Quantum Computing

Jacob Denson

January 28, 2023

Table Of Contents

Quantum Computing is the study of the mode of computation using quantum mechanical phenomena, like the principle of superposition, interference, and entanglement, to solve computational tasks. A *quantum advantage* is found when a quantum algorithm exists that solves a problem faster than any possible classical algorithm.

Historically, quantum computing began in the 1980s, when physicists Richard Feynman and Paul Benioff observed that classical computers were very inefficient at modelling quantum mechanical phenomena. They suggested that a different kind of computer, modelled on quantum mechanic principles could model such phenomena much better, and that quantum computers could potentially speed up the computation of other tasks as well.

In 1985, David Deutsch gave the first formal model of a quantum computer. Certain problems were shown to be much easier solved using a quantum computer, though these problems were mostly contrived, and not interesting practically. And secure cryptographic key exchange, which an eavesdropper cannot observe without irreversibly altering the exchange, was shown to be possible, a task not possible in the classical setting.

In 1994, Peter Shor used *phase estimation* to develop a $\tilde{O}(n^2)$ polynomial time algorithm to factor integers and finding discrete logarithms. The best known classical algorithms for these problems run in time $O(2^{O(n^{1/3})})$ time. A consequence is that commonly used public-key cryptography algorithms, like RSA, can theoretically be broken by a quantum computer. This is not a problem just yet, since the largest integer current quantum computers can factor is 21 due to issues in the *error correction* problem in

quantum computing.

In 1995, Lov Grover used *amplitude amplification* to show that only $O(n^{1/2})$ queries were needed to search for an element in an unordered list of n elements; classical algorithms certainly need $O(n)$ queries in the worst case. Applying Grover's algorithm to the satisfication problem for a boolean function $f : \{0,1\}^n \to \{0,1\}$ requires only $\tilde{O}(2^{n/2})$ queries; we are not aware of any classical algorithm that attain such a bound.

Much research about quantum computation in the 2000s focused on solving *NP-intermediate problems*, problems in NP which are not known to be NP complete or in P, like factorization, the discrete logarithm problem, the approximate shortest vector problem, and the graph isomorphism problem. Lattice algorithms underlie many crytographic systems, but it is not yet known whether quantum computation can solve lattice algorithms efficiently. The technique of *quantum walks* emerged, a variant of classical random walks used to perform *Hamiltonian simulation*, i.e. the simulation of physical quantum systems.

In 2008, using Hamiltonian simulation, Harrow, Hassidim, and Lloyd developed the HHL algorithm, to solve well-conditioned, sparse system of $N \times N$ linear equations, with at most s non-zero coefficients in each equation in $\tilde{O}((\log N)s)$ time; the best known classical algorithm to perform this computation is $O(N^2 s)$.

A major problem in quantum in quantum computing is the issue of *error*. Quantum data is effected by being accessed, a phenomenon known as *decoherence*. Fault-tolerant design solves this problem, provided the amount of time we interact with a certain amount of data stays below a certain threshold. A result of this is that if we want to model a theoretical quantum bit, we often need many physical quantum bits in order to ensure that errors are not introduced. This is one reason why scaling up quantum computers is so difficult.

We do not focus on physical implementations of quantum computers here, we only briefly remark on the theory. To obtain quantum phenomena, one either needs a microscopic system, or a system at a very low temperature. Here are some potential implementation:

- In an atom-based quantum system, atoms are trapped in a lattice created by lasers to maintain their state. The main problem here is the issue of scalability, since to prevent atoms from interacting with one another one requires a large amount of laser power.

- One could perform an electron-based quantum system. This avoids the scalability issue since we have developed sophisticated systems of silicon technology in the development of modern classical computers to scale up the design of this system. But the difficulty is to ensure that neighboring electrons interact reliably with one another.
- Low temperature quantum systems rely on superconducting circuits, and this approach currently seems more promising that microscopic quantum systems.

Chapter 1

Modeling a Quantum Computer

There are several ways to model a classical computer. The most common is the use of a *Turing machine*. However, quantum Turing machine models are more cumbersome to deal with, so the alternate approach of using *elementary instructions* is more convenient. To review, we can view as a computation as a sequence of elementary operations, i.e. with a Turing machine we can move our tape head left and right, and replace symbols on a tape, where the tape head is located. For any m, an *elementary operations* will be a function $f : \{0,1\}^m \to \{0,1\}^m$ which reads and modifies *at most* three of the m bits in the system. A function $F : \{0,1\}^* \to \{0,1\}$ is then *computable* if there is a Turing machine which, for any input integer n, produces an integer $k \geqslant n$, and a sequence of elementary operations $f_1, \ldots, f_l$, such that for any string $s \in \{0,1\}^n$, $f(s)$ is the first bit in the bit-string $(f_l \circ \cdots \circ f_1)(s0^{k-n})$.

We begin constructing our model of quantum computation by modelling all these operations in the quantum regime. In the quantum state, the state of a system is no longer a bit string in $\{0,1\}^m$, but a unit vector in the space H_m of functions $\{0,1\}^m \to \mathbf{C}$. We view such a unit vector as a *superposition* of bit strings, since, if we identify $s \in \{0,1\}^m$ with the state $\langle s|$, which is the indicator function of s, then any state can be written as

$$\sum_{s \in \{0,1\}^m} a_s \langle s|,$$

where $\sum_s |a_s|^2 = 1$. Once we collapse the wave function, turning our quantum state into a classical state, we obtain a random bit string X with

$\mathbf{P}(X = s) = |a_s|^2$. Next, a Boolean function $f : \{0,1\}^m \to \{0,1\}^m$ then corresponds to a unique *linear operator* $T : H_m \to H_m$, such that $T\langle s| = \langle f(s)|$. Such an operator is orthogonal. An *elementary quantum operation* is then an orthogonal transformation $T : H_m \to H_m$ which depends on, and modifies the state of, at most three bits.

After performing a sequence of elementary operations, we get a probabilistic output by measuring the first bit string in the composition, which collapses the wave function.

We consider the model of *sequential gates*; In the classical setting, a circuit C is a directed, acyclic graph,

For each input of size n, we consider a circuit C_n, a directed, acyclic graph, consisting of input nodes, and other nodes, known as *gates*

diagram which shows how to derive an output via a computation of Boolean operations

consisting of a family of operations which performs an algorithm. The family of circuits is called *uniform* if there exists a single Turing machine which, on an input n, produces the circuit C_n. Uniform circuit families are equivalent to Turing machines, up to polynomial factors in running time, since there is a way to simulate any s

Let's review the concept of gates in the classical setting. A state of a computer can be viewed as a string s of zeroes and ones. To begin performing an algorithm, we initialize the state of the computer to all zeroes, and consider an input string to the algorithm, which we may assume form the initial bits of the state of the computer. One might model a computation as being performed by simple subcomputations. For instance, we might take two bits s_i and s_j, compute the *NAND* of these bits, denoted $s_i|s_j$, and replace s_i with $s_i|s_j$.

The Strong Chruch-Turing thesis is that the running time of a deterministic computer can be simulated by a Turing machine, increasing the number of steps by at most a polynomial amount of steps. It is conjectured that this is false for the simulation of probabilistic computations, but yet unproven. And it could also fail for quantum computations (in which case we would conclude that there is *quantum supremacy* for some algorithms).