Quantum Programming Foundations: History and Overview

Jens Palsberg

Jan 4, 2022

Outline

Hook: Quantum computers are here! We have algorithms for them but how do we actually run a quantum algorithm on a quantum computer? To answer this question, we need a sense of what the algorithms are like, how to program them in a programming language, and how to run a program.

Purpose: Persuade you that you can learn quantum programming.

Preview:

1. We will take the time to go through the details.
2. The math is linear algebra, probabilities, and complex numbers.
3. The quantum programming languages look familiar.

Transition to Body: First let me tell you about the pace of the course.

Main Point 1: We will take the time to go through the details.
[January is about foundations and algorithms, ending with a midterm exam]
[February is about languages and error correction]
[March is about implementation and quantum advantage, ending with a final exam]

Transition to MP2: The math that we need is stuff that many other courses use as well.

Main Point 2: The math is linear algebra, probabilities, and complex numbers.
[Double-slit and probabilities as complex numbers]
[Computing with vectors of complex numbers]
[Two levels of probabilities]

Transition to MP3: How weird are the quantum languages?

Main Point 3: The quantum programming languages look familiar.
[Some of the quantum languages are Python libraries]
[Some of them are entirely new designs, yet have familiar aspects]
[The languages tend to abstract away the connections between physical qubits]

Transition to Close: So there you have it.

Review: We will go slow, we will see some math that many people learn as undergraduates, and we will learn the quantum languages.

Strong finish: We have a dream team to do the teaching effort. In addition to me, we have Auguste Hirth as the teaching assistant for the fourth time in a row; he knows this course inside out. We also have one of the veterans from the 2021 version as our reader, who will grade the homework and give you all the feedback you need to succeed.

Call to action: Get going on the first homeworks today! We are ramping up fast, both on the math and on the programming.

Detailed presentation

Hook: Quantum computers are here! Many companies are working on quantum computing, more and more researchers work on quantum computing, and in 2019 the U.S. government started the National Quantum Initiative that allocated more than $1 billion to research in quantum computing. We have algorithms for quantum computers but how do we actually run a quantum algorithm on a quantum computer? To answer this question, we need a sense of what the algorithms are like, how to program them in a programming language, and how to run a program. This course will teach all that.

Purpose: Persuade you that you can learn quantum programming.

Preview:

1. We will take the time to go through the details.
2. The math is linear algebra, probabilities, and complex numbers.
3. The quantum programming languages look familiar.

Transition to Body: First let me tell you about the pace of the course.

Main Point 1: We will take the time to go through the details.

[January is about foundations and algorithms, ending with a midterm exam]
The goal for January is to get a detailed understanding of a handful of algorithms that we will run in February. Those algorithms are short, if we count lines of code, yet also complex, if we count the brain cycles needed for a newcomer to understand them. The way we will get there is to begin with two weeks on foundations and then continue with two weeks on algorithms. In the two weeks about foundations, I will cover the math needed to understand what quantum computing is and how the algorithms work. Let me get into this a little bit already now.

Quantum computing may fundamentally change what is efficiently computable. How? The idea is to scale computation exponentially with the size of the computer. This may allow us to solve otherwise intractable problems in optimization, chemistry, and machine learning. Those three areas are indeed the most promising application areas for quantum computing. Among those three areas, the front runner is optimization. We have all heard of Shor's algorithm for factoring numbers, but actually using it to factor the numbers we use in crypto is way into the future. Much more realistic for near-term quantum computers is a quantum algorithm for finding approximate solutions to optimization problems. Those problems can be NP-complete problems like graph coloring, traveling salesman, and integer linear programming. A paper from MIT in 2014 by Farhi and his colleagues presented such an algorithm and it generated a ton of interest. DARPA has a program devoted entirely to this algorithm and it descendants. If we can use quantum computers to quickly get good approximations to large NP-complete problems, it will be a game changer. We will cover that paper later in the course and you will get to program the algorithm.

Where are we with building quantum computers? IBM released a 127-qubit quantum computer in November 2021, USTC in China has 76-qubit quantum computer, and Google has a 72-qubit quantum computer. So, computers with 100+ qubits are here and 1,000 qubits seem within reach.

How many qubits can we simulate on a classical computer? Recently, NASA simulated 70 qubits on its supercomputer, and academics have simulated 49 qubits. The key here is that the challenge doubles for every additional qubit and nobody believes we can go beyond simulation of

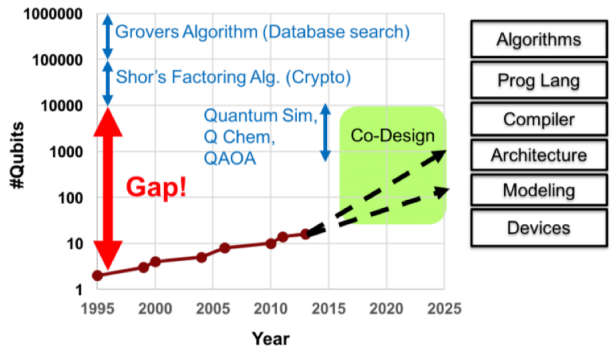
100 qubits. Hartmut Neven at Google has said that they prefer to go no higher than 37 qubits in their simulations. Any higher than that takes way too many resources and he says that they see no new phenomena above 37 qubits. I take all this as a sign that we are moving towards a quantum advantage. You will get both to use a quantum simulator and to program a quantum simulator.

Current quantum computers have high error rates, or, as John Preskill at Caltech has said: they are noisy. He called them NISQ computers, short for Noisy Intermediate-Scale Quantum computers. The errors will limit the potential until better error correction kicks in.

Let's compare two computers: one classical and one quantum.

The classical computer in our comparison is Intel Xeon Phi Processor, also known as "Knight's Corner". It is a commonly used processor in supercomputing clusters. This processor has a clock frequency of 1 GHz. In 2017, researchers measured the "soft error rate", which are radiation-induced errors that can flip states. The got the soft error rate to be 1 error per 1 thousand years.

In contrast, the IBM Q16 Rueschlikon is a quantum computer. It clocks at 5.26 GHz. The statistics are obtained on the IBM calibration data page in October 2018. Its single-qubit gate error is 2 errors for every 1,000 operations, its multi-qubit gate error 44 errors for every 1,000 operations, and its read out error is 6 errors for every 100 measurements. All of those error rates are orders of magnitude more frequent than the error rate of the Knight's Corner.



Two famous quantum algorithms are Shor's algorithm and Grover's algorithm. Shor's algorithm factors integers, which is useful for breaking crypto, while Grover's algorithm searches an unstructured database super quickly. As it happens, those algorithms require more than 10,000 qubits to do anything useful. Optimizing compilers may have a role to play here. Once we look closer at the optimization problem, it looks a lot like a design automation problem of the kind known from FPGA design and the like.

How good is a quantum computer? The key concept is the *quantum volume*, or the space-time product:

$$\text{quantum volume} \;=\; \#\text{qubits} \times \#\text{operations}$$

However, this must be tempered with the error rate of each operation. As we saw before, an operation on a couple of qubits has a much higher error rate than an operation on a single qubit. How far can we go in the near future? People are hoping for:

$$\text{near-future quantum volume} \;=\; 100 \text{ qubits} \times 1{,}000 \text{ operations}$$

The quantum computer stops working when decoherence kicks in, that is, when the quantumness goes away. Today, this happens within a second. Quantum computers today are what classical computers were in the 1950s. As an analogy that I learned from Alan Ho at Google, One of Google's current quantum computers has 72 qubits, and it just so happens that the Harvard Mark 1 computer from 1944 could store 72 numbers. By the way, one of the first programs on the Mark 1 was run by John von Neumann who was working on the Manhattan Project and needed some computation done. So, 72 qubits in 2021 and 72 numbers in 1944. Today's quantum computers are too small and too unreliable to be practical.

Each qubit doubles the size of the computer. Additionally, the reliability of each piece of the quantum computer is getting better all the time. The result is a double-exponential growth.

Let me compare with Moore's law. Moore's law says that the number of transistors on a square-inch die doubles every two years. We can translate this to say that the number of bits available for computing doubles every two years. Intuitively, if we have 2^{100} bits, we have a state space with 2^{100} dimensions. Let us contrast this with quantum computing. In quantum computing, every additional qubit doubles the number of dimensions of the state space. So, if we have 100 qubits, we have a state space with 2^{100} dimensions. Now I am waiting for the quantum equivalent of Moore's law that will talk about that the number of qubits increases by one every so often. How often? At the moment, my impression is that the number of qubits increases by one every month. This is exciting!

Alan Ho at Google talks about Neven's law, after Hartmut Neven. At Google, the reliability of gates is improving linearly over time. Neven observed that as you improve the reliability of the 2-qubit gates, you can linearly increase both the entanglement and the depth of the circuit. Because the equivalent classical computational power is exponential in both number of qubits and gate depth, this leads to a double exponential in growth. At the moment, the most reliable qubit in the world is from UCLA, from Eric Hudson's group in Physics, at 99.97 percent reliability.

What kind of algorithm is a good candidate for running on a quantum computer?

1. Small input, lots of computation, small output. Example: Shor's algorithm (input = an integer; output = two integers).

2. A result that we can verify easily. Example: Grover's algorithm (check that we found what we are looking for).

3. A subroutine for a classical computation. Example: Simon's algorithm (the quantum computer outputs equations, while the classical computer solves the equations).

We are going to spend January on the foundations and on some of the algorithms of quantum computing. On Bruinlearn you will find three useful links to material for January.

The first link is to Jack Hidary's book, which is good for three reasons. First, it gives a brief and accessible overview of quantum computing, both hardware and software. Second, it has a wonderful second half of the book on the math tool kit that we all need for learning quantum computing. Please take a look at whether you know all this and fill in any gaps that you discover. Third, the book uses Cirq for its program examples. Cirq is Google's quantum language and we will use it in this course.

The second link is to John Watrous lecture notes. John Watrous writes super clearly and is great at explaining proofs of quantum algorithms. Likely I have learned more quantum computing from reading John Watrous than from reading any other source.

The third link is to the quantum circuit simulator Quirk that Craig Gidney at Google wrote. This simulator makes it easy to get going on executing quantum programs and it plays a big role in homeworks in January.

Bruinlearn has a ton of homework for January. Some of the homework is on the foundations, which means math that boils down to mostly linear algebra and Boolean functions. I give you this homework for two reasons. One is to get you to page in some linear algebra that you already know and to perhaps learn some more linear algebra. The other reason is to think through some points that are directly relevant to the algorithms that we will cover later in January.

Some other homework in January is programming homework. Some of it is about running quantum circuits, to try out some ideas in the simplest possible setting. Some other programming homework is about getting into the details of the problems that the quantum algorithms can solve. You will do that by programming solutions on classical computers. This means that once we get to the quantum algorithms and the problems they solve, you have already solved each of them on a classical computer. You can use any language you like for those homeworks.

Overall, January has homework due twice a week, for the purpose of getting you ready for programming quantum computers in February. You will do the homework in January individually, and we will have a midterm exam in early February.

We will also have ten online quizzes throughout the course. I give quizzes for three reasons.

The first reason is to communicate some points in a lecture that I think are essential. The second reason is to have a different way to learn. When I took an MBA in UCLA Anderson some years ago, some of the professors gave us online quizzes, which were enjoyable. The third reason is that I see quizzes as part of your participation grade. You can all get 100 percent on the quizzes by taking them again and again so in a way I am not grading you on right or wrong. But I am grading your participation.

[February is about languages and error correction]

Every company that is building quantum computers also has its own quantum programming language. Google has Cirq, IBM has Qiskit, Microsoft has Q#, and Rigetti has PyQuil. In addition to languages from companies, we also have languages from universities, another handful of them. Until recently, each of the companies had built approximately one quantum computer, so we could say that in the quantum world, we had one language per computer. No standardization on that front! The goal should be the same as for classical computing: a language should run on many different computers. Write once, run everywhere. We are slowly moving in this direction. For example, Amazon Braket is working on the "connect everything" problem, and Cambridge Quantum Computing is working on "compile to any target" problem.

Which language will emerge as the winner? This is way to early to say. My guess is: none of them; people will design better languages. But we should try to grasp what language designers care about, how language designers envision language support for quantum algorithms, and how the ideas compare. We will get into all that in February by programming and running the algorithms that we learned in January. We will run on IBM's quantum computer and I am trying to get us access to IonQ's quantum computer.

The homework in February will cover three quantum languages. We will do the programming in Cirq, which is Google's quantum language. The Cirq compiler compiles Cirq programs to QASM, which is a widely used exchange format for quantum programs. Now we can import the QASM program into Qiskit, which is IBM's quantum language. Now we are ready to run on IBM's quantum computer.

In February and March you will work in groups. The maximum group size is three. We have 60+ people in the course so if every group has three people, we will have 20+ groups. Each group will submit quantum programs in February and March. You will form the groups, the deadline for group formation is end of January. Submit the group member names and the group name.

[March is about implementation and quantum advantage, ending with a final exam]
In March, I will talk about how to implement quantum languages, both via interpreters and via compilers. I will also give a lecture on where we are with getting a quantum advantage. I will talk about research questions to give you a sense of my perspective on where the field is going, which obstacles have to be overcome, and how long it will take to get there. Some of you may be interested in getting involved in research on quantum computing yourself; if so, please come and talk with me.

The course will have a final exam.

My plan for grading is straightforward. This is a graduate course and I would much prefer to give only A, B, C grades. If I sense that you really tried, even if you have lots of stuff wrong, I will give you either an A or a B, I hope that this will be true for everybody. My plan is to limit the A's to 45 percent, so my ideal case is that 45 percent of you gets an A and that 55 percent gets a B. However, I reserve the right to give a C or even below if I see little evidence of effort.

Let me make a shoutout to the undergraduate students in the course. I am delighted that you are here and encourage you to speak up. I know that you are outnumbered by the graduate students, yet please contribute to the discussions as much as you can.

Surely you have noticed that I have no slides. This is how I am going to roll the entire quarter. I am going to use the blackboard. The reason is that I want to force myself to slow down. Hopefully the slow pace will give you time to think and to ask questions. I hope you will ask a ton of questions.

We will use piazza in the course and I am going to give ten percent of your grade based on participation on piazza. Some examples of participation: ask a question, answer a question, and send a link to a newspaper article and comment on it. Some other examples could be to list a typo in my lecture notes and suggest a fix, show a work-through of a new example, and link to a research paper and comment on it. Plan on posting at least ten times on piazza under your own name this quarter. The goal is to build a community around quantum computing at UCLA.

Transition to MP2: The math that we need is stuff that many other courses use as well.

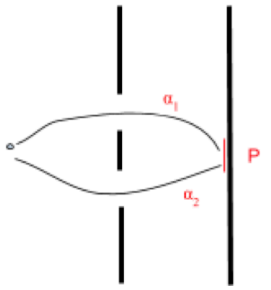
Main Point 2: The math is linear algebra, probabilities, and complex numbers.

[Double-slit and probabilities as complex numbers]
What is a qubit? Answer: it is a vector of two complex numbers:

$$\begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

If we have 72 qubits, then we are working with 2^{72} complex numbers. This exponential is the key to the power of quantum computing. If one day we have 300 qubits, we will be computing with more qubits than the number of atoms in the known universe. That day is likely to come within a few years.

Why are all the numbers complex? This goes to the heart of quantum mechanics. Feynman said that the essence of quantum mechanics is the Double Slit Experiment. This experiment goes back to Thomas Young in 1801.



The experiment works with of a source of light, a wall with two slits, and a back wall. In the experiment, we shoot photons toward the wall with the two slits. Where each photon lands on a second wall is probabilistic. If we plot where photons appear on the back wall, some places are highly likely, some unlikely.

So far this is straightforward: we might justify this behavior by a theory where each photon has a degree of freedom that determines which way it goes. What is weird is the following. For some interval on the second wall, let us define three probabilities:

- let P be the probability that the photon lands in the interval with both slits open;
- let P_1 be the probability that the photon lands in the interval if only slit 1 is open; and
- let P_2 be the probability that the photon lands in the interval if only slit 2 is open.

We would think that $P = P_1 + P_2$. But experiments show that this is false! Even places that are never hit when both slits are open, can sometimes be hit if only one slit is open.

Slogan: God plays dice, but they aren't normal dice.

The way to make this work is to change probabilities from real numbers to complex numbers. The people in quantum mechanics use the word *amplitudes* instead of probabilities.

The central idea of quantum mechanics is that to fully describe a system, you need a probability (an amplitude) for each possible configuration.

The Born rule: if the amplitude of a configuration is α, then the probability P of seeing that configuration as an outcome is

$$P \quad = \quad |\alpha|^2$$

Now we can redo the Double Slit Experiment with amplitudes instead of probabilities.

- let α be the amplitude that the photon lands in the interval with both slits open;
- let α_1 be the amplitude that the photon lands in the interval if only slit 1 is open; and
- let α_2 be the amplitude that the photon lands in the interval if only slit 2 is open.

We do get $\alpha = \alpha_1 + \alpha_2$. Example: suppose $\alpha_1 = \frac{1}{\sqrt{2}}$ and $\alpha_2 = -\frac{1}{\sqrt{2}}$. By the Born rule, the probability that the photon lands in the interval with one bit slit open is $\alpha_1^2 = \alpha_2^2 = \frac{1}{2}$. But if both slits are open

$$P \quad = \quad |\alpha_1 + \alpha_2|^2 \quad = \quad |\frac{1}{\sqrt{2}} + (-\frac{1}{\sqrt{2}})|^2 \quad = \quad 0$$

So, in quantum mechanics, probabilities (amplitudes!) can cancel each other out.

Now let us ask "how does a particle that went through the first slit know that the other slit is open?" In quantum mechanics, this question is ill-formed. Particles don't have trajectories, but rather take all paths simultaneously, in superposition. This is where we get the power of quantum computing.

[Computing with vectors of complex numbers]
Here is a way to sum up a central idea of quantum mechanics. Question: what if probabilities could be negative? Answer: we get quantum computing. Those complex numbers from quantum mechanics are the building blocks. Classical computing is about computing with bits, while quantum computing is about computing with complex numbers. Just like the state of a classical computer is given by a vector of bits, the state of a quantum computer is given by a vector of complex numbers. What can we do with those complex numbers? That is the topic of this course.

[Two levels of probabilities]
Quantum computing has two levels of probabilities. Probabilities on top of probabilities. At the bottom level we have probabilities from quantum mechanics. At the top level we have probabilities of the kind that we find in statements like: this algorithm gives the correct answer with high probability. The essence of quantum programming is to connect those levels. The top layer turns out to be what quantum algorithms are good at: finding approximate answers really fast. In contrast, quantum computers cannot get exact answers to hard problems much faster than a classical computer. For example, a quantum computer cannot find exact solutions to NP-complete problems in polynomial time. However, a quantum computer may be much better than a classical computer at finding approximate solutions to NP-complete problems.

Transition to MP3: How weird are the quantum languages?

Main Point 3: The quantum programming languages look familiar.

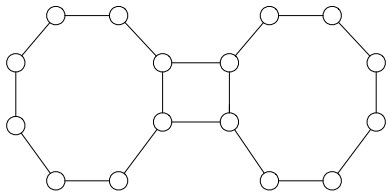
[Some of the quantum languages are Python libraries]
Cirq from Google and PyQuil from Rigetti are both Python libraries. This means that we can write Python code that runs on a classical computer and makes calls to a quantum computer. This is convenient because a lot of the code we need to write is not quantum at all. Rather, much of the code is for set up, post-processing, and input-output, which all can be written in Python. So, Python is the host language and all the quantum stuff is done by calling a quantum library.

[Some of them are entirely new designs, yet have familiar aspects]
Q# is a domain-specific language that was designed entirely for the purpose of expressing quantum algorithms. Program execution proceeds in a manner similar to the Python libraries: a classical computer controls the computation and makes calls to a quantum computer. However, in Q# the boundary is more fluid and up to the compiler.

[The languages tend to abstract away the connections between physical qubits]
For classical computers, while many companies make them, really they are all the same in one important respect: they are all based on CMOS. For quantum computers, this is not at all the case. Different companies build qubits in radically different ways. For example, IBM, Google, and Intel build superconducting qubits, while IonQ builds ion-trapped qubits. This affects quantum computing in particularly one important way: what if we want to do an operation on two qubits;

are they connected? On superconducting quantum computers, any operation on two qubits requires that they are connected. On ion-trapped quantum computers, an operation on two qubits can work on any two qubits.

Rigetti has a superconducting quantum computer with 16 qubits that are arranged as two octagons with two connections that cut across.



Quantum languages tend to abstract away the connections between physical qubits. The idea is "write once, run everywhere". of writing efficient programs. The compiler plays a key role in making that work. The problem of mapping the qubits in a program to the qubits on a specific quantum computer is akin to problems in design automation. Specifically, the mapping problem for quantum computing reminds of the mapping problems in FPGA design and ASIC design.

Transition to Close: So there you have it.

Review: We will go slow, we will see some math that many people learn as undergraduates, and we will learn the quantum languages.

Strong finish: We have a dream team to do the teaching effort. In addition to me, we have Auguste Hirth as the teaching assistant for the fourth time in a row; he knows this course inside out. We also have one of the veterans from the 2021 version as our reader, who will grade the homework and give you all the feedback you need to succeed.

Let me introduce the Quantum Computing Drinking Game. In the company of friends and fellow students, you fire up a youtube video on quantum computing or you read lecture notes. Now you listen and watch for the following quotes.

> Albert Einstein: "God does not play dice." (1926)
> Richard Feynman: "Nature isn't classical, dammit." (1981)
> Everybody: "$\frac{1}{\sqrt{2}}$" (every year)

Every time you hear or see one the quotes, you take a sip.

This course is the first in a series of two courses on quantum computing. The second course will be in Spring and my hidden agenda is to make as many of you as possible interested in taking the second course.

Call to action: Get going on the first homeworks today! We are ramping up fast, both on the math and on the programming.