# Quantum Computers: A Brief Overview

Merciadri Luca
Luca.Merciadri@student.ulg.ac.be

**Abstract.** A promising technology is the "quantum computers," and this paper gives a general overview about this subject.

**Keywords:** quantum computers.

## 1 Introduction

A promising technology is the "quantum computers." More and more scientists are interested in it because of the performances' enhancement it could bring to the today's computing world.

Quantum computers require *quantum logic*, which is fundamentally different to classical Boolean logic. This difference leads to a *greater efficiency* of quantum computation over its classical counterpart. [13]

However, many articles about quantum computers are still a bit difficult to understand for the "absolute beginner." Thus, the main aim of this paper is to give a simple overview, without coming into technical details, of quantum computing, basing principally on [3,9,10,13].

## 2 Prerequisites

We give here some important prerequisites about quantum computers and quantum computing.

### 2.1 Dirac's Notation

*Dirac* invented the "*bra-ket*" *notation*. It is very useful in quantum mechanics. The notation defines the "*ket*" vector, denoted by $\psi\rangle$, $\langle\phi|$ being its conjugate transpose (also called Hermitian conjugate: the "*bra*"). The "*bracket*" is then defined by $\langle\phi|\psi\rangle$. [14]

The inner product is linear, and defined by

$$\langle\phi|\psi\rangle = c, \tag{1}$$

$c$ being a number. [10] If $c = \langle\phi|\psi\rangle$ the complex conjugate is

$$c^* = \langle\phi|\psi\rangle^* = \langle\psi|\phi\rangle. \tag{2}$$

The state of a physical system is identified with a ray in a complex separable[1] Hilbert space, denoted by $\mathcal{H}$, or equivalently, by a point in the projective Hilbert space of the system. [14]

Each vector in the ray is called a "ket." Evidently, in the ket $\psi\rangle$, $\psi$ can be replaced by any symbols, letters, numbers, or even words. The ket can be viewed as a column-vector and (given a basis for the Hilbert space) written out in components,

$$|\psi\rangle = (\widetilde{c_0, c_1, \cdots}), \tag{3}$$

when the considered Hilbert space is finite-dimensional. In infinite-dimensional spaces there are infinitely many components and the ket may be written in complex function notation, by prepending it with a bra. [14]

Dirac's notation's description is often omitted as it is supposed to be known, but it is here an introductary article, and it has thus to be explained.

### 2.2 Qubit

**Definition 1 (Qubit).** *A qubit is a quantum system in which the Boolean states* 0 *and* 1 *are represented by a prescribed pair of normalised and mutually orthogonal quantum states labeled as* $\{|0\rangle, |1\rangle\}$. *The* $|0\rangle$ *is called "ground state;" the* $|1\rangle$ *is the "excited state." As the most general electronic state is a*

[1] Remind that a topological space is *separable* if it countains a countable dense subset.

[2] More generally, a general normalized vector can be expanded in an orthonormal basis for the space of dimension $2^N$, $N$ being the number of qubits, by

$$\sum_{x=0}^{2^N-1} a_x|x\rangle, \tag{5}$$

*superposition of the two basic states, we then have*

$$|\Psi_1\rangle = a|0\rangle + b|1\rangle, \tag{4}$$

*that is, a normalized*[2] *vector, with* $a, b \in \mathbb{C}$. *[3,9,13]*

The two states form a "computational basis" and any other (pure) state of the qubit can be written as a superposition $\alpha|0\rangle + \beta|1\rangle$ for $\alpha$ and $\beta$ such as $|\alpha|^2 + |\beta|^2 = 1$. [3] Habitually, a *qubit* is a microscopic system, such as an atom, a nuclear spin, or a polarised photon. [3]

### 2.3 Quantum Register

A collection of $n$ qubits is called a *quantum register* of size $n$. [3]

*Example 1.* For example, a quantum register of size 4 can store individual numbers such as 13:

$$|1\rangle \otimes |1\rangle \otimes |0\rangle \otimes |1\rangle \equiv |1101\rangle \equiv |13\rangle,$$

where $\otimes$ denotes the tensor product.

It can also store the two of them simultaneously. [3]

### 2.4 Quantum Logic Gates

For this, and for other manipulations on qubits, unitary operations have to be performed. Naturally, the definitions of quantum logic gate and quantum network follow.

**Definition 2 (Quantum logic gate).** *A quantum logic gate is a device which performs a fixed unitary operation on selected qubits in a fixed period of time. [3]*

**Definition 3 (Quantum network).** *A quantum network is a device consisting of quantum logic gates whose computational steps are synchronised in time.*

A quantum gate acts on superpositions of different basis states of qubits, whereas classically this option is nonexistent. [13]

Basic gates used in quantum computation are namely the Hadamard gate, the NOT gate, the C-NOT (Controlled-NOT, also known as as the XOR or the measurement gate [3]) gate, the controlled phase-shift gate, the Toffoli gate and the Fredkin gate. [3,9,13] We will now describe them.

**Hadamard Gate** The *Hadamard gate* is a single-qubit gate $H$ which performs the unitary operation known as the Hadamard transform whose action is the following:

$$|0\rangle \longrightarrow |0\rangle + |1\rangle \tag{6}$$
$$|1\rangle \longrightarrow |0\rangle - |1\rangle. \tag{7}$$

[3] More formally, if we start with a register of size $n$ in some state $y \in \{0,1\}^n$, then

$$|y\rangle \mapsto 2^{-\frac{n}{2}} \sum_{x \in \{0,1\}^n} (-1)^{y \cdot x} |x\rangle, \tag{8}$$

where the product of $y = (y_{n-1}, \cdots, y_0)$ and $x = (x_{n-1}, \cdots, x_0)$ is taken bit by bit:

$$y \cdot x = (y_{n-1}x_{n-1} + \cdots + y_1x_1 + y_0x_0). \tag{9}$$

[3] It is universal[3].

**NOT Gate** The square root of *NOT gate* is, from a classical point of view, unusual in its behaviour. A single square root of NOT gate produces a completely random output with equal probabilities of the output being equal to 0 or 1. However two such gates linked sequentially produce an output that is the inverse of the input, and thus behave in the same way as the a classical NOT gate. We then have

$$|0\rangle \longrightarrow \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \tag{10}$$
$$|1\rangle \longrightarrow \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle). \tag{11}$$

Thus, an input of $|0\rangle$ leads to an equal and opposite amplitude of the output being $|0\rangle$ or $|1\rangle$. An input of $|1\rangle$ leads to an equal amplitude of the output of the gate being $|0\rangle$ and $|1\rangle$. [8]

the $a_x$'s being in $\mathbb{C}$, satisfying $\sum_x |a_x|^2 = 1$.

[3] We shall see the definition of the *universality of a gate* in subsubsection 2.4.

**Controlled-NOT Gate** *Controlled-NOT gate* (*C-NOT*, for short) is a two-qubit gate, where the value of the first qubit (called *control*) determines what will happen to the second qubit (called *target*) qubit. [13] More precisely, it flips the second qubit if the first qubit is $|1\rangle$ and does nothing if the control qubits is $|0\rangle$. [3] The gate is thus represented by the matrix

$$\begin{pmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&0&1\\ 0&0&1&0 \end{pmatrix}, \tag{12}$$

written in the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. [3] This gate has attracted much interest in the field of quantum computation as it is reversible while requiring only[4] two inputs. [8]

**Controlled Phase-Shift Gate** Another common two-qubit gate is the *controlled phase-shift gate* $B(\phi)$ defined as

$$\begin{pmatrix} 1&0&0&0\\ 0&1&0&0\\ 0&0&1&1\\ 0&0&0&e^{i\phi} \end{pmatrix}, \tag{13}$$

also written in the computational basis $\{|00\rangle, |01\rangle, |10\rangle, |11\rangle\}$. [3]

**Toffoli Gate** *Toffoli gate* is a three-qubit gate where, if the first two bits are set, the third bit is flipped. It is then defined by

$$\begin{pmatrix} 0&0&0&0&0&0\\ 0&0&1&0&0&1\\ 0&1&0&0&1&0\\ 0&1&1&0&1&1\\ 1&0&0&1&0&0\\ 1&0&1&1&0&1\\ 1&1&0&1&1&1\\ 1&1&1&1&1&0 \end{pmatrix}, \tag{14}$$

written in the computational basis

$$\{|000\rangle, |001\rangle, |010\rangle, |011\rangle\, |100\rangle, |101\rangle, |110\rangle, |111\rangle\}.$$

[21] It is universal.

**Fredkin Gate** A last common three-qubit gate is the *Fredkin gate*. It is universal. It is described with the following matrix:

$$\begin{pmatrix} 0&0&0&0&0&0\\ 0&0&0&0&0&0\\ 0&0&1&0&0&1\\ 0&1&0&0&1&0\\ 0&1&1&0&1&1\\ 1&0&0&1&0&0\\ 1&0&1&1&1&0\\ 1&1&0&1&0&1\\ 1&1&1&1&1&1 \end{pmatrix}, \tag{15}$$

the first input being directly mapped to the C output. If $C = 0$, no swap is performed; $I_1$ maps to $O_1$, and $I_2$ maps to $O_2$. Otherwise, the two outputs are swapped so that $I_1$ maps to $O_2$, and $I_2$ maps to $O_1$.

Put briefly, it swaps the last two bits if the first bit is 1. One can notice that the number of 0 and 1 are conserved throughout. An analogy is the billiard ball model. Using this analogy, the number of balls is conserved: the same number of balls are output as input. [15]

## Universality

**Definition 4 (Universality of gates).** *Gates are called universal if they can be used to create any logic circuit, like the NAND (the conjunctive denial) gate in classical boolean-based circuits. [8]*

An extremely useful result of this *universality* is that any quantum computation can be done in terms of a C-NOT gate and a single-qubit gate (which varies), although, of course, it might sometimes be more convenient to use other gates as well. [13]

Also, a simple concatenation of the Toffoli gate and the C-NOT gate gives a simplified quantum adder, which is a good starting point for construction of full adders, multipliers, and more elaborate networks. [3]

If a phase gate is defined as a gate that flips the phase of the upper state of the target qubit only if the control qubit is in the upper state ([13]), the Hadamard and phase gates are sufficient to construct *any* unitary operation on a single qubit. [3]

[4] In fact, the difficulty of building a quantum gate greatly rises with the number of inputs to gate. [8]

## 3 Quantum Reversibility

Consider the Boolean AND gate. There is no way of completely deducing the inputs of an AND gate from the outputs, and thus the AND gate appears not to be reversible, because of its truth table.

A gate of AND type produces waste heat when working (*i.e.* giving outputs to its inputs). The "lost" information about the inputs are contained in this waste heat.

In quantum computers, we cannot allow this situation to occur. The radiation of the heat would depend on the state of the inputs to the quantum gate.

Thus, in effect, the radiation of the heat would be a measurement on the inputs and decoherence would ensue. The universes would be so far apart as to be unable to interfere with each and the result, which depends upon the interference of these universes, would be invalid. Thus, quantum gates have to be reversible.

Reversible gates must, by their very definition, have an equal number of inputs and outputs. [8] More formally, an $n$-bit reversible gate is a bijective mapping $f$ from the set $\{0,1\}^n$ of $n$-bit data to itself. [17]

Reversible gates are also useful as they would be the only potential way to improve the energy efficiency of computers beyond the fundamental von Neumann-Landauer limit of

$$kT \ln 2$$

energy dissipated per irreversible bit operation, where $k$ is Boltzmann's constant of $1.38 \cdot 10^{-23}$ J/K, and $T$ is the temperature of the environment into which unwanted entropy will be expelled. [20]

## 4 Quantum Entanglement

We say that a pure state of two qubits is *entangled* if it cannot be written as a product of the individual states of the two qubits (thus, with a tensor product), such as $|\nu_1\rangle \otimes |\nu_2\rangle$.

For example, the EPR (Einstein-Podolski-Rosen) state is not decomposable into a direct product of any form, and is therefore entangled:

$$|\Psi_{\text{EPR}}\rangle = \frac{(|01\rangle + |10\rangle)}{\sqrt{2}}, \tag{16}$$

as two qubits in this state display a degree of correlation impossible in classical physics and hence violate the Bell inequality which is satisfied by all local (*i.e.* classical) states. [13]

At the opposite, for two qubits ($n = 2$), the state

$$\alpha|00\rangle + \beta|01\rangle = |0\rangle \otimes (\alpha|0\rangle + \beta|1\rangle) \tag{17}$$

is separable: $|\Psi_1\rangle = |0\rangle$ and $|\Psi_2\rangle = \alpha|0\rangle + \beta|1\rangle$. [3]

The exploitation of a number of entangled qubits can lead to a considerable computational speed-up in a quantum computer over its classical counterpart. [13]

This leads us to the interest of using quantum computers.

## 5 Quantum Interest

### 5.1 Finding Prime Factors

Suppose we wish to find prime factors of $N$. It is equivalent to finding the smallest $r$ such that $a^r \equiv 1 \bmod N$, where $\gcd(a, N) = 1$. In other words, we want to determine the period of the function

$$a^r \bmod N.$$

There is no known *efficient* classical algorithm to factorize $N$. The time of computation is proportional to the number of divisions we have to perform, and this is $\sqrt{N} = 2^{\frac{l}{2}}$. [9]

Put simply, there are two distinct stages in this algorithm. Initially, two registers are at the input to the quantum computer. Then, the first register is prepared[5] in a superposition of consecutive natural numbers, while leaving the second register in 0 state to obtain

$$|\Psi\rangle = \sum_{n=0}^{M-1} |n\rangle|0\rangle,$$

where $M = 2^m$ is some sufficiently large number. In the second register, the function $a^i \bmod N$ is computed. This can be achieved unitarily and the result is

$$|\Psi_1\rangle = \sum_{n=1}^{M-1} |n\rangle|a^n \bmod N.$$

[9] Then, the period $r$ is extracted from the first register. [9]

However, the Shor's algorithm is probabilistic: it does not always give the correct answer. Anyway, it

[5] By *preparing*, we want to say that $N$ qubits are put in a standard initial state such as $|0\rangle|0\rangle\cdots|0\rangle$, or $|x = 0\rangle$. [9]

is not a real problem, as the answers' validity can be checked very easily: the factors are multiplied and must equal $N$. If it does not equal $N$, the algorithm loops until the factorization is correct. [9]

In fact, factoring is an example of *intractable* problem with the properties [9]:

1. The solution can be easily verified, once found,
2. But the solution is hard to find.

That is, it cannot be solved in a time bounded by a polynomial in the size of the input, in this case $\log n$. [9]

The best known factoring algorithm (called "the number field sieve") requires a time similarly equal to

$$\exp\left(c\left(\ln n\right)^{\frac{1}{3}}\left(\ln\left(\ln n\right)\right)^{\frac{2}{3}}\right), \tag{18}$$

where $c = (64/9)^{\frac{1}{3}} \simeq 1.9$. The current state of the art is that the 65 digits factors of a 130 digit number can be found in the order of one month (!) by a network of hundreds of work stations. [9] Shor showed that a computer can factor in polynomial time, e.g. in time $\mathcal{O}\left(\ln(n)^3\right)$. [9] It would result in a big difference, as shown in the next paragraph.

If we had a quantum computer that could factor a 130 digit number in one month, running Shor's algorithm could factor 400 digit numbers in less than 3 years, where it would take about $10^{10}$ years for a classical computer. [9]

It has also practical importance, as the *difficulty of factoring* is the basis of schemes for public key cryptography, such as RSA. [9] If quantum computers were able to break current RSA algorithms, it would be useful (and imperative!) to use quantum cryptography.

There are also some digital signature schemes that are already believed to be secure *against* quantum computers. For example, Lamport signatures often use crypto hash functions. Unfortunately each Lamport key can only be used to sign a single message. However, combined with hash trees, a single key could be used for many messages, making this a fairly efficient digital signature scheme. [17]

Put this way, quantum computing would have better performances to time ratio's than classical computing. Other examples are simulation of quantum physical processes, from chemistry and solide state physics, approximation of Jones polynomials, and solving Pell's equation[6]. [17] Anyway, it is not always as simple, as quantum computation is very fragile. [9] The justification of this fragility is given in the following section.

## 6 Quantum Problem

This section comes from [9]. A big quantum system cannot be perfectly isolated from its environment. To perform a complex quantum computation, a delicate superposition of states of a relatively large quantum system has to be prepared.

As this sytem cannot be perfectly isolated from its environment, this superposition decays very rapidly. Thus, contact between the computer and the environment (*decoherence*) cause errors that degrade the quantum information.

Decoherence is not the only problem. To improve performance, the bits, after each gate, could be cooled. By this way, small errors that could be made would wind up, heating the environment rather than compromising the device's performance.

Unfortunately, a quantum computer cannot be cooled this way: contact with the environment would destroy encoded quantum information. A classical error-correcting code is a repetition code: a bit we wish te protect is then replaced by, say, three copies of this bit. Using it, even if a bit flips, the bit can still be decoded correctly, by majority voting. Of course, it is possible for more than one bit to flip. It can be improved using longer codes. Approaching a Gaussian, the majority vote is said to fail (for large $N$) at

$$P_{\text{error}} \simeq e^{-Ne^2}. \tag{19}$$

[6] Recall that Pell's equation is any Diophantine equation of the form

$$x^2 - ny^2 = 1$$

where $n$ is a nonsquare integer, and $x, y \in \mathbb{Z}$. Many solutions of this equation exist, and they yield good rational approximations of the form $\frac{x}{y}$ to the square root of $n$. [7,16]

[7] It was in 1995, and, later, a more general theory of quantum error correction was developed [13]. This development has continued and has led to an avalanche of different codes that were optimized in different respects and adapted to special situations. [13] The rough idea is to entangle our information-carrying qubit with some auxiliary qubits such that we can distribute the information about the information-qubit over many auxiliary qubits. [13]

Anyway, it is not as simple to use this process with quantum computers: even if Shor has discovered[7] an error-correcting code, there can still be phase errors, which are serious, as they make the state

$$\frac{1}{\sqrt{2}}\left[|0\rangle + |1\rangle\right] \tag{20}$$

fliping to the orthogonal state

$$\frac{1}{\sqrt{2}}\left[|0\rangle - |1\rangle\right]. \tag{21}$$

As

$$\sigma_z = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{22}$$

if we consider a phase error ($\sigma_z$ operator) in a rotated basis, it then appears as an amplitude error, and *vice-versa*. This new basis that we obtain from the $\{|0\rangle, |1\rangle\}$ by using a Hadamard transformation is given by $|\widetilde{0}\rangle = \frac{(|0\rangle + |1\rangle)}{\sqrt{2}}$ and $|\widetilde{1}\rangle = \frac{(|0\rangle - |1\rangle)}{\sqrt{2}}$. In this new basis, a phase error has the effect of an amplitude error, and has thus the effect $\sigma_z|\widetilde{0}\rangle = |\widetilde{1}\rangle$, $\sigma_z|\widetilde{1}\rangle = |0\rangle$. Therefore, instead of encoding the state $\alpha|0\rangle + \beta|1\rangle$, we encode it as

$$\alpha|0\rangle + \beta|1\rangle \rightarrow \alpha|\widetilde{000}\rangle + \beta|\widetilde{111}\rangle. \tag{23}$$

[13]

Furthermore, measuring the qubits to detect errors would disturb the quantum information they encode. Replicating qubits is also difficult, as copying quantum information cannot be copied with perfect fidelity.

## 7 Quantum Programming

This section comes from [18].

Using *quantum programming*, one can allow the expression of quantum algorithms using high-level constructs [18]. Its aim is to provide a tool for researchers to understand better how quantum computation works and how to formally reason about quantum algorithms.

### 7.1 Imperative Quantum Programming Languages

**Quantum Pseudocode** Firstly, a *quantum pseudocode* was proposed by E. Knill. It was the first formalised language for description of quantum algorithms.

**Quantum Computer Language** After it, a *Quantum Computer Language* (*QCL*) was proposed. It is one of the first implemented quantum programming languages. Its syntax resembles syntax of the C programming language and classical data types are similar to data types in C.

The basic built-in quantum data type in QCL is the `qureg` (quantum register). It can be interpreted as an array of qubits (quantum bits). An example of such a code would follow the following model.

```
qureg x1[2]; // 2-qubit quantum register x1
qureg x2[2]; // 2-qubit quantum register x2
H(x1); // Hadamard operation on x1
H(x2[1]); // Hadamard operation on the first qubit of the register x2
```

As the `qcl` interpreter uses `qlib` simulation library, it is possible to observe the internal state of the quantum machine during execution of the quantum program:

```
qcl> dump
: STATE: 4 / 32 qubits allocated, 28 / 32 qubits free
0.35355 |0> + 0.35355 |1> + 0.35355 |2> + 0.35355 |3>
+ 0.35355 |8> + 0.35355 |9> + 0.35355 |10> + 0.35355 |11>
```

Fortunately, the `dump` operation is different from measurement, since it does not influence the state of the quantum machine and can be realised only using a simulator. Mainly, the QCL standard library provides standard quantum operators used in quantum algorithms such as:

1. controlled-not with many target qubits,
2. Hadamard operation on many qubits,
3. parse and controlled phase.

The most important feature of QCL appears to be the support for user-defined operators and functions. Like in modern programming languages, it is possible to define new operations which can be used to manipulate quantum data.

**Q Language** *Q Language* is the second implemented imperative quantum programming language.

It was implemented as an extension of C++ programming language. It provides classes for basic quantum operations like `QFourier`, `QHadamard`, `QNot`, and `QSwap`, which are derived from the base class `Qop` . New operators can be defined using C++ class mechanism. Quantum memory is represented by class `Qreg`.

Here is an example of Q code.

```
Qreg x1(); // 1-qubit quantum register with initial value 0
Qreg x2(2,0); // 2-qubit quantum register with initial value 0
```

Computation process is executed using provided simulator. Noisy environment can be simulated using parameters of the simulator.

**qGCL** *Quantum Guarded Command Language* (*qGCL*) was defined by P. Zuliani in his PhD thesis. It is based on Guarded Command Language created by Edsger Dijkstra. It can be described as a language of quantum programmes specification.

### 7.2 Functional Quantum Programming Languages

During the last few years many quantum programming languages based on the functional programming paradigm were proposed. *Functional programming languages* are well-suited for reasoning about programs.

**QFC, QPL** *QFC* and *QPL* are two closely related quantum programming languages defined by Peter Selinger. They differ only in their syntax: QFC uses a flow chart syntax, whereas QPL uses a textual syntax.

These languages have classical control flow, but can operate on quantum or classical data. Selinger gives a denotational semantics for these languages in a category of superoperators.

**QML** *QML* is a Haskell-like quantum programming language by Altenkirch and Grattage. Unlike Selinger's QPL, this language takes duplication, rather than discarding, of quantum information as a primitive operation. Duplication in this context is understood to be the operation that maps $|\phi\rangle$ to $|\phi\rangle \otimes |\phi\rangle$.

**Quantum Lambda Calculi** *Quantum lambda calculi* are extensions of the lambda calculus, introduced by Alonzo Church and Stephen Cole Kleene in the 1930s. The purpose of *quantum lambda calculi* is to extend quantum programming languages with a theory of higher-order functions[8].

The first attempt to define a quantum lambda calculus was made by Philip Maymin in 1996. His lambda-*q* calculus is powerful enough to express any quantum computation. This language can efficiently solve NP-complete problems, and therefore appears to be strictly stronger than the standard quantum computational models (such as the quantum Turing machine[9] or the quantum circuit model[10]).

In 2003, André van Tonder defined an extension of the lambda calculus suitable for proving correctness of quantum programs. He also provided an implementation in the Scheme programming language.

In 2004, Selinger and Valiron defined a strongly typed lambda calculus for quantum computation with a type system based on linear logic.

## 8 The Future

### 8.1 Overview of Research

As said in [3], research in quantum computation and in its all possible variations has become vigorously active and any comprehensive review of the field must be obsolete as soon as it is written.

The following text comes from [13].

### 8.2 Practical View of Realising a Quantum Computer

To *realize a quantum computer* (or indeed any other computer) we have to have a physical medium in which to store and manipulate information. It is here that quantum information becomes very fragile and it turns out that the task of its storage and manipulation requires a lot of experimental ingenuity.

[8] Remind that *high-order functions* are functions which can take one or more functions as an input, and output a function.

[9] As in classical computers, a *Turing machine* is an abstract machine which aims to model the effect of a computer. Here, the computer is a quantum computer. The Turing machine thus provides a very simple model which captures all of the power of quantum computation. Any quantum algorithm can be expressed formally as a particular quantum Turing machine; thus, quantum Turing machines have the same relation to quantum computation that normal Turing machines have to classical computation. Quantum Turing machines can be related to classical and probabilistic Turing machines in a framework based on transition (stochastic) matrices, as shown by Lance Fortnow. [4,19]

[10] This latter is often prefered to quantum Turing machines. Here, a computation is a sequence of reversible transformations on a quantum mechanical analog of an $n$ bit register. This analogous structure is referred to as an $n$-qubit register. [17]

A very beautiful proposal for an *ion-trap quantum computer* was made by Cirac and Zoller. Subsequently, other realistic suggestions such as quantum computation based on nuclear magnetic resonance methods have been made. Although these new proposals are very interesting, we confine ourselves here to the summarized description of the *linear ion trap implementation* of Cirac and Zoller.

**Linear Ion Trap** The *linear ion trap* is one of the more promising proposals for a physically realizable quantum computer. Here, information is stored into electronic states of ions, which are in turn confined to a linear trap and cooled to their ground state of motion. Laser light is then used to manipulate information in the form of different electronic transitions.

However, the uncontrollable interactions of ions with their environment induce various errors known as decoherence (such as e.g. spontaneous emission in ions) and thus severely limit the power of computation.

There is a method to combat decoherence during computation known under the name of *quantum error correction*. This then leads to the notion of *fault-tolerant quantum computation*, which is a method of performing reliable quantum computation using unreliable basic components (e.g. gates) providing that the error rate in this components is below a certain allowed limit.

Much theoretical work has been undertaken in this area at the moment and there is now a good understanding of its powers and limitations. The main task is now with the experimentalists to try to build the first fully functional quantum computer, although it should be noted that none of the present implementations appear to allow long or large scale quantum computations and a breakthrough in technology might be needed.

Despite the fact that at present large computational tasks seem to lie in the remote future, there is a lot of interesting and fundamental physics that can be done almost immediately with the present technology. A number of practical information transfer protocols use methods of quantum computation.

However, implementing large numbers of quantum gates on many qubits is not easy because noise from all kind of sources will disturb the quantum computer. Of course, also a classical computer suffers from the interaction with a noisy environment and nevertheless works very well.

The advantage of a classical computer is, however, that it is a classical device in the sense that one bit of information is represented by the presence or absence of a large number of electrons. Therefore a small fluctuation in the number of electrons does not disturb the computer at all.

On the contrary, in a quantum computer, the qubit is stored in the electronic degree of freedom of a single atom. Even worse than that, a quantum computer *crucially* depends on the survival of quantum mechanical superposition states which are notoriously sensitive to decoherence and dissipation. This makes a quantum computer extremely sensitive to small perturbations from the environment. It has been shown that even rare spontaneous emissions from a metastable state rule out long calculations unless new ideas are developed.

We hope that quantum factorization and other large and important quantum computations will be realized eventually. Fortunately, there is a vast amount of effort and ingenuity being applied to these problems, and the future possibility of a fully functioning quantum computer still remains very much alive.

## References


1. Berkeley University, *The Mathematical Formalism of Quantum Mechanics*, (2006). Physics 221A, University of California, Berkeley; http://bohr.physics.berkeley.edu/classes/221/0708/notes/hilbert.pdf.
2. Dowek, Gilles and Arrighi, Pablo, *Linear-algebraic Lambda-calculus: higher-order, encodings and confluence*, Quantum Physics, (2006). http://arxiv.org/abs/quant-ph/0612199.
3. Ekert, Artur, Hayden, Patrick and Inamori, Hitoshi, *Basic concepts in quantum computation*, (2001).
4. Fortnow, Lance, *One complexity theorist's view of quantum computing*, (2002). http://www.sciencedirect.com/science?_ob=ArticleURL&_udi=B6V1G-44M2G3W-4&_user=10&_rdoc=1&_fmt=&_orig=search&_sort=d&view=c&_acct=C000050221&_version=1&_urlVersion=0&_userid=10&md5=60761b323b55062ac8cd2ea22abb88b0.
5. Grattage, J, *QML: A Functional Quantum Programming Language*, 2009. http://sneezy.cs.nott.ac.uk/qml/.
6. Iriyama, Satoshi and Ohya, Masanori, *On generalized quantum Turing machine and its language classes*, (2007). http://wseas.us./e-library/conferences/2007dallas/papers/567-277.pdf.
7. Lenstra, H.W. Jr., *Solving the Pell Equation*, American Mathematical Society, (2002). http://www.ams.org/notices/200202/fea-lenstra.pdf.
8. Marshall, Jonathan, *Simulating Quantum Circuits*, 2009.
9. Preskill, John, *Quantum Information and Computation*, 1998.
10. Saffman, M., *Dirac Notation and rules of Quantum Mechanics*, (2006). Atomic and Quantum Physics; http://hexagon.physics.wisc.edu/teaching/2007f_ph448/diracnotation.pdf.
11. Selinger, Peter, *Towards a Quantum Programming Language*, (2003). http://www.mathstat.dal.ca/~selinger/papers/qpl.pdf.
12. van Tonder, Andr, *A Lambda Calculus for Quantum Computation*, SIAM Journal on Computing, (2004). http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=SMJCAT000033000005001109000001&idtype=cvips&gifs=yes; http://www.het.brown.edu/people/andre/qlambda/.
13. Vedral, Vlatko and Plenio, Martin B., *Basics of Quantum Computation*, (2002).
14. Wikipedia, *Bra-ket notation - Wikipedia, the free encyclopedia*, 2009.
15. ———, *Fredkin gate - Wikipedia, the free encyclopedia*, 2009.
16. ———, *Pell's equation - Wikipedia, the free encyclopedia*, 2009.
17. ———, *Quantum computer - Wikipedia, the free encyclopedia*, 2009.
18. ———, *Quantum programming - Wikipedia, the free encyclopedia*, 2009.
19. ———, *Quantum Turing machine - Wikipedia, the free encyclopedia*, 2009.
20. ———, *Reversible computing - Wikipedia, the free encyclopedia*, 2009.
21. ———, *Toffoli gate - Wikipedia, the free encyclopedia*, 2009.


The articles coming from Wikipedia have naturally been rechecked. The URI related to these documents have voluntarily been given, but keep in mind that these pages are often subject to numerous modifications. Links with no given author (which constitute the most of the given links) are classed by date of browsing.