Intern Projects: Summer 2017

Harley Patton

Rigetti Computing

Table of Contents

Introduction

I spent the Summer of 2017 as a Software Engineer Intern for Rigetti Computing. The following paper is a record of the different projects I contributed to during that time.

1 Grove

Grove is an open source Python library containing algorithms implemented using pyQuil and the Rigetti Forest toolkit. This library is organized into modules for the various algorithms, each of which has its own self-contained documentation. I contributed to the following modules:

1.1 Deustch-Jozsa Algorithm

The Deustch-Jozsa Algorithm [1] is a quantum algorithm that can determine whether a function $f : \{0,1\}^n \to \{0,1\}$ is constant (either always returns 0 or always returns 1) or balanced (returns 0 for exactly half of all possible inputs, and 1 for the other half), provided that f is guaranteed to be one of the two. Any classical algorithm would require a number of calls to f that is exponential in n, but a quantum computer running the Deutsch-Jozsa algorithm could solve this problem in a single iteration with a circuit size linear in n.

I wrote a module [2] implementing this algorithm, and merged it into Grove. In doing so, I also formulated the framework for how quantum "Blackbox Oracle" algorithms could be implemented. Fellow intern Joseph Lin was able to take the methods that I wrote to convert bitstring mappings into unitary circuits for this module and use them in his implementations [3, 4] of the Bernstein-Vazirani Algorithm [5] and Simon's Algorithm [6], both of which were merged into Grove as well.

1.2 Order Finding and Shor's Algorithm

The quantum order finding algorithm can find the order of a modular exponential function $f(x) = a^x \pmod N$, where the order of f is defined as the lowest nonzero r such that $f(r) = 1$. This algorithm takes advantage of quantum superposition, allowing for the calculation of $f(x)$ for all x simultaneously. Then, the Quantum Fourier Transform can be used to tease the correct answer out of the resulting superposition. Order finding is most often used as a subroutine to Shor's Algorithm [7], which provides a polynomial-time reduction from prime factorization to order finding.

I wrote a module [8] implementing the order finding subroutine, and fellow intern Aaron Vontell wrote the classical code reducing prime factorization to my subroutine. As of the time of publication, this module in under code review to be merged into Grove.

1.3 Arbitrary State Generation

Arbitrary classical information can be encoded into the probability amplitudes of a quantum state. For example, the classical data $[\alpha, \beta, \gamma, \delta]$ would be encoded into the states of two qubits as $|\psi\rangle = \alpha\,|00\rangle + \beta\,|01\rangle + \gamma\,|10\rangle + \delta\,|11\rangle$.

I designed a method to create a unitary operator mapping the ground state of a set of qubits to the desired outcome state using QR factorization. I, along with fellow intern Joseph Lin, wrote a module [9] to decompose this unitary transformation into a series of Bloch Sphere rotations and phase adjustments [10], and have merged it into Grove.

2 PyQuil

PyQuil is an open source Python library developed at Rigetti that constructs programs for quantum computers using Quil (the Quantum Instruction Language) [11]. I added the following features:

2.1 Measurement Upgrade

PyQuil did not originally support the ability for the user to measure multiple qubits into multiple locations in classical memory simultaneously. The user would have to either manually loop through each qubit and specify where to measure it individually. I wrote an upgrade [12] to the measurement functionality and merged it into pyQuil.

2.2 Program Reversal

Any quantum circuit that does not involve a collapse of the wavefunction can be expressed as a unitary transformation, which by its very definition must be reversible. I wrote an upgrade to pyQuil that could reverse any quantum program so long as it does not collapse the wavefunction (i.e. no measurement, no classical control flow, and no resource allocation). I merged this upgrade [13] into pyQuil, and also used it to greatly simplify the module [14] in Grove for the inverse Quantum Fourier Transform.

3 Palm

Palm is an internal-use library I created to house a variety of machine learning and optimization algorithms implemented using pyQuil. Under the advice of Dr. Nicholas Rubin, I added the following to Palm:

3.1 Quantum Metropolis Sampling

The Metropolis-Hastings Algorithm [15] is a Markov chain Monte Carlo (MCMC) method for obtaining a sequence of random samples from a probability distribution for which direct sampling is difficult. This sequence can be used to approximate the distribution or estimate its attributes (such as its expected value). This algorithm can be extended to the quantum realm [16, 17] where it can be used to perform a random walk on the state space of n qubits in order to find the system's ground eigenstate.

I wrote a module [18] that implements Quantum Metropolis Sampling for an arbitrarily large system of qubits evolving under an arbitrary Hamiltonian. This module included an experimental implementation that finds the Gibb's states of the Heisenberg ferromagnet on two spin-1/2 particles.

3.2 Simulated Cooling

The Simulated Cooling Algorithm [19] is used to simulate low-temperature properties of physical and chemical systems that are intractable with classical methods. The cooling process is modeled as a one-dimensional random walk by keeping track of the sequence of measurement outcomes of an ancillary qubit. By employing a feedback-controlled loop that favors low-energy updates, this algorithm is effectively a realization of the thought experiment of Maxwell's demon [20].

I wrote a module [21] that implements the Simulated Cooling Algorithm for an arbitrarily large system of qubits evolving under an arbitrary Hamiltonian. This module included an experimental implementation that can cool a one-qubit system down to the ground states of the Pauli operators.

4 Cedar

Cedar is an internal-use library created by fellow intern Aaron Vontell and myself to house a collection of quantum communication protocols designed and implemented using Quil and pyQuil. All modules in Cedar are highly abstracted, with the intention that a user would not even need to know that they are communicating over a quantum channel. With that in mind, we created server and client objects for the following quantum communication protocols:

4.1 BB84 Encryption

BB84 [22] is a quantum key distribution scheme, and is the world's first quantum cryptography protocol. The protocol is provably secure [23], and is used a method of securely communicating a private key from one party to another for use in one-time pad encryption.

I wrote a module [24] that implements client and server object capable of connecting to each other over both quantum and classical channels in order to transmit encrypted information using keys generated from BB84. This module included an experimental implementation that could be run on a single quantum chip, using half of its qubits as the client device and the other half as the server.

4.2 Superdense Coding

Superdense coding [25] is a technique used in quantum information theory that can be used to send two bits of classical information using only one qubit by making use of shared entanglement.

Fellow intern Aaron Vontell and I wrote a module [26] that implements client and server objects capable of sending bit strings of length $2n$ by transmitting only n qubits across a quantum channel. As with the BB84 module, we included an experimental implementation that could be run on a single quantum chip.

5 Race Against A Quantum Algorithm

As part of the buildup to the June 20th launch of pyQuil, fellow intern Aaron Vontell and I designed and implemented an interactive website for demonstrating the capabilities of Forest. In our demo, the user races against the Quantum Approximation Optimization Algorithm (QAOA) [27] to find a solution to MAX-CUT on a randomly generated graph. Aaron implemented the backend server and frontend UI, while I implemented the graph visualizations, interactions, and algorithms.

As of the time of this publication our demo has been viewed by thousands of people from over fifty countries. It is hosted at "demo.rigetti.com".

6 Single-Shot Data Storage

Measurements that require storage of the full single-shot readout data for many runs and repetitions are being limited by the amount of data that can be inserted into a remote postgres database in a single commit, leading to an increasing overhead for database communication. I worked with Dr. Guen Prawiroatmodjo on the Full-Stack Quantum Engineering (FSQE) team to move this measurement data to a data lake hosted on Amazon Web Services.

As part of this effort, I wrote a Python package [28] to read and record single-shot data. Each measurement can kick off sub-measurements; for example, each run of a flux bias automation measurement will start a repark flux bias measurement, which it turn kicks off a series of measurements (cavity spectroscopy, qubit spectroscopy, power rabi). My package implemented an API allowing users to write data to an HDF5 file in a locally mounted volume or in a cloud storage solution, stored in a schema defined by the hierarchy of a measurement and its sub-measurements. It has since been moved to a branch on Willow and is being integrated into the sqlalchemy relational database.

7 Circuit Utilities

Arbitrary single-qubit gates can be decomposed [29] into sequences of elementary rotation gates, and larger multi-qubit controlled gates can often be reduced [30] to instances of simpler two-qubit controlled gates.

Fellow intern Joesph Lin and I put together a circuit utility library [31] that compiles together all of these decompositions. Our library contains the Euler Decomposition for arbitrary single-qubit gates, a decomposition of arbitrary n-qubit controlled gates into n-Toffoli gates and rotations, and a decomposition of controlled unitary time-evolutions of Pauli operators.

8 Quantum University

Our entire intern class has been working to develop a series of lessons and demos [32] to teach quantum computing to both the general public and future interns/employees. We split into three teams (Software, Hardware, Physics) which are

writing their own lessons in parallel. I am on the software team with fellow interns Aaron Vontell and Joseph Lin. I wrote lessons explaining the quantum programming paradigm, how to use Quil and pyQuil, and how to connect to the Quantum Virtual Machine (both the synchronous connection and the asynchronous jobqueue), as well as how to connect to the Quantum Processor Unit (QPU).

9 Talks

During my time at Rigetti, I gave some talks on assorted subjects:

9.1 Quantum Machine Learning (Intern Talks)

As part of our biweekly Intern Talks series, I gave a talk on Quantum Machine Learning on June 29th. I described how quantum computation could potentially speed up training of several existing machine learning algorithms, and used a proposed method for for a Quantum Support Vector Machine [33] as an example. Then, I went into detail on the utility of a quantum sampling device for random walks along Markov Chains, using Quantum Metropolis Sampling [16, 17] as an example.

9.2 Shor's Algorithm (Bay Area Quantum Computing Meetup)

I gave a talk on Shor's Prime Factorization Algorithm [7] at the Bay Area Quantum Computing Meetup on August 17th. I explained the reduction from factorization to order finding, described the quantum circuit for order finding and how it is used as a subroutine of Shor's Algorithm, and used the Shor's Algorithm module [8] in Grove (written by fellow intern Aaron Vontell and myself) to demonstrate a toy example in which I used a quantum program to factor 15.

References

[1] http://rspa.royalsocietypublishing.org/content/439/1907/553
[2] https://github.com/rigetticomputing/grove/pull/26
[3] https://github.com/rigetticomputing/grove/pull/39
[4] https://github.com/rigetticomputing/grove/pull/40
[5] http://dl.acm.org/citation.cfm?id=167097
[6] https://courses.cs.washington.edu/courses/cse599/01wi/papers/simon_qc.pdf
[7] https://arxiv.org/pdf/quant-ph/9508027.pdf
[8] https://github.com/rigetticomputing/grove/pull/30
[9] https://github.com/rigetticomputing/grove/pull/41
[10] http://140.78.161.123/digital/2016_ismvl_logic_synthesis_quantum_state_generation.pdf
[11] https://arxiv.org/abs/1608.03355
[12] https://github.com/rigetticomputing/pyquil/pull/48
[13] https://github.com/rigetticomputing/pyquil/pull/51
[14] https://github.com/rigetticomputing/grove/pull/46
[15] https://arxiv.org/abs/1504.01896
[16] https://arxiv.org/abs/0911.3635
[17] https://arxiv.org/abs/1011.1468
[18] http://bitbucket.lab.rigetti.com/projects/QCS/repos/palm/browse/quantum_metropolis_sampling
[19] https://arxiv.org/abs/1208.2256
[20] https://arxiv.org/abs/1009.5287
[21] http://bitbucket.lab.rigetti.com/projects/QCS/repos/palm/browse/simulated_cooling
[22] http://researcher.watson.ibm.com/researcher/files/us-bennetc/BB84highest.pdf
[23] https://arxiv.org/pdf/quant-ph/0003004.pdf
[24] http://bitbucket.lab.rigetti.com/projects/QCS/repos/cedar/browse/cedar/protocol/bb84
[25] https://pdfs.semanticscholar.org/8393/66c5e0b37a1b2078f7e25f40981f369ea220.pdf
[26] http://bitbucket.lab.rigetti.com/projects/QCS/repos/cedar/browse/cedar/protocol/superdense
[27] https://arxiv.org/abs/1411.4028
[28] http://bitbucket.lab.rigetti.com/users/harley/repos/file_storage/browse
[29] http://d.umn.edu/ vvanchur/2015PHYS4071/Chapter4.pdf
[30] https://arxiv.org/pdf/quant-ph/9503016.pdf
[31] http://bitbucket.lab.rigetti.com/users/joe.lin/repos/circuit_utils/browse
[32] http://bitbucket.lab.rigetti.com/projects/RU/repos/rigetti-university/browse
[33] https://arxiv.org/pdf/1307.0471.pdf