Elementary workflows for programming quantum computers with QASM and Qiskit
Final research report
Dylan Miracle
ICS 698-02
Spring 2021
Apr 20, 2021
Dr. Jigang Liu

Elementary workflows for programming quantum computers with QASM and Qiskit

Dylan Miracle
Department of Computer Science
Metropolitan State University
St. Paul, Minnesota, USA
dylan.miracle@my.metrostate.edu

May 1, 2021

Contents

Abstract

Quantum computing is in its infancy but programmers can become familiar with the fundamentals of programming a quantum computer. A quantum computer can be programmed using a quantum circuit. In this paper a model for quantum computing is developed using technologies from IBM that include an intermediate representation of a quantum circuit in QASM. Then the circuit is manipulated in Python using the library Qiskit. Finally a general workflow is illustrated for quantum computing that will help traditional software developers understand how quantum computers will be used.

1 Introduction

In the early days of digital computing a programmer needed a deep understanding of electrical engineering to program computers. The programming was low level involving digital logic gates and binary code. Abstractions were slowly built in to allow programmers to manipulate abstract data types so that now you can program a computer in a human readable language that uses high level concepts such as object inheritance and lambda calculus. Quantum computing is in its early phase and will likely progress to human human readable abstractions quickly. Developers who want to learn to program quantum computers will be served well to learn a programming framework to experiment with as opposed to taking a PhD in physics before learning the basics of quantum computing. It is not necessary to understand how a computer stores a string to build a web page, we should expect the same kinds of abstractions from quantum computing in the coming decade.

Frameworks such as qiskit and others already give a programmer access to many tools for interpreting and experimenting with quantum computers. This work aims to show an interested developer how to get started programming and interpreting quantum results. After reading this paper, for example, a developer would be able to create a webpage to display quantum results because they could understand how the result data is returned. Without needing to know the ins-and-outs of quantum computing they would be able to manipulate result data with a classical computer. This is a valuable tool because quantum computers will always be used with classical computers.

Quantum computing has a rich theoretical background but practical quantum computers have only recently become available to researchers and business users. These computers are still rare and operated by a small number of hardware vendors. The dominant paradigm for programing quantum computers is the gate model of quantum computing that defines a sequence of operations on a set of qubits. In this paper we introduce a fundamental quantum circuit and demonstrate its implementation using an intermediate representation, QASM, and a higher level quantum computing framework Qiskit.

In this paper we will explore how quantum computing will fit into a standard computing workflow. This work gives an overview, from mathematical development of a simple quantum circuit to implementation with a high level quantum framework. Readers will get a taste of the complete quantum pipeline. This is valuable because much work focuses on one area of the ecosystem instead of walking those new to quantum mechanics through the complete process of how to implement some quantum code. For example, there are many papers detailing the use of a particular language, or that go into depth on specific quantum algorithms. Even though we focus on a specific framework the model of understanding quantum development proposed by this paper will be useful to anyone who wants to delve further into quantum computing.

2 Background and Literature Review

A basic discussion of quantum computing will involve developing two fundamental concepts in quantum computing, namely superposition and entanglement. Specifically we will use the example of the Bell states. These are the most basic states that exhibit the main features of quantum computing, namely superposition and entanglement of qubits. The development of this basic quantum circuit will start by exploring qubits and some basic quantum operations [1, 2].

2.1 Qubits

Qubits are represented as two element column vectors. The elements are complex numbers. Since we measure qubits to be either 0 or 1 we will talk about them as being in the computational basis. This is not the only basis we can measure qubits. A single qubit can be visualized as a vector inside a unit sphere. This sphere is known as the Bloch sphere.

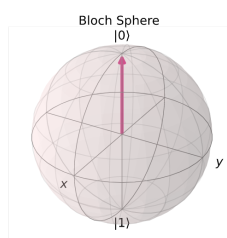

Figure 1: Bloch vector for qubit $|0\rangle$.

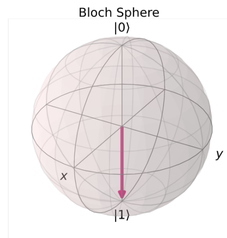

Figure 2: Bloch vector for qubit $|1\rangle$.

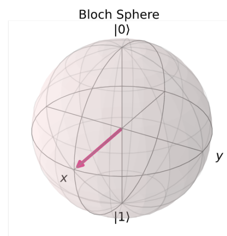

Figure 3: Bloch vector for qubit $(|0\rangle + |1\rangle)/\sqrt{2}$.

Fig. 1, Fig. 2 and Fig. 3 show a representation of 3 different pure states for a qubit. Measuring the state in Fig. 1 will always result in 0. Measuring the state in Fig. 2 will always give a 1. Measuring the state in Fig. 3 will give a 1 or 0 with equal probability. For the purposes of this introduction it is only important to know that each of these sates is represented by a column vector. The state shown in Fig. 1 is represented by

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}.$$

The state shown in Fig. 2 is represented by

$$|1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

Finally the state in Fig. 3 is represented by

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}.$$

In most quantum circuits qubits start in teh $|0\rangle$ state and are then manipulated using gates that can be represented as matrices. The single qubit gate used to take the starting state $|0\rangle$ into state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$, called a superposition and shown in Fig. 3, is the Hadamard gate and can be represented as the matrix

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

With these states defined graphically and mathematically we are ready to move on to our treatment of the Bell state.

2.2 Bell state

We achieve a bell state by using a CNOT gate to combine a $|0\rangle$ with a superposition like $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$. This creates what is known as an entangled state. This combination results in a state represented as

$$\frac{|00\rangle + |11\rangle}{\sqrt{2}}$$

With the property that any individual measurement of on of the qubits is random, but the measurement of the second qubit is entangled with the measurement of the first. So a measurement of 1 for qubit 0, means we will get a 1 for qubit 1 as well. In the next section we will simulate such a state and visualize it with a histogram.

2.3 Qiskit and QASM

Quantum computing ecosystem includes many different software architectures, compilers and languages. The dominant tool used in developing quantum algorithms is the quantum gate model acting on arrays of qubits. IBM Quantum Experience allows users to simulate and implement quantum circuits on IMB simulators and hardware. Users of the IBM QE are able to develop circuits graphically using a drag and drop circuit builder of as code using the QASM language [5]. The purpose of QASM is to act as an Intermediate Representation between an imagined quantum circuit and a compiled quantum code, real signals that control a quantum computer. The intermediate representation is similar to x86 assembly code acting between a C program and machine code.

QASM allows developers to declare two kinds of data, quantum registers, qubits, and classical registers, bits. These are declared as `qreg, creg` respectively. Each register is a 1D array of either qubits or classical bits. These are referenced as C arrays. For example the following code snippet will create two qubits and three classical bits. Then will measure the value of the first qubit and store the output in the first classical register. The syntax is similar to C and creates a human readable method for manipulating quantum bits.

Qiskit is the most widely used framework for programming quantum computers. We consider three types of users: algorithm developers, circuit designers, and quantum physicists [6].

A user can use QASM to define a quantum circuit and use Qiskit to submit the circuit to simulators and quantum hardware at IBM. The quantities of interest are different for different types of users. While some users want a simulator that gives access to idealized quantum states, others are interested in the impacts of noise on the performance of certain gates.

A general overview of the Qiskit API consists of a Provider (hardware if available), a Backend (simulator or hardware handler), Job (execution) and

Result Data Structure. This method will outline all qiskit experiments and gives users a framework to design and implement runs on quantum computing hardware or simulators. This is also important because it provides a consistent data structure for result data that means we can write tools for interpreting results that are extensible.

3 Findings

3.1 Quantum and classical workflows

The central finding of this paper is that quantum computing will employ workflows from classical digital computers that software developers are used to using to implement new types of computation that will me new to most software developers. We show that the preparation of a quantum circuit is created with a domain specific language, then loaded into a python program that will look familiar to python developers. The quantum workload will be sent to a quantum backend where the quantum computation will be done, then the results will be collected into a data structure that can be analyzed and displayed using standard methods. We demonstrate a simple quantum program and decompose its workflow into the classical and quantum parts so that we can understand how developers will used quantum computing in the near future.

Developing quantum algorithms is found to follow a flow from the quantum mechanics, to a quantum circuit, to intermediate representations, to higher level frameworks. The flow illustrated in Fig. 4 shows the findings of this paper and will serve as an outline for our exploration of quantum computing.

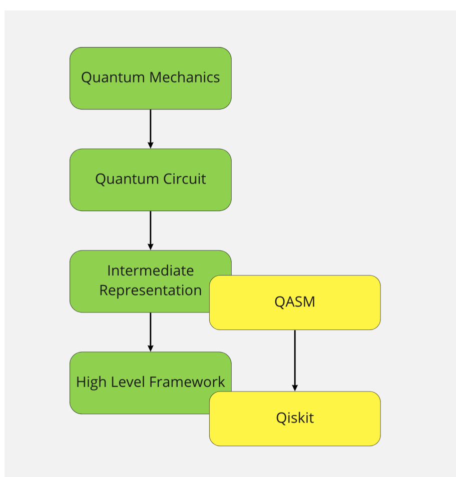

Figure 4: The quantum development flow.

3.2 Program bell state in QASM

QASM is the basic assembly level intermediate representation for quantum code. It allows us to create two types of registries: quantum registries and classical registries. The quantum registries hold the information in qubits while the classical registries hold information in bits. Higher level languages use python libraries to represent the different parts of the quantum program.

Pure quantum computing involves superposition and entanglement. We will use a hybrid approach, preparing our problem on a classical computer, then submitting a job to a quantum computer, finally gathering and displaying our results using a classical computer.

In this program we will implement the Bell state by preparing two quantum registers and two classical registers. The quantum registers will then be put into a Bell state by sending one qubit through a Hadamard gate to create a superposition, then both qubits through a CNOT gate to entangle them. Finally a measurement will be performed on each qubit independently.

Fig. 5 shows the standard circuit representation for gate model quantum computing. This is a simple circuit that creates a Bell state with two starting qubits, then shows the measurements of those qubits being written into classical bits.

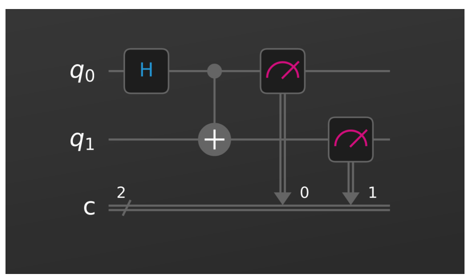

Figure 5: A quantum circuit that will create a Bell state with two qubits.

The QASM implementation of the Bell state is shown in Fig. 6

Once we have defined the circuit using the intermediate representation of QASM, we can import the circuit into a higher level language that will be used to submit the circuit to a simulator or to quantum hardware. In the following example we will use Qiskit to load our circuit and submit it to the builtin simulator.

3.3 Load circuit into Qiskit and submit job

The Qiskit program that follows will expand our workflow. We use standard methods of python programming familiar to anyone used to using the language to load our libraries, import the circuit, then submit the circuit to the simulator backend. The only change we would need to make to submit this program to actual quantum hardware is the change the backend to point at a quantum

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[2];
h q[0];
cx q[0],q[1];

measure q[0] -> c[0];
measure q[1] -> c[1];
```

Figure 6: bell.qasm

processor. The Qiskit code used to run our circuit against a simulator is shown in Fig. 7.

```
import qiskit
from qiskit.visualization import plot_histogram

qc = qiskit.QuantumCircuit.from_qasm_file("bell.qasm")
qc.h(0)

backend = qiskit.Aer.get_backend('qasm_simulator')
result = qiskit.execute([qc], backend, shots=100)
count = result.result().get_counts()
plot_histogram(count)
```

Figure 7: bell-qiskit.py

3.4 Summary of findings

By running our simulation 100 times we can gather statistics on the results of the measurements. Here is a histogram of measurements from the simulation shown in Fig. 8.

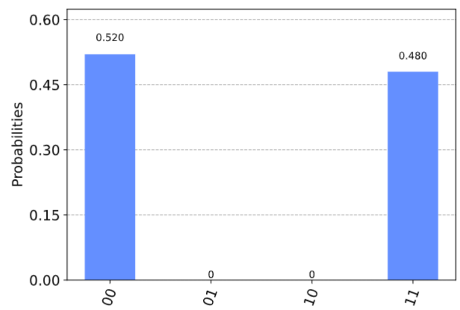

Figure 8: Simulations of measurements of Bell state.

While QASM and Qiskit are specific to IBM, the model they use of intermediate representation and higher level language is present with other vendors (see Rigetti) and a good way to understand and control a quantum computer. An intermediate language gives us a method to write out specific gates used in most quantum computing algorithms, while the higher level framework lets us use standard computing.

4 Evaluation

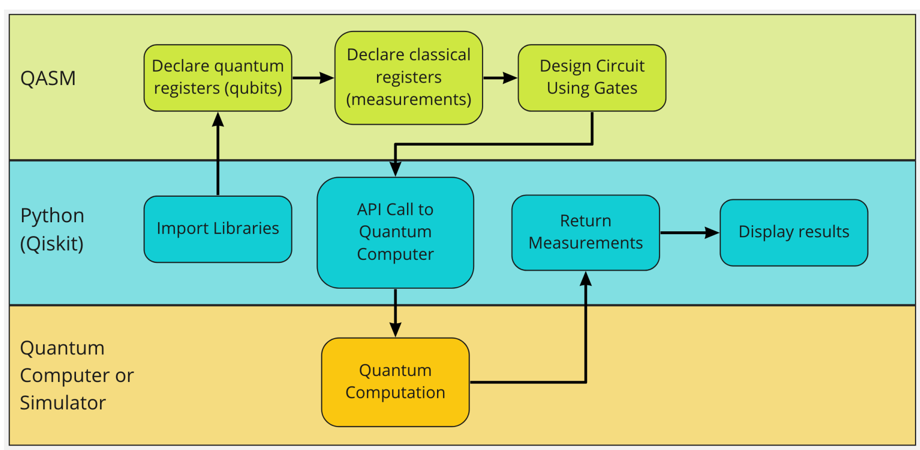

Figure 9: The quantum computing workflow.

This work allows us to compare the process of programming a classical digital computer with a quantum computer. From this work we can see that much of the workflow that we will use to create quantum programs will be familiar to software developers, even if the actual quantum algorithms are not. The model used by qiskit to run jobs and return results means that programmers can predict what kind of data they will be dealing with before they fully understand the algorithms being used on the quantum side. As more software has a quantum components having well defined result data and job definitions means that software developers can create code for interpreting results and dealing with the output of quantum computers without becoming quantum computing experts. The process of implementing quantum computations is illustrated in Fig. 9

Developers can write code for quantum computers by learning frameworks in already existing languages. Several frameworks studied in the literature review have python frameworks. Python is often used because it is broadly adopted by the scientific community and creates a lower barrier to entry than some other languages might. For example Microsoft's Q# language uses a combination of F# and C# that is less familiar to people not used to writing code in both of those languages. A good framework gives developers access to the tools needed to run quantum gates, but also standard computing tools that are helpful in interpreting results.

Conclusion

We should understand quantum computation as a workflow between the classical computer and the quantum computer. In this example we will create a simple quantum program and follow its execution using standard open source tooling available for programming IBM quantum computers. At the highest level we can understand that the classical computer is used to define the computation we will do, then an API is used to communicate the quantum job to the chosen backend, in our case a simulator, though a similar method will be used to submit the job to actual quantum hardware.

Quantum computing will not stand on its own, but will always be employed in tandem with classical computing. This is as simple as using standard computers to create flow control or as advanced as using the fully featured APIs needed to connect to actual quantum hardware. As discussed the data returned by quantum algorithms will be interpreted and represented by classical computers. It is possible to currently develop software that will use quantum results in the future because well designed frameworks like qiskit have a standard workflow and results data structure.

Much work has been done to develop quantum computing frameworks. These frameworks are racing to increase functionality and capabilities for advanced users. This paper should give an overview of some of the fundamentals of quantum computing for an advanced undergraduate or software developer without a background in quantum mechanics. The audience for most quantum computing work is physics PhDs making it inaccessible to a typical software developer. This has a chilling effect on a bourgeoning industry. When we think about the great contributions to our modern information age we often end up thinking about hackers in a garage just as much as engineers in a clean room.

5 Next Steps

The next part of this study would be to extend our workflow understanding to encompass what tools are available to build hybrid workflows for the different kinds of quantum applications that will be available in the short term. The quantum computers that are available now are known as noisy intermediate scale quantum (NISQ) computers. These can only run a certain subset of quantum algorithms. Next steps would be to create workflows for NISQ computers.

To extend our workflow we would need to implement a more complicated algorithm than the one used for this paper. One candidate is known as the Variational Quantum Eigensolver. This algorithm can be used to compute the ground state of molecules exponentially more efficiently than standard molecular energy solvers available on classical computers. Such an algorithm would fit into the workflow developed in this paper, however that would only encompass a single iteration of such an algorithm. We would need to design a workflow that could send data to the quantum hardware, interpret that data on the classical computer, then create a new submission for the quantum computer.

6 References

References

[1] M. S. Saraivanov, Quantum Circuit Synthesis using Group Decomposition and Hilbert Spaces, MS, Portland State University, 2013.

[2] M. A. Nielsen and I. L. Chuang, Quantum Computation and Quantum Information, 10th Anniversary ed., Cambridge, 2016, pp. 171–215.

[3] R. LaRose, "Overview and Comparison of Gate Level Quantum Software", Quantum Journal, 3/25/2019

[4] F. Leymann, J. Barzen, "Hybrid Quantum Applications Need Two Orchestrations in Superposition: A Software Architecture Perspective", arXiv:quant-ph/2103.04320, 2021

[5] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open Quantum Assembly Language," arXiv:quant-ph/1707.03429, 2017.

[6] A. Asfaw, et al., Learn Quantum Computation Using Qiskit, 2020, http://community.qiskit.org/textbook, University of Waterloo, 2008.