THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

EFFICIENT SCHEDULING AND COMPILATION OF QUANTUM PROGRAMS

JAVIER MONTANER
SPRING 2022

A thesis
submitted in partial fulfillment
of the requirements
for baccalaureate degrees
in Physics and Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Dr. Swaroop Ghosh
Professor of Computer Science and Engineering
Thesis Supervisor

Dr. John Hannan
Associate Head of the Computer Science and Engineering Department
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

In the decades since the conception of a quantum computer, significant progress has been made toward making them a reality. Current implementations are severely limited; however, they are a significant step in the development of the field and allow for the continuation and expansion of research in the area. While much of the research is focused on improving the robustness and scalability of the logical units as well as finding algorithms which fully utilize the power available to the quantum system, there are other areas which require attention to develop a fully functioning quantum computer. One such consideration, and the focus of this work, is the efficient management of the execution of quantum programs on a device. Current work seeks to find the factors which most directly influence the execution of programs and should therefore be considered to perform efficient operation and scheduling. The work also seeks to maximize their performance given the limitations of the current generation of quantum computers. Through utilizing IBM's quantum backends, multiple scheduling heuristics were tested to determine the factors which contribute to the execution of programs and their relative importance. These included the error rates of the hardware, the traffic level measured by the queue time for each device, and the depth of the implemented quantum circuit. In this way this thesis is able to explore the field of quantum compilation and understand the current state of its development.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

The field of quantum computation and information is newly emerging, rapidly developing, and becoming one of the most exciting areas in current scientific research. The strange quantum mechanical properties employed give these machines the possibility to vastly outperform the classical computers currently in widespread use. The actualization of a full quantum computer will require a wide range of research into each aspect of the system. While the construction of the hardware and development of quantum algorithms face obstacles unique to the field, other considerations are applicable to all computers and just require adoption to the quantum case. One such example, and the focus of this paper, is resource allocation and the scheduling of quantum programs. However, before getting into specifics, it is important to understand the theoretical concepts behind how a quantum computer works.

As the classical bit (0 or 1) is the basis of a classical computer, the quantum bit, or qubit, is the basis of a quantum computer. A qubit is a two-dimensional quantum system with basis states defined as $|0\rangle$ and $|1\rangle$. The way this state can be manipulated is what gives quantum computers their computational advantage over classical ones and are given from the following quantum mechanical properties:

- **Superposition –** A quantum state can be in multiple basis states simultaneously. While a classical bit can only be a 0 or a 1, a qubit can be in a superposition of 0 and 1, interpreted as a linear combination of the basis states $|0\rangle$ and $|1\rangle$ and written as:

$$|\Psi\rangle = a_0|0\rangle + a_1|1\rangle$$

This can be interpreted as the state being both a 0 and a 1.

- **Measurement** – Although a quantum state can be in multiple states at once, when a measurement is performed, the state will 'collapse' to one of the basis states according to its coefficient, which is a probability amplitude. By measuring the state $|\Psi\rangle$ above, we will get a 0 with probability $|a_0|^2$ and a 1 with probability $|a_1|^2$. Following the measurement, $|\Psi\rangle$ will be equal to the basis state corresponding to the result of the measurement, destroying the superposition. This is typically interpreted as the measurement 'collapsing' the superposition to the measured state.
- **Entanglement** – Two quantum states are considered entangled to one another when the measurement of one qubit influences the measurement of the other. For example, consider the following state containing two qubits:

$$|\Psi\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

If you measure the first qubit, there is an equal probability of it being a 0 or 1. However, because the second qubit was entangled with the first, if we measure it, it is guaranteed to be the value measured by its entangled partner.

- **Unitary Evolution** – Every operation on a quantum system is unitary. Therefore, Unitary transformations mathematically describe quantum computation (Quantum Compiling). This quality also guarantees that any quantum operation can be represented by a unitary matrix if formulating the system's evolution in terms of linear algebra.

Similar to classical computers, quantum computers perform computations by imposing a series of logical gates on the system's qubits. These gates are what make up quantum circuits: an ordered sequence of quantum gates. However, unlike classical computation, quantum computation is most easily understood through linear algebra. A state containing n qubits can be represented as a 2^n dimensional vector and a quantum gate on n qubits can be represented as a $2^n \times 2^n$ dimensional matrix. In this way, applying a quantum gate is equivalent to performing matrix multiplication. As mentioned by the unitary evolution of quantum systems, every gate is a reversible process and can be represented by a unitary matrix. Due to this, it is common that the term 'unitary' is interchangeable with 'gate'. The following gates are the most important in understanding how quantum computers are able to utilize quantum mechanical principles to their advantage:

- **NOT** – This gate is equivalent to the classical NOT gate. It will flip a $|0\rangle$ state to a $|1\rangle$ state and vice versa. This is a one-qubit gate and can therefore be represented by the following matrix:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

The operator is denoted as X because it is equivalent to the Pauli-X matrix, σ_X. Similarly, the Y and Z unitaries are represented by the Pauli matrices σ_Y and σ_Z respectively, however, their functionality is less intuitive.

- **Hadamard** – The Hadamard gate is a single qubit gate that puts the qubit into a superposition. Acting on the basis states results in the following transformations:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

It can then be seen that the Hadamard gate has the following matrix representation:

$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

- **Controlled-NOT (CNOT)** – The CNOT gate is a two-qubit gate and used to create an entanglement. Of the two qubits, one is the control qubit and the other is the target. If the control qubit is $|0\rangle$, then nothing is done to the target qubit. However, if the control qubit is $|1\rangle$, then a NOT gate (described above) is applied to the target qubit. The unitary has the following matrix representation:

$$CNOT = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

Although the gate may appear intuitive to understand, it can exhibit strange behavior. The following example shows how CNOT can be used to create entanglement. This is possible because the checking of whether the control qubit is $|0\rangle$ or $|1\rangle$ is not a measurement and can be done while maintaining a superposition:

$$CNOT\left(\frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)\right) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

This idea can also be generalized to create any arbitrary controlled n-qubit unitary by expanding the number of target bits. If the control bit is $|0\rangle$, then still nothing will be done to the other qubits. However, if the control bit is $|1\rangle$, then apply the desired unitary on those n qubits.

- **Phase Gates** – The phase gate is a one-qubit gate that alters the phase of the system according to the parameter φ. These gates do not affect the measurement probabilities of the system. The general form of these gates is:

$$P(\varphi) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\varphi} \end{bmatrix}$$

However, the most common of these gates given their own name as:

$$T = P\left(\frac{\pi}{4}\right) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{4}} \end{bmatrix}$$

$$S = P\left(\frac{\pi}{2}\right) = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\frac{\pi}{2}} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & i \end{bmatrix}$$

The previously mentioned Z gate is also a phase gate as:

$$Z = \sigma_Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & e^{i\pi} \end{bmatrix} = P(\pi)$$

The CNOT gate, Hadamard gate, and S gate described above form the Clifford group — the group of unitary operators that map the group of Pauli operators to itself under conjugation [2]. These gates are important because the Clifford group and the T gate are a conventional form of a universal set of quantum gates [13]. This universality inspires the Solovey-Kitaev Theorem: any unitary operation can be represented as an ordered sequence of transformations acting on one and two-qubit subsystems [1,10,13], or equivalently, any unitary on an arbitrary number of qubits can be implemented as a circuit of the quantum gates described above.

In order to execute a computation, the quantum computer implements a quantum circuit: a sequence of operations from the universal set of gates defined above. The quantum circuit is the compiler's main consideration when deciding how to most ideally schedule and implement a given process. While information about things like the number of CNOTs and which qubits they

are between can help the compiler effectively map the circuit to hardware, a simple and useful piece of information the compiler can find is the circuit depth. The circuit depth is the number of layers of quantum operations stacked upon each other [15] and gives the compiler a general idea of the time needed to perform the computation, which is useful when optimizing resource use.

An understanding of relevant quantum mechanical principles and the standard gates which construct quantum circuits is important in quantum computing, however, similarly important is the history of the field as well as its current state of development. The idea of a computer able to utilize quantum mechanical properties to perform more difficult computations was first conceptualized in the 1980s. However, it was the publication of Peter Shor's factoring algorithm in 1994 that many attribute to the inception of the field of quantum computing. This algorithm exhibits an exponential speedup over the best-known classical factoring algorithm and gave the first glimpses at the power of a quantum computer. Since then, research in developing the hardware necessary to actualize a full quantum computer as well as research in quantum algorithms and the theoretical bounds on a quantum computer's power have made large strides. If successful, there are numerous potential applications and many still to be found. The most intuitive application is the simulation of quantum systems. This would contribute to fields like quantum chemistry as even the evolution of simple molecules is difficult to simulate classically [3].

Despite this progress, we are still a ways away from a quantum computer large and fault-tolerant enough to implement the quantum algorithms that give it a decisive computational advantage over classical computers. The machines in the current epoch of quantum computers are referred to as Noisy Intermediate-Scale Quantum (NISQ) Computers. These are characterized as containing around 10-100 qubits and relatively high rates of gate error and state decoherence.

Although imperfect, these NISQ devices have significantly expanded access to quantum technologies, however limited they may be. Developers such as IBM offer up their quantum computers to run any program you would want. This expanded access has hastened research as more people can get involved. Also, with some working hardware, it opened new areas of research that were not possible before, including the focus of this paper: scheduling and resource allocation.

Chapter 2

Literature Review

The ultimate goal of a quantum compiler is to provide robust control of the computation and map the quantum computation into ordered sequences of gates implementable on real quantum hardware [10]. While the quantum nature of the computations requires many necessary considerations, at a high level, quantum compilation has the same functionality as a compiler on a classical computer: implementing a process on the available hardware. Current quantum compilers must also be cognizant of the limited functionality of NISQ machines. This includes considerations of the limited availability of qubits, high error rates, and state decoherence. Therefore, much of the research in quantum compilation seeks to mitigate the effects of the error-prone, unstable hardware on the result of the computation through efficient compilation and scheduling algorithms.

Despite their differences, there are several concepts from classical compilation that are universal to process management and can therefore be applied to the quantum case. One example is being aware of operation dependencies while scheduling a program to be executed. The compiler must be conscious of the order in which operations should be performed to preserve the integrity of the computation. One such way is to produce a Directed Acyclic Graph (DAG) to describe the dependencies of each operation. This DAG would also inform the compiler of which processes are independent and can be run simultaneously, thus maximizing parallelism [18]. Another example is an understanding of the resources and time needed to perform the computation. The high implementation cost of qubits and operations emphasizes the importance of a compiler which can quickly estimate what resources will be consumed [7].

While the universal set of quantum gates described in the Introduction can be used to implement any arbitrary unitary, it is not guaranteed to be able to implement it exactly. In some cases, the quantum compiler may receive a quantum circuit expressed with high-level, multi-qubit unitaries, that it then must approximate as a sequence of low-level operations native to the processor [5,10]. However, it is suggested that this process accounts for error as the abstraction of high-level unitaries can perturb the evolution of the underlying quantum states, leading to a less reliable outcome [17]. A good framework to understand this issue is to consider the computation as the evolution of the system's Hamiltonian. On n qubits, the system's Hamiltonian will have dimension $2^n \times 2^n$. However, by formulating the evolution as a series of low-level gates, we are representing the Hamiltonian as a sum of sub-system interactions [13]. The entire Hamiltonian has no guarantee that it can be perfectly represented in this way which is the source of the error. A suggested solution to this issue is to aggregate the operations by creating a custom control pulse optimized to enact the high-level transformation. This allows the compiler to consider the evolution of a larger number of qubits, rather than just its decomposition into one and two-qubit gates [17]. Another proposed solution is to train a deep-learning algorithm to generalize how to convert any unitary into a sequence of elementary gates [11]. While this implementation will still be imperfect due to the approximation, the program can learn techniques to produce the most optimal approximation.

Another problem the compiler must consider that is unique to quantum computation is dealing with qubits which have been entangled. Temporary values are common in programming and while a classical compiler can reassign the resources storing these values with little consideration, a quantum compiler must be aware if any of these temporary values are entangled with a qubit which will be used later in the computation. These qubits which store temporary

values which aid in the computation are referred to as ancilla qubits. If an ancilla qubit will not be used for the remainder of a computation, the compiler may want to reassign it to another program's exeution where it can be used. However, if this ancilla qubit is entangled with a qubit vital to the computation (a data qubit), its wavefunction is integrated with the wave function of the data qubits. Therefore, if you were to alter the ancilla qubit's quantum state (through measurement or use after reassignment) you affect the probabilities of state measurements for the data qubits which can result in an incorrect output [7]. This problem can be mitigated through a process called uncomputation. This exploits the reversibility of quantum processes to return the qubit to its original, unentangled state by subjecting it to the inverse of the operation which entangled it. As seen, these transformations can be represented by unitary matrices so their inverse is simply their conjugate transpose, making them efficient to implement [14]. This process protects the state of the data qubits while allowing ancilla qubits to be safely reassigned to other processes, maximizing resource use.

A quantum compiler on NISQ devices must also be aware of the limitations posed by the system's imperfect resources. Along with the high gate error rates and the lack of sufficient qubits to perform error correction, it is important that the compiler seeks to mitigate the loss of quantum information due to state decoherence [9]. Each qubit only has a short amount of time that it can be considered viable. Therefore, the compiler should schedule operations on the qubit such that all its gates finish before the decoherence time [12].

While the considerations mentioned are vital for an ideal quantum compiler, capable of implementing any arbitrary quantum program, current quantum compilers commonly require a quantum circuit as an input. This means the compiler is only concerned with implementing the given circuit on the hardware available, the user must be the one to ensure

proper resource management and use which will implement their goal. However, there are still many measures the compiler can take to maximize the effectiveness of the computation. One of the most intuitive ways is to prioritize scheduling programs to be executed on the most reliable qubits. Information about error rates for individual qubits is available to the compiler which allows it to make these decisions. One suggestion on how to most effectively implement this idea is to design a greedy scheduling algorithm in which the qubit with the highest degree (most operations) or the two qubits with the most CNOTs between them are assigned the qubits with the lowest error rates [12].

One of the best ways for the compiler to maximize the computation's success probability is to minimize the total number of CNOT operations. This is because the CNOT gate has low fidelity and is thought to be the limiting factor in near-term NISQ machines [5]. While CNOT gates described in the input quantum circuit cannot be avoided, the minimization of CNOTs largely comes from the minimization of SWAP operations. These SWAPs are made up of three CNOT gates and swap the states of two qubits. They can be minimized through topological considerations of the hardware qubits.

The topology of the quantum hardware is one of the most important considerations a quantum compiler must make while efficiently scheduling quantum algorithms. When mapping qubits from the input quantum circuit to qubits on the hardware, the compiler must strive to maximize parallelism while minimizing communication [8]. Different quantum hardwares have different qubit connectivities. Figures 1 and 2 show the topology of two, five-qubit quantum devices provided by IBM.

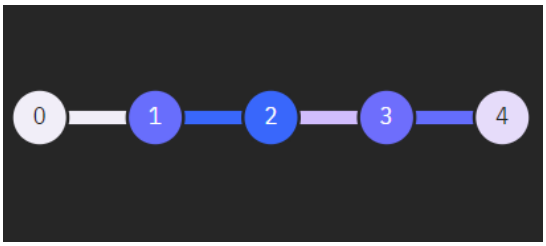


Figure 1: ibmq_bogota Topology

The topology of IBM's Bogota quantum computer. Source: [16]

Topological considerations are important because two-qubit gates can only be applied to adjacent qubits. For example, a CNOT could applied on qubits 2 and 3 in the device in Figure 1, but not the device in Figure 2. Therefore, if the quantum circuit calls for a CNOT to be applied on two qubits which are not adjacent in the hardware, a series of SWAP operations must be performed until they are adjacent. The compiler should be able to optimize the mapping which minimizes qubit movement [12], which contributes to maximizing the computation's probability of success. Besides the high error rates, minimizing the amount of CNOT and SWAP operations and having the compiler be communication-aware is important because high communication inflicts a significant cost to global memory [8].

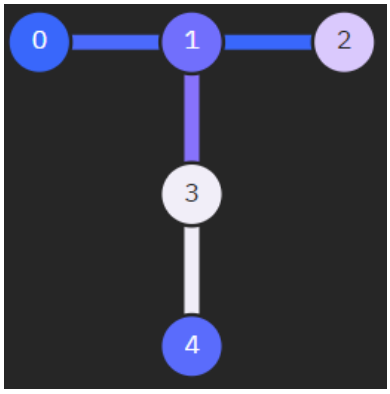


Figure 2: ibmq_lima Topology

The topology of IBM's Lima quantum computer. Source: [16]

Chapter 3

Methodology

The quantum hardware owned and operated by IBM, referred to as backends, serves as a valuable resource to collect data on the scheduling, compilation, and execution of quantum programs. IBM maintains a number of quantum computers ranging from 5 to 127 qubits which are always operating and accepting quantum circuits from users. Due to the focus on scheduling, the more powerful backends were not required with all of the work being done using IBM's various 5-qubit hardwares: ibmq_lima, ibmq_belem, ibmq_quito, ibmq_bogota, and ibmq_santiago. Through utilizing the services provided, one can get information about these backends and send quantum circuits to be run on the quantum computer. To allow for wide access of the services available, IBM developed a special package for the programming language Python called Qiskit.

Qiskit is an open-source software platform that allows the user to interact with the quantum machine language [10]. It provides all the functionality to construct quantum circuits, manage the execution of the program, and send them to be run on an IBM backend. It is through the creation and manipulation of Qiskit programs that allows anyone to design their own quantum program and get the results from a real quantum computer. Additionally, IBM also allows users to send quantum circuits to be implemented by one of their simulators. These simulate the execution of the circuit classically and therefore do not have the same limitations of high error rates and state decoherence. However, this work is focused on efficient scheduling given the limitations of current quantum technologies, requiring the use of backends rather than simulators. Similar to Qiskit, QASM is a quantum assembly language which provides an

alternate method of defining quantum circuits. However, QASM cannot send these circuits to be executed and must be imported through Qiskit to do so.

With the focus of this work being compilation and scheduling, the process being sent to the quantum backend is arbitrary. The chosen quantum circuit which was used to gather the following data implements the Quantum Approximate Optimization Algorithm (QAOA) to solve the max-cut problem for an input graph. QAOA is a hybrid quantum-classical algorithm which works by running the quantum portion with tunable parameters, having the result fed into a classical optimizer which gives improved parameters, and running the quantum circuit again with these improved values from the optimizer. This process then continues iteratively but for the purpose of this work, the circuit is passed with ideal parameters and run only once. QAOA was chosen because it has a fundamentally simple structure which makes it an ideal process for NISQ devices [4]. QAOA is used to solve the max-cut problem which, given a graph, seeks to separate the vertices into two sets such that the number of edges between these sets is maximized. A simple, 5-node graph was used as the input for the experimental quantum circuit.

Through the work with quantum scheduling and compilation, we looked into the considerations needed to implement a scheduler over a set of available quantum hardware. While most current work is concerned with a compiler which can only map processes to a single quantum computer, this work sought to receive quantum circuits and optimally schedule them to be run on one of five available backends. These backends were IBM's five-qubit hardwares described above. Through this work we were able to learn about which factors influence a processes compilation and execution time such as the backend's queue length, the circuit depth, and the backend's reliability.

The primary method of data collection came from implementing a scheduling algorithm, running it for a queue of identical quantum programs described above (solving max-cut problem with QAOA), and tracking the elapsed time between sending the circuit to IBM and receiving a result. The scheduling algorithm chose the backend to send the program to according to multiple heuristics: randomly, to the backend with the minimum queue, or the backend with the highest performance/reliability. The quantum circuits sent also had variable depths to observe the effect this would have on the scheduling and execution time. The QAOA algorithm can be implemented as a quantum circuit with arbitrary depth which allowed for the creation of a set of simple and comparable circuits of different depths to use for data collection. For this thesis, the circuits which were used implemented QAOA with depths of 1, 5, 10, 15, and 20 to understand the effect the input circuit's depth would have on its execution. This procedure was performed at different times to have data for both heavier and lighter traffic of the backends.

In order for the work to model a real scheduler, multithreading was used to overcome the way jobs are sent to the backends with Qiskit. The Qiskit command execute takes parameters which include the quantum circuit and is the function which sends out the program to be executed on the IBM backends [6]. This function is atomic meaning the program will not progress to the next instruction until the result of the computation has been returned by IBM. However, the scheduler should not have to wait for a job to finish before considering the next one in the queue, ideally it should be able to send out a circuit to be run and then immediately begin to consider where to schedule the next process without needing to wait for a result. Multithreading helped with this issue as it allowed me to incorporate more simultaneity to the scheduler. The only limitation of this is that IBM limits the number of simultaneous jobs a user

can run to five. The requests to execute jobs would be periodically declined if too many of the previously sent programs had not finished executing yet.

A difficulty with this procedure, however, is that the results are constrained by IBM's scheduling algorithms for its individual backends. Once a program is sent to the queue of jobs to be performed on the hardware, it is IBM's scheduling algorithm which determines the order in which they will be run, which the program written to enact this procedure has no control over. IBM orders the computations generally as a typical first in first out (FIFO) queue but the lack of control of the program after it has been sent to IBM is still a concern. However, this is a product of utilizing their services and it would require a local quantum computer to have total control over process scheduling and execution. Regardless, it is still possible to gain useful information from this procedure and this discussion simply serves to identify a limitation of it.

Chapter 4

Results and Discussion

Through implementing various scheduling heuristics using constructed queues of comparable circuits with variable depth, we was able to determine the most important factors for developing a quantum compiler and scheduler. Through considerations of the depth of the program's quantum circuit as well as information about the implementing hardware like its performance capabilities and job queue length, a quantum computer would be able to efficiently schedule and execute computations. The data being collected by sending out circuits to IBM and waiting for them to return was definitely not ideal and lead to various difficulties and unexpected results, however, the process was still informative and satisfied the purpose of the work.

At the start, we were primarily concerned with the scheduling algorithm being implemented and wanted to test out the most intuitive heuristic: sending the quantum circuit to the backend with the minimum queue length. It was not until later that the depth of the quantum circuit was incorporated as an important consideration so at this point, all of the circuits being used had a depth of 5. A queue of 25 of these circuits was created where each to be scheduled on a specific backend according to some scheduling heuristic and then sent to be executed, with the times to complete each one being recorded. To gauge the effectiveness of scheduling the program on the backend with the minimum queue, the same procedure was performed with the backend chosen randomly and with the backend with the highest reliability chosen for comparison.

The results from the minimum queue scheduling can be seen in Table 1 where the average time to execute a circuit as well as the average minimum queue length is shown.

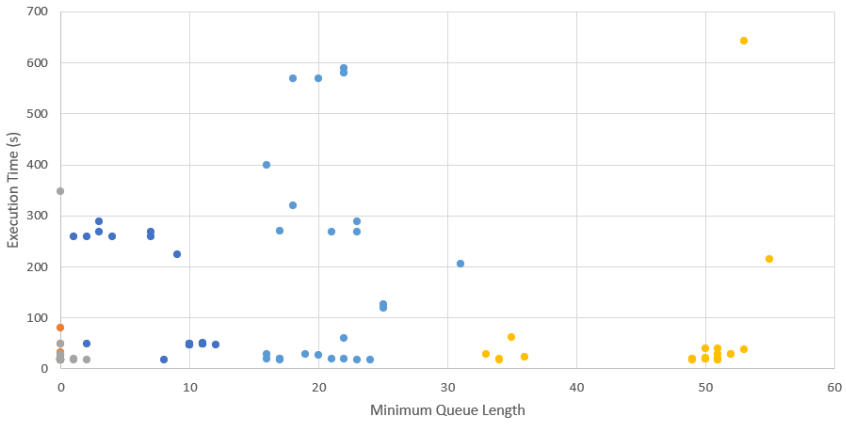
Table 1: Minimum Queue Scheduling

Average Execution Time (s)	Average Minimum Queue Length
1.301596	0.1875
1.4708502	1.5333333
1.188492	2.625
2.625	23.7333333

While, on average, there seems to be a slight trend indicating a longer queue will result in longer execution time but when the backend was chosen randomly, the average execution was 1.40475 seconds and 1.1533 seconds when selecting the most reliable backend. Many of the data points also indicated that the queue length alone was not a perfect indicator of the time it would take to execute. It was at this point that we began to suspect that time the depth of the circuits was affecting the result. The scheduling algorithm IBM uses to manage the job queues for their backends prioritizes the execution of circuits with larger depths. This meant that while sending circuits with the same depth, we saw similar execution times regardless of the length of the queue.

In order to try and account for IBM prioritizing the execution of circuits with greater depth, the queue of circuits maintained on our end was changed to include circuits of varying depth. This allowed for us to gauge how the depth of the circuit would affect the execution time and if a trend between queue length and execution time could still be seen when accounting for circuits of different depths. The results from performing the procedure with circuits of varying depths where the backend with the minimum queue length is selected are shown in Table 2. The circuit depth, average execution time, and average queue length

Figure 3: Depth Conscious Scheduling



are shown while running the procedure at 5 different times. The data is also represented graphically in Figure 3. While Table 2 shows averages, Figure 3 shows every data point plotted on a scatter plot of the queue length of the selected backend and the execution time. The 5 different colors correspond to the five runs performed to collect the data.

Table 2: Depth Conscious Scheduling

Circuit Depth	Average Queue Length	Average Execution Time (s)
1	0	18.54374
1	0.2	18.6827
1	7	205.2663
1	25	239.5091
1	52.8	188.5003
5	0	30.87362
5	0	20.77894
5	1.8	176.5415
5	21.4	200.5158
5	51	24.55472
10	0	20.39888
10	0	86.67842
10	9	90.65643
10	17.8	240.9247
10	49.6	24.76164
15	0	21.3227
15	0.2	24.91586
15	10.4	48.27044
15	16.4	96.85856
15	50	19.03058
20	0	27.17698
20	0.4	18.56392
20	10.6	48.80586
20	22.4	194.9741
20	34.4	29.79254

The results shown do not give an ideal trend for the relationship between input circuit depth, queue length, and average execution time. However, this is largely due to IBM's scheduling algorithm used to manage the queue after the circuit has been sent out. There were

many data points where it appeared that the program being tracked was getting starved out due to other circuits in the queue. Essentially, the circuit being sent would not be prioritized in favor of some other processes, regardless of when they were received, resulting in a significantly longer execution time even if the queue length was not too large. This resulted in significant outliers in the distribution of data that made drawing a clear conclusion difficult.

Despite the imperfections in the data, they are still telling of the factors which influence the execution of a quantum program, the ability to draw conclusions is just muddied by the allocation algorithm used by IBM to manage their backends. Circuits with higher depths will take longer to execute because they contain more layers of sequenced instructions. This is also why it was a factor in IBM's scheduler and should be a considered factor in any scheduling heuristic. Similarly, the queue of jobs waiting to be executed will also affect the execution time as there will be some delay before the program can begin to execute. While this concept is intuitive, it can be applied to a scheduler when scheduling a sequence of programs onto the same region of hardware. In this way it would be able to balance the cost of waiting for the resources to be available with scheduling it on another, potentially less reliable, but available region. These considerations outline the basis under which quantum programs should be scheduled: making efficient resource allocation decisions to maximize the reliable execution of processes while minimizing the time needed to do so.

Chapter 5

Conclusion

The research which contributes to the development of a fully functioning quantum computer has made considerable strides in the decades since the field's inception, however, it will require much more to meet this goal. While the limitations in the robustness and scalability of qubits are a primary concern, other necessary factors for any computer, such as the management of the execution of processes, are similarly important. This motivates research into the development of quantum compilers and schedulers which can efficiently allocate system resources to manage the execution of quantum programs, with a focus on accommodating the current, error-prone NISQ devices in use. Through consideration of the quantum computer's error rates, its topographical layout, and the circuit's parameters, it has been shown how a quantum compiler can maximize the efficiency of its operation: a necessary component of a quantum computer.

Through the work described, this thesis was able to confirm some of the factors mentioned which influence the effectiveness of a computation and therefore, are important when designing a quantum compiler. While the procedure left us unable to map the computation onto a specific subset of qubits on a single device like a quantum compiler would do, the results produced are still informative. The individual backends chosen from can be considered analogous to different regions of a quantum computer with different error rates and topologies. In this way, the work done with limited control over IBM's quantum computers is still relevant and applicable to current research. Further research could be performed where the researcher has robust control over a local quantum computer. This would allow for better management of the assignment of resources in the execution of programs, giving more unambiguous and direct data about different scheduler configurations or heuristics. Further research could also incorporate other concepts that further complicate the system such as the handling of entangled qubits or mapping the

process onto specific qubits based on their respective error rates. These considerations are required to fully implement a quantum compiler, with this work being concerned more with the considerations needed for efficient scheduling, and their relative importance.

This work contributes toward the larger goal of implementing a fully functioning, fault-tolerant quantum computer. If achieved, it would be a momentous milestone in our scientific and technological development. The full possibilities are still being realized but it is already known that quantum technologies will offer significantly more computational power that what's available to classical ones today. The applications are still being realized but are very exciting and sure to have a significant impact on society. With interest around the field growing, there is a lot of confident that this goal will be realized in the near future and a lot of excitement around the possibilities it will allow for.

BIBLIOGRAPHY

[1] Barenco, Adriano, et al. “Elementary Gates for Quantum Computation.” *Physical Review A,* vol. 1, 23 Mar. 1995.

[2] Bravyi, Sergey, and Alexei Kitaev. “Universal Quantum Computation with Ideal Clifford Gates and Noisy Ancillas.” *Physical Review A,* vol. 71, ser. 022316, 22 Feb. 2005. *022316.*

[3] Cao, Yudong, et al. “Quantum Chemistry in the Age of Quantum Computing.” *Chemical Reviews,* vol. 119, no. 19, 28 Dec. 2018.

[4] Choi, Jaeho and Joongheon Kim. "A Tutorial on Quantum Approximate Optimization Algorithm (QAOA): Fundamentals and Applications," *2019 International Conference on Information and Communication Technology Convergence (ICTC),* 2019.

[5] Davis, Marc, et al. “Toward Optimal Topology Aware Quantum Circuit Synthesis.” *IEEE International Conference on Quantum Computing and Engineering (QCE),* 12 Oct. 2020.

[6] “Executing Experiments (Qiskit.execute_function)¶.” *Executing Experiments (Qiskit.execute_function) - Qiskit 0.34.2 Documentation,* https://qiskit.org/documentation/apidoc/execute.html.

[7] JavadiAbhari, Ali, et al. “ScaffCC: A Framework for Compilation and Analysis of Quantum Computing Programs.” *ACM Conference on Computing Frontiers,* 20 May 2014.

[8] Heckey, Jeff, et al. “Compiler Management of Communication and Parallelism for Quantum Computation.” *International Conference on Architectural Support for Programming Languages and Operating Systems,* 2 Mar. 2015.

[9] Hu, X, et al. “Decoherence and dephasing in spin-based solid state quantum computers. In Foundations Of Quantum Mechanics In The Light Of New Technology: ISQM—Tokyo’01”, pages 3–11. World Scientific. 2002

[10] Maronese, Marco, et al. “Quantum Compiling.” *ArXiv Preprint ArXiv:2112.00187,* 1 Dec. 2021.

[11] Moro, Lorenzo, et al. “Quantum Compiling by Deep Reinforcement Learning.” *Communication Physics,* vol. 4, 31 May 2021.

[12] Murali, Prakash, et al. “Noise-Adaptive Compiler Mappings for Noisy Intermediate-Scale Quantum Computers.” *International Conference on Architectural Support for Programming Languages and Operating Systems,* 30 Jan. 2019.

[13] Nielsen, Michael A., and Isaac L. Chuage. *Quantum Computation and Quantum Information*. Cambridge University Press, 2010.

[14] Paradis, Anouk, et al. “Unqomp: Synthesizing Uncomputation in Quantum Circuits.” *Programming Language Design and Implementation*, 23 June 2021.

[15] Pirhooshyaran, Mohammad and Tamas Terlaky. “Quantum circuit design search.” *Quantum Mach. Intell.* **3,** 25, 2021.

[16] “Services.” *IBM Quantum*, https://quantum-computing.ibm.com/services?services=systems.

[17] Shi, Yunong, et al. “Optimized Compilation of Aggregated Instructions for Realistic Quantum Computers.” *International Conference on Architectural Support for Programming Languages and Operating Systems.* 17 Feb. 2019

[18] Tzvetan S. Metodi, et al. "Scheduling physical operations in a quantum information processor," Proc. SPIE 6244, Quantum Information and Computation IV. 12 May 2006

ACADEMIC VITA

Javier Montaner

EDUCATION

PENNSYLVANIA STATE UNIVERSITY: SCHREYER HONORS COLLEGE (AUG 2018-MAY 2022)

- Majors: Physics, Computer Science
- Minors: Math, Philosophy, Cybersecurity Computational Foundations

CONESTOGA HIGH SCHOOL (AUG 2014-JUNE 2018)

- GPA (Weighted): 4.98
- GPA (Unweighted): 4.00

RESEARCH PROJECTS

QUANTUM COMPUTING COST MODEL (JAN 2020-PRESENT)

- Done under Dr. Swaroop Ghosh, Penn State, Dept. of Electrical Engineering and Computer Science
- Worked on a variety of problems and computations by writing and modifying code written using Qiskit and similar quantum packages in Python.
- Used Quantum Approximate Optimization Algorithm (QAOA) to solve Max-Cut problems for different problem graphs and circuit configurations to understand the costs for a given quantum operation.
- Looked at scheduling costs associated with running quantum programs such as tradeoffs between resource use and quality of the output as well as efficient scheduling algorithms and the differences as compared to classical ones

QUANTUM MACHINE LEARNING (DEC 2020-JUNE 2021)

- Done under Dr. Sean Hallgren, Penn State, Dept. of Electrical Engineering and Computer Science
- Worked to address theoretical obstacles faced by quantum neural networks when training.
- Involved thorough reading and discussion of literature and then meeting with the fellow researcher writing the neural network and giving suggestions/warnings that will optimize training.
- Done in part with an interdisciplinary group (quantum computation, machine learning, and biochemistry) looking to use quantum machine learning to generate ligands that will bind most effectively to an input receptor pocket, then used to enhance drug development.

MACHINE LEARNING IN MOLECULAR ASTRONOMY (JUNE-AUGUST 2019)

- Done under Dr. Robert Forrey, Penn State Berks, Atomic and Molecular Astrophysics
- Trained a machine learning program to reduce the error of data used to represent molecular collisions in large astrophysical simulations.
- NSF funded internship
- Co-authored in a publication from this work (see Publications)

SCHOLARSHIPS AND RECOGNITIONS

- Schreyer Honors College: Schreyer Scholar (2019-present)
- Eberly College of Science: Braddock Scholar (2018-present)
- Elbasch Scholarship in Physics (2020)
- Sigma Pi Sigma Physics Honors Society (2021-present)
- John and Elizabeth Holmes Teas Scholarship (2021)

LEADERSHIP

THE QUANTUM INFORMATION AND COMPUTATION CLUB: PRESIDENT AND FOUNDER

- Founded in February 2021 to introduce undergraduates to quantum computation as well as to foster a community that allows for networking of like-minded individuals who are drawn together by the interdisciplinary nature of the field (physics, math, computer science, computer/electrical engineering)
- As president I have given talks on introductory topics, invited professionals in the field to come speak, pooled and distributed resources to the members, and coordinated the administrative necessities of operating a club at Penn State.

PENN STATE BLUE BAND: SECTION LEADER

- Selected as section leader of the Trombone section for the 2020 and 2021 seasons.
- Responsibilities include keeping the section focused, giving feedback on proper marching form, enforcing the director's plans, and establishing the appropriate tone for rehearsals and performances.
- Helped lead the band through the difficulties of the COVID-19 pandemic by enforcing safety restrictions and maintaining a positive atmosphere despite our limited capabilities.

ACTIVITIES

COLLEGE

- Quantum Information and Computation Club (2021-present)
- Penn State Blue Band: Trombone (2018-present)
- Society of Physics Students (2018-2020)
- Spark Program (2019)
- DevPSU (2019)

HIGH SCHOOL

- National Honors Society (2016-2018)
- Tri-M Music Honors Society (2016-2018)
- New Voters: Voting Registration Drive (2017-2018)
- PMEA All-State Band: Trombone (2017, 2018)
- Second Degree Black Belt in Tang Soo Do (2005-2015)

PUBLICATIONS

Jasinsky, A, Montaner, J, et al. "Machine Learning Corrected Quantum Dynamics Calculations." *Physical Review Research*, vol. 2, no. 3, 27 Aug. 2020.

- Based on work done with Dr. Forrey in Summer 2019. A copy is available upon request

SKILLS

- Experienced quantum computing researcher with a strong academic background and a good understanding of different topics and current research in the field
- Experience with high level math such as numerical analysis, probability and statistics, differential equations, and vector calculus
- Experience with high level physics such as theoretical mechanics, electromagnetism, statistical mechanics, and quantum mechanics
- Proficient in multiple programming languages (Python, C, Java, Racket)