Modeling of and Co-Design for the Duke STAQ Platform

Quentin Sieredzki - qvsiered

North Carolina State University

qvsiered@ncsu.edu

Faculty Mentor - Dr. Frank Mueller

Gitlab Link: https://gitlab.com/fmuelle/qisdax

***Abstract*— The goal of this research project is to expand the previous Qisdax project, which allows for the translation of high-level IBM Qiskit programs into low-level Duke ARTIQ extensions (DAX). IBM Qiskit programs already have an extensive existing userbase, and have a rich variety of quantum algorithms available. Translating these programs into DAX will allow for researchers to run more sophisticated programs easier on the STAQ program's ion trap quantum hardware platforms. The prior Qisdax project had functionality for translating most simple gates from Qiskit to Dax, but most of the more complicated gates had not been implemented. As a result of the efforts this semester, the implementation of many of these more complicated gates has been completed in Qisdax. However, there is still work to be done to complete some of the more difficult gates, and to test the output DAX programs using a simulator to ensure the quality of the generated programs.**

I. Problem Statement

Quantum computing has the potential to provide a significant advantage over classical computing in terms of algorithmic complexity. The STAQ project is focused on demonstrating such an advantage on an ion trap quantum hardware platform developed at Duke with 64 or more qubits [1]. We propose to devise a co-designed software control framework to program quantum devices at a low level and demonstrate this capability with STAQ devices. The objective is to enable existing quantum programs to become compatible with STAQ, to translate to low-level control code, to provide control over alternating classical and quantum execution, and to enable novel experimental circuit design beyond existing capabilities. As depicted in Figure 1, we will create a transpiler infrastructure to translate IBM Qiskit programs into low-level Duke ARTIQ extensions (DAX) [7], which provide controls over the ion trap lasers via an ARTIQ FPGA interface. This will instantly provide access to a rich variety of quantum algorithms available as Qiskit programs, which can then be run on STAQ devices.

Specifically, this project focuses on expanding on previous efforts to create the translation program (known as Qisdax) to add functionality necessary for practical usage. Previously, many of the fundamental operations necessary for quantum algorithms had been implemented. However, some of the more complicated operations necessary for more important algorithms had not been implemented into Qisdax. This is what the work on the project this semester primarily concentrated on.

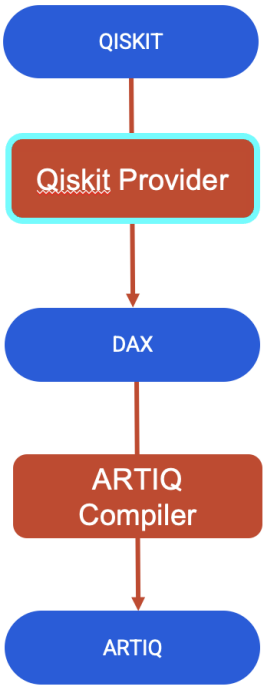

Figure 3: Transpiler Infrastructure

II. Short Literature Summary

The main source of literature came from the documentation for Qiskit [2]. This was necessary for figuring out how the gates worked, and how to use the parameters from Qiskit into Qisdax.

Additionally, research papers were used to figure out the combination of gates necessary to implement operations. IBM quantum machines use a superconducting architecture of qubits, and the Duke STAQ machines use ion-trapped qubits. Due to this difference, there is a difference in native gates. In other words, some gates that are callable on IBM machines by themselves cannot be natively realized on STAQ machines, and need to be decomposed into a combination of gates that are callable (and vice-versa). Therefore, it was necessary to figure out these combinations of gates through reading peer-reviewed research papers. Specifically, research papers were used to calculate the C-NOT gate [4], the MS gate [6], and were being used to develop other gates including the Toffoli gate (which still poses the problem of a global phase difference) [5]. However, often these papers would have slightly wrong calculations. They would often have the right combination of gates, but forget a negative sign or have the wrong values for some rotation (theta). Additionally, sometimes the convention for a gate would be different than the implementation in Qiskit, which made things more complicated. Therefore, it was necessary to double check the combination of gates every time after they were found in the literature.

III. Methodology / Process

The main process for the research project was through weekly calls with Dr. Mueller, and then working through the additions necessary for that week. The tasks completed, tasks underway, and tasks to do were discussed during these weekly meetings in the form of a Powerpoint presentation. This allowed for the direction of the project to be constantly clear, and the requirements to be changed as necessary. Additionally, this allowed for accountability by writing down what was expected to be done.

Most of the work was completed within the qobj_to_dax.py file, which contains the rules necessary for the translation from Qiskit to Dax [7]. This is effectively where most of the functionality was changed, and where the bulk of the work was done. Additionally, new test files were constantly created as necessary to ensure that everything was working as expected.

Many of the new gates added into the program needed to have changes due to the differences in architecture between the IBM devices and the STAQ devices. For instance, some gates that are native for IBM devices need a combination of multiple gates in order to perform the same operation in STAQ devices. The process of figuring out the combination was a little difficult, as often the research done to prove these equivalencies are slightly off, and double checking every formula is necessary. This was mostly done by Dr. Mueller, but it still took time to figure out and implement.

IV. Results / Contributions

To give a broad overview of the contributions made this semester, some of the notable gates that were implemented into the Qisdax project over the course of this semester include the Rx, Ry, and Rz gates, the C-NOT (Cx) gate, the XX gate, and the XX4 gate. Additionally, the GMS gate [3] was started but not completely finished.

The Rx, Ry, and Rz gates were by far the easiest to implement into Qisdax. These gates result in a rotation of theta around each of their respective axes (Rx around the x axis, Ry around the y axis, Rz around the z axis). All three have simple parameters of theta and the target qubit, both in Qiskit and DAX. Therefore, translating it was not too complicated as the main difference was just changing the formatting.

Next, the C-NOT (Cx) gate was implemented. This gate was already attempted to be implemented before the start of the semester, but had not been

completely finished or tested. In fact, the initial implementation had both the wrong order of gates, and the wrong formatting. This gate is more involved to translate from Qiskit to DAX than the previous gates discussed due to differences in how the quantum computers are created. The gate is native to IBM quantum computers, which use superconducting qubits rather than ion traps. This effectively means that while in Qiskit only one gate is required to implement a Cx operation, in DAX a series of five gates is required to perform the same operation.

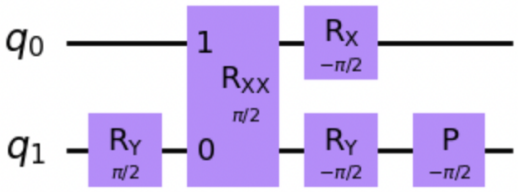

Figure 2: Combination of gates to create Cx

This series of gates was determined by another paper [4], but was ultimately slightly wrong (as discussed in the literature section). After working out the proper values, the gate was implemented correctly into Qisdax.

Finally, the XX and XX4 gates were implemented into Qisdax. These two gates can be grouped together because the XX4 gate is simply a specific version of the XX gate. Essentially, a XX gate has the parameters of theta (rotational angles), and two target qubits. However, DAX has a native gate for a XX gate with a specific theta of pi/4 that is optimized for the machines. Therefore, implementing these gates into Qisdax had the added caveat that the parameter needs to be checked to see if the theta parameter is pi/4.

Another challenge in implementing the XX and XX4 gates into Qisdax is that there are multiple ways to achieve the operation in Qiskit. This is due to the fact an XX gate affects two qubits, but there is a more general version of this that can achieve the same operation on an arbitrary amount of qubits. Therefore, Qiskit has both the specific operation for two qubits (called rxx in the Qiskit implementation), and for an arbitrary amount of qubits (there are two gates that achieve this in Qiskit, the MS gate and the GMS gate). Translating rxx into an XX or XX4 gate was not too difficult, but the arbitrary amount of qubits version was more complicated.

At the moment, the STAQ devices that run DAX only support a two qubit version of the XX operation. However, because this might change in the future, Qisdax needed to extend the functionality to an arbitrary number of qubits if the MS or GMS gates are called in Qiskit. This was achieved for the MS gate. The current implementation also includes a variable that can be adjusted in Qisdax that limits the number of qubits allowed to be entangled to limit the software to the proper amount of qubits. Therefore, the MS gate can be limited to simply allowing the XX gate at the moment, and changed to allow for future advances by simply changing this variable. Unfortunately, while the implementation for the MS gate was completed, the GMS gate was not finished by the end of this semester.

V. Reflection Elements

As a result of the work this semester, the Qisdax program is much further along to be eventually usable by others. This experience was extremely valuable, and it let me develop my understanding of both Quantum Computing and software development in general. I especially found it valuable how it was not too focused on the nitty gritty of Quantum Computing, and allowed me to also refine my general programming skills. With that being said, my understanding of Quantum Computing and Qiskit definitely improved tremendously as a result of my involvement in this research project. Specifically, I feel much more comfortable with my understanding of XX gates, and understand the differences between the ion-trapped and superconducting qubits implementation of quantum computers more thoroughly. I also found that trying to translate Qiskit into Qisdax allowed me to really understand the decisions made in Qiskit, which in turn allowed

me to become more comfortable in how Qiskit can be used. As I am interested in potentially working more on Quantum Computing in the future, this could be extremely valuable as I pursue future opportunities.

However, that is not to say that the project did not have some difficulties. In particular, the two main difficulties that I encountered involved implementing the GMS gate, and testing the DAX programs by using the simulator that Duke provided us halfway through the semester. These two problems were what I was trying to solve for the better part of the last month, and impeded me from accomplishing as much as I wished. For the GMS issue, the difficulty arose from trying to use the parameter provided as an input into Qiskit. Unfortunately, the way that Qiskit implements GMS is completely different from the other gates, and it creates a circuit object from which is difficult to extract the parameters. I spent hours looking through the different directories, but was unable to figure out how to extract the theta parameter. People who work on this project in the future will have to figure this out, as the MS gate is currently being phased out of Qiskit for the GMS gate, so GMS functionality is critical.

The second problem I had was with the DAX simulator. The Duke team provided us with a simulator partway through the semester to try out the generated DAX code. However, it did not work on Mac, which is the computer that I have. I downloaded a VM, and this did not work either because of the architecture of my Macbook Air. Eventually, we realized that the best way would be to use the Linux computers available in the computer labs - however, by this time there was not enough time to install everything. From these difficulties, the main thing I learned was to not feel guilty asking for help. It is better to inconvenience someone for a little bit rather than struggle through for hours, and that was something that took a little bit of time to admit to myself.

References

[1] Artiq, (2021), GitLab repository, https://gitlab.com/duke-artiq

[2] "Circuit Library." Circuit Library - Qiskit 0.32.1 Documentation, IBM Qiskit, https://qiskit.org/documentation/apidoc/circuit_library.html.

[3] "GMS." GMS - Qiskit 0.32.1 Documentation, Qiskit, https://qiskit.org/documentation/stubs/qiskit.circuit.library.GMS.html.

[4] Javadi-Abhari, Ali, et al. "Write Once, Target Multiple Architectures." IBM Research Blog, Qiskit, 5 Nov. 2019, https://www.ibm.com/blogs/research/2019/11/qiskit-for-multiple-architectures/.

[5] Martinez, Esteban A, et al. "Compiling Quantum Algorithms for Architectures with Multi-Qubit Gates." Iopscience, 24 June 2016, https://iopscience.iop.org/article/10.1088/1367-2630/18/6/063029.

[6] Maslov, Dmitri, and Yunseong Nam. "Use of Global Interactions in Efficient Quantum Circuit Constructions." Iopscience, Iopscience, 2 Mar. 2018, https://iopscience.iop.org/article/10.1088/1367-2630/aaa398.

[7] Mueller, Frank, Qisdax, (2021), GitLab repository, https://gitlab.com/fmuelle/qisdax