;login:

Quantum Computing Notes:

Why Is It Always Ten Years Away?

January 13, 2025

Authors: Konstantin V. Shvachko

Article shepherded by: Rik Farrow

Busy, busy, busy. … It's what we Bokononists say, …
when we feel that a lot of mysterious things are going on.
Kurt Vonnegut

Introduction

Why do they always say that Quantum Computing is ten years away? I first heard this prognosis in the late nineties when the fundamental Shor's algorithm was developed, and the first physical qubit was tested. A lot has changed in the field since then, but the ten-year horizon for practical Quantum Computing keeps sliding with its evolution.

Quantum computers promise execution of tasks beyond the capability of classical computers. Contemporary classical computer chips have already reached levels of density where quantum effects occur. Quantum computers should be a natural next step in miniaturization of chips where quantum effects are embraced rather than prevented.

The goal of this article is to understand why Quantum Computing is hard, what are its potential advantages, challenges, and boundaries. It reviews quantum computing via a prism of computer science and software engineering. In the end, as with traditional programming software engineers do not think about physical representation of bits, properties of transistors and integrated circuits, or Boolean gates. So as with Quantum Computing programmers should have a high enough level of abstraction to focus on computation rather than effects of quantum physics, principles of qubit implementation or even quantum gates.

Historical Notes

Quantum computing started in the early 1980s with the founding ideas of

- Paul Benioff, who in 1980 constructed a quantum mechanical model describing the computational process of Turing machines, which set the theoretical foundation of Quantum Computing [1],[2].
- Yuri Manin who recognized in 1980 that quantum states possess much larger capacity than classical and therefore a single quantum automaton can represent states of multiple classical automata simultaneously [3].
- Richard Feynman who in his influential keynote lecture “Simulating Physics with Computers” [4] in May 1981 stated that classic computers are inadequate to describe physical systems governed by the laws of quantum mechanics and that an exponentially larger computer is needed for the task – a quantum computer. He outlined the basis of the Quantum Computational model.

Notable achievements in Quantum Computing include:

- Early results in quantum computational complexity. Deutsch-Jozsa algorithm (1992) showed that quantum algorithms can be exponentially faster than any classical algorithm [6]. This was an important separation result, but is of little practical use since the problem it solves is specifically designed for the benefit of quantum computation.
- In 1994 Peter Shor published an integer factorization algorithm [7],[8] known as Shor's algorithm. It showed that a quantum computer can find prime factors of an integer in polynomial-time. The practical importance of this algorithm is due to the fact that modern public-key cryptography heavily relies on the fact that the problem of integer factorization is exponentially hard, making decryption of a cipher without a private key impractical. Even though Shor's algorithm cannot be used today, since there aren't enough physical qubits manufactured so far, it possesses a real threat in the future due to the surveillance strategy known as Store-Now-Decrypt-Later. Shor's algorithm opened a new research branch of post-quantum cryptography, which designs alternative cryptographic schemes not relying on prime factorization.
- Classic Fourier Transform is widely used in different areas of science. Don Coppersmith in 1994 developed Quantum Fourier Transform [10], which calculates Fourier transform of a quantum state in poly-logarithmic time – exponentially faster than classic algorithms.
- Quantum error-correction was developed by Peter Shor in 1995 [9].

- In 1995 Christopher Monroe and David Wineland following the Cirac-Zoller proposal built a physical system of two qubits implemented with trapped ions and demonstrated the operation of quantum logical gates on them including two-qubit CNOT gate [11].
- Lov Grover in 1996 developed a quantum algorithm known as Grover database search algorithm [12]. This algorithm allows polynomial speedup of NP-complete problems. The speed up is not as drastic as Shor's, but it has a wider application area.
- First physical implementations of two quantum algorithms were demonstrated in 1998 on a 2-qubit nuclear magnetic resonance (NMR) quantum computer.
    - Jonathan Jones and Michele Mosca implemented Deutsch's algorithm [13].
    - Isaac Chuang, Neil Gershenfeld, and Mark Kubinec demonstrated Grover's search algorithm [14].
- Michael Nielsen and Isaac Chuang published a prominent textbook (2000) on quantum computation and information [5].
- First experimental realization of Shor's algorithm was done in 2001. Number 15 was factored on a 7-qubit NMR quantum computer [15].
- The Harrow–Hassidim–Lloyd algorithm or HHL algorithm (2009) is a quantum algorithm for numerically solving a system of linear equations – one of the key problems of linear algebra [16]. Under specific restrictions the quantum algorithm solves the system in poly-log time – an exponential speedup over classic algorithms with the same restrictions. The HHL algorithm found applications in quantum machine learning [17].

Quantum Circuit Computational Model

In classical computation a bit represents a basic unit of information. A bit can be either 0 or 1. Quantum computers operate on qubits (quantum bits) [5],[18].

Qubits

Qubit states are composed of two base logical states denoted $|0\rangle$ and $|1\rangle$. A qubit state is a linear combination or a *superposition* of the base states: $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where α, β are complex numbers, called *amplitudes*, such that $|\alpha|^2 + |\beta|^2 = 1$. Geometrically, quantum states are represented as points on the surface of a unit 3D sphere known as the *Bloch Sphere* using spherical coordinates.

A qubit state can have infinitely many values compared to a binary classical bit. But due to quantum mechanics principles one cannot determine its quantum state at any given moment since measurement destroys quantum state, which *collapses* into a base state $|0\rangle$ or $|1\rangle$ with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. Note that measurement of a qubit state is probabilistic, while for a classical bit you get the same value whenever you check it.

For a system with multiple qubits the number of base states increases exponentially. For example, for a 2-qubit system there are 4 base states $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$ and the system state is a superposition of the base states:

$$|\psi\rangle = \alpha_0|00\rangle + \alpha_1|01\rangle + \alpha_2|10\rangle + \alpha_3|11\rangle$$

Here the squared amplitudes $|\alpha_x|^2$ represent probabilities of the system to be in the respective states, and the sum of the probabilities equals to one:

$$|\alpha_0|^2 + |\alpha_1|^2 + |\alpha_2|^2 + |\alpha_3|^2 = 1$$

In general, in a system of n qubits the number of amplitudes $\{\alpha_x\}$ describing the state is 2^n. This number grows very fast. For n=100 the number 2^{100} of complex-number coefficients exceeds many times the size of today's Internet. Emulating a quantum computation with such a large amount of data using classical computers would be infeasible.

Quantum Gates

Even though the qubit state cannot be known precisely, the state can be modified using quantum operators. There are different ways of describing quantum operations. The most traditional approach as of today uses *quantum gates*. This is analogous to classical logic gates such as NOT, AND, OR, which can be combined to define an arbitrary Boolean function.

As mentioned earlier quantum states can be viewed as points on a 3D unit sphere – the *Bloch Sphere*. Then 1-qubit gates represent different rotations on the sphere. For example, X, Y, and Z gates known as Pauli gates define 180° rotations of the state on the sphere around the corresponding axes. If the qubit state is $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, then

$$X|\psi\rangle = \beta|0\rangle + \alpha|1\rangle$$
$$Y|\psi\rangle = -i\beta|0\rangle + i\alpha|1\rangle$$
$$Z|\psi\rangle = \alpha|0\rangle - \beta|1\rangle$$

More complex rotations are presented by widely used Hadamard gate H and phase gate P:

$$H|\psi\rangle = \frac{\alpha+\beta}{\sqrt{2}}|0\rangle + \frac{\alpha-\beta}{\sqrt{2}}|1\rangle$$

$P|\psi\rangle = \alpha|0\rangle + e^{i\varphi}\beta|1\rangle$, where an angle $\varphi \in [0, 2\pi]$

An example of a 2-qubit gate is the controlled-NOT or CNOT gate. It transforms a 2-qubit state by swapping the last two coefficients. If

$$|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

then

$$\text{CNOT}|\psi\rangle = \alpha|00\rangle + \beta|01\rangle + \delta|10\rangle + \gamma|11\rangle$$

A composition of quantum gates forms a quantum circuit, which is represented as a directed acyclic graph. As traditional Boolean circuits, quantum circuits define quantum computations.

Quantum computing as quantum mechanics itself is alternatively expressed in the linear algebra language of vectors, matrices and operations on them. In linear algebra notation quantum states are described as vectors of amplitudes

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \quad \alpha|0\rangle + \beta|1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix}$$

and quantum operations are represented as unitary matrices. The unitary constraint guarantees that quantum operators produce valid quantum states.

Universal Quantum Gates

For classic logic gates one can select a finite set of gates called universal, e.g., {AND, OR, NOT}, composition of which allows defining any Boolean function. The number of quantum gates is infinite, in fact uncountable. In the quantum case a finite set of quantum gates is *universal* if any quantum operation can be approximated with arbitrary accuracy by a quantum circuit composed of the gates from this set. For example, the set of 1-qubit gates listed earlier plus the CNOT gate is universal.

Entangled Qubits

Entanglement is an intrinsic phenomenon of quantum physics and one of the key features of quantum computing. In quantum computing qubits are *entangled* if their states are correlated. That is, the states depend on each other so that they cannot be changed independently. Rather the entire entangled system evolves as a whole. Since the states of the entangled qubits are correlated so are the results of measurements. Once one qubit randomly collapses into a certain value the other qubits collapse as well into values deterministically dependent on the former.

Unentangled states are called *separable*. Mathematically it means they can be represented as a product of individual qubit states. Let's consider a two-qubit system and assume that the qubits are in the following states, respectively

$$|\psi_1\rangle = H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$|\psi_2\rangle = |0\rangle$$

Then the combined state of the system is the product of the two states and is therefore separable:

$$|\psi_1\rangle\ |\psi_2\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\ |0\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle)$$

Multi-qubit gates are used to entangle qubits. For example, if we apply CNOT gate to the above state:

$$\text{CNOT}\ \frac{1}{\sqrt{2}}(|00\rangle + |10\rangle) = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$$

the result is entangled, since it cannot be decomposed into a product of individual qubit states.

The latter state is known as one the *Bell's states*. An intrinsic property of this Bell's state is that when one of the qubits is measured then both qubits collapse and into the same value, which is either $|0\rangle$ or $|1\rangle$ with probability ½. For another entangled Bell's state $\frac{1}{\sqrt{2}}(|01\rangle + |10\rangle)$ the qubits collapse into opposite states. In both cases the results of measurement are correlated.

Quantum Computation

Conceptually the quantum computation process is similar to any computational model. It consists of the following steps

1. *Input*: prepare qubits initial states. It suffices to initialize qubits into the same state first, say $|0\rangle$, and then transform each into a desired state by applying 1-qubit gates.
2. *Compute*: apply a quantum circuit to the qubit system. The circuit is designed to solve the target problem in the first place. Due to the probabilistic nature of the model, it should also intend to maximize the probability of the correct answer when measurements are performed.
3. *Output*: measure the states of the qubits. This yields a classical result, which could be passed to classic devices for further processing.
4. *Error check*: the obtained results are correct with a certain probability $p > ½$, which is sufficient in e.g., statistical analysis. If precise computation is required then the obtained result should be verified and if incorrect the quantum computation should be repeated. In practice only a constant repetition of quantum runs is needed.

The principles of the quantum computational model differ from the classical. They are different in many aspects including algorithmic and programmatic. From an algorithmic perspective, a series of new algorithms need to be invented, since quantum algorithms are based on different principles and can be more powerful than the classical. From a programming viewpoint, new high-level programming languages should be developed. The traditional operators like assignments or condition checking do not have direct equivalents in the quantum world. Quantum principles do not allow duplicating a quantum state as in assignment. And `if` conditions imply measurements, which destroy the state being measured.

Quantum Algorithms

The main advantage of quantum algorithms is that they provide computation speedup compared to classical counterparts. Several quantum algorithms were developed to demonstrate the advantage.

Deutsch–Jozsa Algorithm

Deutsch–Jozsa algorithm [6] is one of the first algorithms that showed the high potential of Quantum Computing.

> *Consider a Boolean function $f:\{0,1\}^n \rightarrow \{0,1\}$, which is either constant on all 2^n inputs or is balanced. Balanced here means that f(x) = 0 on exactly half of the inputs and equals 1 on the other half. Determine if a given f is constant or balanced.*

With a classical deterministic algorithm function f must be evaluated at least $2^{n-1} + 1$ times. While a quantum algorithm needs only one evaluation of f. The quantum algorithm is based on the technique known as *quantum parallelism*, which allows computing f(x) for all input values x simultaneously.

The Deutsch–Jozsa algorithm shows exponential speedup of quantum computation compared to classical deterministic algorithms. A generalization of this algorithm called Simon's algorithm provides exponential speedup compared to classical probabilistic computers as well. Both algorithms, while theoretically important, have little if any practical application.

Quantum Fourier Transform

Traditional *Discrete Fourier Transform* (DFT) is a linear transformation of a sequence of complex numbers $\{x_k\}$ of length N to another sequence $\{y_k\}$ of the same length. DFT has a lot of applications in different scientific areas. Algorithmic applications include fast multiplication of matrices, large integers, polynomials.

Quantum Fourier Transform (QFT) is applied to an n-qubit state and transforms its amplitudes. Given the state $|x\rangle = \sum_{k=0}^{N-1} x_k |k\rangle$ with $N = 2^n$ QFT transforms it to the state $|y\rangle = \sum_{k=0}^{N-1} y_k |k\rangle$ where amplitudes y_k are calculated as

$$y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j w_N^{kj} \text{ for } 0 \le k < N \text{ and } w_N = e^{i2\pi/N}$$

The best classical algorithm, known as *Fast Fourier Transform* (FFT), calculates DFT in time O(N log(N)). The quantum algorithm discovered by Don Coppersmith (1994) does it with $O(\log^2(N))$ quantum gates, which constitutes an exponential speedup [10]. The algorithm also exploits quantum parallelism as the Deutsch–Jozsa algorithm. Thus, the latter can be considered as the predecessor of QFT, and QFT is the key building block of Shor's algorithm.

Prime Factorization

Prime factorization is the problem of decomposing an integer into a product of prime numbers. The problem is believed to be hard to solve for classic computational models. There is no known deterministic classic algorithm, which solves it in polynomial time, and the best known algorithms solve it in exponential time. The computational hardness of factorization made it a principal component of public-key cryptography, where encryption is performed using a public key known to anybody, but decryption is practically impossible without the private key kept in secret. "Practically impossible" here means that decryption without the private key will take millions of years and an enormous amount of compute resources.

Shor's Algorithm

In 1994 Peter Shor developed a quantum algorithm, which factors an integer with a polynomial runtime upper bound of $O(n^3)$ where n is the bit length of the number being factored [7],[8]. It is an exponential speedup compared to complexity $O(c^n)$ of known classic deterministic algorithms.

The efficiency of Shor's algorithm raised alarm for cryptography and cryptocurrency. It spurred the development of new intractable (that is, hard to solve efficiently) problems and cryptographic standards. A new branch of cryptography was spawned known as *post-quantum cryptography*.

Shor's factoring algorithm is a complex construct combining classical and quantum computation steps. Each step could require an entire article to fully explain it.

It is also hard to implement. In practice Shor's algorithm was used to factor numbers 15 and 21 with a handful of qubits. Factorization of larger numbers requires more qubits. Latest estimates [19] show that factoring of 2048-bit RSA integers would take only 8 hours, but will require 20 million qubits. The World has not produced anywhere near that many qubits yet. In 2023 Atom Computing and IBM announced the first quantum computers with over 1000 qubits. It could be a while until people will be able to actually break meaningful ciphers.

Grover's Search Algorithm

Grover's Algorithm [12] solves the following problem:

> *Given a Boolean function $f:\{0,1\}^n \rightarrow \{0,1\}$, such that there is only one argument x for which f(x) = 1, find that argument.*

This problem becomes a database search when arguments are treated as indexes in a database table and function f is a search criterion. The classic algorithm requires exhaustive search in the worst case to solve the problem and therefore has O(N) time complexity, where $N = 2^n$. A quantum algorithm proposed by Lov Grover in 1996 can obtain the solution with probability p > ½ in time $O\left(\sqrt{N}\right)$. The algorithm can be run multiple times in order to increase the probability of the correct answer. On average it suffices to run it twice to obtain the correct result, so the complexity remains the same.

Grover's algorithm is still exponential but provides a polynomial speedup compared to the classical algorithm. It has a wide area of applications, since it can accelerate NP-complete problems.

Quantum Algorithms for Linear Algebra

Linear algebra studies linear operations on vector spaces. As mentioned earlier, quantum states and computation can be expressed in linear algebra terms of vectors and matrices, where quantum states are vectors of amplitudes and quantum operations are unitary matrices.

Intuitively, associating classic vectors with amplitudes of quantum states should convert a linear algebra problem into a quantum one. Such association provides an exponentially more compact representation of data, since a system of n qubits encodes 2^n amplitudes. This also prompts more efficient algorithms as a polynomial classic algorithm may translate into a quantum poly-logarithmic one.

Solving linear systems of equations is one of the common problems of linear algebra:

*Given a N*N matrix A and a vector b find vector x such that Ax = b*

A classic solution of this problem has lower bound $\Omega(N^2)$ and the best algorithm solves it in time $O(N^{2.376})$.

Aram Harrow, Avinatan Hassidim, and Seth Lloyd in 2009 developed a quantum algorithm for this problem – the HHL algorithm [16], which solves the problem in $O(\log N\, k^2 / \varepsilon)$, where k is the condition number defined as the ratio of the largest and smallest eigenvalues of matrix A, and ε is the error parameter. The time bound holds under certain restrictions: the matrix A should be sparse, and if the condition number k is too large or the error parameter ε is too small the estimate degrades to linear O(N). The HHL algorithm calculates the solution x as amplitudes of a quantum state, which cannot be measured exactly. So instead of the exact value of x it produces a value of an operator on x such as $x^\dagger Mx$ for some matrix M.

The HHL algorithm and its modifications found implementation in machine learning where training is reduced to solving linear systems of equations. Other applications include chemistry, and finance. *Quantum machine learning* [17] is a new scientific field. Unlike classical machine learning it still remains purely theoretical and any significant practical results are yet to be demonstrated.

Quantum Turing Machines

Another way to define a computational model is *Turing Machines* (TM) introduced by Alan Turing in 1936. TM is a mathematical abstraction of a computational device. The simplicity of TMs makes them ideal to study theoretical computational problems. In computational complexity theory TMs are used to compare different complexity classes and computational models as shown in the next section.

A TM consists of

- an input-output tape,
- a head that can read from and write to the tape and moves along it in either direction one cell at a time,
- an internal state that is modified according to a finite state transition table based on the current state and the observed symbol on the tape.
- Some states are marked as final indicating that the machine must stop.

The goal of a TM computation implementing a Boolean function is to accept or reject the input sequence initially written on the tape.

There are many different variants of TMs. Four types or TMs considered here have different ways of defining their state transition tables.

- In *deterministic Turing machines* the state transition is a 1-1 mapping and is always deterministic.
- *Nondeterministic Turing machines* can have multiple choices to choose the next state. The machine accepts the input sequence if at least one of the series of choices accepts the input.
- *Probabilistic Turing machines* (PTM) also have multiple choices for state transitions, but they choose the next step probabilistically. Probabilistic TMs produce correct results with a certain probability. The goal is to maximize that probability, otherwise computation is no better than tossing a coin.
- *Quantum Turing machines* (QTM) are similar to PTM, but the state transition is defined with unitary operators on a quantum state using amplitudes instead of probabilities.

The QTM was first defined by Paul Benioff in 1980 [1],[2]. Here I present a simplified description of QTMs. For in-depth details see the Bernstein and Vazirani paper [20].

Let us consider a probabilistic TM first. The state transition can be viewed as a function P:

$$P(a, q, b, r, m) \rightarrow p \in [0,1]$$

where a – is the symbol PTM currently observes on the tape, q – is the machine's current state, b – is the symbol it writes to the tape, r – is the new state the PTM transitions to, and $m \in \{-1, +1\}$ defines whether the head moves left or right on the tape. The result of the function p – is the probability of the transition. So, if there are two possible transitions from the current configuration (a,q) then

$$P(a, q, b_0, r_0, m_0) + P(a, q, b_1, r_1, m_1) = 1$$

For QTMs a similar function on state transitions is defined, but it maps transitions into complex numbers:

$$A(a, q, b, r, m) \rightarrow \alpha \in \mathbb{C}$$

Then transitions are viewed as base quantum states $|a,q,b,r,m\rangle$, the values of function A are amplitudes, and the QTM's quantum state is defined as a superposition:

$$|\psi\rangle = \alpha|a,q,b_0,r_0,m_0\rangle + \beta|a,q,b_1,r_1,m_1\rangle, \text{ where } |\alpha|^2 + |\beta|^2 = 1$$

The QTM starts with an initial state $|\psi_0\rangle$ and applies a unitary operator U on each step to its quantum state. U defines the computation of the machine

$$|\psi_n\rangle = U^n|\psi_0\rangle$$

The unitary restriction on the operator U guarantees that the resulting state $U|\psi\rangle$ remains quantum.

Quantum circuits and QTMs are different quantum computation models. They are equivalent in polynomial time [21] meaning that an algorithm expressed in one model can be simulated using another model in polynomial time.

Complexity Classes

Theoretically, algorithms are classified using computation complexity classes. A complexity class asymptotically restricts the amount of compute resources: time or space, that can be used to solve a problem within a certain computational model. Thus, a complexity class combines all problems that can be solved by an algorithm with these restrictions. A computational model is customarily represented by a variant of Turing machines.

Fundamental complexity classes known in classic computing are:

- **P** – polynomial time: class of problems that can be computed in polynomial time on deterministic Turing machines
- **NP** – nondeterministic polynomial time: class of problems computable in polynomial time on nondeterministic Turing machines

- **PSPACE** – polynomial space: class of problems computable with polynomial space on TMs. It is known that **PSPACE** = **NPSPACE**, so deterministic and nondeterministic space classes are indistinguishable.

The relationship between the complexity classes is as follows:

$$\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE}$$

It is not known if any of the relations are strict. That is, if **P** is strictly smaller than **NP** or if any of the two is strictly smaller than **PSPACE**. It is commonly believed that the three classes are separable and proving it is a fundamental unsolved problem.

Probabilistic and quantum TMs need to additionally restrict the probability of an error. The corresponding classes are called *Bounded-error Probabilistic Polynomial time* (**BPP**) and *Bounded-error Quantum Polynomial time* (**BQP**)

- **BPP** – class of problems computable in polynomial time on probabilistic Turing machines with the probability of an error $\varepsilon < \frac{1}{3}$.
- **BQP** – class of problems computable in polynomial time on quantum Turing machines with the probability of an error $\varepsilon < \frac{1}{3}$.

It is known that

$$\mathbf{P} \subseteq \mathbf{BPP} \subseteq \mathbf{BQP} \subseteq \mathbf{PSPACE}$$

It is not known if any of the relations are strict. It is also not known how classes **BPP** and **BQP** are related to **NP**. Shor's algorithm shows that **BQP** contains some hard problems from **NP**, such as factorization, but since factorization is not NP-complete it does not imply that **BQP** contains **NP**. The reverse is also unknown.

> The relations between complexity classes give us a good picture about the algorithmic power of quantum computing. They confirm common intuition that quantum algorithms **BQP** should be more powerful than classic both deterministic **P** and probabilistic **BPP**.

Of course, if it would turn out to be, although unlikely, that **P = PSPACE**, then all these classes will collapse into one and will be equivalent.

Quantum Supremacy

Quantum supremacy or *quantum advantage* is an effort to build a programmable quantum computer, which would solve a problem that is not feasible for any classical computer. This is a practical challenge to prove the potential of current quantum computing, different from theoretical asymptotic complexity.

One of the first claims of quantum supremacy was made by Google in 2019. Its Sycamore processor was “used to perform a series of operations in 200 seconds that would take a supercomputer about 10,000 years to complete". The claim was challenged by IBM suggesting that their fastest at the time supercomputer Summit could perform the task in 2.5 days rather than thousands of years. Later, improvements in algorithms reduced the speed of classic execution and allowed it to match or even be less than the 200 seconds runtime of Google’s quantum implementation.

In 2020 a group in the University of Science and Technology of China (USTC) announced achieving quantum supremacy on the photonic quantum computer Jiuzhang. The computation performed on Jiuzhang in 200 seconds was estimated to take 600 million years on the fastest supercomputer of the time, Fugaku. The results were further improved later with the USTC’s next-generation quantum computers Jiuzhang 2.0 and Zuchongzhi.

Canadian Xanadu (2022) and US D-Wave Systems (2024) have also reported quantum supremacy. The latest Google’s quantum computer Willow (December 2024) achieved quantum supremacy with logical qubits and error correction.

These results show that quantum computing already demonstrates a tremendous computational power. Although critics of the effort note that the problems used in quantum supremacy experiments are not practical enough and that advancements in classical algorithms and hardware may diminish quantum advantage.

Quantum Hardware

Over the last four decades substantial efforts have been devoted to the experimental development of quantum computers. A number of physical realizations were proposed including nuclear magnetic resonance, superconducting, trapped ions, semiconductor quantum dots, photonic, topological quantum computation platforms, and more. Today it is not clear which physical implementation will provide the best qubits and gates.

Common requirements for quantum hardware are

1. Qubits: an adequate representation of quantum states.
2. An implementation of a universal set of quantum gates.
3. A reliable qubit state initialization.
4. Quantum state measurement.

Qubit Coherence

Theoretical quantum computation deals with ideal qubits and genuine quantum operators, but in all current physical realizations qubits and quantum gates are unreliable due to *quantum noise* and the law of entropy. In quantum systems interaction with the environment causes state *decoherence*, which leads to information loss and computation errors. This is similar to classic mechanical systems where energy is lost due to various types of friction being converted to heat. High coherence of qubits can be achieved by isolation of qubits. The majority of today's quantum computers run at cryogenic temperatures in order to minimize interaction with the environment. But they cannot be completely isolated since operations and measurements need to be performed on them. This is one of the tradeoffs of experimental quantum computers design.

Coherence rates of existing qubits range from fractions of a second to hours. This limits the computation time, since when qubits lose coherence their states become random i.e. meaningless. Individual quantum gate execution times also vary depending on the realization. Dividing the qubit coherence time by the gate execution latency gives us the number of operations that a quantum computer is limited to for meaningful results. The computational capability can be extended by increasing the lifetime of qubits and by optimizing gate latency.

Quantum Gate Fidelity

Physical quantum gates are devices that emit different types of fields depending on the hardware platform to alter the quantum state of qubits. Quantum gates are also prone to errors due to quantum noise. Gate *fidelity* characterizes the reliability of quantum gates. It measures the precision of a physical gate compared to the ideal gate. Sequential execution of gates leads to error accumulation so that long chains of gates become unreliable. Therefore, shallow quantum circuits are preferred for lower error rates.

Scalability

The primary challenge of practical Quantum Computing is *scalability*, that is the ability to connect a large number of qubits without losing system reliability / coherence. A reasonable application of Shor's algorithm as stated earlier requires 20 million qubits [19]. Keeping such a large number of qubits entangled, which provides parallelism in quantum computation, and being able to control them with low error rates is a hard engineering and scientific problem.

Quantum Error Correction

Mitigation of decoherence effects is possible with the *quantum error correction* (QEC) technique introduced by Shor. In 1995 he presented a nine-qubit quantum error-correcting code [9]. In classic computing, error correction can be done using redundancy. This is not possible in the quantum world due to the no-cloning *theorem*, which states that an arbitrary qubit state cannot be cloned.

An ensemble of error correcting physical qubits represents a single *logical qubit*. The state of the logical qubit is shared between entangled physical qubits so that the logical state remains correct even if physical qubits are corrupted. Logical qubits are the building blocks of future fault tolerant quantum computing.

Critics of QEC raise concerns that logical qubits are still not perfect, that it requires too many physical qubits to mitigate errors with existing noisy quantum systems, and advocate for developing new QEC schemes.

Practical quantum computing has many fascinating hard problems that are being solved today or waiting to be solved or discovered. It is a fast evolving and well-funded area of research and engineering.

> Low qubit coherence times and noisy quantum gates pose a major obstacle for practical quantum computing. Without reliable scalable hardware the progress of quantum computing is limited.

Quantum Programming

Today's quantum programming is based on the quantum circuit model. Constructing quantum circuits gate-by-gate is inefficient and error-prone. The gate-by-gate approach works for a small number of qubits, but for large scale computations with thousands or millions of qubits circuits become unmanageable.

Quantum programming languages (QPL) have been developed to offer higher-level programming instruments and facilitate productivity of quantum computing. Many present QPLs use traditional programming languages like Python, C, C++, Java augmented by qubit variables to hold quantum states and built-in constructs for quantum gates.

Qubit variables are of a special type in QPLs with restrictions intrinsic to quantum states. Particularly, their values cannot be reassigned or passed in a function by value due to the non-cloning theorem. Classical loops and if statements are used to control quantum computation and simplify the description of quantum circuits.

A quantum program is then compiled to a quantum circuit. Since different quantum processors have different sets of universal quantum gates, the compiled circuit is further translated into hardware supported quantum gates and instructions of the physical quantum device for execution. Quantum compilation is an active area of research as it can optimize quantum execution in many ways, including

- Optimally translate quantum gates used in the program into hardware supported gates.
- Optimize the circuit by reducing its depth or taking advantage of the topology of qubit connectivity in the processor.

Some QPLs allow non-classic conditions on qubit variables in `if` and `loop` statements. This is not to be confused with classical Boolean conditions since a comparison of quantum states implies measurement, which collapses the states. Instead, quantum conditions are compiled into a composition of gates typically using CNOT gates.

QPLs can be divided into two main groups: *imperative* and *functional*. Imperative are based on procedural programming languages – C, Java, Python. Python is dominant as a base language for imperative QPLs and quantum software development kits (SDK).

Functional QPLs employ the *functional programming* paradigm and are based on Haskell-like languages or lambda-calculus. The advantage of the functional approach is that it does

not have to deal with effects of the no-cloning theorem as there are no assignments. Examples of functional quantum languages include QPL [22], QML [23], and Quipper [24].

Another interesting approach to quantum programming could be *logic programming* using declarative languages like Prolog and Datalog, which define relations between objects and their properties, or facts and rules, rather than a step-by-step computational process.

Quantum Software

Some examples of QPLs and SDKs that are under active development and use include

- OpenQASM is a descendant of Quantum Assembly Language (QASM). It is a part of IBM quantum SDK Qiskit. See documentation for more details.
- Cirq is an open-source Python library for quantum circuits, part of Google Quantum AI.
- Q# a high-level open-source programming language for developing and running quantum algorithms. Part of the Microsoft Quantum Development Kit.
- Amazon Braket SDK supports different languages and quantum hardware providers.
- PennyLane is a cross-platform Python library for quantum computing, which integrates with various quantum frameworks, devices, and simulators.
- Qrisp is an open-source Python framework supporting EU quantum computing.
- The quantum modeling language Qmod is a part of the Classiq quantum platform.
- Ocean is an open-source Python SDK for annealing quantum computing of D-Wave Systems.

As small-scale quantum computers become available for experiments and research, it is natural to see that the development of quantum software is more active within the systems that can and do provide access to live quantum equipment. Early QPLs such as cQASM, QCL, QPL, QML, LanQ, Silq, Scaffold, Quipper were able to run primarily in simulated mode due to lack of quantum hardware.

Quantum Computing Simulation

Quantum circuit simulator (QCS) is a software program that emulates execution of quantum circuits on traditional computers. QCSs were developed when physical quantum devices were unavailable. QCS uses brute-force calculation of the evolution of quantum states.

Since the amount of information grows exponentially with the number of qubits, classical simulators are limited by the computational power of classic computers.

QCSs are not a replacement for a physical quantum computer, but they play an important role in quantum computing as they allow real-time debugging of quantum algorithms, which is problematic with real quantum devices since one cannot check qubit states in the middle of a quantum computation. QCSs are widely used for developing new algorithms, as well as testing, debugging, and education frameworks.

There are numerous quantum simulators out there. Examples of hardware-optimized simulators that utilize multi-core CPUs and GPUs are Intel-QS and NVIDIA cuQuantum.

Quantum Memory and Storage

In von Neumann's architecture, traditional computers have the central processing unit (CPU), volatile random access memory (RAM), and persistent long-term storage. The quantum circuit model does not assume memory. Rather qubits' initial states are set up on the initialization stage, then a quantum circuit is executed, and the output is obtained by measurements.

Quantum memory, that is a system for storing and retrieving quantum states, is hard to build because of the no-cloning theorem and short qubit coherence times. Classical RAM stores information by creating a copy of it, which is not possible for quantum states. Also, information in RAM lives as long as the computer is on, while qubits susceptible to decoherence cannot hold quantum states long.

In 2008 a technique for *Quantum Random Access Memory* (QRAM) was suggested by V. Giovannetti, S. Lloyd, and L. Maccone called Bucket-Brigade [25], which uses a binary tree of qubits for addressing the memory with the leaf nodes serving as the memory cells. Error correction is used to increase coherence times.

Quantum long-term storage would be an interesting device. The amount of information stored in qubits grows exponentially on the number of qubits. "Quantum state drives" may be a very compact way of storing information. Unfortunately, such technology does not yet exist.

There is a vast amount of quantum software developed, which seems just waiting to be utilized when scalable and reliable quantum computers arrive. On the other hand,

> The state of the art of quantum software for gate-model quantum computers is still at the level of an assembly language with evident enhancement features but provides little towards higher-level abstractions.
> Further proliferation of quantum software is expected as high-level quantum programming languages are still yet to be developed.

Challenges of Quantum Computing

So far it was shown that quantum computers are based on a more powerful computational model than classic computing, promising exponential acceleration in computational power for various tasks. Returning to the question raised in the beginning – why Quantum Computing is an elusive ever-shifting target, this section will summarize the main challenges of Quantum Computing and let you answer this question for yourself.

Quantum Devices

Production of physical quantum devices is a major theoretical and technological challenge. It faces the following major problems:

1. *Reliability*: lowering error rates, increasing qubit coherence time, and improving gate fidelity.
2. *Scalability*: practical quantum computation requires millions of qubits. More qubits mean higher compound error rates and the higher complexity of maintaining the state of interconnected qubits.
3. *Fault-Tolerance*: error-correction is imperative for building logical qubits – the core building blocks of resilient quantum devices.
4. *Cost*: present quantum computers are custom manufactured. Most of them operate at very low temperatures requiring costly cooling systems and use of other expensive equipment.

Production of reliable quantum devices depends on solving these problems. Multiple approaches exist today for physical realization of quantum computers, each having its pros and cons. It may take time to find the winner and work out a mass-production technology.

Quantum Algorithm Design

Existing quantum algorithms already show drastic acceleration of classic computing for several important classes of problems. Since quantum algorithms are based on different principles and can be more powerful than the classical, a new series of algorithms is still waiting to be invented that take advantage of properties of the quantum model. Not all algorithms will benefit from the quantum approach. For example, comparison-based quantum sorting will remain lower bounded by Ω(n log n) the same as in classic [26]. In perspective, a series of volumes need to be written of "The Art of *Quantum* Computer Programming" summarizing quantum algorithms [27], [28], [29] the way Donald Knuth does [30] for classic computing.

High-Level Quantum Programming

Today's quantum programming is based on the quantum gate model, which is essentially a low-level assembler language since it is so closely tied up to the machine code instructions. Classic computing also started with logic gates and relay switching circuits [31] before they were replaced by vacuum tubes, then transistors, and integrated circuits. It took decades to level up the abstractions. The same should inevitably happen with quantum programming. For example, one can fantasize that quantum parallelism will be represented with a special construct in the language, which will be compiled into a circuit entangling qubits with multi-qubit gates. Such high-level abstractions do not exist yet, and need to be invented.

Conclusion

Quantum Computing is an exciting and fast evolving area of research, engineering, and technology. Quantum algorithms are powerful. Small scale quantum computations are already possible. But there is so much yet to be done and discovered in the field before such computations become practical. Quantum computing is not expected to replace classic computing. More likely quantum chips will be used along with traditional computers as accelerators similar to how GPUs are used to accelerate graphics and machine learning.

> Quantum Computing is upon us, but it is still ten years away

References

[1] P. Benioff, "The Computer as a Physical System: A Microscopic Quantum Mechanical Hamiltonian Model of Computers as Represented by Turing Machines," *J. Stat. Phys.* **22**, 563–591, 1980

[2] P. Benioff, "Quantum mechanical Hamiltonian models of Turing machines," *J. Stat. Phys.* **29**, 515–546, 1982

[3] Yu. I. Manin, "Computable and Uncomputable," *Sovetskoye Radio*, Moscow, 1980

[4] R. P. Feynman, "Simulating physics with computers," *Int. J. Theor. Phys.* **21**, 467–488, 1982

[5] M. A. Nielsen, I. L. Chuang, "Quantum Computation and Quantum Information," *Cambridge Univ. Press*, Cambridge, 2000

[6] D. Deutsch R. Josza, "Rapid solutions of problems by quantum computation," *Proc. Roy. Soc. London Se. A* 439, 553–558, 1992

[7] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," *35th Annual Symposium on Foundations of Computer Science*, 124-134, 1994

[8] P. W. Shor, "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer," *SIAM J. Sci. Statist. Comput.* 26, 1997

[9] P. W. Shor, "Scheme for reducing decoherence in quantum computer memory," *Physics Review A* 52 4, R2493–R2496, 1995

[10] D. Coppersmith, "An approximate Fourier transform useful in quantum factoring," *IBM Research Report RC 19642*, 1994

[11] C. Monroe, D. M. Meekhof, B. E. King, W. M. Itano, D. J. Wineland, "Demonstration of a Fundamental Quantum Logic Gate," *Phys. Rev. Lett.* **75** 25, 4714-4717, 1995

[12] L. K. Grover, "A fast quantum mechanical algorithm for database search," *STOC '96: 28th annual ACM symposium on Theory of Computing*, 212-219, 1996

[13] J A. Jones, M. Mosca, "Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer," *Chem. Phys.* 109 5, 1648–1653, 1998

[14] I. L. Chuang, N. Gershenfeld, M. Kubinec, "Experimental Implementation of Fast Quantum Searching," *Phys. Rev. Lett.* **80** 15, 3408-3411, 1998

[15] L. M. K. Vandersypen, M. Steffen, G. Breyta, C. S. Yannoni, M. H. Sherwood, I. L. Chuang, "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance," *Nature* **414**, 883–887, 2001

[16] A. W. Harrow, A. Hassidim, S. Lloyd, "Quantum algorithm for linear systems of equations," *Phys. Rev. Lett.* **103** 15, 2009

[17] S. Lloyd, M. Mohseni, P. Rebentrost, "Quantum algorithms for supervised and unsupervised machine learning," *arXiv: Quantum Physics*, 2013

[18] R. Hundt, "Quantum Computing for Programmers," *Cambridge Univ. Press*, Cambridge, 2022

[19] C. Gidney, M. Ekerå, "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits," *arXiv:1905.09749*, 2021

[20] E. Bernstein, U. Vazirani, "Quantum Complexity Theory," *SIAM J. Comput.* 26, 1411–1473, 1997

[21] A. C.-C. Yao, "Quantum circuit complexity," *IEEE Annual Symposium on Foundations of Computer Science*, 352–361, 1993

[22] P. Selinger, "Towards a Quantum Programming Language," *Mathematical Structures in Computer Science* 14 4, 527–586, 2004

[23] T. Altenkirch, J. Grattage, "A functional quantum programming language," *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, 249-258, 2005

[24] A. S. Green, P. L. Lumsdaine, N. J. Ross, P. Selinger, B. Valiron, "Quipper: A Scalable Quantum Programming Language," *ACM SIGPLAN Conference on Programming Language Design and Implementation*, 333–342, 2013

[25] V. Giovannetti, S. Lloyd, L. Maccone, "Quantum random access memory," *Physical review letters, 100 16*, 2008

[26] P. Høyer, J. Neerbek, Y. Shi, "Quantum Complexities of Ordered Searching, Sorting, and Element Distinctness," *ICALP 2001, Lecture Notes in Computer Science* 2076, 346-359, Springer, 2001

[27] S. Jordan, "Quantum algorithm zoo," online

[28] A. M. Childs, "Lecture Notes on Quantum Algorithms," online, 2022

[29] R. de Wolf, "Quantum Computing: Lecture Notes," *arXiv: Quantum Physics*, 2023

[30] D. E. Knuth, "The Art of Computer Programming," *Addison-Wesley*. 1968-...

[31] C. E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits," *Trans. AIEE*. **57** 12, 713–723, 1938