QISKit and Quantum Computing

developerWorks Open Tech Talk

March 29th 2017


>> MARC-ARTHUR: Open source projects. My name is Marc-Arthur Pierre Louis, series coordinator and moderator. Today we have a thought provoking innovative presentation, quantum programming. It took about 80 years, as you will learn for us to be where we are today. We are only beginning to unravel the power used in applications of quantum theory as illustrated by quantum computing. When you were in college, computer science, solving problems that could be solved in linear or polynomial time, this will jog your memory. You will recall also the dreaded problems. Quantum programming takes a stab at those, as you will soon find out.

I'm not telling the outcome, not willing to steal the presenter's thunder. Suffice it to say that the presenter knows what he is talking about, Mr. Lev Bishop, PhD, physics, will take you through quantum computing and will break down heavy concepts for you. The video will be available and can be replayed as many times as you want. Who knows, quantum computing begins here. I'm proud to introduce to you Dr. Lev Bishop from IBM research. You have the floor.

>> LEV BISHOP: Thank you very much for inviting me to give this talk. Yeah, as you mentioned, I'm a theoretical physicist is my background. I will teach you about what is exciting about quantum computing and introduce to you IBM's new open source tools.

Let me pull up my slides. My outline for the talk is that I'll tell you about why is quantum computing so exciting and the summary is that it allows you to solve problems that are completely intractable with standard computers.

All computers that exist today are classical computers, opposite of quantum. They cannot solve the quantum computers that are able to solve. I'll tell you what application areas are relevant to quantum computing and I'll tell you about the different forms of quantum computing that are coming in particular, distinguishing between [inaudible] because this is a new domain, I'll tell you about what quantum computing hardware looks like today, this isn't really important from the point of
view of the user any more than you need to know what kind of transistor technology Intel uses, in order to use a Intel machine. But it's probably useful to have a sense of where the science is and where the hardware looks like just to have a flavor. In particular I'll focus on IBM's super conducting qubit based hardware. Of more interest to developers, what they can do with these things, I'll show you what a quantum program looks like because they look very different from the kinds of programs that you are used to writing. I'll give you a introduction to the IBM quantum experience, which is a Web based tool that gives you the chance to start playing with a real quantum computer in the Cloud. You will see from that that it's, that is useful as a learning experience, but it's not very developer friendly.

It's a way for you to learn, but if you want to start actually writing programs in a standard editor and start building up larger circuits, you don't want to be doing that by dragging boxes around in a Web GUI. You want to be able to use your usual development environments.

We started beginning to lay the foundations for a quantum information science SDK and API, so you can start interacting with the hardware in a more standard way. For that the open source project we are calling QISKit. I'll give you a demo, and I'll finish with ideas of projects that you can work on using this technology.

The outline of the quantum computing time line is that this is technology that's been, as Marc said, some 80 years in the making. Quantum mechanics has been around since 19, before 1935, the fact that it was very different from the standard way of thinking about information, started to crystallize in the '70s. As you go through the time line all the way through to today, you can see that IBM has been at the forefront of this development of quantum information science right from the beginning, with the first conference on physics of computation in 1980 and various world firsts, including all the way up to 2016 when we made the first quantum computing platform available in the Cloud.

Why is quantum computing exciting? Quantum computing has been, as an idea has been around for a long time, it has been crystallized as a concept in the early 1980s, although many scientists were skeptical at that time. But interest and funding took off in the '90s after the discovery of Shor's factoring algorithm and fault tolerance threshold. That made it believable that quantum computing can solve problems that are important and impossible for standard classical computers. In 2017 we are on the cusp of breaking through and being on the edge of being able to do something useful with these machines.
However, to dampen down enthusiasm, despite the recent progress, there are still no rigorous proofs that a quantum computer can do anything better than a classical computer. As yet there are no fair comparisons where quantum computer feeds the classical computer and technical challenges remain for building a large scale machine that can do anything useful. No matter what the Economist Magazine says, it's not a matter of engineering. There is science to solve before we can get there. Right now people are excited because there is amazing potential. The hope is progress is picking up at the moment.

Given background, there is a famous legend which, if you follow this YouTube link, nice retelling of it, about a story where there was a, as a reward for doing good work to the emperor, the man is asked what his reward should be. He replies, I only wish for this, for 64 days I'll come back and on the first day give me one grain of rice, on the second day give me two grains of rice, so on, doubling the grains of rice for each square on the chessboard until you fill the 64 squares. The emperor agreed because that is a small reward. That will be easy to afford. But of course we know that exponential scaling builds up fast. On the first day you have one grain of rice, that is nothing. After one week it's 127 grains of rice, no big deal. After one month, we are talking a sizable stack of rice, 5,000-kilos of rice. After the full 64 days we are talking as much rice as may be the size of Mount Everest or 1,000 times the global production of rice. The point of the story is there are problems where the difficulty scale exponentially are problems that you have no hope of attacking using any method that scales exponentially.

The real problems in this class, the space of problems that we can try and solve with our computer into easy problems and hard problems, problems are the ones we can solve in polynomial time, and there are standard well-known algorithms for those, things like linear programming.

There are hard program problems where the best known algorithms are exponential. In particular with hard problems, there is a subset of the hardest problems, so called NP complete problems which are known to be as hard as any other problem in NP.

(someone coughing).

If you can solve one of these problems in polynomial time you can solve all of them but so far no such algorithm is known.

There are many important NP hard problems which are of interest for both academic reasons and real business problems. So optimizing stock or logistics running or electronic circuit design and automation, chemistry and protein folding. Lots of important problems.
Nothing we are doing with standard computers will make any dent in solving these hard and important problems.

The hope is that quantum computing comes along and it has a different set of easy problems, compared to what is hard and easy for classical computers.

In a schematic way, I'm dividing my space up as before, there is easy problems, solved in polynomial time, hard problems that can be only solved in exponential time and there is a classic problems that are easy for a quantum computer which the so-called BQP problems, bound in quantum polynomial. Problems that can be solved in polynomial resources on a quantum computer with bounded error is what this means. We know one algorithm that falls into this category. It's integer factorization problem.

It's believed that simulating quantum mechanics is easy for a quantum computer because it's based on quantum mechanics itself. It is not known whether the problem is even inside NP or it might be a harder problem than that.

It's a interesting question to know what are the problems that are in this realm where they can be easy for quantum computers [inaudible] classical. Saying anything rigorous about computational complexity is tricky. Even the classical level, it is a unproven statement that the set of problems that I've been calling easy and hard, P and NP are even different. It might be the case that they are in fact the same if you use the smart enough algorithm. Most computer scientists believe that they are different. Certainly after decades of trying nobody found any polynomial time solution to NP hard problems. It would be a major breakthrough if progress is made on this.

It's the case that Shor's factoring algorithm has no polynomial equivalent, classical equivalent. But it's not a NP complete problem. It exists in middle ground between P and NP complete. Even if somebody found a polynomial time classical factoring algorithm it wouldn't imply P equal NP. Despite much effort for real good reasons to try and find ways to factor integers sufficiently, nobody has found one.

There are no proofs that P is super set of, BQP is super set or subset of NP. But Shor's algorithm suggest that the containment is P is a subset of BQP is a subset of NP. This is interesting, because before quantum computing came along, there was something called the extended church turing piece that states all possible physical models of computation can be efficiently simulated, with polynomial overhead, efficiently emulate each other.

This means that if the problem is impossible for your X86 architecture it is just as impossible if you are using power or running on GPU or any other kind of classical computer.
This is known true for academic models of computation, turing machine, lambda calculus, random, superficial resemblance to a real desktop or laptop computer and many others.

Scott Aaronson in his PhD thesis pointed out the implication of this is at least one of the three statements must be true. Either the extended church turing thesis is wrong or fast classical factoring algorithm.

(distorted audio).

Before quantum computing came along any one of those three statements sound crazy. But at least one of them must in fact be wrong.

(coughing).

We believe and we are hoping that the one that breaks down is not two or three but the first one and that quantum computers are strictly more powerful than classical computers. That is the hope.

To point out that it's been a open question whether P equals NP since 1956. Even in 2002, when quantum computer scientists were polled 61 percent believe P is not equal to NP. 9 percent believe they were. 30 percent believe you can't answer the question. (overlapping speakers).

>> MARC-ARTHUR: I have a question for you. One of the reasons why they are saying that quantum computing will be more powerful is because of the qubit that can hold more than one state. Is that a big advantage?

>> LEV BISHOP: That's correct. I'll come in a couple of slides to where the quantum computing gets its power from. But yes, that is the idea.

A poll of researchers is not the way you answer mathematical questions. But it gives you a idea of where people think the likely power is. To try and answer Marc's question, we would like to know where does a quantum computer get its power from? The fundamental aspects of quantum mechanics is that it tells us that trying to observe a system in general -- I'll answer somebody -- the idea is that observing a system, inevitably causes disturbance to it. That is the uncertainty principle.

It's the case that two systems can exist in a entangled state which causes them to behave in ways that cannot be explained by supposing each system has a state of its own. The whole point of quantum applications is finding out how to use the two principles in a new model of computation.

The way you should think about this is a both classical computer and quantum computer you start by giving them a N bit input. We want to be able to deal with the output of the system, we want to get a N bit output. The difference is in the internal space of the computer. A classical computer as it's working on running its algorithm on its N bit input is updating
a N bit or at least a finite number of bits intermediate state that it's processing as it works on the inputs.

Conversely, a quantum computer because of the super position principle and possibility of entanglement, the state of a quantum computer requires exponentially many numbers to describe, in fact 2 to the N complex amplitudes, one for each of the possible states of a N bit register which gives the quantum computer more room for maneuvering.

This is a schematic slide, but the idea is to give a slight flavor of how this works. There are many computational paths from the initial state to the final state. As you go along the paths, as the algorithm progresses each path accumulates a complex phase such as 1 minus 1, I or some more arbitrary phase angle. If you are able to write the right kind of algorithm, at the end of the algorithm the alpha probability will be concentrated in the final state where almost all the paths arrive with the same or approximately the same phase.

This is often incorrectly summarized as saying that quantum computers get their additional power because they are able to explore all of the paths in parallel, exponentially many paths in parallel, but this is not true. It is only true for algorithms with special additional structure that at the end of propagating down all these paths, the phases line up in the right way. Shor's algorithm is such, but you can't do it for arbitrary exponential paths.

Shor's algorithm addresses the problem of integer factorization, which is the idea it's easy to take two large integers and multiply them together to find out their product, and it's hard to take the product and figure out which integers you can multiply together to make that number.

The assume hardness of the problem is the basis of RSA public key crypto and the basis of how SSL security works on the public Internet, and many other cryptographic algorithms. You can download Google SSL public key, 1024 bits, if you are able to factorize that into two integers which is just short of impossible, then you would be able to break RSA. Shor's algorithm gives you a way of taking the best classical algorithm that is known for this problem which scales exponentially and the number of bits in the key, and turn it into a mere qubit and number of bits in the key. This is the algorithm that jump starts the quantum computing. I'll come back later in the talk to how, as we propagate through the early development of quantum computing, it has a lot of parallels to the early days of classical computing and early days of classical computing, the initial interest was code breaking in World War II. Once the hardware became available, unexpected uses were found outside of crypto. That became a minor part of the application of
practical computers. We are hoping that the same is true of quantum computers.

Somebody asked what about for the discrete logarithm problem which is used for public key crypto. It turns out the Shor's algorithm with a simple modification can apply to discrete log, they are effectively the same problem.

We have at least one algorithm that says we can in principle do something useful with quantum computers but there is a problem and that is coherence. Quantum states are extremely fragile. Every quantum state has 2 to the N complex volume amplitude which may be small and they must all be preserved so at the end of the algorithm they are intersect back together and give you the right answer. This feels like analog computing, which was big in the early days of classical computing but been obsolete since the 1970s, where you get some advantages to small problems but as you scale up to larger problems, no linearity, noise and imperfections in general in your analog circuits eventually swamps any advantage. The answer to this is no, Peter Shor, the same guy behind the algorithm shows that you can do quantum error correction in order to remove any noise that you get in your circuit.

The way this works is similar to how for classical information you can add extra parity check operators and correct errors as long as they are rare enough.

In particular, there is a quantum threshold theorem, that says if you can get all the errors below threshold which is, depends on the way you are doing the encoding, but the best practical codes that are known now is 1 percent error rate, then in principle you can use error correction to push the effective error rate arbitrarily close to zero.

In practice the overhead is high unless errors are well below the threshold. But this gives justification for why we should care about this, and why noise isn't the problem in principle.

There are three types of quantum computing I'd like to bring up. One of them is the universal fault tolerant quantum computer, that takes advantage of the thresholds terms, runs below the error correction threshold, uses error correction and allows you to run useful quantum algorithms that go to the scaling regime and get exponential speed over the classical counterparts. You need to have low errors. We are not there yet. You are going to need millions of qubits. Right now we are nowhere near that in terms of the hardware we can build.

This is great, something to aim for, where you can execute Shor's algorithm and others, but in the short term, we would like to do something more modest, to go for an approximate quantum computer which doesn't go as far as, fault tolerance. We won't go to large end scaling but we will demonstrate a
useful application by taking a combination of quantum computer and interact it with classical computing system. There were ideas that seem likely to be useful in this domain, and the one that everyone talks about right now is quantum chemistry.

We estimate that we need 1,000 to 5,000 qubits to do anything useful in this area.

However, we are not even there yet. We would like to demonstrate before we get as far as quantum computer is some kind of quantum advantage, so before any useful quantum computer is built, we would like to come up with special purpose applications, whose output cannot be assimilated as fast as using existing classical computers. We believe that in the 60 to 100 qubits range with enough coherence, you can do 50 consecutive operations in a row without decohering. Then you will be able to do this demonstration, which would cement quantum computing as a viable platform.

I'd like to emphasize none of these has been demonstrated yet but we are getting close.

For approximate quantum computing, what is the reason this might have a advantage? These ideas date back to the 1930, where Nobel Prize winner Durac found underlying laws necessary for physics is completely known. The difficulty is solving them is giving you equations that are too difficult to be solvable. We need to come up with approximate solutions.

Another Nobel Prize winner in physics pointed out in 1982 that when you are doing approximation, you shouldn't use a classical approximation, because nature isn't itself classical. If you want to make a simulation of nature, it ought to be a quantum mechanical simulation. Feynman pointed out this is a wonderful problem, because it's a hard one as we have seen. To give you a flavor of how you might do a approximate quantum algorithm on a classical computer, you would use the quantum computer in the middle of the process as a way of evaluating one part of the problem.

You could use your classical computer to run a set of, do the mapping of your chemistry problem on to the content system, vary some controls, use the quantum computer to evaluate a cost function effectively, and use the classical optimizer and we have powerful classical optimization algorithms to vary the parameters and try to come up with optimal solutions to this problem, where the cost function is evaluated quantum mechanically. With that introduction of why we should think that quantum computing has anything to offer, I'll go into what hardware looks like.

David while he was at IBM pointed out that there are five criteria that you need to have a viable platform for quantum computing. You need to have a system that has, that you can set
single out from it well-defined qubits, two level systems analogous to 0s and 1s of the binary classical computer. You need to be able to prepare qubits in a defined state. You need to be able to keep them in that state without very much decoherence for a long period of time. You need to do operations on those qubits analogous to classical operations, you need to be able to measure those qubits at the end of the algorithm.

There are a few different proposed architectures for building a quantum computer. To make the analogy to the early days of classical computing, it is not clear what the ultimate large scale, scalable computer will look like. We are still at the early days. Analogous to back in the '40s and '50s where people were trying to decide is it better to use a ferrite core for memory or is it select tron a better way of storing information. The real answer is that neither and something completely different came along, static ram and dynamic ram.

Nevertheless, there are two at the moment systems architectures that are the most mature, super conducting qubits as used at IBM and ion trapped qubits. Super connecting circuits are fabricated on silicon wafers. We can leverage decades of know how. The systems need to operate at extremely low temperatures, which means this is a unconventional in terms of something you put in a laptop. But it does have the advantage that scaling qubits has obvious paths in terms of putting lots of qubits on a silicon wafer.

Electrostatic and it lays the trapping and control technology, you need high vacuum systems, scaling to many qubits is less clear, but they do have advantages because they have much better connectivity and coherence properties compared to super conducting qubits. There has been a experimental comparison between those technologies, which tries to point out that the current state-of-the-art, it's pretty equal, the state-of-the-art, super conducting and ion trap are neck and neck.

In terms of benchmarking metrics, nonfault tolerance computing hardware as we go forward, high performance computing, we use Linpack benchmarks. We are proposing quantum volume which you can follow the link to learn more about that. There are more exotic and less mature technologies that are, super conducting qubits and ion traps are the leading candidates right now. What do superconducting qubits look like? You don't need to understand this for the rest of the talk, but I'll give you a flavor.

These things are lithographically defined circuits, so they are built from resistors, capacitors, inductors, that you are
familiar with, and they are also built from one additional element that you are probably not familiar with which is called a Josephson junction which you can build only if you have superconductors which are cold metal when they get cold enough, they lose all the resistance and they exhibit superconductivity and when you have superconductors you can build Josephson junction which as a circuit element behaves like a nonlinear inductor, uses a capacitor, and this Josephson junction nonlinear inductor you can build a oscillator whose frequency depends on the amplitude of the oscillation. That frequency dependence is what allows you to address just the lowest two levels of this oscillator, and address it as a two level qubit because the frequency of the lowest two levels of the circuit compared to the next two levels is sufficiently different that you can address those specifically.

There has been a equivalent of Moore's law and exponential growth of coherence time with superconducting qubits versus time. The first demonstrations were in the late '90s, and we seem to be gaining a factor of 10 every couple of years, in terms of the T1s and T2s which are the various measurements of the coherence of qubits. We are now reaching the hundred microsecond level of coherence which is about where we need to be to get close to the fault tolerance thresholds.

Similarly, there is a similar exponential improvement not as fast, in the overall gate error, versus time. We have picked up a factor of 10 in the last five years or so, in terms of the gate error.

As you start taking these superconducting qubits and putting them together as a larger processor, we can start thinking about how we build a network of these. The nice thing is that this is all lithographically defined circuits. You can put together large numbers on the lattice. The fault tolerance we are aiming towards eventually is demonstrated on the left, where you have a square lattices of qubits that can talk to their nearest neighbors, and we have started implementing small pieces of that. We have got a 4 qubit chip here, a 8 qubit chip here, and a 2Q bit chip here. They all look similar, it's a matter of laying them out to get the topology that you want.

That is what the processor looks like at the lowest level. But these things have supporting hardware around them in order to make them work. Here is a picture of a quantum computing laboratory today. The qubits themselves live in these dilution fridge raters that hang from the ceilings, in this lab there is a couple refrigerators. Inside at the bottom, we are able to cool the system down to 10,000th of a degree above absolute zero. These racks of electronics on the side are what do all of the signal control and readout generation, in order to control
these qubits and make them do the operations and measurements that we need to do to do quantum computing. To look inside one of the dilution refrigerators, there is a series of stages where we cool first from room temperature down to 70 Kelvin and down to 4 Kelvin and down to a hundred millikelvin and at the bottom down to the 10 millikelvin. We have to be careful as you bring the signal, the refrigerator, that we symbolize the signals correctly so we don't bring in extra heat from room temperature and ruin everything. There is difficult vacuum and cryogenics technology that goes into the systems.

>> MARC-ARTHUR: Quick question again. I see that this very very elaborate setup to do the quantum machines. You are telling me that those are different than the classical computers? Or are you being very modest?

>> LEV BISHOP: The largest system that we have demonstrated so far has only five qubits. That is the one we put in the Cloud. We have a few more than that, that we are working with internally.

Because the power of the quantum computer grows exponentially, you only start to get the power once you have a few more qubits. Five qubits or ten or 20 qubits, my Intel based laptop can simulate them faster than they can execute themselves.

Once we have about 50 qubits, the world's largest super computer that governments can afford to buy might just be able to simulate it off line, it might take the government super computer some weeks to do what their computer can do in a second. Once you get to 60 qubits or 70 or a hundred, you are in the realm where the only thing that can do this is a quantum computer. We are in the limit where we are just on the steep part of the exponential, where we can still, we haven't got any real speed up but we are hoping that soon we will.

>> MARC-ARTHUR: How many qubits have you done so far at IBM?

>> LEV BISHOP: I think we have published work with 8 qubits, I believe. We are working to build that to larger systems as time goes on.

>> MARC-ARTHUR: Thanks.

>> LEV BISHOP: Yeah, 5 qubit computer we have put on line in the Cloud. We have put that on line in May, 2016. And I'll give a very brief demo of what that looks like. This is really an educational tool.

It's a website that you can go to. It has a user guide which if you go through this it gives you a very basic introduction to quantum computing, all the way up to the state-of-the-art algorithms. As you go through this, there are demonstrations that you can run live, either with quantum simulator that runs in the Cloud, or you can run this on real quantum hardware.
Here I ran a simple circuit that generated a super position of 32 states of a 5 qubits and ran as a simulated experiment. But now let's run a real algorithm on a real quantum device. I'll do a simple one. I'll first of all bring in a error gate that puts qubits into a super position of 0 and 1, put in a qubit generalization of the exo gate and put in two measurements. I'll run this on real hardware. Because this has to go out to the Cloud and there is only one quantum computer, it goes into a queue and when your turn comes up, it will run. It takes a few second before the results are available. We have to wait.

But this is going out to a machine that is inside a dilution refrigerator in a lab at IBM Yorktown Heights New York. You see that we got our results. The histogram of the results is we should be getting the two results with a large 000 and 001 and we had noise because these quantum computers are not, they do have some errors.

You can interact with the circuits, either by dragging boxes around in the classical user interface or you can write code in the editor. Both of those will give you ability to generate quantum circuits. You can look at previous circuits you have run. We have in addition to the user guide a community of quantum computing users where you can ask questions and learn more about all of this stuff.

>> MARC-ARTHUR: Can you tell us a very basic book people can get to get their arms around quantum computing or do you have that in the references?
>> LEV BISHOP: Yeah, at the end of the talk there is going to be a list of references, including textbooks on all of this stuff.
>> MARC-ARTHUR: Thanks.
>> LEV BISHOP: Yeah, so I basically gave you the demo, it gives you a interface to try out and learn about quantum computing in a hands-on way, where you can run through every circuit in the tutorial and play around with it yourself through a drag and drop interface.

That is great, but as you can see, this is a hands-on, intensive way of programming your computer, it is really different from how you, as developers are used to building software.

Recently in the last month, we have now started coming out with tools that are aimed at developers. We have come up with a Cloud quantum computing talk for developers, which we are calling QISKit, names are difficult. And basically this is our attempt at starting to build a complete open source stack for how would you run a quantum algorithm on quantum hardware.

We divide the steps that are needed to run a quantum
algorithm into several phases. On the right we have circuit execution phase. This is what is going to happen in realtime, with access to real quantum hardware in a lab somewhere. On the left side of this, this is the quantum programming that a high level developer would do rising at some high level language equivalent to C++ only targeted at quantum algorithms. We have circuit generation and doing feedback closing a loop doing feedback back to the quantum hardware. This very much matches with the kind of quantum simulation for quantum chemistry and optimization paths that I was alluding to earlier.

Some parts of this stack exist, some parts of the stack are a stated goal of where we want to be eventually. But in particular, I'd like to highlight this concept of a quantum circuit with classical control.

This is the representation of the circuits that I was showing graphically in the quantum experience demo. We have come up with a standard for writing these in a form of a quantum programming language, representation of quantum circuits which I'll discuss further, but this we are calling open QASM that is available now. On the right side we have the quantum experience basically wraps up all of the hardware. In between a developer and the quantum experience hardware in the Cloud, we have a API that you use to interact with that, that is what we are binding with Python API binding which is now open source.

Finally, on top of all this, we need tools that take and manipulate quantum circuits and build them algorithmically from scratch. This is, the beginnings of this toolbox, we have work we are calling the, building up as a Python SDK.

What is open QASM? Intermediate representation of quantum circuits, in the nonfault tolerant limit, and this is a representation of circuits that you can use for things like circuit level optimization, it's a representation that can be generated by high level tools, for example, a front end parser for high level content language or it can be consumed by hardware specific back end.

Your classical compilers, NL VM or GCC have many such intermediate representations inside them, that your various optimization and parsing passes targets and open QASM is basically our version of this targeted at its quantum circuit. It's based on various previous so called quantum assembly languages, published by many researchers in the field. But I'd like to emphasize that this assembly language concept is only an, analogy. It is better to think of it as a intermediate representation.

A single piece of open Qasm represents a basic block of a quantum program, straight line circuit with classical feedback like what I was drawing on the web GUI using the quantum
experience tool. This is focused on short term requirements. We have only included features that are likely to be present in near term intermediate representations, but we intend it to be forward compatible.

We don't anticipate breaking any existing open QASM code as we go forward. It is intended to be hardware agnostic though right now it interfaces only with the IBM quantum experience free on-line qubit system and simulators via the http RESTful API and Python wrapper.

The features of the language is simple. It supports parameterized unitary operations, I'll drill down into that. It's like the gates and hands and knots of classical computing. It gives a subroutine like mechanism for specific parameterized gates in terms of other gates. This allows us to specify different bases for quantum computing. If you for example open up your standard textbooks of quantum algorithms and they specify an algorithm in one basis and your hardware works in a different basis, you can use this subroutine mechanism to translate between them.

We provide simple quantum classical interface instructions, so the ability to measure and reset qubits using classical instructions and basic feedback between the classical computer and the quantum computer. We provide opportunities for optimization control that prevents gate reordering and combinations that might be important for various reasons I'll get into.

We provide a few miscellaneous other features, comments and file inclusion, declaration of registers and opaque gates.

In the near term we are still in the process of building up the quantum computing programming stack. We provide syntactic sugar that makes it nicer to look at as a human rather than -- it isn't relevant for machine generated open QASM, but it's nice for if you are writing it by hand or having to read it.

Unitary operations are built from two operations, one is a general single qubit operation, parameterized by 3 parameters, so called SU2. This is like a generalization of not for a single classical bit. We have CNOT which is like a quantum generalization of Xor for quantum. By putting these two together we can build any quantum circuit in the same way you build any classical circuit using only NAND gates. We provide a subroutine way of parameterizing generating new gates in terms of built in basis. I give an example of building a controls unitary in terms of, if you want to do a operation, you on the target only if the control is in the one state, then you can build out this way.

This allows you to declare a hardware basis set, allows you
to save typing, and the important thing is that it only allows you to build up a unitary operation which simplifies the analysis of open QASM programs.

We can measure a qubit into a classical bit which if quantum hardware allows, qubit available for further operations, we can reset qubit into ground state. We can do classical feedback. If a classical register has a particular value, then conditionally do a particular quantum operation. This is the most obvious area for future extensions. We will definitely revisit this as the capabilities of the hardware get better.

Finally, we provide optimization controls so this prevents circuit optimization routines from doing reordering and gate combinations, if you don't want them to. By inserting this barrier on the register R and on the qubit 0 we prevent these two operations. In the absence of this statement, this circuit is performing the identity and smart optimizer could remove these.

Right now because we don't mention qubit 1 these two operations which can be combined to go away, can be still eliminated. This is useful for things like testing how well your machine can run algorithms, one famous example of doing, of testing your hardware, is to generate a random sequence of gates that is generated such that at the end of the sequence of gates you should end up back where you started.

That is so called randomized benchmarking algorithm. By running this algorithm you see whether you ended up back where you started or not. The deviation of that is a measure of how good your gates are and how much error you have. Into this process, if you have a optimizing compiler it doesn't just look at the circuit and if there is a smart enough optimizer compiler realizes that this circuit does nothing and optimizes it away to identity. We support comments in C++ form. We declare registers. We can do file inclusion, for example standard headers.

We allow defining so called opaque gates, which is like a external declaration of a operation, where the implementation is not mentioned explicitly. There is a lot of technical uses for such a gate.

Finally, the syntactic sugar, if Q, R and S are all quantum registers of the same size and C is classical register of that size, in the obvious way we can thread over those. That saves a lot of typing. This line, CCX on Q, R and S will expand out to, if each of those is a five bit register will expand to being equivalent to cc. XQI0X0 or (indecipherable).

This saves a lot of typing.

An example of such open QASM, on the left is the code you would enter. On the right this is the circuit that this
generates. This is an example of a randomized benchmarking sequence that you might use for testing how good your gates are, with your quantum hardware. More complicated example is a adder, the primitives of the language supports don't even have such basic operations as adding two integers together but they give you the pieces you need to build that from scratch.

You can by defining several gates, there is a majority gate, unmajority gate, those are defined in terms of a gate defined in the standard header in terms of SU2 and CNOT. You can build up a adder with a 4 bit replica adder. The typing that you save by using these gates is exhibited here. If you add all the gates out explicitly as SU2 and 2 not there would be over a hundred of them. By using these gates you can save typing and make it more concise.

We have now provided a Python API, Python wrapper around the http API so you can execute these codes on the quantum experience in a easy way. It is simple. We have gave several Jupyter notebook examples that you can do it. It's a few lines, one line to set up the API, one line to submit the Java to the quantum hardware and wait for it to complete. At the end you get the results back. I'll give a quick demonstration of that right now.

You set up, before you do anything, you have to request from the website a API token that gives you access to this API and to the configuration file. Now it's a matter of importing the relevant packages. Then you can easily put the QASM program into a string and submit it, with a line of Python to the quantum experience. Now that is running on the quantum --

(audio breaking up).

I was able to do through the Web GUI. This is a job submitted by the purely Python process. We wait for the job to finish. We go to the results. We can now plot a histogram of the data. This is as I ran that previously using the GUI.

We provide several such notebooks to give examples of algorithms that you might like to run. Our goal is to start building a library of all of the standard textbook algorithms that you might come across.

One interesting one that drives home how far we have come over the last couple years is the so-called CHSH inequality which is a way of running a experiment that proves that you have some something quantum that you couldn't possibly do with classical hardware.

This is, if you have a quantum mechanical background, this is an example of a so-called Bell test. This is a way of proving the quantum mechanics is different from classical mechanics. You can run this code and it takes, you submit some hundreds of different jobs to the back end, to the API. This would be
something that would be extremely tedious to do through the Web GUI to drag boxes around so, to submit a hundred jobs. But, at the end of it, we get back a graph that looks like this. The important thing is that there is this code that you should get in red, the actual result that we got with blue points, and the threshold in blue that is the best of classical machine can do. You can see that we have exceeded this value of 2 by a substantial margin.

If you are able to do this experiment in 2009, then you would have been able to publish the same paper that was written by the UCSB group in Nature Magazine, a important paper that at this point has over 300 citations. You can see that their violation of the bound of 2 was much less than the violation we just did in two minutes through a notebook interface. This graph on the right represents several graduate students' entire thesis projects, and on the left it's something that you can run right now after the talk.

I think this is really exciting. Even before we had the API released so that you can inject this with Python, researchers all over the world have been jumping on our Cloud quantum hardware, and they have already published some 15 publications just by dragging boxes around. You can imagine how this is going to pick up now they can use Python.

What comes next? What will you do with this QISKit? There is obvious ideas, we want to build a library of textbook examples. It will be great to be able to build circuit level optimizers and rewriters. These are things that take a open QASM and rewrite it to reduce the number of operations like something like a optimizing compiler, GC3-03. Here it's worth spending efforts, even if it takes you several hours to optimize the algorithm, if it's difference between success and failure it's worth doing. There are projects along these lines where they need more work, we need debuggers, benchmarking, workarounds (indecipherable) Nonfault tolerant limit. We need new quantum algorithms. Because I'm running short of time, I'll go through this quickly. I'll point out that even in the classical domain, algorithms are very important and very hard to find. Between 1988 and 2003, a example program picked up a factor of, went from taking 82 years with the technology of the time to one minute in 2003, and 1,000 fold increase from hardware improvements, 43,000 fold was from algorithm improvements.

Even classical algorithms are hard to find. It took decades on classical computers before people figured out how to do fourier transform. Importantly many classical algorithms are hard to analyze theoretically. The only way to verify the performance is to run them on real hardware with real problems.
All of the above is likely true, for quantum computers. Now that we have those, that quantum hardware beginning to become available, now is the time to start doing that empirical investigation of algorithm performance.

There are only a couple of known quantum algorithms. You get the chance to be the guys making the fundamental contributions to a brand-new, whole new branch of computer science. You should be looking for problems that aren't just slow on your kind of hardware, it is not like a GPU where you get a ten fold speedup by moving a matrix multiply from the CPU to GPU. You want to look for problems that are impossible, that you wouldn't attempt to solve on a classical hardware.

In conclusion, it's a exciting time to be working on quantum. I'm excited to share our work beyond the restricted community of quantum computing researchers to a larger set of developers. There is a lot to do, challenges to overcome with new realms of physics, computer science, software engineering. We may get to experience the early excitement and chaos of the early days of digital computers. Thank you very much.

The last couple of slides, there is a list of references, textbooks, papers, lecture notes and the open source stack that we have. So thank you.

>> MARC-ARTHUR: Thank you very much for that very good presentation in quantum computing. We will for sure have you back, Lev, to talk more about optimal QISKit to do quantum computing and hands-on manner. But we will also like to invite you to our next tech talk, which is going to be on April 12, it's going to be a update on Brunel visualization. If you are presenting data in software, you can do that. You can take advantage of Brunel's capability to show off pattern visually and presenters will be Dan Rope and Graham Wills. It is going to be a great presentation on April 12.

Having said so, nothing left but to thank Lev for the great presentation and tell you good-bye, and we will see you on April 12 for the Brunel visualization update.

>> KATHY GHANEEI: Thank you, everyone. We will end the presentation. We will post the recording and the Q&A transcript on the link to that is in your calendar invitation. It will be on our videos page. Thanks, everyone. We will go ahead and end the meeting.

(end of call at 11:01 a.m. CST)