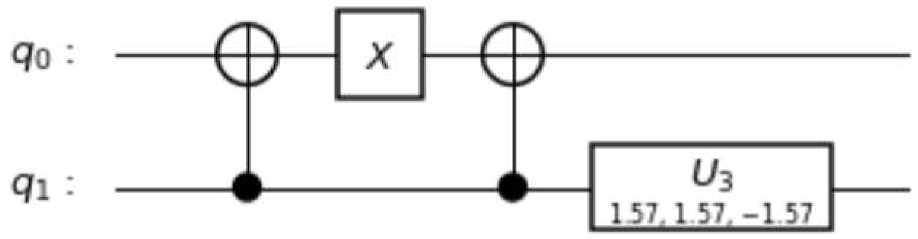
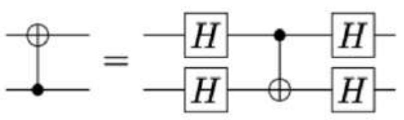


Figure 4.2: The circuit for one timestep of the first quantized version with two sites. q_0 stores the particle's position, and q_1 stores the particle's velocity.

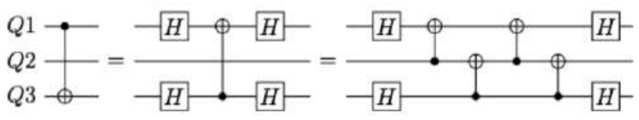
There is only one unique two-qubit gate, a CNOT from q_1 to q_0. Looking at Figure 4.1, we can see that this gate is directly supported between these qubits by both architectures, so we can compile the circuit exactly as it is. This circuit contains four gates and has a depth of four. It is simple enough that IBM also compiles it exactly onto both architectures. Thus, all the compiled versions of this circuit are identical to the circuit in Figure 4.2.

4.2 Implementing Unsupported CNOTs

In both architectures, compiling a CNOT from qubit A to qubit B is not necessarily a straightforward task. For example, in the ibmqx4 architecture, one can perform a CNOT with Q1 as the control and Q0 as the target, but the reverse direction is not supported. Fortunately, it is simple to reverse the direction of a CNOT. Figure 4.3(a) shows that adding Hadamard gates to either side of the CNOT on both qubits involved reverses the direction of the CNOT.



(a) Reversing a CNOT[12].



(b) Connecting two unconnected qubits[12].

Figure 4.3: Two examples of implementing an unsupported CNOT.

Implementing a CNOT between two unconnected qubits can be accomplished by going through a third qubit to which they are both connected. For example, Figure 4.3(b) depicts a CNOT from Q1 to Q3 in the ibmqx4, and we can remove the Hadamard gates at either end to get a CNOT from Q3 to Q1. Thus, we can see that reversing the direction of a CNOT gate adds four Hadamard gates and two timesteps to the circuit while implementing a CNOT gate for two unconnected qubits can add up to seven gates and five timesteps depending on the direction.

4.3 Implementing Advanced Two-Qubit Gates

As shown in Figure 3.5, the second quantized version of the quantum walk uses swap gates in its circuit. This two-qubit gate is not natively supported at the hardware level. We can implement a swap gate as a series of three alternating CNOTs between the two qubits getting swapped[13]. We already know how to reverse the direction of a CNOT, so implementing a SWAP gate between two connected qubits will require five timesteps and seven gates. Figure 4.4 shows this decomposition assuming that we can perform a CNOT from q_0 to q_1. Note that the direction of the CNOTs can go either way in this case, so we could just as easily implement this swap gate given a supported CNOT from q_1 to q_0 without adding any extra gates or timesteps.

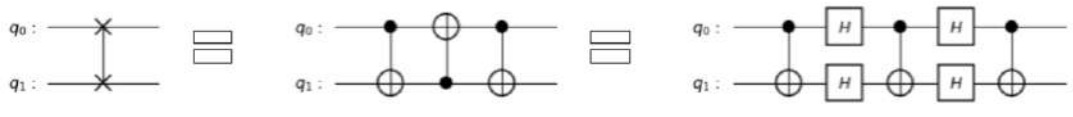


Figure 4.4: Reducing a SWAP gate to a series of CNOTs.

Figure 3.5 also depicts a controlled general unitary gate, or controlled-U gate. This is more tricky to implement, as the parameters to the controlled-U gate determine its decomposition into CNOTs and single-qubit gates. In general,

$$U(\theta, \phi, \lambda) = R_Z(\lambda)R_Y(\theta)R_Z(\phi), \tag{4.1}$$

where R_Z, R_Y are rotations about the Z-axis and Y-axis, respectively[14]. Let

$$A = R_Z(\lambda)R_Y(\frac{\theta}{2}), \tag{4.2}$$

$$B = R_Y(-\frac{\theta}{2})R_Z(-\frac{\phi+\lambda}{2}), \tag{4.3}$$

$$C = R_Z(\frac{\phi-\lambda}{2}). \tag{4.4}$$

Then the equivalence shown in Figure 4.5 holds[14].

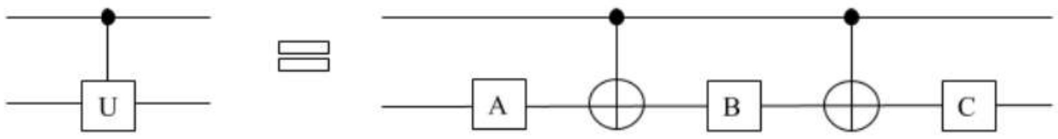


Figure 4.5: Circuit for reducing a controlled unitary gate to a series of single-qubit and CNOT gates[14].

If the control qubit is in the state $|0\rangle$, then the circuit should not change the target qubit; *i.e.*, $ABC = I$ should be true, and if the control qubit is in the state $|1\rangle$, then the circuit should apply U to the target qubit; *i.e.*, $AXBXC = U$ should be true[14]. Note that for any rotation gate, R, and the X gate, the following equivalences hold[14]:

$$R(\theta + \phi) = R(\theta)R(\phi), \tag{4.5}$$

$$R(0) = I, \tag{4.6}$$

$$XR(\theta) = R(-\theta)X, \tag{4.7}$$

$$X = X^{-1}. \tag{4.8}$$

We can use these identities to prove $ABC = I^{14}$:

$$\begin{aligned}
ABC &= R_Z(\lambda)R_Y(\frac{\theta}{2})R_Y(-\frac{\theta}{2})R_Z(-\frac{\phi+\lambda}{2})R_Z(\frac{\phi-\lambda}{2}) && (4.9)\\
&= R_Z(\lambda)R_Y(\frac{\theta}{2}-\frac{\theta}{2})R_Z(-\frac{\phi+\lambda}{2}+\frac{\phi-\lambda}{2}) && (4.10)\\
&= R_Z(\lambda)R_Y(0)R_Z(-\lambda) && (4.11)\\
&= R_Z(\lambda)IR_Z(-\lambda) && (4.12)\\
&= R_Z(\lambda)R_Z(-\lambda) && (4.13)\\
&= R_Z(\lambda-\lambda) && (4.14)\\
&= R_Z(0) && (4.15)\\
&= I && (4.16)
\end{aligned}$$

Furthermore, we can show $AXBXC = U^{14}$:

$$\begin{aligned}
AXBXC &= R_Z(\lambda)R_Y(\frac{\theta}{2})XR_Y(-\frac{\theta}{2})R_Z(-\frac{\phi+\lambda}{2})XR_Z(\frac{\phi-\lambda}{2}) && (4.17)\\
&= R_Z(\lambda)Ry(\frac{\theta}{2})R_Y(\frac{\theta}{2})XXR_Z(\frac{\phi+\lambda}{2})R_Z(\frac{\phi-\lambda}{2}) && (4.18)\\
&= R_Z(\lambda)R_Y(\frac{\theta}{2}+\frac{\theta}{2})IR_Z(\frac{\phi+\lambda}{2}+\frac{\phi-\lambda}{2}) && (4.19)\\
&= R_Z(\lambda)R_Y(\theta)R_Z(\phi) && (4.20)\\
&= U && (4.21)
\end{aligned}$$

Thus, we have decomposed a controlled general unitary gate into CNOTs and single-qubit rotations.

For our scattering operation, we have $\theta = \phi = \frac{\pi}{2}$ and $\lambda = -\frac{\pi}{2}$. Thus,

$$\begin{aligned}
A &= R_Z(-\frac{\pi}{2})R_Y(\frac{\pi}{4}), && (4.22)\\
B &= R_Y(-\frac{\pi}{4})R_Z(0) = R_Y(-\frac{\pi}{4}), && (4.23)\\
C &= R_Z(\frac{\pi}{2}). && (4.24)
\end{aligned}$$

Figure 4.6 shows the decomposed circuit for the controlled general unitary gate shown in Figure 3.5 in terms of rotation gates and CNOTs, using the equivalence shown in Figure 4.5, with the A, B and C gates defined in Equations 4.22, 4.23, and 4.24.

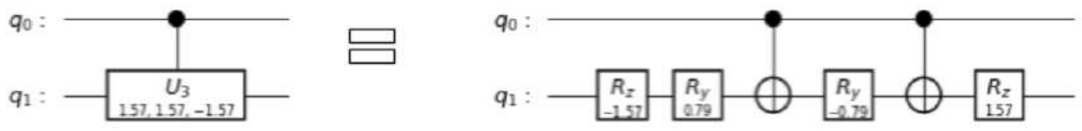


Figure 4.6: The decomposed circuit for the controlled unitary gate we use for scattering in the second quantized version.

5 Compiling the First Quantized Version

We already compiled the circuit for the first quantized version with two sites, but it was a trivial example. It is more interesting to compile the circuit shown in Figure 3.3. That circuit includes CNOTs from q_2 to q_0, q_2 to q_1, and q_0 to q_1. The ibmqx4 supports CNOTs from q_2 to q_0 and q_2 to q_1, but the CNOT between q_0 and q_1 goes from q_1 to q_0. Therefore, we can compile the circuit by switching q_0 and q_1, so the CNOT instead goes from q_1 to q_0, which is supported. Figure 5.1 shows my compiled circuit. Note that this circuit is identical to the one in Figure 3.3, just with q_0 and q_1 exchanged. This circuit has seven gates and a depth of six, identical to the size and depth of the circuit in Figure 3.3.

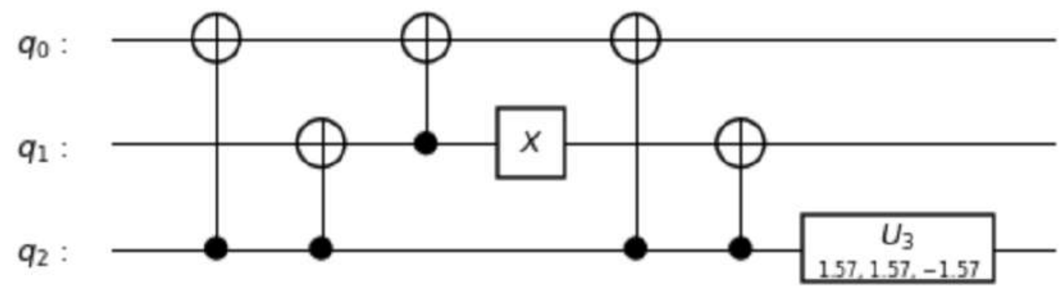


Figure 5.1: The circuit from Figure 3.3 I compiled onto the ibmqx4 architecture.

IBM, however, does not produce such a concise circuit when it compiles the one in Figure 3.3 onto the ibmqx4. IBM's documentation of qiskit does not tend to match what is actually implemented in the latest version, so it is unclear exactly how IBM compiles circuits and what degree of randomness goes into each compilation. For simpler circuits, such as the ones we have compiled so far, IBM always produces the same compiled circuit. For larger circuits, however, IBM sometimes produces different circuits of varying depth and size for subsequent compilations of the same circuit.

Figure 5.2 shows the circuit IBM produces when compiling the circuit from Figure 3.3. It has ten gates and a depth of eight. The extra three gates and two timesteps come from having to switch the direction of the CNOT from q_0 to q_1. Note that the Hadamard gate on q_0 after the reversed CNOT gets combined with the following X gate into a single unitary gate operation, outlined in purple. IBM did not switch q_0 and q_1 and thus could not perform the CNOT between them directly. This demonstrates that choosing the initial assignment of physical qubits to be different from the order given in the uncompiled circuit can lead to savings in both gates and depth in the compiled circuit.

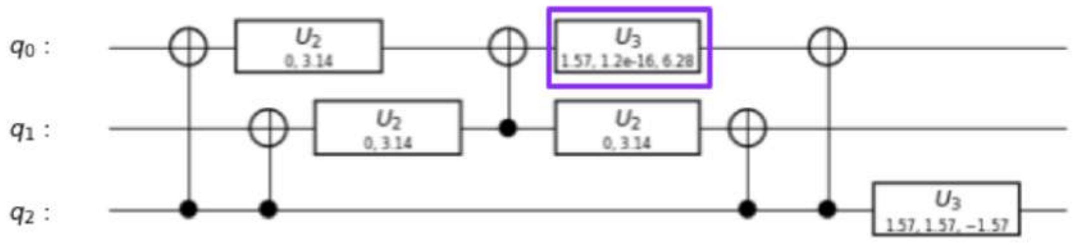


Figure 5.2: The circuit from Figure 3.3 IBM compiled onto the ibmqx4.

We can also compile the circuit from Figure 3.3 onto the ibmq 16 melbourne. The square layout of this architecture does not lend itself as readily to the compilation of this particular circuit as the triangle layout of the ibmqx4 did. By choosing $q_0 = q_{12}$, $q_1 = q_3$, and $q_2 = q_{11}$, we can directly compile the CNOTs from q_2 to q_0 and q_2 to q_1 as CNOTs from q_{11} to q_{12} and q_{11} to q_3 are supported, but the CNOT from q_0 to q_1 has to go through another qubit, from q_{12} to q_2 to q_3 as q_{12} and q_3 are not connected. Using the circuit from Figure 4.3(b) without the Hadamard gates, we can compile this CNOT, only adding three gates and three timesteps to the circuit. Figure 5.3 shows my compiled circuit with ten gates and a depth of nine.

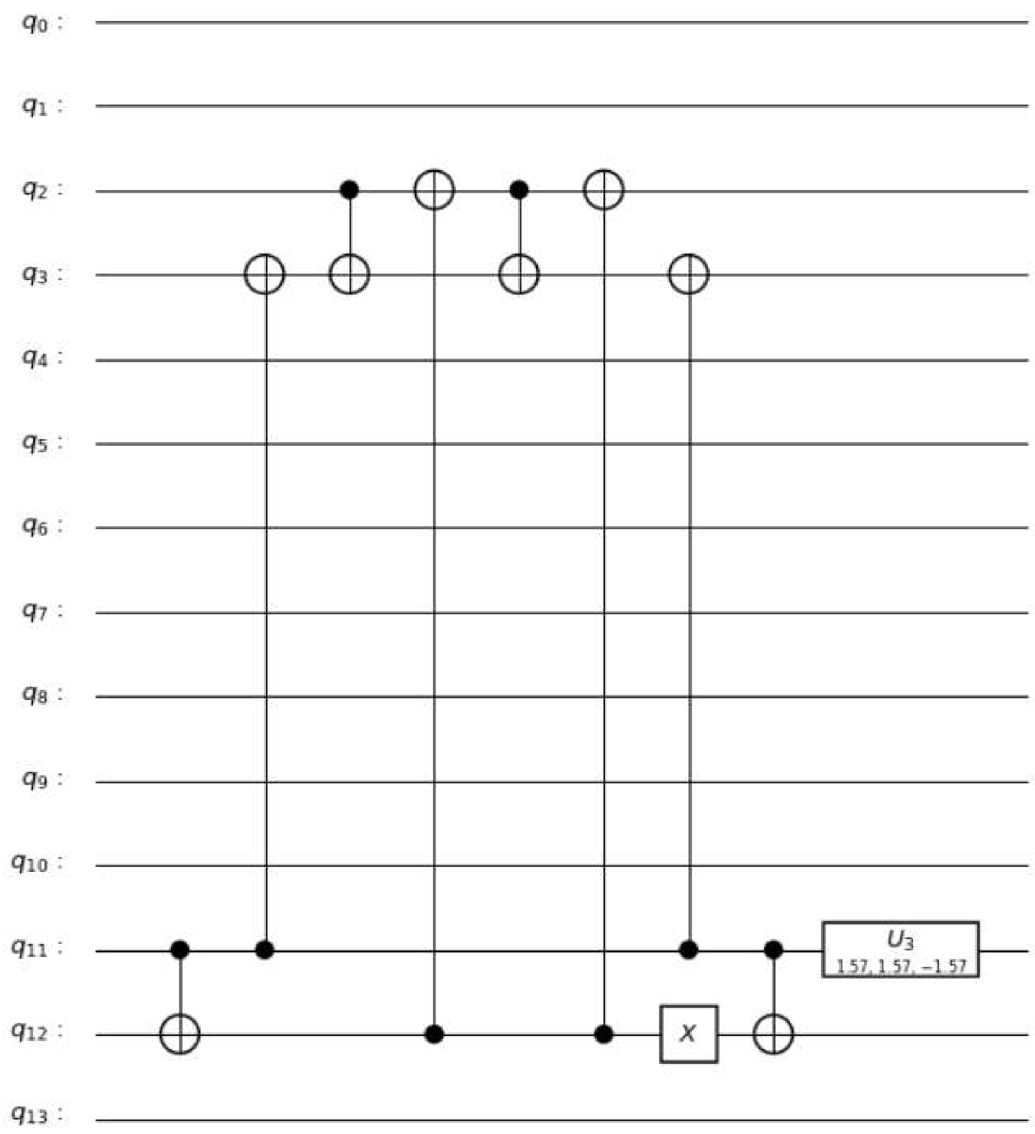


Figure 5.3: The circuit from Figure 3.3 I compiled onto the ibmq 16 melbourne, showing all the available qubits in the architecture.

Figure 5.4 shows the circuit IBM compiled onto the ibmq 16 melbourne, which has twenty-two gates and a depth of sixteen. IBM chose to use the same three qubits indicated by the uncompiled circuit, q_0, q_1, and q_2, which are arranged linearly in this architecture. This layout requires more manipulation of CNOTs to compile the circuit than my choice of four qubits arranged in a square does. IBM did, however, preserve the original number of qubits in the circuit while I added one for convenience.

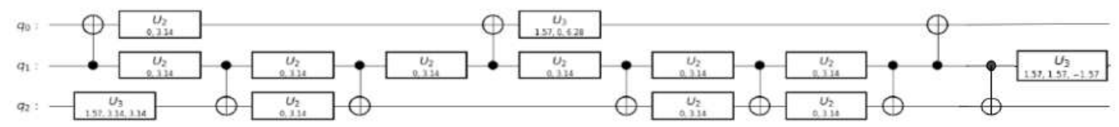


Figure 5.4: The circuit from Figure 3.3 IBM compiled onto the ibmq 16 melbourne.

6 Compiling the Second Quantized Version

6.1 Two Sites

The second quantized version with two sites requires four qubits, so it will fit onto both the five qubit and fourteen qubit architectures provided by IBM. Let us start by compiling it onto the ibmqx4. We can perform the two initial swaps from Figure 3.5 by first swapping q_0 and q_2, then simultaneously swapping q_2 with q_3 and q_0 with q_1, and finally swapping q_0 and q_2 again. Figure 6.1 shows this series of swaps, tracking the positions of the original qubits' values.

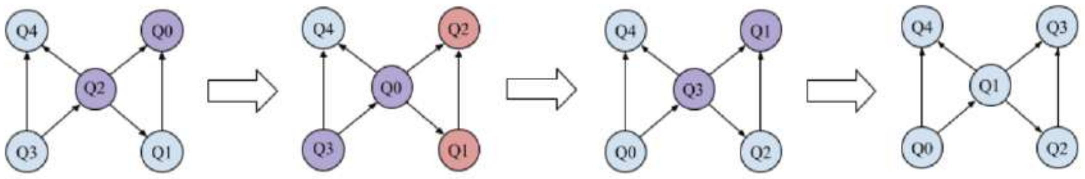


Figure 6.1: The steps to swap qubits q_0 with q_3 and q_1 with q_2. The colored pairs at each stage indicate which two qubits will be swapped next. Note that in this diagram it is the original values of the qubits I am tracking. The physical layout of qubits in the architecture remains constant.

Figure 6.2 shows the circuit I compiled onto the ibmqx4, which has fifty gates and a depth of twenty-four. Outlined in purple, more than half of the gates and timesteps are spent implementing the swaps at the beginning of the circuit. The remaining inner swaps among pairs and controlled scattering compile well onto this architecture. The gates outlined in blue indicate the compilation of the controlled scattering. Due to combining the X gates surrounding the controlled scattering in Figure 3.5 and Hadamard gates to flip the direction of the CNOTs with the rotation gates in Figure 4.6, this sub-circuit does not exactly match the one from Figure 4.6. The circuit in Figure 4.6 was isolated, and when placed into the larger circuit from Figure 3.5, its single-qubit gates combine with other single-qubit gates to produce unitary gates with different parameters.

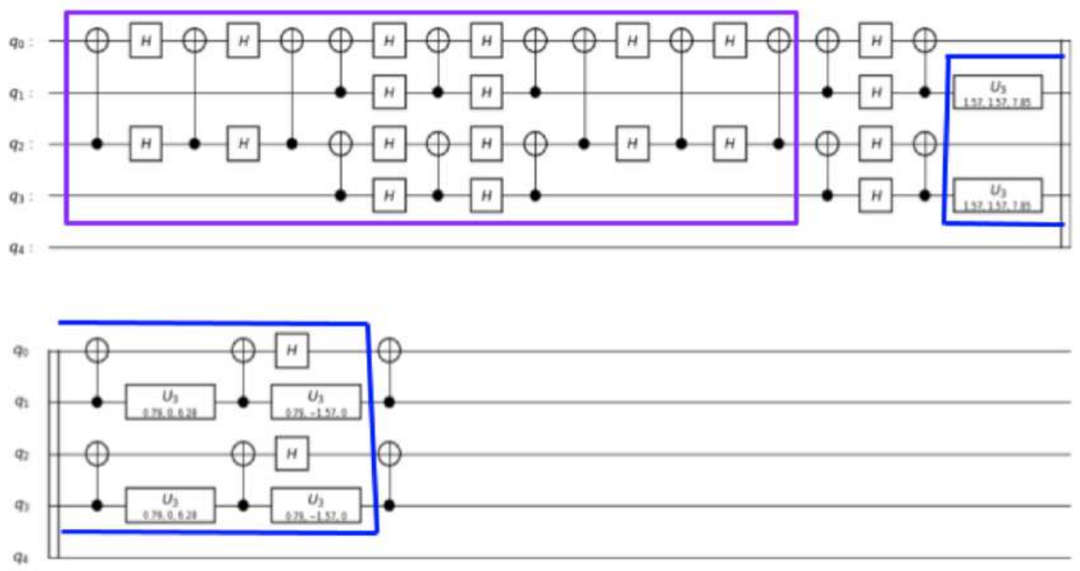


Figure 6.2: The circuit from Figure 3.5 I compiled onto the ibmqx4 architecture.

Figure 6.3 shows the circuit IBM compiled with sixty gates and a depth of twenty-eight. IBM employs a similar initial swap strategy as I do, but does not implement it as efficiently. By switching qubits around, IBM's compiler could have benefitted from CNOTs running in opposite directions to the ones it used.

I also compiled the circuit shown in Figure 3.5 onto the ibmq 16 melbourne. As is clear from Figure 6.4, this architecture with its squares is more suited to the second quantized version than the triangles of the ibmqx4 are. On this architecture, only thirty-six gates and fourteen timesteps are required. The only optimization I made was in the initial swap routines; what comes after is identical to the ibmqx4 compiled circuit. The square architecture allows implementing the two swaps at

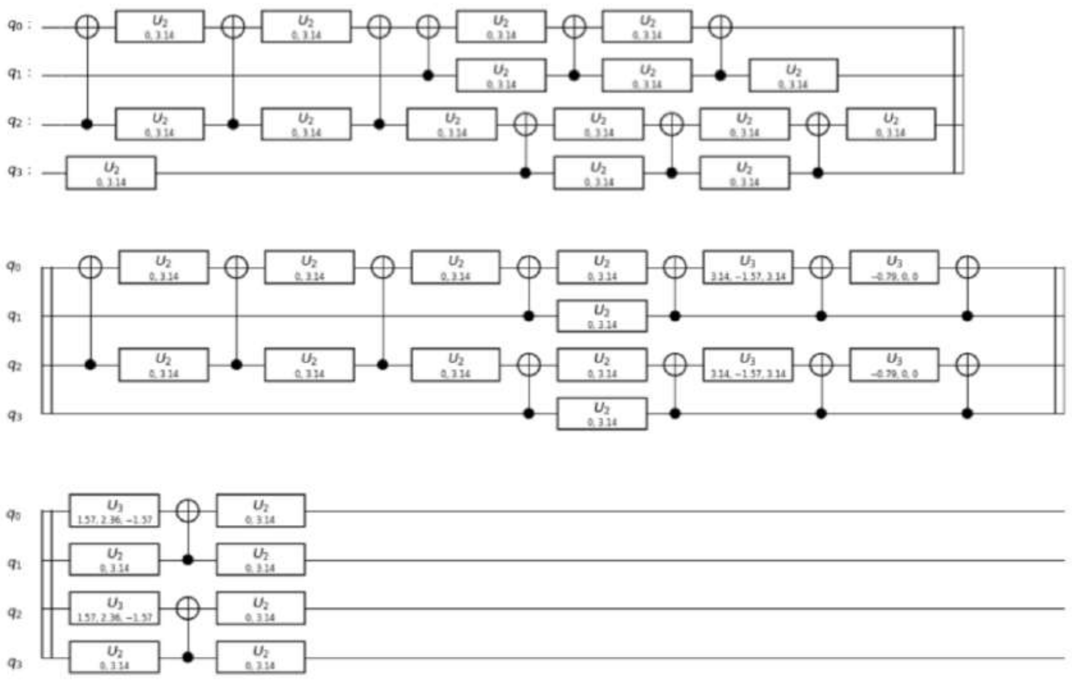


Figure 6.3: The circuit from Figure 3.5 IBM compiled onto the ibmqx4.

the beginning directly and in parallel, leading to a swap circuit with only fourteen gates and five timesteps, compared to the twenty-eight gates and fifteen timesteps used to implement the same two swaps on the ibmqx4.

IBM's compiler behaved atrociously on this compilation of the circuit from Figure 3.5, producing a circuit with 149 gates and 104 timesteps, much larger than its compilation onto the ibmqx4 was. IBM chose qubits q_0, q_1, q_2, and q_{13} at the left corner of the machine. This layout was worse than the triangles the ibmqx4 offers. Figure 6.5 shows the long circuit across several pages.

Despite the worse performance of IBM's compiler on the ibmq 16 melbourne architecture, it is clear from my compilations that this machine provides a natural architecture for the second quantized version while the ibmqx4 does not. This is primarily due to the square structures allowing the initial inter-cell swaps to be executed in parallel as well as the intra-cell swaps and scattering that can be executed in parallel on both architectures.

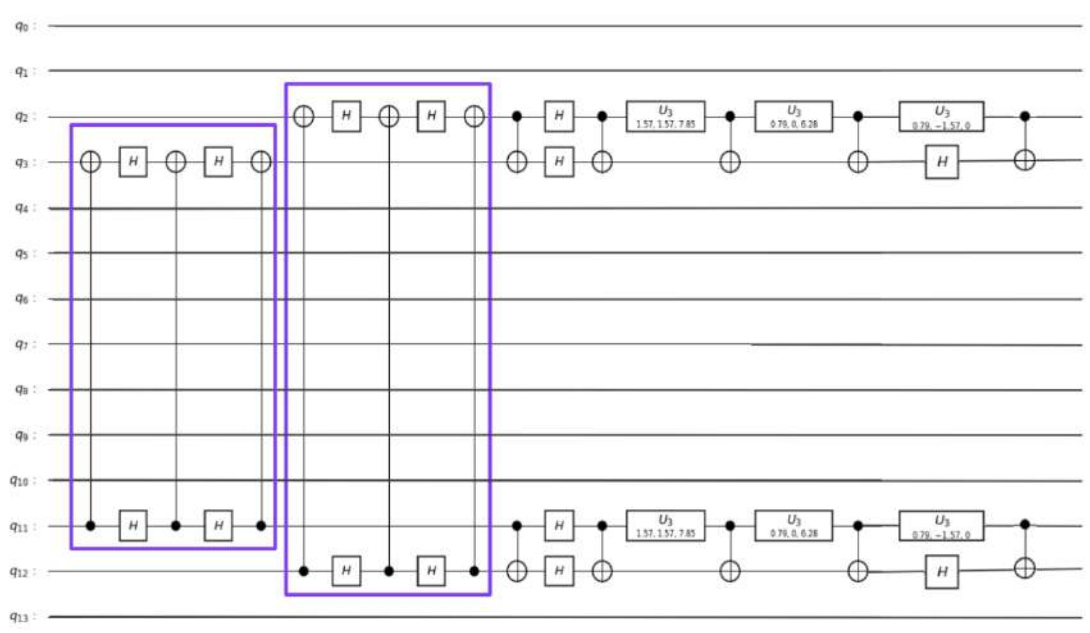
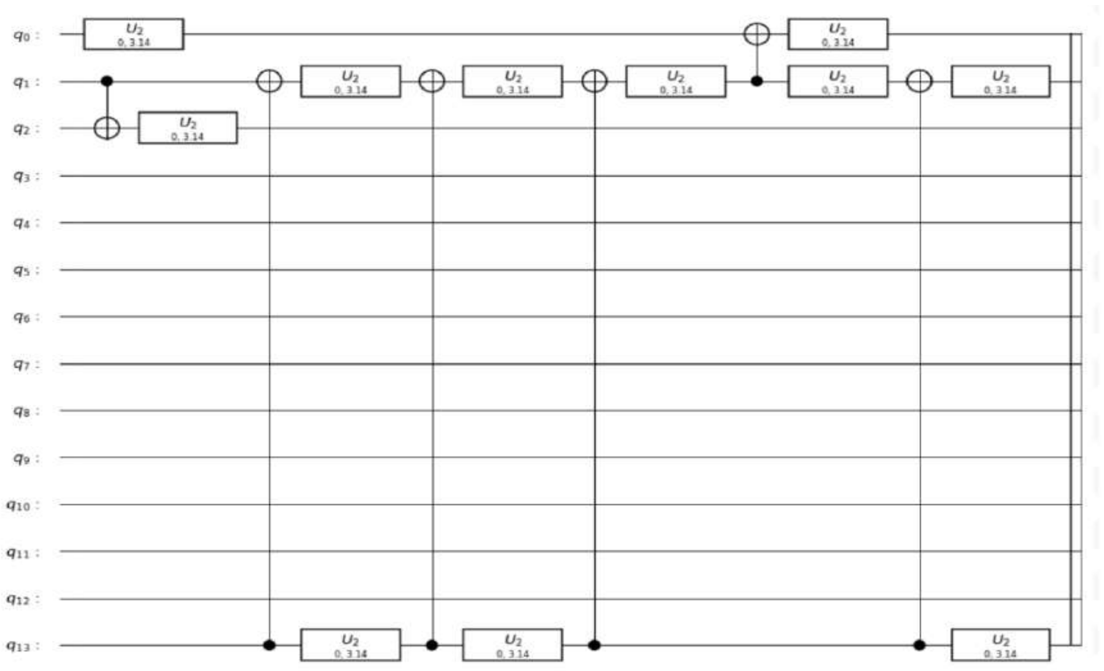
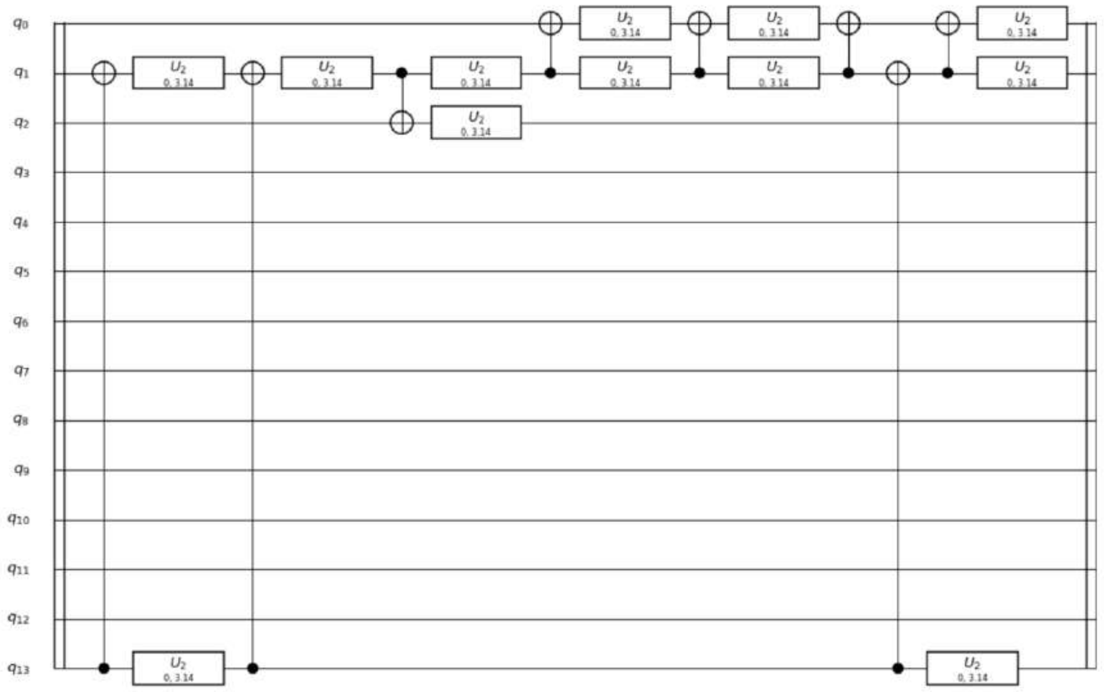


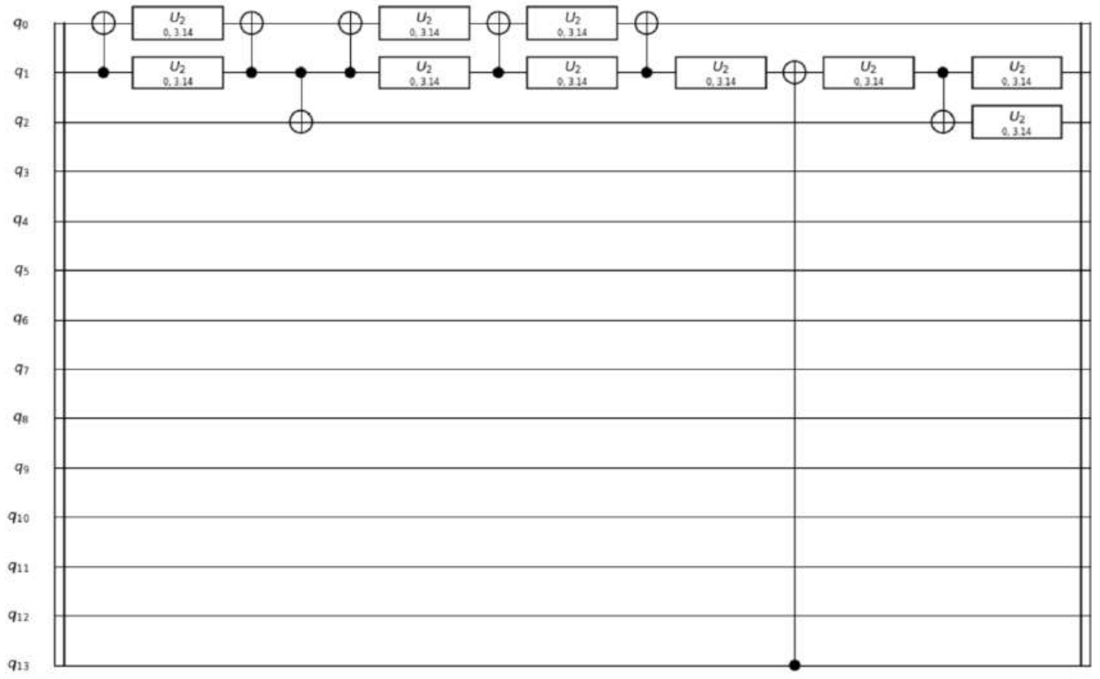
Figure 6.4: The circuit Figure 3.5 I compiled onto the ibmq 16 melbourne architecture. The two swap subcircuits outlined in purple can be executed in parallel.

It is notable in this case that simply choosing the right architecture is not enough to compile a good circuit. The most important factor is the layout of qubits chosen; the architecture matters only in what layouts it provides as options.

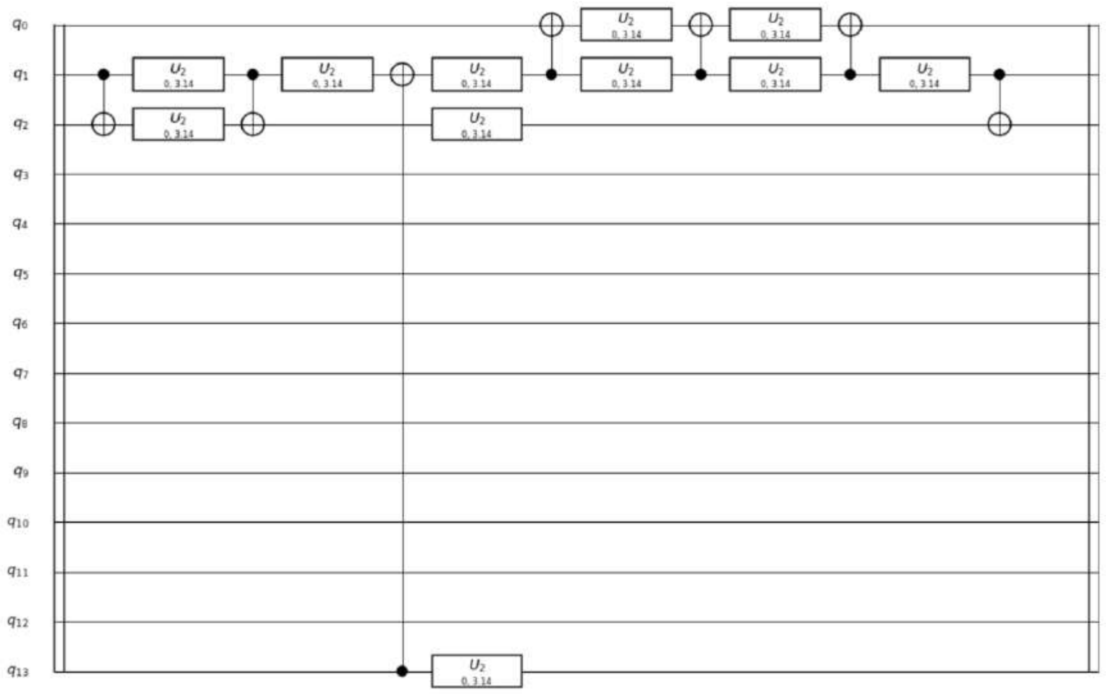


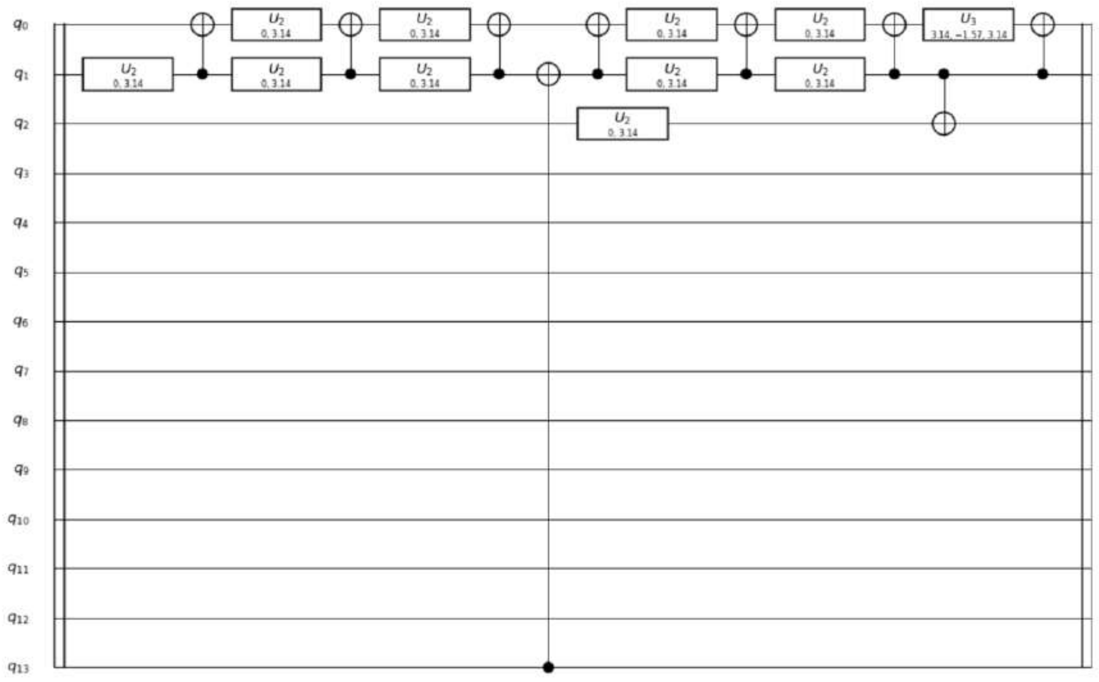


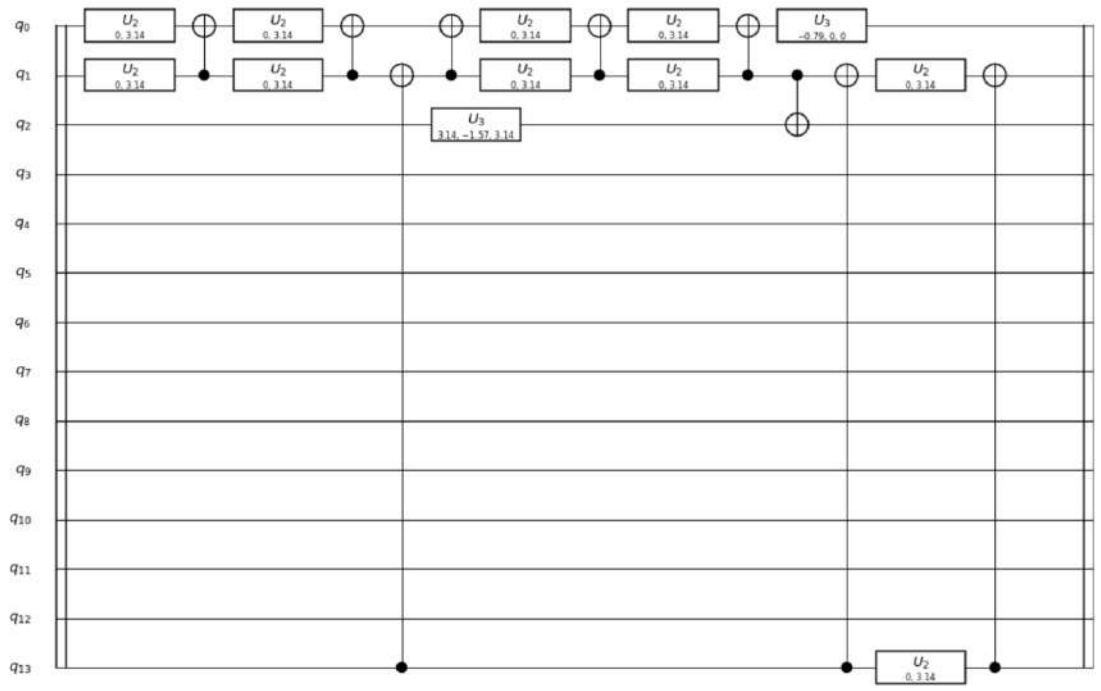
q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
q_{10}
q_{11}
q_{12}
q_{13}
U_2
0, 3.14

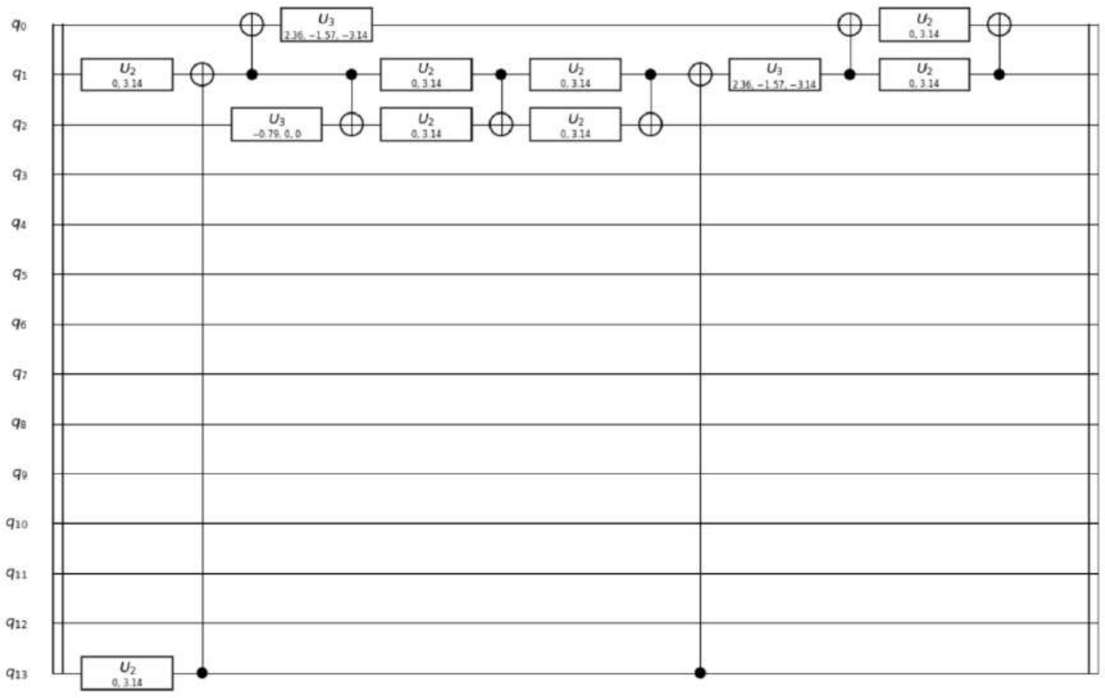


q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
q_{10}
q_{11}
q_{12}
q_{13}
U_2
0, 3.14


q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
q_{10}
q_{11}
q_{12}
q_{13}
U_2
0, 3.14


q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
q_{10}
q_{11}
q_{12}
q_{13}
U_2
0, 3.14
U_3
3.14, −1.57, 3.14


q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
q_{10}
q_{11}
q_{12}
q_{13}
U_2 0, 3.14
U_3 3.14, −1.57, 3.14
U_3 −0.79, 0, 0


q_0
q_1
q_2
q_3
q_4
q_5
q_6
q_7
q_8
q_9
q_{10}
q_{11}
q_{12}
q_{13}
U_2 0, 3.14
U_3 2.36, −1.57, −3.14
U_3 −0.79, 0, 0

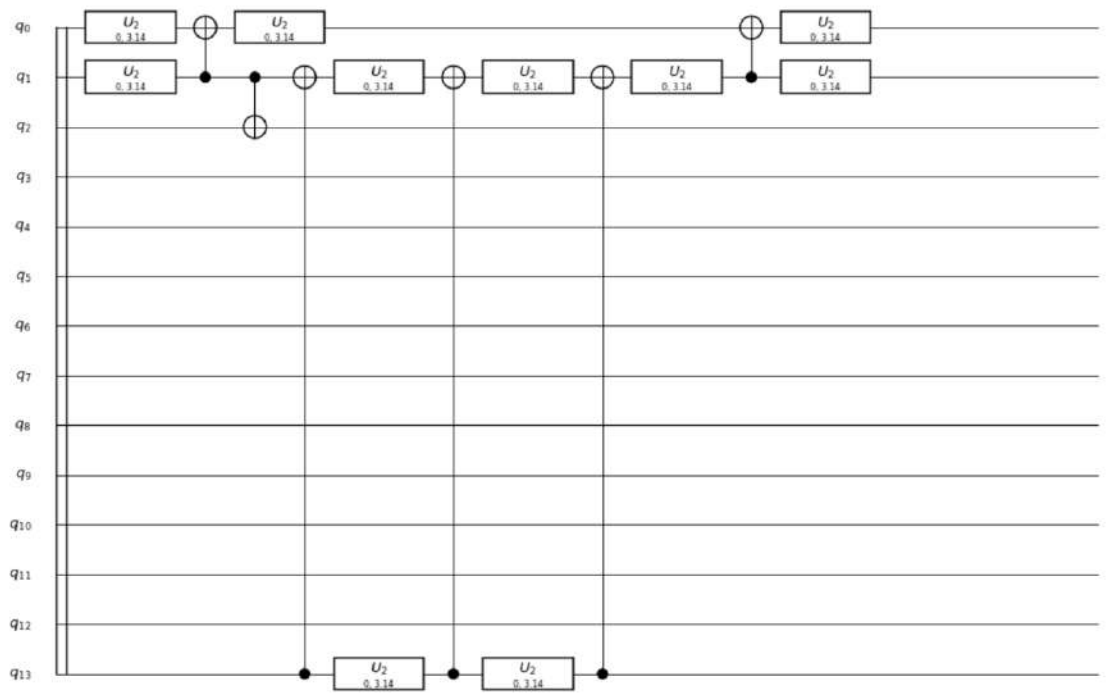


Figure 6.5: The circuit from Figure 3.5 IBM compiled onto the ibmq 16 melbourne architecture.

6.2 Four Sites

The advantages of the ibmq 16 melbourne architecture for the second quantized version can further be seen by compiling the circuit for the second quantized version with four sites. As four sites require eight qubits to implement, this circuit will not fit on the ibmqx4 architecture, which only has five qubits. Figure 6.6 shows the uncompiled circuit, and Figure 6.7 shows my compiled circuit on the ibmq 16 melbourne architecture. Both circuits are essentially the same as their two site counterparts, just doubled in number of qubits and gates. In fact, the compiled circuit on the ibmq 16 melbourne still only requires fourteen timesteps while the number of gates has doubled to seventy-two. By choosing the right rectangle of qubits, the inter-cell swaps can still all be done in parallel, followed by the remaining intra-cell swaps and scattering done in parallel as before. This suggests that the second quantized version might scale better than the first quantized version in terms of circuit depth. However, it scales linearly rather than logarithmically in terms of number of qubits, making it quickly impossible to compile for larger N onto IBM's publicly available architectures.

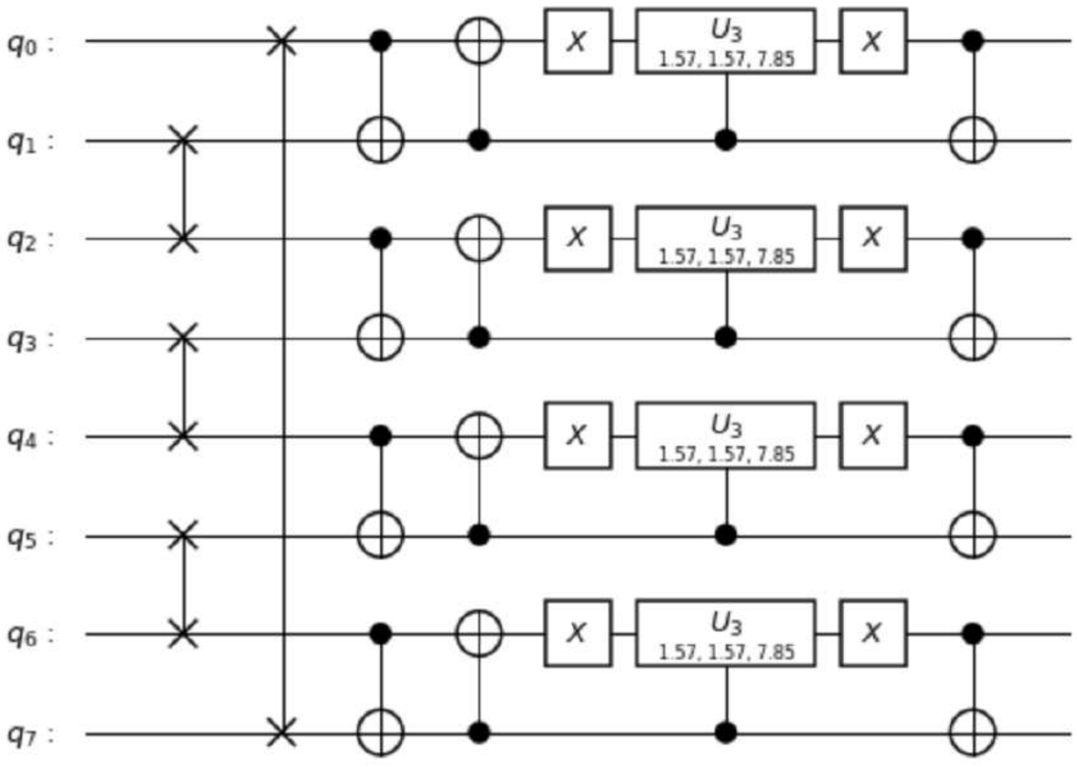


Figure 6.6: This circuit shows one timestep of evolution for the second quantized version of the quantum walk with four sites.

IBM produced a much better circuit compiling the one in Figure 6.6 onto the ibmq 16 melbourne than it did compiling the circuit in Figure 3.5 onto the same architecture. In fact, IBM's compiled circuit contains eighty-one gates and has a depth of sixteen, only nine gates and two timesteps more than my compiled circuit. Figure 6.8 shows IBM's circuit. It is noticeably shorter and more efficient than the compiled circuit it produced for the two site second quantized version. This improvement can be explained by the layout of physical qubits IBM chose to compile onto. Instead of choosing an odd shape of qubits at the edge of the ibmq 16 melbourne architecture, IBM used the same rectangle of qubits as I did. This allowed IBM to execute most of the circuit in parallel among pairs of qubits, reducing the number of gates and timesteps it used swapping qubits around in the two site version.

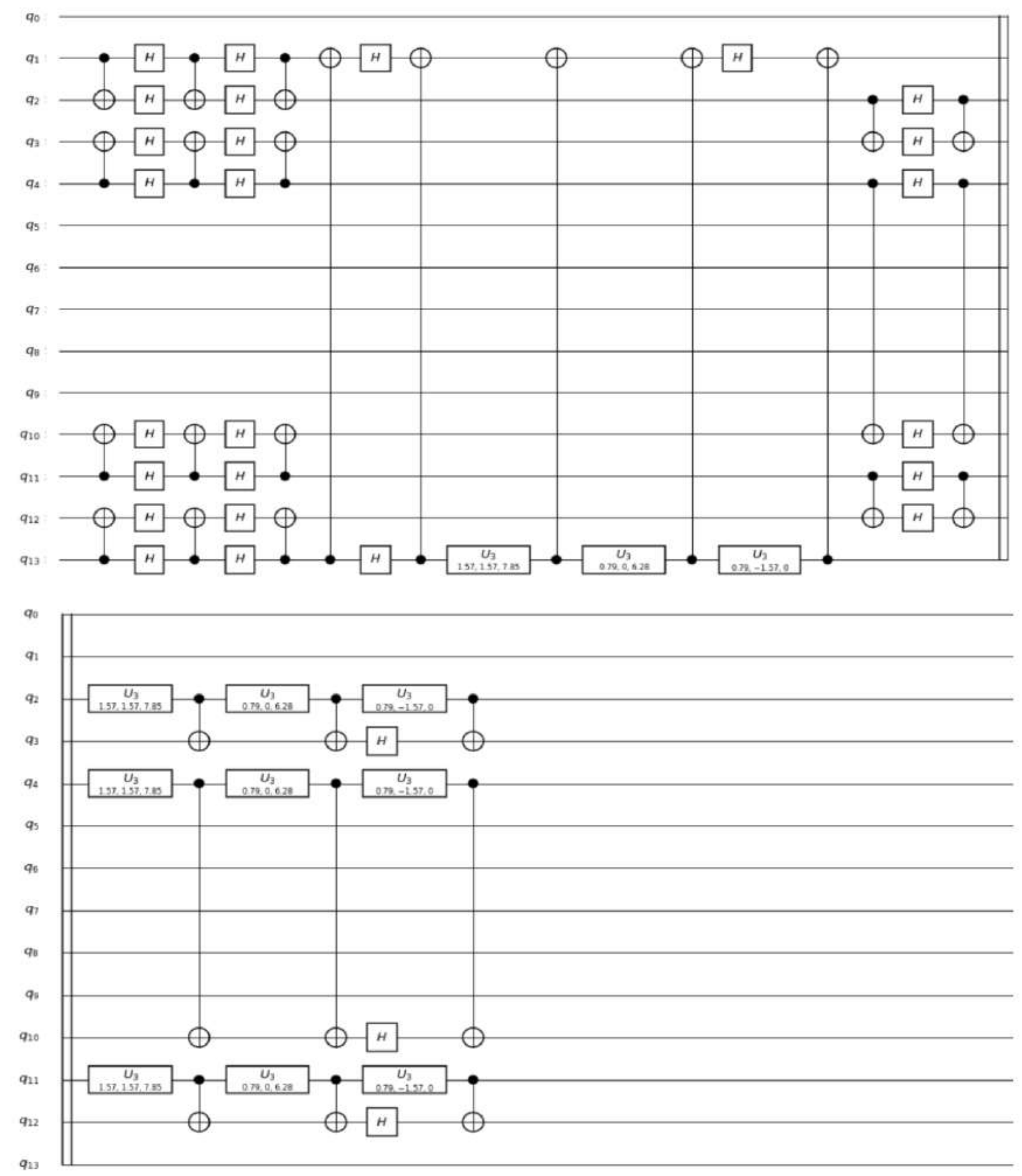
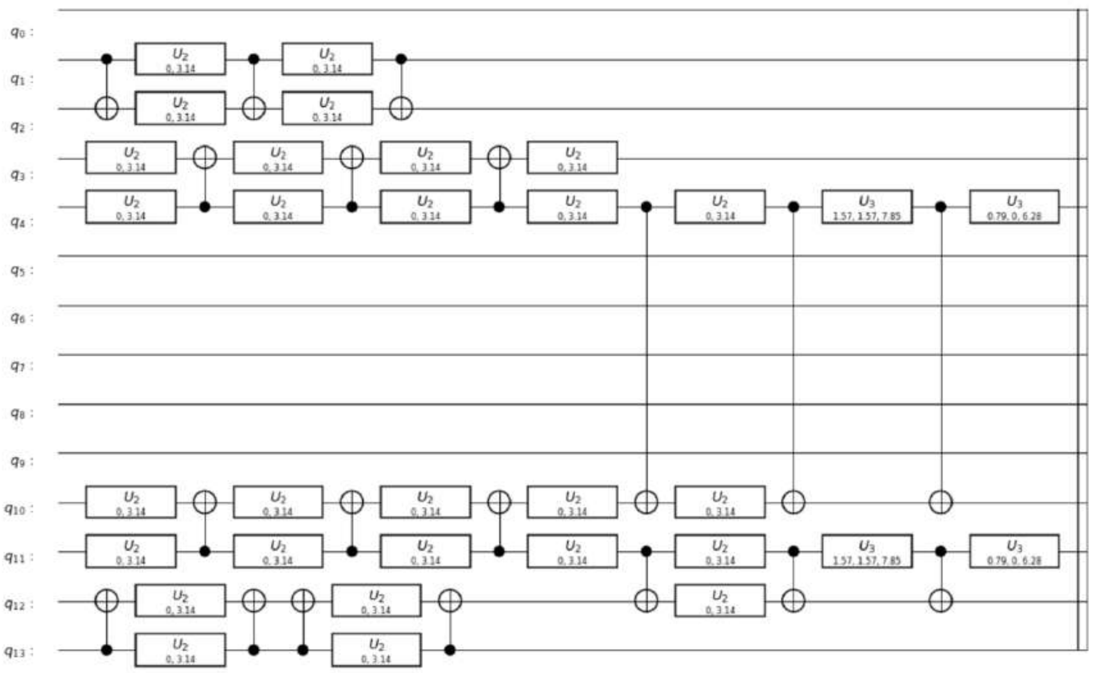
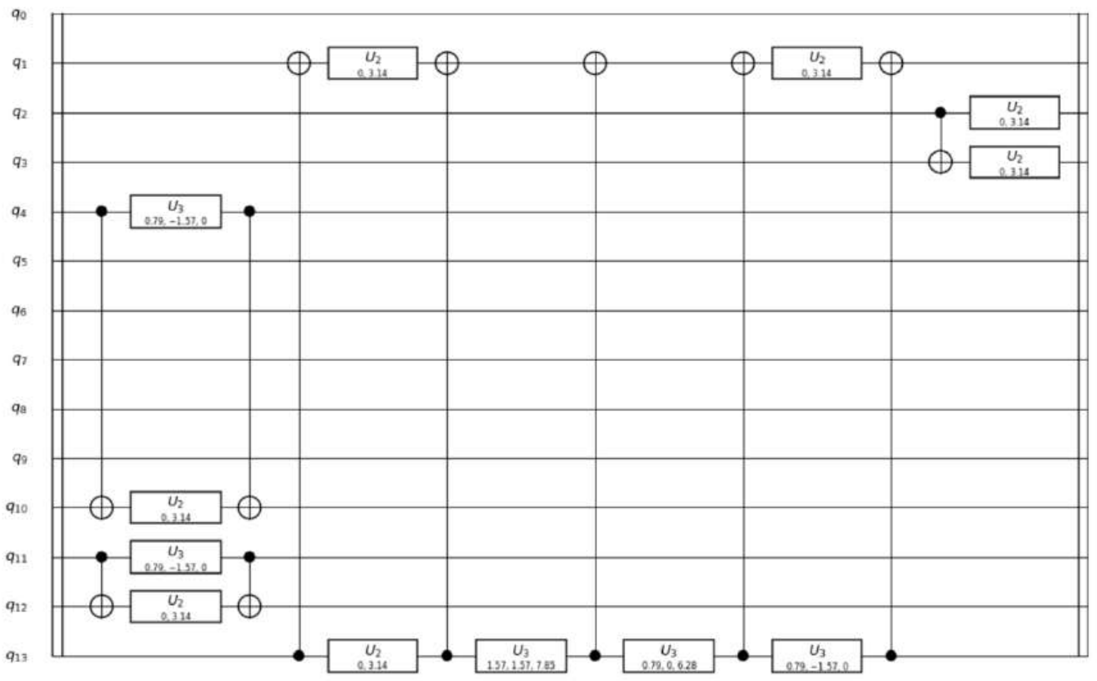


Figure 6.7: The circuit from Figure 6.6 I compiled onto the ibmq 16 melbourne architecture.



q0 :
q1 :
q2 :
q3 :
q4 :
q5 :
q6 :
q7 :
q8 :
q9 :
q10 :
q11 :
q12 :
q13 :
U2 0, 3.14
U3 1.57, 1.57, 7.85
U3 0.79, 0, 6.28



q0
q1
q2
q3
q4
q5
q6
q7
q8
q9
q10
q11
q12
q13
U2 0, 3.14
U3 0.79, −1.57, 0
U3 1.57, 1.57, 7.85
U3 0.79, 0, 6.28

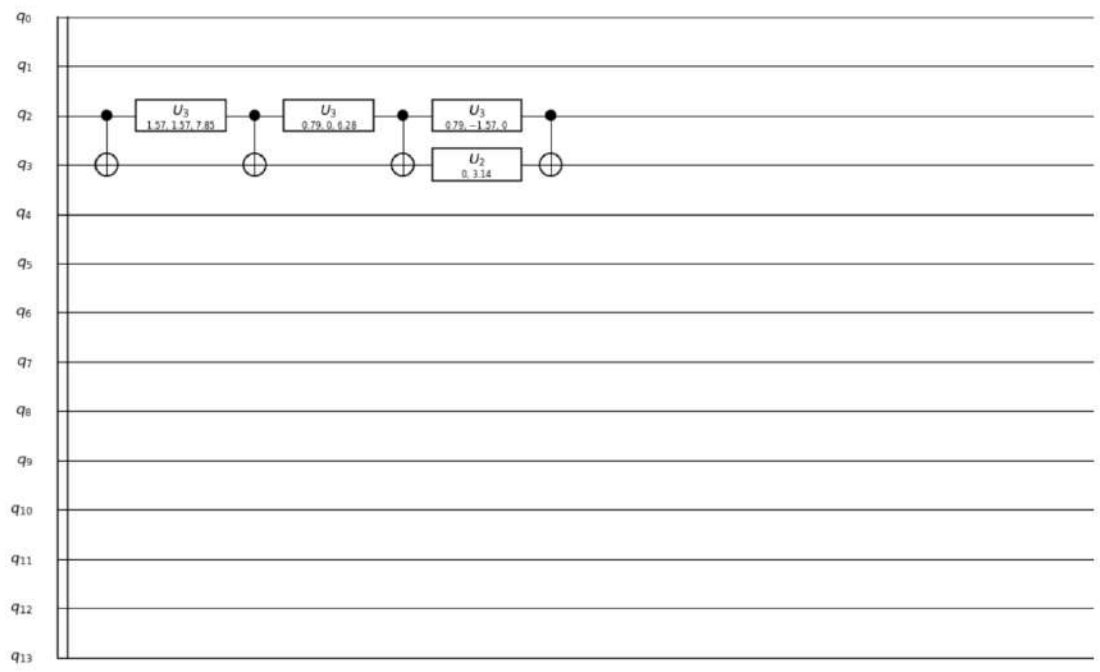


Figure 6.8: The circuit from Figure 6.6 IBM compiled onto the ibmq 16 melbourne.

7 Running Circuits

I ran the circuits compiled in previous chapters to experimentally compare them. For each run, I computed the ℓ^1 distance between the distribution of measured states and the expected distribution of states. I also computed the ℓ^1 distance between the corresponding uniform distribution of states and the expected distribution to quantify what pure noise would look like. The possible values of the ℓ^1 distance between two distributions range from 0 (when the two distributions are identical) to 1 (when the two distributions are disjoint). Thus, the lower the ℓ^1 distance between them, the closer two distributions are to each other.

7.1 First Quantized Version, Two Sites

Figure 7.1 shows the expected distribution of states after running the circuit shown in Figure 4.2 along with the corresponding uniform distribution of states generated by two qubits. The ℓ^1 distance between these two distributions is 0.5. Thus, we expect the ℓ^1 distance between the distributions of measured states and the expected distribution for the first quantized version with two sites to be significantly less than 0.5. If the ℓ^1 distances are not significantly less than 0.5, then we can conclude that the measured distributions of states are no better than noise.

Figure 7.2 shows example distributions of runs of the circuit from Figure 4.2 on the ibmqx4 and ibmq 16 melbourne. The circuit run on the ibmqx4 produces a distribution with an ℓ^1 distance of 0.177, and the circuit run on the ibmq 16

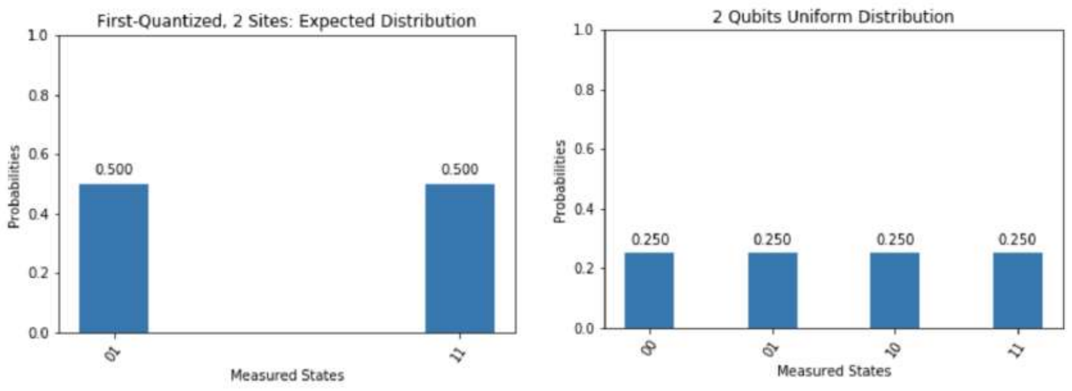


Figure 7.1: Expected distribution of states for the first quantized version with two sites (left). Uniform distribution of states generated by two qubits (right).

melbourne produces a distribution with an ℓ^1 distance of 0.099. Both ℓ^1 distances fall well below 0.5, so we can conclude that these distributions contain the signal we expected to see. Furthermore, because the same circuit was run on both machines, these results suggest that the ibmqx4 is noisier than the ibmq 16 melbourne.

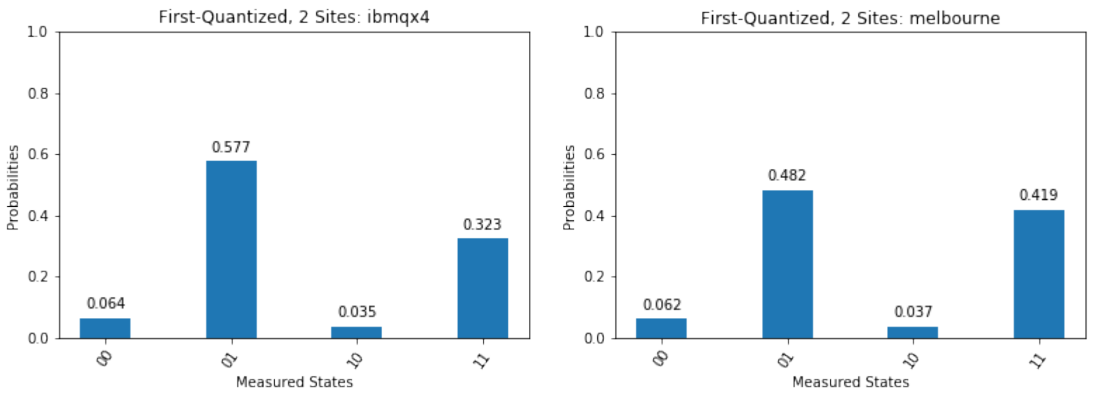


Figure 7.2: Example distributions of measured states after running the circuit for the first quantized version with two sites.

7.2 First Quantized Version, Four Sites

Figure 7.3 shows the expected distribution of states after running the circuit shown in Figure 3.3 along with the corresponding uniform distribution of states generated by three qubits. The ℓ^1 distance between these two distributions is 0.75. Thus, if the ℓ^1 distances between the distributions of measured states and the expected distribution are not significantly less than 0.75, we cannot conclude that the measured distributions of states are any more significant than noise.

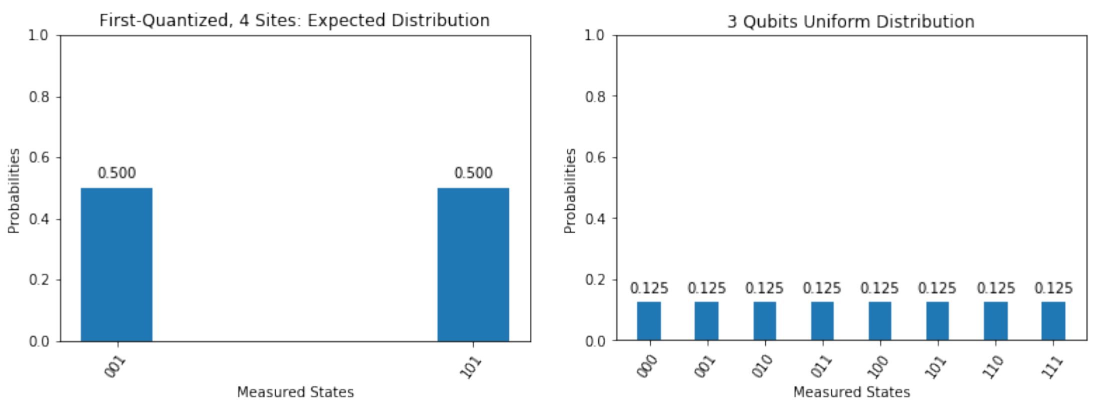


Figure 7.3: Expected distribution of states for the first quantized version with four sites (left). Uniform distribution of states generated by three qubits (right).

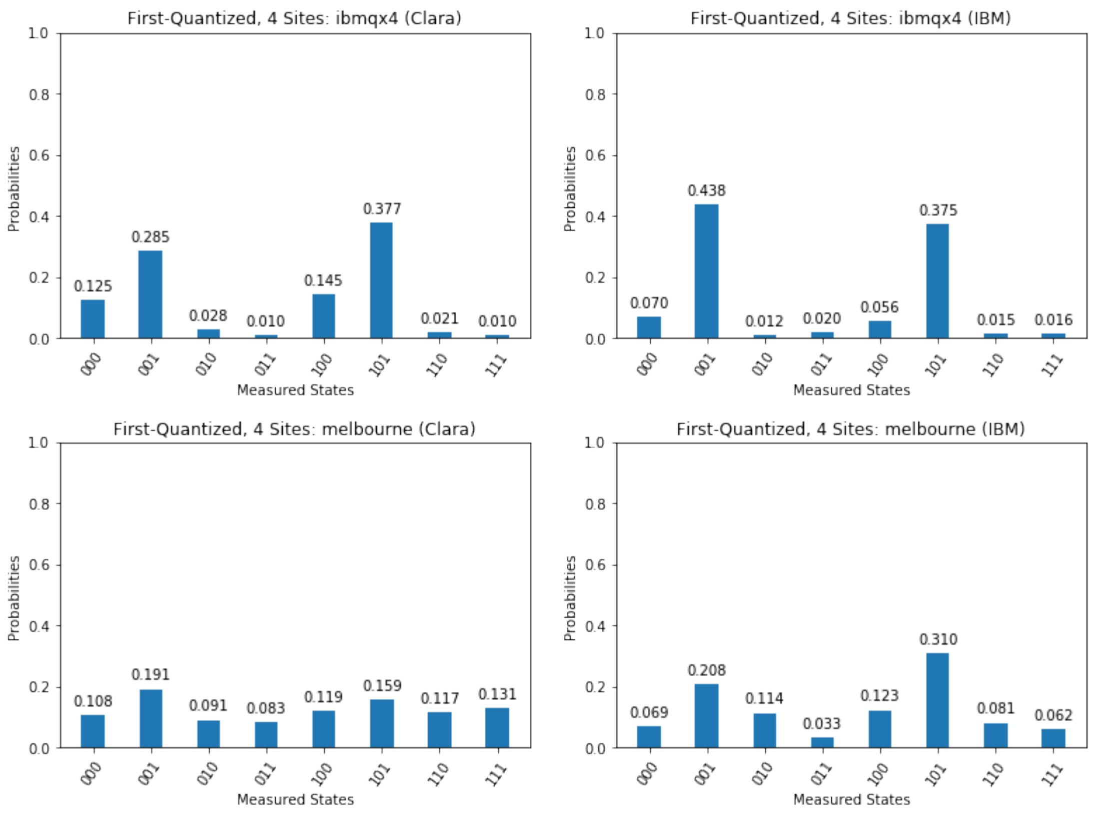


Figure 7.4: Example distributions for the first quantized version with four sites.

Figure 7.4 shows example distributions of runs on the ibmqx4 and ibmq 16 melbourne for both the circuits I compiled and the circuits IBM compiled for the first quantized version with four sites. The circuit I compiled onto the ibmqx4 produces a distribution with an ℓ^1 distance of 0.338, which is larger than the 0.188 ℓ^1 distance of the distribution the circuit IBM compiled onto the ibmqx4 produces. These ℓ^1 distances are clearly below the 0.75 threshold, with IBM's circuit performing better than mine. This is surprising considering that IBM's circuit uses the same three qubits as mine, yet has three more gates and two more timesteps. The circuit I compiled onto the ibmq 16 melbourne produces a distribution with an ℓ^1 distance of 0.649, which is larger than the 0.482 ℓ^1 distance of the distribution the circuit IBM compiled onto the ibmq 16 melbourne produces. Again, this is surprising considering my circuit has less than half as many gates as IBM's. I do, however, use a different set of physical qubits than IBM. The qubits I chose may be noisier than the qubits IBM chose, which could lead to worse results even though my circuit contains fewer gates.

7.3 Second Quantized Version, Two Sites

Figure 7.5 shows the expected distribution of states after running the circuit shown in Figure 3.5 along with the corresponding uniform distribution of states generated by four qubits. The ℓ^1 distance between these two distributions is 0.875. Thus, the distributions of measured states after running this circuit are only meaningful if their ℓ^1 distances from the expected distribution are significantly less than 0.875.

Figure 7.6 shows example distributions of runs from circuits compiled by myself and IBM onto the ibmqx4 and ibmq 16 melbourne. The circuit I ran on the ibmqx4 produces a distribution with an ℓ^1 distance of 0.559, which is close to the 0.578 ℓ^1 distance of the distribution the circuit IBM ran on the ibmqx4 produces. Both of these ℓ^1 distances are below the 0.875 cutoff for noise, but they still demonstrate a heightened level of noise compared to the best runs for the first quantized version. The circuit I ran on the ibmq 16 melbourne produces a distribution with an ℓ^1

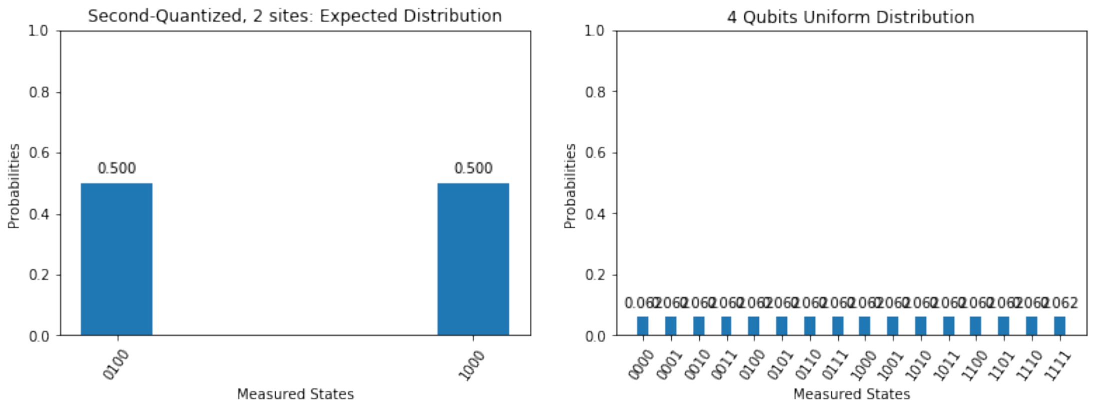


Figure 7.5: Expected distribution of states for the second quantized version with two sites (left). Uniform distribution of states represented by four qubits (right).

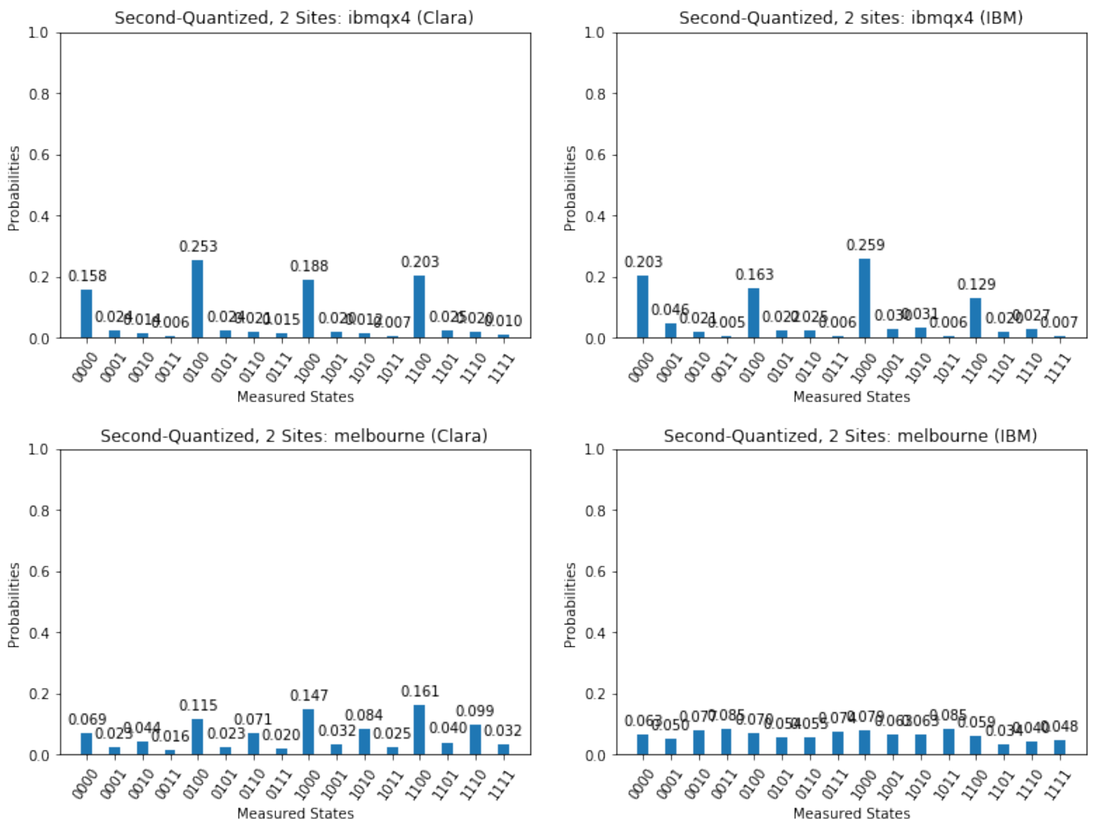


Figure 7.6: Example distributions of measured states after running the circuit for the second quantized version with two sites.

distance of 0.738, which is lower than the 0.851 ℓ^1 distance of the distribution the circuit IBM ran on the ibmq 16 melbourne produces. While the ℓ^1 distance of the distribution my circuit produces on the ibmq 16 melbourne is technically below the cutoff noise level, it would be difficult to pick out the expected states without knowing what the expected distribution looks like. The ℓ^1 distance of the distribution IBM's circuit produces on the ibmq 16 melbourne is not significantly below the noise cutoff, and, indeed, the distribution of measured states looks almost uniform.

7.4 Second Quantized Version, Four Sites

Figure 7.7 shows the expected distribution of states after running the circuit shown in Figure 6.6. The corresponding uniform distribution of states generated by eight qubits cannot be plotted nicely, but each state has a probability of $\frac{1}{256}$. The ℓ^1 distance between these two distributions is 0.9921875. The noise threshold is almost 1 in this case because, as there are so many possible states, the uniform distribution is almost disjoint from the expected distribution.

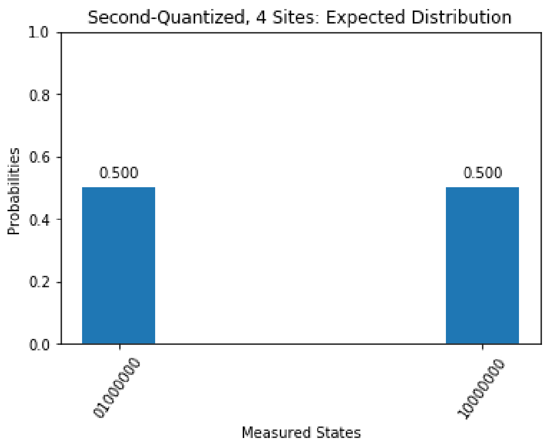


Figure 7.7: Expected distribution of states for the second quantized version with four sites.

Example distributions for runs of this circuit contain 256 different states, so plots are less helpful for visualization purposes here. My circuit run on the ibmq 16 melbourne produced a distribution with an ℓ^1 distance from the expected distribution of 0.984, and IBM's circuit produced a distribution with an ℓ^1 distance of

0.985. While these ℓ^1 distances fall slightly below that of the uniform distribution's they are not significantly below, and the distributions of measured states are still mostly, if not entirely, made up of noise.

8 Discussion

After compiling two different algorithms onto two different architectures, it is evident that the choice of architectural layout influences the size and depth of the compiled circuit. Furthermore, there does not appear to be a generally superior architecture, as each algorithm preferred a different configuration of qubits and connections between them. For example, the first quantized version compiles more neatly onto the ibmqx4 when $N = 4$ whereas the second quantized version compiles more neatly onto the ibmq 16 melbourne when $N = 2$.

Table 8.1 shows all the compiled circuits presented in this paper with their number of gates and depth. Comparing the best compiled circuits for each version for $N = 2$ and $N = 4$, the first quantized version is more efficient than the second quantized version with regard to the number of qubits used, number of gates, and depth. Even when compiled onto the ibmq 16 melbourne, the first quantized version still produced smaller and shorter circuits than the second quantized version also compiled onto the ibmq 16 melbourne. It is possible that the second quantized version scales better in terms of circuit depth for large N, but this would require more qubits to test than are currently available on IBM's publicly available quantum computers.

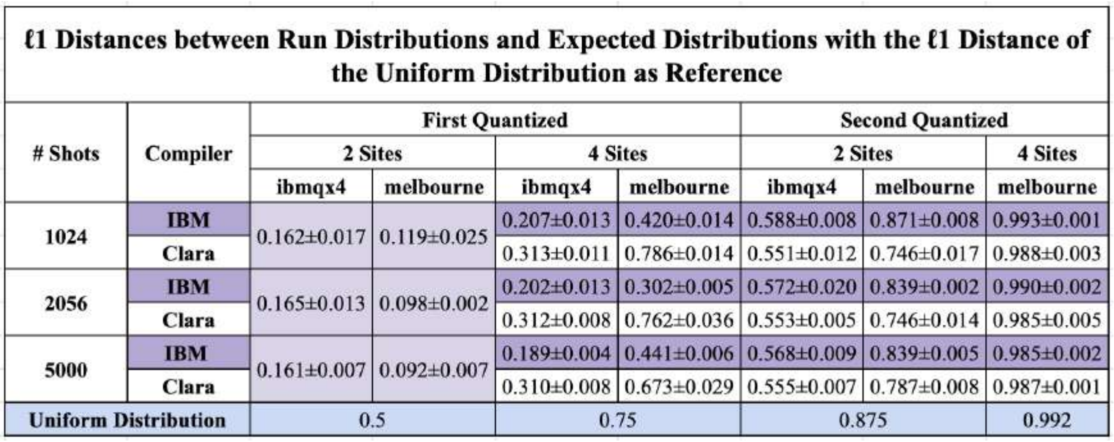
When IBM's compiler chose the same layout of physical qubits as I did, its compiled circuits were longer than mine, but by a smaller margin than they were when it chose a different layout of physical qubits. Its main fault lay in its initial assignment of physical qubits, which led to unnecessary CNOT reversals and more complicated swaps.

Circuit	Number of Gates	Depth
fq-2	4	4
fq-4-ibmqx4-Clara	7	6
fq-4-ibmqx4-IBM	10	8
fq-4-melbourne-Clara	10	9
fq-4-melbourne-IBM	22	16
sq-2-ibmqx4-Clara	50	24
sq-2-ibmqx4-IBM	60	28
sq-2-melbourne-Clara	36	14
sq-2-melbourne-IBM	149	104
sq-4-melbourne-Clara	72	14
sq-4-melbourne-IBM	81	16

Table 8.1: Summary of compiled circuits with their number of gates and depth. The rows highlighted in green indicate the best circuit for that combination of version and number of sites. The first quantized version with two sites has no highlighted row because all the compiled circuits were identical.

Running the compiled circuits on the ibmqx4 and ibmq 16 melbourne allowed us to see how the circuits performed in practice. Table 8.2 shows the average ℓ^1 distances with errors for the distributions the compiled circuits produced when run on their respective architectures. I calculated these averages and errors from five distributions produced one after another. The low error values are thus indicative of a relatively constant amount of noise at any given time in IBM's quantum computers and do not reflect variations in noise across hours, days, or weeks.

For the first quantized version with two sites, the ibmq 16 melbourne produced distributions with smaller ℓ^1 distances than the ibmqx4. Because the same circuit ran on both architectures, this suggests that the qubits used in the ibmqx4 are noisier than their corresponding qubits in the ibmq 16 melbourne. Both architectures, however, produced distributions with very low ℓ^1 distances, so we can easily pick out the expected states.



ℓ1 Distances between Run Distributions and Expected Distributions with the ℓ1 Distance of the Uniform Distribution as Reference

# Shots	Compiler	First Quantized				Second Quantized		
		2 Sites		4 Sites		2 Sites		4 Sites
		ibmqx4	melbourne	ibmqx4	melbourne	ibmqx4	melbourne	melbourne
1024	IBM	0.162±0.017	0.119±0.025	0.207±0.013	0.420±0.014	0.588±0.008	0.871±0.008	0.993±0.001
	Clara			0.313±0.011	0.786±0.014	0.551±0.012	0.746±0.017	0.988±0.003
2056	IBM	0.165±0.013	0.098±0.002	0.202±0.013	0.302±0.005	0.572±0.020	0.839±0.002	0.990±0.002
	Clara			0.312±0.008	0.762±0.036	0.553±0.005	0.746±0.014	0.985±0.005
5000	IBM	0.161±0.007	0.092±0.007	0.189±0.004	0.441±0.006	0.568±0.009	0.839±0.005	0.985±0.002
	Clara			0.310±0.008	0.673±0.029	0.555±0.007	0.787±0.008	0.987±0.001
Uniform Distribution		0.5		0.75		0.875		0.992

Table 8.2: Summary of ℓ^1 distances for the first and second quantized versions compiled onto the ibmqx4 and ibmq 16 melbourne by myself and IBM. I ran each circuit for the indicated number of shots to produce a distribution, and I took five such distributions to calculate the average ℓ^1 distance with the standard deviation used as error.

For the first quantized version with four sites, IBM's circuits produced better distributions than mine on both the ibmqx4 and the ibmq 16 melbourne despite my circuits having fewer gates and smaller depths. IBM used a different set of qubits on the ibmq 16 melbourne than I did, so I suspect that those three qubits were less noisy than the four qubits I used. Thus, when picking physical qubits to compile a circuit onto, we may also want to consider the noise of each qubit instead of only optimizing for number of gates and depth. I do not currently have any hypotheses as to why IBM's circuit ran better on the ibmqx4 because I used the same three qubits and CNOTs among them as IBM, but IBM used three more gates. For all circuits except the one I compiled onto the ibmq 16 melbourne, the ℓ^1 distances of the produced distributions fall below the uniform distribution's ℓ^1 distance. This confirms that we can run one timestep of the first quantized version with four sites and get significant results as long as we choose the correct circuit to run. In this case, the circuit IBM compiles onto the ibmqx4 performs best.

For the second quantized version with two sites, my circuits slightly outperformed IBM's on both architectures. This met my expectations as my circuits were shorter with fewer gates than IBM's. However, my circuit runs on the ibmqx4 did better than my circuit runs on the ibmq 16 melbourne even though the circuit I compiled onto the ibmq 16 melbourne was smaller. Again, I presume that this difference stems from the qubits I chose in the ibmq 16 melbourne being noisier than the qubits I chose in the ibmqx4. This pushed my circuit runs on the ibmq 16 melbourne close to the ℓ^1 distance of the uniform distribution, suggesting that they no longer contain the signal we expect. The ℓ^1 distances from the ibmqx4, however, fall comfortably below the uniform distribution ℓ^1 distance. Therefore, even though they are higher than the ℓ^1 distances from the first quantized version, we should still be able to pick out the signal if we know what the correct answer is (which we do).

For the second quantized version with four sites, there was no real difference in the ℓ^1 distances calculated from the runs of my compiled circuit, runs of IBM's compiled circuit, or the uniform distribution. Despite my circuit having the same depth as the two site case, the number of gates and qubits doubled, which increased the noise. Thus, the constant depth for the second quantized version does not outweigh the cost of the linear scaling in number of qubits and gates. We cannot get any significant results for one timestep of the second quantized version with four sites.

References

[1] https://www.research.ibm.com/ibm-q/technology/devices/

[2] https://quantumexperience.ng.bluemix.net/qx/editor

[3] https://qiskit.org/

[4] https://github.com/Qiskit/openqasm

[5] Andris Ambainis, Quantum walk algorithm for element distinctness, SIAM J. Comput. 37 (2007), no. 1, 210239, arXiv:quant-ph/0311001, preliminary version in FOCS 2004.

[6] F. Magniez, M. Santha, and M. Szegedy, Quantum algorithms for the triangle problem, Proc. 16th ACM-SIAM Symposium on Discrete Algorithms, pp. 11091117, 2005, quant-ph/0310134.

[7] E. Farhi, J. Goldstone, and S. Gutmann, A quantum algorithm for the Hamiltonian NAND tree, Theory of Computing 4 (2008), no. 1, 169190, quant-ph/0702144

[8] https://github.com/Qiskit/ibmq-device-information/blob/master/backends/tenerife/images/ibmqx4-connections.png

[9] https://github.com/Qiskit/ibmq-device-information/blob/master/backends/melbourne/images/melbourne-connections.png

[10] https://github.com/Qiskit/ibmq-device-information/tree/master/backends/tenerife/V1

[11] https://github.com/Qiskit/ibmq-device-information/tree/master/backends/melbourne/V1

[12] Gerhard W. Dueck, Anirban Pathak, Md Mazder Rahman, Abhishek Shukla, and Anindita Banerjee, Optimization of Circuits for IBMs five-qubit Quantum Computers, arXiv:1810.00129v1 [cs.ET] 29 Sep 2018.

[13] https://quantumexperience.ng.bluemix.net/proxy/tutorial/full-user-guide/004-Quantum_Algorithms/061-Basic_Circuit_Identities_and_Larger_Circuits.html

[14] John Hayes and Igor Markov (2001). Quantum Circuit Decomposition from unitary matrices into elementary gates [PowerPoint slides]. Retrieved from http://vlsicad.eecs.umich.edu/Quantum/EECS598/lec/7a.ppt.