Ryan Baker
Senior Seminar

Quantum Computing

Section 1.0 - Introduction to Quantum

In this evolving world of technology that we live in, things continue to become smaller and faster. Eventually we are going to hit a point where we cannot physically make computer components any faster with our current configuration, and we will have to switch to different means. This is where quantum computing is going to come into play. Quantum computing changes the game when it comes to computer hardware. Quantum computing changes how the architecture itself works. This would involve switching from binary, where bits are zero and one, to Qubits, where bits can be anything based on their position and entanglement. Quantum Computing will be the next generation of computing.

Section 1.1 - Background and Physics

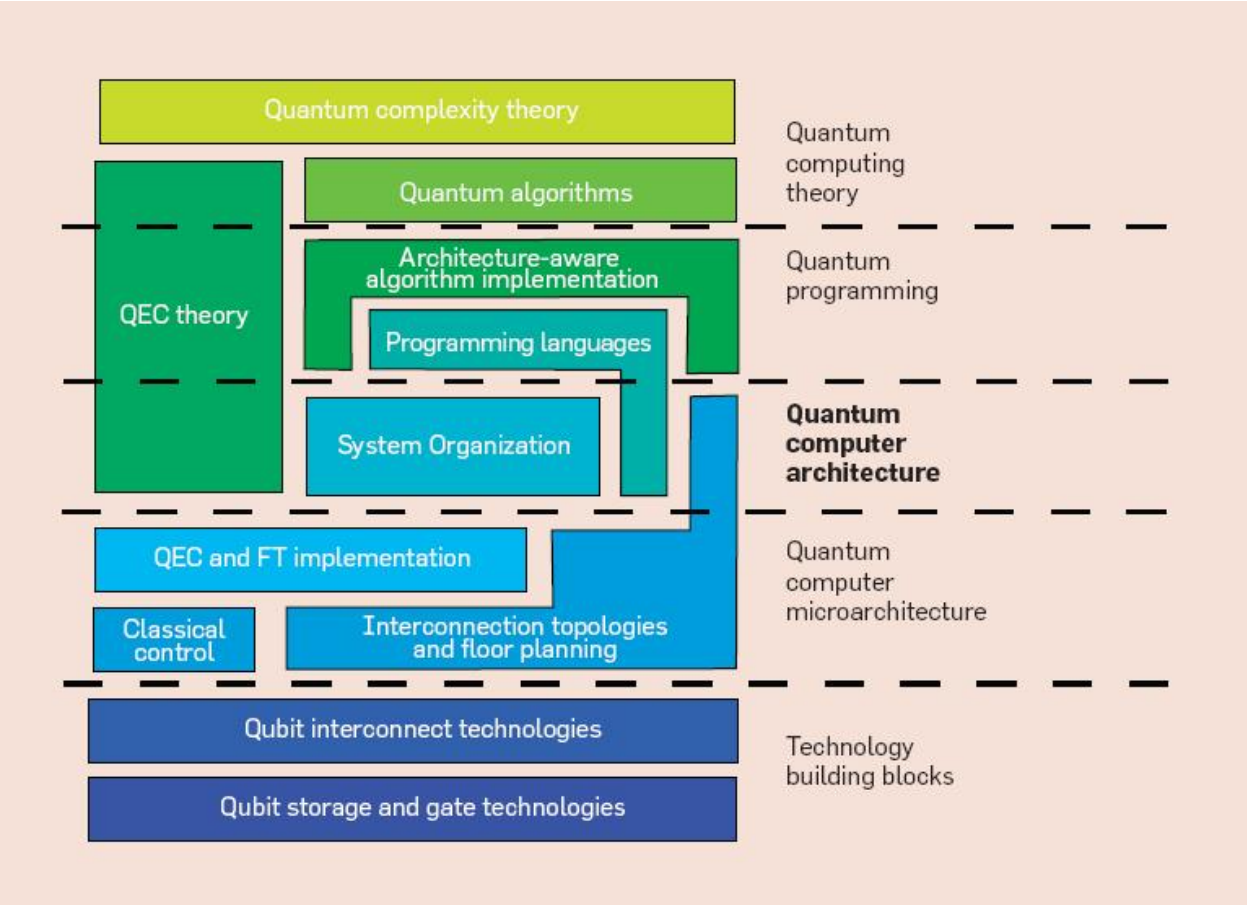
Quantum computing was started by the work of a man named, Paul Benioff. In order to understand how computers will work on the Quantum level, there are a few physics terms and theories we need to discuss. First, we will talk about entanglement. The definition is, "Quantum entanglement is a quantum mechanical phenomenon in which the quantum states of two or more objects have to be described with reference to each other, even though the individual objects may be spatially separated." When we talk about Qubits, entanglement means that the Qubits will interact with each other, regardless of the separation of space between any number of them. Quantum State refers to the state of the quantum system. The quantum states are pure, and mixed. A pure quantum state is represented

by a vector called a state vector in space. Quantum superposition is, “any two (or more) quantum states that can be added together ("superposed") and the result will be another valid quantum state; and conversely, that every quantum state can be represented as a sum of two or more other distinct states.” This is put into place by Schrodinger’s Cat, in this experiment there is a cat that is both dead and alive at the same time, yet when it is observed, the cat is either dead or alive, but not both.

Section 1.2 – Architecture

Computations that are performed inside of a quantum computer are done in a way that is far different from conventional means. This type of computing involves data that is encoded, “in the state of objects governed by the laws of quantum physics.” One of the biggest questions involved with this new type of Architecture is how do we program such manipulation in data when it is not done by “flipping switches” from zero to one.

This is answered by the Quantum Coprocessor Model. In this model, there is a classic type of computer linked to the Quantum Unit, where the classic computer performs the basic operations such as compilation, syntactic corrections, and the preparation of code for the Quantum Computer. The aforementioned coprocessor performs only the Quantum operations. Quantum operations such as, initializations, unitary operations, and measurements. This model described is known as Knill’s QRAM Model. Out of all of the quantum computing models we have, this model is by far the most realistic model for quantum computing. The one major issue with this model is that, yes we do have a quantum computer working, however it is being bottlenecked by being attached to a classical computer. Researchers believe that the coprocessor will communicate with the classic computer through some kind of message queue. On this message queue, the classical computer will then tell the quantum computer what to do.

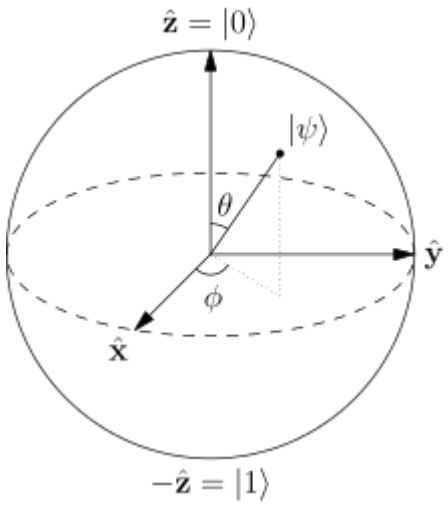


Referencing the Figure shown. At the lowest level, there are the technological building blocks for the quantum machine we are trying to build. A viable quantum computer must have five things. First, a two level physical system to function as a qubit. Second, a means to initialize the qubits into a known state. Third, a universal set of gates between the qubits. Fourth, memory. Finally, a quantum computer must have a long memory lifetime.

Section 1.3 – Introduction to Qubits

In Quantum Computing, Qubits, or quantum bits, are used to hold and manipulate quantum information. A qubit is a two-state quantum-mechanical system. In a classic system, the bits would be in one state or another, on or off. In a quantum system, the qubit is allowed to be in the superposition of both states at the same time. The two states of a qubit are known as the basis states. A pure qubit

state is a superposition of the base states. What this means is that the qubit can be represented as a combination of 0 and 1. The possible states for one qubit can be modeled using a Bloch sphere.



To represent a classic bit on this sphere, it would reside either on the north, or south pole, representing 0 and 1 respectively. All of the rest of this sphere is inaccessible to a classic bit as there are no other options for it. On the other hand, a pure qubit state can be represented by any point on the surface of the sphere. The state space, "has two local degrees of freedom. It might at first sight seem that there should be four degrees of freedom, as α and β are complex numbers with two degrees of freedom each. However, one degree of freedom is removed. Another, the overall phase of the state, has no physically observable consequences, so we can arbitrarily choose α to be real, leaving just two degrees of freedom." Qubits have a wide variety of things they can represent. However, for the sake of the Bloch Sphere and for this paper, we will be sticking to just two degrees of freedom and exclude the other two.

Section 1.4 – Programming

When we say Quantum Programming, we expect nothing less than how we currently program on classical computers. We are hoping for something that allows us to implement quantum algorithms

at a level close to how we naturally think. Knill has taken the time to lay out some guidelines for how quantum programming should work. This includes:

1. Allocation and Measurement
2. Reasoning About Subroutines
3. Quantum Oracles
4. Quantum Data Types
5. Specification and Verification
6. Resource Sensitivity and Estimation

While discussing all of these topics could cover a paper completely separate, I will be picking a few of these and diving into them in detail. First, Quantum Data Types. Much like in classical programming, data types are used to allow the programmer to think about the data abstractly instead of worrying about every single word or digit that could be entered by the end user. Many quantum algorithms need richer data types to allow this kind of abstraction. For example, "the Quantum Linear System Algorithm requires manipulation of quantum integers and quantum real and complex numbers that can be represented through a floating-point or fixed-precision encoding." Second, Specification and Verification. As you would expect, in classical programming there are a multitude of ways for making sure the programs we write are correct, not only logically, but also syntactically. However, this is different in quantum computing. This is mostly due to the fact that once you observe the quantum machine, you are prone to change its current state. One of the biggest hardships to overcome will be the fact that in order to assure the correctness of the quantum algorithm, we would have to come up with a way to debug, without actually observing the machine itself, as that would change the state. Finally, Resource Sensitive and Estimation. In the beginning of their life, Quantum Computers are most likely not going to have many qubits with which to operate on. In order to make a programming

language at the quantum level more successful, it will have to have a way to detect what kind of resources it has, and also what resources the program it is trying to execute needs. One of the most successful Quantum Languages so far is Quipper.

Section 1.4.1 – Quipper

Quipper is a functional language for quantum computation. Quipper is a deeply embedded domain specific language inside of the language Haskell. Quipper was made with the intention of bringing a unified all-purpose programming frame work for quantum computing. Some of the main features of Quipper are as follows:

1. Hardware Independence – Quipper was made to view the computation of the quantum computer at the level of the logic circuits. Error correction and mapping are left to other pieces of hardware farther down in compilation.
2. Extended Circuit Model – Initialization and termination of bits is tracked by Quipper.
3. Hierarchical Circuits – Quipper has subroutines at the logical circuit level. This allows a compact representation of memory.
4. Versatile Circuit Description Language – Quipper is multifaceted in that it can handle multiple programming styles, both procedural and functional. This also allows for higher levels of circuit manipulation.
5. Two Runtimes – The first runtime is circuit generation, whereas the second runtime is circuit execution. There are two modes in which this operates, batch and online. In batch mode, these two runtimes happen one right after the other. However, in online mode, the two may happen at the same time.

6. Parameter and Input Distinction – Quipper is able to determine the difference between parameter, things that we implement before run time, and inputs, which are things that are only known after a particular circuit has been executed.
7. Automatic Generation of Quantum Oracles – Quipper has the ability to turn ordinary Haskell code into a reversible circuit. After the Haskell code is made into a reversible circuit it is then made ready for quantum operation.[1]

```
plus_minus:: Bool -> Circ Qubit

plus_minus b = do

q <- qinit b

r <- hadamard q

return r
```

Above is a brief example of code from Quipper. The very first line shown tells us what kind of function we are dealing with. In this case, the input is going to be Boolean. On the other side, the output is going to come in the form of a Circ Qubit. The word Circ, is actually an operator and is used to show that the operation can have a, "physical side effect" when it is evaluated. The final part explains itself, the Qubit part of the first line tells us that the type of output to be expected is going to be a Qubit. The function body normally begins with a do, immediately followed by some operations to be executed in the given order. The qinit operator initializes a new qubit in a state that is corresponding to b. The hadamard operator applies this specific type of gate to the qubit mentioned above. Overall, there are many similarities to how quantum code operates with how we code in a classical way.

Section 1.5 – Conclusion

So, in conclusion, Quantum computing works in a familiar fashion to classical computing. The main difference to take away from Quantum computing is the Quantum Physics that go into the operation of tasks at compilation. Superposition and entanglement change how we interact with data and we become extremely powerful once we figure out how to get rid of the bottleneck that is the classical computer that runs all of the Quantum Code and then sends it to the quantum computer for execution. Quantum computing will be a huge area in the coming years as more and more companies invest more time into figuring out how to make this a standalone thing.

Works Cited

[1]Benoît Valiron, Neil J. Ross, Peter Selinger, D. Scott Alexander, and Jonathan M. Smith. 2015. Programming the quantum future. Commun. ACM 58, 8 (July 2015), 52-61. DOI=10.1145/2699415 http://doi.acm.org.ezproxy.lib.csustan.edu:2048/10.1145/2699415

[2]Bohr, N. (1927/1928). The quantum postulate and the recent development of atomic theory, Nature Supplement 14 April 1928, 121: 580–590.

[3]Cohen-Tannoudji, C., Diu, B., Laloë, F. (1973/1977). Quantum Mechanics, translated from the French by S.R. Hemley, N. Ostrowsky, D. Ostrowsky, second edition, volume 1, Wiley, New York, ISBN 0471164321.

[4]Green, Alexander S., Neil J. Ross, and Peter Selinger. "Introduction to Quantum Programming in Quipper." (2006): n. pag. 19 Apr. 2013. Web. 5 Dec. 2015.

[5]Nielsen, Michael A.; Chuang, Isaac L. (2010). *Quantum Computation and Quantum Information*. Cambridge University Press. p. 13

[6]Rodney Van Meter and Clare Horsman. 2013. A blueprint for building a quantum computer. Commun. ACM 56, 10 (October 2013), 84-93. DOI=http://dx.doi.org.ezproxy.lib.csustan.edu:2048/10.1145/2494568