Classic bit vs qubit

no-cloning theorem

Mario Cuomo

An article for Microsoft Q#Advent Calendar 2021

09.12.2021

CONTENTS

1 ABSTRACT

When we talk about quantum computers, we mistakenly associate this technology with the modern age, certainly after the 2000s. In reality, the studies on this new computation model were born even earlier. Consider, for example, that the ***no-cloning Theorem***, the final object of this article, was demonstrated in the 1980s (although there is a practical demonstration of the phenomenon 10 years earlier, in 1970).
Why this happened?

The computational model of quantum computers is based on quantum mechanics which, as we know, was born in 1900 with the first studies of Max Planck and, subsequently, of Ewinr Schrödinger (have you ever wondered why references to cats often appear when talking about quantum computing?) and Paul Dirac.
The advent of this new calculation model poses a strong challenge to the ***strong Church Turing Thesis***. The *Church Turing Thesis* states that if there is a humanly computable problem then there also exists a Turing probability machine that can solve this problem efficiently (or, to put it another way, the class of computable functions coincides with that of functions computable by a Turing machine). The adjective *strong* indicates the impossibility of the existence of a more expressive model than TM.
A classic example in the literature of this challenge is the ***Shor's algorithm***.

Given an integer N expressible as the product of two prime numbers p and q, it is possible to obtain p and q in a *polynomial* time to N.
The classic algorithms return these values in a *super polynomial* time with respect to N .

The computability theory is based on the *Church Turing Thesis.* Unfortunately, this theory is not completely expressive as it is based on classical physics to model the universe but, as mentioned, the universe is ***quantum physical***.
We can therefore say that a quantum computer is a machine made **specifically** to perform all those tasks that a classical machine cannot do (by its nature, not because *'it is not powerful enough'*). In the same way, however, tasks performed efficiently by classical computers can be realized in an inefficient way by quantum computers: it follows that quantum computing is a valid tool only for certain classes of problems.
Although not yet available, work is underway on general-purpose quantum computers.

2 INTRODUCTION TO THE SUPER-POSITION

The fundamental computation unit for quantum computers is the ***qubit***.
To better understand what a *qubit* is we can think of the physics of an atom, for example a *electron*.
It is generally stated that a *electron* rotates on an orbit around the atom.
Based on its energy, it rotates on a different orbit: the states of a particle are therefore *quantized* in a discrete set.
Quantum mechanics, on the other hand, states that these parameters, at a certain instant of time, are not fixed but **vary**.
An *electron* can therefore be in multiple orbits at the same time.
The newly introduced property is the ***super-position***.

We can propose a comparison with the classical calculation which uses the *bit* as an element of information. A *bit* is a binary digit that can take only two possible values: 0 or 1.
The *qubit*, on the other hand, can take ***simultaneously*** both states or the intermediate states between the two extremes.
By *simultaneously* we mean the impossibility of defining, in a given instant and with certainty, the value of the qubit. However, it is possible to define the *probability* with which the qubit is in one state rather than another.

Formally we denote ***super-position*** as

$$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle, \alpha_0, \alpha_1 \in C$$

In addition to *super-position*, another fundamental axiom of quantum computing is ***decay after observation***.
Returning to the example of the electron that rotates around the atom, if we try to measure it and ask ourselves what orbit it is on, it loses its *super-position* property and collapses on one of the two known states.
The probability with which it collapses on one value rather than another depends on the two parameters α_0 and α_1. In particular, the qubit, after observation, collapses to 0 with probability $|\alpha_0|^2$, while it collapses to 1 with probability $|\alpha_1|^2$.
It follows that the normalization constraint for the *super-position* must be checked.

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

The *super-position* described above uses the *ket-notation* devised by Dirac to represent column vectors.

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix} e \, |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The qubit $|\psi\rangle$ can therefore also be represented as

$$|\psi\rangle = \alpha_0 \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \alpha_1 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

3 MEASUREMENT OF A QUBIT - Q#

It is possible to use the Q#language to develop applications and algorithms that exploit quantum computing. The basic idea is to have two separate systems: the first is that of the quantum simulator that executes the Q#code, the second is that of the application (for example Python, as in the following examples) which exploits the results obtained by computation.
Here is the documentation of how to set up a Q#environment.
https://docs.microsoft.com/it-it/azure/quantum/install-overview-qdk
A first experiment that can be carried out concerns the measurement of a qubit which has exactly the same probability of being in one of the two basic states. A qubit thus described is the following:

$$|\psi\rangle = \frac{1}{\sqrt{2}}\alpha_0 |0\rangle + \frac{1}{\sqrt{2}}\alpha_1 |1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\0\end{bmatrix} + \frac{1}{\sqrt{2}}\begin{bmatrix}0\\1\end{bmatrix}$$

The probability that $|\psi\rangle$ is in 0 is $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$.
The probability that $|\psi\rangle$ is in 1 is $|\frac{1}{\sqrt{2}}|^2 = \frac{1}{2}$.

```
import qsharp
from Qrng import SampleQuantumRandomNumberGenerator

i=0
while i<10:
    print(SampleQuantumRandomNumberGenerator.simulate())
    i=i+1
```

```
namespace Qrng {
    open Microsoft.Quantum.Intrinsic;

    operation SampleQuantumRandomNumberGenerator() :
        Result {
         use q = Qubit(); // Allocate a qubit.
         H(q);            // Put the qubit to
            superposition. It now has a 50% chance of
            being 0 or 1.
         let r = M(q);    // Measure the qubit value.
         Reset(q);
         return r;
    }
}
```

In the proposed example a qubit is generated. The Qubit() constructor returns a qubit that is in the state $|\psi\rangle = |0\rangle$.
If we measure the qubit in this situation it will always return the value 0.
To place the qubit in an equally probable intermediate state between $|0\rangle$ and $|1\rangle$ we apply the *Hadamard* H operator, described in the next paragraphs.
A possible execution of the script could produce the output:

$$0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad 1 \quad 0 \quad 0$$

If we had not applied the H operator:

$$0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0$$

4 OPERATORS ON A SINGLE QUBIT

As with classical bits, it is possible to define fundamental operators that operate on qubits. In this chapter the focus is on the operators that work on the single qubit.
Applying an operator to a qubit means multiplying the starting qubit by a U matrix. The U matrix must be a *unitary transformation.*
An operator is *unitary* when the product between it and the *Hermitian adjoint* returns the *identity* matrix.

$$UU^* = I$$

There are 4 important single qubit operators, the *Pauli operators.*

- ***X*** operator, *bit flip*
  The X operator, often referred in literature as σ_X, is described by the square matrix:
  $$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
  As the name suggests, the final result after applying the operator is to swap amplitude values α_0 and α_1.
  $$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |0\rangle$$
  $$X |\psi\rangle = \alpha_1 |0\rangle + \alpha_0 |0\rangle$$
- ***Z*** operator, *phase flip*
  The Z operator, often referred in literature as σ_Z, is described by the square matrix:
  $$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$
  As the name suggests, the final result after applying the operator is to reverse the phase of the qubit.
  $$|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |0\rangle$$
  $$Z |\psi\rangle = \alpha_0 |0\rangle - \alpha_1 |1\rangle$$
- ***Y*** operator, $Y = XZ$
  The Y operator can be obtained from the X and Z operators as follows .
  $$Y = XZ = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$
- ***I*** operator, *identity*
  Applying this operator does not change the *super-position* of the qubit.

5 PAULI MATRICES FOR A QUBIT - Q#

You can easily test Pauli operators using Q#.
Here are some examples.

```
import qsharp
from Qrng import X_application
from Qrng import Z_application
from Qrng import Y_application
from Qrng import YasXZ_application


print("By measuring X|0> we obtain " + str(X_application.simulate()))
print("By measuring ZZX|0> = ZZ|1> = Z(-|1>) we obtain " +
                                    str(Z_application.simulate()))
print("By measuring Y|0> we obtain " + str(Y_application.simulate())+
                                    " that is the same of XZ|0>:
                                    "+str(YasXZ_application.simulate()))
```

```
namespace Qrng {
    open Microsoft.Quantum.Intrinsic;

    operation X_application() : Result {
        use q = Qubit(); // Allocate a |0> qubit.
        X(q);            // Apply X to the qubit, we get
            |1>
        let r = M(q);    // Measure the qubit value.
        Reset(q);
        return r;
    }

    operation Z_application() : Result {
        use q = Qubit(); // Allocate a |0> qubit.
        X(q);            // Apply X to the qubit, we get
            |1>
        Z(q);            // Apply Z to the qubit, we get
            -|1>
        Z(q);            // Apply Z to the qubit, we get
            |1>
        let r = M(q);    // Measure the qubit value.
        Reset(q);
        return r;
    }
    operation YasXZ_application() : Result {
        use q = Qubit(); // Allocate a |0> qubit.
        X(q);            // Apply X to the qubit, we get
            |1>
        Z(q);            // Apply Z to the qubit, we get
            -|1>
        let r = M(q);    // Measure the qubit value.
        Reset(q);
        return r;
    }

    operation Y_application() : Result {
        use q = Qubit(); // Allocate a |0> qubit.
        Y(q);            // Apply Z to the qubit, we get
            -|1>
        let r = M(q);    // Measure the qubit value.
        Reset(q);
        return r;
    }
}
```

The *Hadamard* operator H is characterized by the matrix

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

Applying the operator to an unbalanced qubit on a base state, for example $|0\rangle$, places the qubit in a *super-position* balanced between the two base states.

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$$

6 QUBIT ENTANGLED

As with traditional computers, it is possible to think of multiple qubits together in quantum computing.
Given a set of any two qubits, they must be thought of as a single entity and not separate.
The principle behind this idea is the ***entanglement***.

$$\alpha_0 |0\rangle + \alpha_1 |0\rangle$$

$$\beta_0 |0\rangle + \beta_1 |0\rangle$$

The overall state described by the two qubits is the tensor product between the two.

$$\begin{bmatrix}\alpha_0\\ \alpha_1\end{bmatrix} \otimes \begin{bmatrix}\beta_0\\ \beta_1\end{bmatrix} = \alpha_0\beta_0 |00\rangle + \alpha_0\beta_0 |01\rangle + \alpha_1\beta_0 |10\rangle + \alpha_1\beta_1 |11\rangle$$

The base states are $|00\rangle, |01\rangle, |10\rangle$ and $|11\rangle$ and they represent a *orthonormal* base.
The normalization constraint remains valid

$$|\alpha_0\beta_0|^2 + |\alpha_0\beta_1|^2 + |\alpha_1\beta_0|^2 + |\alpha_1\beta_1|^2 = 1$$

After observing one of the two qubits, the description of the overall state changes.
If you measure the first qubit, and observe for example 1, the system changes its state.

$$\frac{\alpha_1\beta_0 |10\rangle + \alpha_1\beta_1 |11\rangle}{\sqrt{|\alpha_1\beta_0|^2 + |\alpha_1\beta_0|^2}}$$

Two qubits are ***entangled*** when it is not possible to separate them and uniquely identify the characteristics of each. They influence each other, even from a distance.
A classic example in literature is *Bell* state. A *Bell* state expresses quantum correlation and is achievable starting from two basic states with the application of Hadamard and *CNOT* (operator applicable to two qubits).

$$|\psi\rangle = \frac{1}{\sqrt{2}} |00\rangle + \frac{1}{\sqrt{2}} |11\rangle$$

If we wanted to represent Bell's state as a combination of two qubits, we would have an impossible system of equations.

$$\alpha_0\beta_0 = \frac{1}{\sqrt{2}} \quad \alpha_0\beta_1 = 0 \quad \alpha_1\beta_0 = 0 \quad \alpha_1\beta_1 = \frac{1}{\sqrt{2}}$$

7 OPERATORS ON TWO QUBITS

one of the main features of unit operators U is that if we apply the operator twice to the same qubit we get the starting qubit. They are *invertible* operators.
If we apply a U operator to two qubits in input, we get two qubits as output. This does not happen with classic operators that apply to bits (*AND*, *OR*, ...).

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$
$$U|\psi\rangle = \alpha'_{00}|00\rangle + \alpha'_{01}|01\rangle + \alpha'_{10}|10\rangle + \alpha'_{11}|11\rangle$$
$$U(U|\psi\rangle) = |\psi\rangle$$

Let us consider two widely used operators between two (or more) qubits.

- $CNOT$ operator, *Controlled Not*
  The $CNOT$ operator, in its basic version, takes two qubits as input: a a qubit which is the ***control qubit*** and a b qubit which is the ***qubit target***.
  The idea is to apply a *bit flip* on the b target qubit if and only if the a control qubit is $|1\rangle$.

  $$CNOT|00\rangle = |00\rangle$$
  $$CNOT|01\rangle = |01\rangle$$
  $$CNOT|10\rangle = |11\rangle$$
  $$CNOT|11\rangle = |10\rangle$$

  More generally, we have that

  $$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$$
  $$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{11}|10\rangle + \alpha_{10}|11\rangle$$

  The matrix that describes $CNOT$ operator is

  $$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

  $CNOT$ operator is used to generate two entangled qubits when using $|0\rangle$ as the target bit.
  Given a generic control qubit $|\psi\rangle = \alpha_0|0\rangle + \alpha_1|1\rangle$ with the target qubit $|0\rangle$ we have a system in an overall state
  $(\alpha_0|0\rangle + \alpha_1|1\rangle) \otimes |0\rangle = \alpha_0|00\rangle + \alpha_1|10\rangle$.
  Applying the $CNOT$ operator to this yields the entangled state
  $\alpha_0|00\rangle + \alpha_1|11\rangle$.

- *H* operator, *Hadamard*
  As we have seen, the Hadamard operator allows you to place a completely unbalanced qubit in one of the two basic states in *super-position*.
  The same idea can be applied when there are two qubits in input.
  It is possible to think of the Hadamard operator on two qubits as a single ***composed operator*** which contains within it two *Hadamard* operators (H_1 and H_1) that each work on the single qubit.
  The resulting operator is the tensor product between the two matrices representing the two operators.

$$H_2 = H_1 \otimes H_1 = \frac{1}{\sqrt{2}} \begin{bmatrix} H_1 & H_1 \\ H_1 & -H_1 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ \end{bmatrix}$$

  If we apply H_2 to a completely unbalanced state, for example $|00\rangle$, we obtain

$$\frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$$

  which is a state that is in every possible base state with a probability of 25%

The operators $H, X, Z, CNOT$ are used in the ***Quantum Teleportation Protocol*** to teleport information of a qubit.
The protocol is based on sharing a pair of *entangled* qubits.

8 OPERATORS ON TWO QUBITS - Q#

You can test the operation of the *CNOT* operator using Q#.
As we have seen, we have two qubits as input: the *controller* and the *target*.
The target changes value according to the controller value: if this is $|1\rangle$ then the value of the target qubit is inverted, otherwise it remains unchanged.

```
namespace Qrng {
    open Microsoft.Quantum.Intrinsic;

    operation CNOT_application() : Result {
        use control = Qubit(); // Allocate a |0> qubit.
        use target = Qubit(); // Allocate a |0> qubit.
        X(control);            // Apply X to the control
           qubit, we get |1>

        CNOT(control,target);
        let r = M(target);    // Measure the qubit
           value, it returns 1
        Reset(control);
        Reset(target);
        return r;
    }
}
```

Notice that if the X operator had not been applied to the control qubit, the result would have been 0.

Entanglement can also be tested.
In the following example, two qubits are declared, one in the state $|0\rangle$ and the other in the state $|+\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$.
Entanglement is done through the *CNOT* operator.
The final state is

$$|\psi\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$$

Notice how the behavior of one qubit is described by the behavior of the other.
If you measure the first qubit and get 0, then the second qubit will also have value 0.
If you measure the first qubit and get 1, then the second qubit will also have value 1.

```
namespace Qrng {
    open Microsoft.Quantum.Intrinsic;

    operation Entangle_application() : Int {
        use control = Qubit(); // Allocate a |0> qubit.
        use target = Qubit(); // Allocate a |0> qubit.
        H(control);            // Apply H to the control
           qubit, we get a superposition in the middle

        CNOT(control,target);
        let m1 = M(target);    // Measure the qubit
           value, it now has a 50% chance of being 0 or
           1.
        let m2 = M(control);   //if m1==0 -> m2==0, if
           m1==1 -> m2==1
        Reset(control);
        Reset(target);
        mutable x = -1;
        if(m1==m2){
            if(m1==One){
                set x=1;
            }
            else{
                set x=0;
            }
        }
        return x;
    }
}
```

9 NO-CLONING THEOREM

In computers that use the traditional calculation model, one of the most frequently performed operations is the *copy* of bits.
Unfortunately, there is no similar operation in the quantum computer model: it is not present, and ***cannot*** even exist.
Given a qubit $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$ it is not possible to build any machine starting from $|\psi\rangle$ that produces two identical copies of $|\psi\rangle$ in the overall state $|\psi\rangle \otimes |\psi\rangle$.
Remember that the values of α_0 and α_1 are not known: if you take the measurement you lose the *super-position* property.
The ***No Cloning Theorem*** therefore imposes severe restrictions on the programmer.

The proof is very simple, using Reductio ad absurdum.
We assume the existence of a unitary operator U that can perform the following operation:

$$U(|\psi\rangle \otimes |0\rangle) = |\psi\rangle \otimes |\psi\rangle$$

If such an operator existed, it would also be valid for base states.

$$U(|0\rangle \otimes |0\rangle) = |0\rangle \otimes |0\rangle$$
$$U(|1\rangle \otimes |0\rangle) = |1\rangle \otimes |1\rangle$$

Let the qubit to be cloned, for example, $|\psi\rangle = |+\rangle$

$$|\psi\rangle = \frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle$$

The U operator is applied to $|\psi\rangle \otimes |0\rangle$

$$U((\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle) \otimes |0\rangle) = \frac{1}{\sqrt{2}} U((|0\rangle \otimes |0\rangle) + (|1\rangle \otimes |0\rangle)) =$$
$$= \frac{1}{\sqrt{2}}(U(|0\rangle \otimes |0\rangle) + U(|1\rangle \otimes |0\rangle)) = \frac{1}{\sqrt{2}}(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$$

What you get is different from what we want to have.

$$|\psi\rangle \otimes |\psi\rangle = (\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle) \otimes (\frac{1}{\sqrt{2}} |0\rangle + \frac{1}{\sqrt{2}} |1\rangle)$$
$$= \frac{1}{2}(|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle)$$

It follows that it is ***impossible*** to implement the U operator.

10 NOTE

The code is available here
https://github.com/mariocuomo/Microsoft-Q-Advent-Calendar-2021

This article is strongly inspired by the course of *Next Generation Computing Models* - *Roma Tre University.*