



Department of Computer Science
www.cs.bris.ac.uk

What is Computer Science ? # 08

Winning a Game Show with a Quantum
Computer

Ashley Montanaro

August 21, 2008

Computers have come a long way over the last 50 years. But, surprisingly, in many ways today's desktop PC is very similar to yesterday's mainframe, and the basic architecture of a modern computer would be familiar to an engineer of the '50s. Today's computer has smaller components, and runs a lot faster, but some problems which were intractable on a 50-year-old computer remain intractable today.

This chapter is about a fundamentally different type of computer: a *quantum* computer. A quantum computer is a machine built to use the mysterious principles of *quantum mechanics*. Quantum mechanics claims that many surprising things are true about our universe. For example: atoms can interfere with each other, like waves [6]; doing something to a particle on the Earth can affect a particle on the Moon [3]; and cats can be both alive and dead at the same time [9]. People have come up with clever ways to use properties like these to do things that the computers that we are familiar with – which are known as *classical* computers in contrast to quantum computers – cannot. In particular, it is known that quantum computers can solve certain problems faster than classical computers can.

Large-scale quantum computers don't exist yet, although physicists are working on it! But when they do, they are expected to be a revolutionary technological advance. A critic might object here that today's computers are made from devices like transistors, which are of course quantum mechanical systems. But the key difference is that algorithms that run on today's computers don't take advantage of anything quantum mechanical.

Here's a frivolous story to illustrate one simple problem for which quantum computers have an advantage.

1 Meal or No Meal

It's 2050. The most popular TV show in the UK for the last decade has been the game show *Meal or No Meal*. The rules of the game are simple. There are 100 rounds, and in each round, the contestant is shown two closed saucepans on a table. Each saucepan may or may not contain a delicious meal (say, a roast chicken), so there are somewhere between zero and two meals on the table. The contestant is allowed to open at most one saucepan, and then has to say whether or not they think there is exactly one meal on the table. If they are right a hundred times in a row, they win a billion dollars (approx 50p in today's money).

Nobody has ever collected the prize. Why? Imagine the game show host tosses a coin to decide whether to put a chicken in each saucepan. Now, whichever saucepan the contestant opens, there is a 50% chance that the other saucepan contains a chicken. So, whether the first saucepan had a chicken in it or not, the probability of there being exactly one chicken between the two saucepans is 50%. This means that, whatever the contestant says, they have only a 50% chance of being right each time – a probability of 0.5. So the probability of being right a hundred times a row is $0.5 \times 0.5 \times \dots \times 0.5 = 2^{-100}$ – negligibly small!

This chapter will explain how, if a contestant had a quantum computer, they could win the billion-dollar prize with certainty. That is, a quantum computer can tell whether or not there is exactly one chicken between two saucepans, with only one peek into a saucepan.

2 The mathematics of quantum computing

Quantum mechanics sounds like a pretty complicated subject. But it turns out that, to understand the basics of quantum computing, you only need some quite simple maths, which doesn't even require a pocket calculator. In the following section, we'll introduce the basic mathematical formalism behind quantum computing. There are a couple of things to bear in mind before we start.

First, this will be a cut-down and simplified version of quantum mechanics. But it's still accurate, and is enough to understand simple quantum algorithms. One notable simplification is that general quantum mechanics works with complex numbers, but here we'll only use real numbers.

Second, this may seem like an abstract mathematical game. But experiments tell us that there are physical systems that really do behave in this way!

2.1 Just one qubit

Just as the most basic element of a classical computer is the *bit* (which is just a zero or one), the elements from which a quantum computer is built are quantum bits, or *qubits*. A qubit is at least as interesting as a classical bit, because it can also be in a state of zero or one. We write the “zero” state as $|0\rangle$, and the “one” state as $|1\rangle$.

But it turns out that a qubit is a little more exciting than a classical bit: it can be, in some sense, both zero and one at the same time! The notation we have for this is simple. An arbitrary state of a qubit is written as

$$a|0\rangle + b|1\rangle,$$

for some numbers a and b , where $a^2 + b^2 = 1$. This is called a *superposition* of states $|0\rangle$ and $|1\rangle$. There are no other restrictions on a and b ; in particular, they can be negative. So the following are examples of valid states of a qubit.

$$|0\rangle, -|1\rangle, \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, -\frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle.$$

So a qubit can be in any of an infinite number of different states. But, surprisingly, when we look at it – which we call *measuring* the qubit – we only ever see one of two states: zero or one. The probability of measuring 0 is a^2 , and the probability of measuring 1 is b^2 . As $a^2 + b^2 = 1$, we always see either zero or one when we measure. More surprisingly, when we perform further measurements on the qubit, we always get the same answer.

For example, imagine that we have a qubit in the state

$$-\frac{1}{2}|0\rangle + \frac{\sqrt{3}}{2}|1\rangle.$$

Let’s say we measure it and get the answer 1 (as we will with probability $3/4$). Now any further measurements will also give the answer 1. The qubit is said to have *collapsed* to the state $|1\rangle$.

2.2 Operations on a qubit

So far, it’s hard to see what qubits are good for. But things start changing when we introduce *operations* that can be performed on a qubit. These are known as *quantum gates*, by analogy with the logic gates in a classical electronic circuit. A quantum circuit is thus a sequence of quantum gates. We can write these gates in terms of what they do to the state $|0\rangle$, and what they do to the state $|1\rangle$. For example, consider the classical NOT gate that maps 0 to 1, and 1 to 0. We can write this action as

$$|0\rangle \mapsto |1\rangle, |1\rangle \mapsto |0\rangle.$$

So what does NOT do to a more general state of a qubit, $a|0\rangle + b|1\rangle$? We can work this out using a basic principle of quantum mechanics: all operations that can be performed on quantum states are *linear*. This means that, for any operation U that we can perform on a qubit,

$$U(a|0\rangle + b|1\rangle) = aU|0\rangle + bU|1\rangle.$$

So, in particular,

$$\text{NOT}(a|0\rangle + b|1\rangle) = a\text{NOT}(|0\rangle) + b\text{NOT}(|1\rangle) = a|1\rangle + b|0\rangle,$$

which is exactly what we’d expect. The NOT gate is in fact the only non-trivial operation that can be performed on a classical bit. However, in the quantum world we have considerably more freedom. For example, we could perform the following operation.

$$|0\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle, |1\rangle \mapsto \frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle.$$

Circuit diagrams

We can express quantum gates graphically using *circuit diagrams*, which are based on classical circuit diagrams. Each wire represents a qubit, and blocks represent gates acting on qubits. Diagrams are read left to right.

The following diagram illustrates the NOT gate.

$$a|0\rangle + b|1\rangle \text{ --- } \boxed{\text{NOT}} \text{ --- } b|0\rangle + a|1\rangle$$

This operation is known as the *Hadamard gate* H [5], and will be important for the quantum algorithm that we discuss in the next section. Playing with the Hadamard gate illustrates some important features of quantum computing.

For example, let's say we start with a qubit in state $|0\rangle$, and apply a Hadamard gate. We are now in the state

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

If we were to measure this qubit now, we would get 0 or 1 with equal probability – so the Hadamard gate has acted like a fair coin. But what if we don't measure, but instead apply another Hadamard gate? We get

$$H^2|0\rangle = \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \right) + \frac{1}{\sqrt{2}} \left(\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle \right) = |0\rangle,$$

so we are back where we started! Note that the reason that this happened is that some of the terms in the sum cancelled out – this is a fundamental property of quantum mechanics known as *interference*.

So qubits seem more powerful than classical bits. However, there are some restrictions on the operations which we can perform on them which make them appear weaker in some respects, too. Consider the basic operation of “zeroing” a bit: mapping $0 \mapsto 0$, $1 \mapsto 0$. We can write down a “quantum” gate (call this Z) which performs

$$|0\rangle \mapsto |0\rangle, |1\rangle \mapsto |0\rangle,$$

which looks OK. But what happens when we input a more general quantum state to this operation?

$$Z(a|0\rangle + b|1\rangle) = aZ|0\rangle + bZ|1\rangle = (a + b)|0\rangle.$$

We need $(a + b)^2 = 1$ for this to be a valid quantum state, but this will only be true if one or other of a and b is 0 (remember $a^2 + b^2 = 1$). So the Z operation isn't a valid quantum gate. In general, it turns out that we can classify which quantum gates are valid and which are not by making a correspondence to so-called *unitary matrices* [12], but we won't discuss these further here.

It also turns out that, although not every valid classical operation can be implemented directly on a quantum computer, any classical circuit can be efficiently encoded in such a way that it *can* be implemented as a quantum circuit [11]. Thus quantum computers are at least as powerful as classical computers.

2.3 More than one qubit

If we have several bits, we can combine them into a bit-string like 01101. Similarly, qubits can be combined by writing their states one after the other. Let's focus on the case where we have two qubits. The following are examples of valid states.

$$|0\rangle|0\rangle, |0\rangle|1\rangle, |1\rangle|0\rangle, |1\rangle|1\rangle.$$

Just as with one qubit, the pair of qubits can be in a superposition of these states. In fact, for any numbers a, b, c, d with $a^2 + b^2 + c^2 + d^2 = 1$, the state

$$a|0\rangle|0\rangle + b|0\rangle|1\rangle + c|1\rangle|0\rangle + d|1\rangle|1\rangle$$

is allowed.

Of course, we can apply a single-qubit quantum gate to one, or other, or both of the qubits. For example, if we apply the NOT gate to the first qubit of the above state, we get

$$a|0\rangle|0\rangle + b|0\rangle|1\rangle + c|1\rangle|0\rangle + d|1\rangle|1\rangle \mapsto c|0\rangle|0\rangle + d|0\rangle|1\rangle + a|1\rangle|0\rangle + b|1\rangle|1\rangle.$$

We can also define quantum gates that act on more than one qubit. The only such gate we'll need here is one that we can think of as a “function evaluation” gate, F . This will allow us to implement

on a quantum computer any classical function that maps a bit to another bit. Given any such function $f(x)$, where x is 0 or 1, F behaves as follows.

$$|x\rangle|y\rangle \mapsto |x\rangle|y \oplus f(x)\rangle,$$

where x and y are each 0 or 1. \oplus here means the “exclusive or” operator, which acts as follows.

y	$f(x)$	$y \oplus f(x)$
0	0	0
0	1	1
1	0	1
1	1	0

For example, consider the classical function $f(x) = x$. Then F performs

$$|0\rangle|0\rangle \mapsto |0\rangle|0\rangle, |0\rangle|1\rangle \mapsto |0\rangle|1\rangle, |1\rangle|0\rangle \mapsto |1\rangle|1\rangle, |1\rangle|1\rangle \mapsto |1\rangle|0\rangle.$$

An exercise for the reader: why didn't we define F as $|x\rangle|y\rangle \mapsto |x\rangle|f(x)\rangle$?

3 Back to Meal or No Meal

We finally have enough background knowledge to understand the quantum algorithm that solves the *Meal or No Meal* problem from Section 1. Let's express it mathematically.

Problem. Consider an arbitrary function f which takes as input 0 or 1 (“select first or second saucepan”), and returns 0 or 1 (“no chicken or chicken in saucepan”) for each input. Output 1 if f takes the value 1 on exactly one input.

It's easy to see that there are 4 different possible functions of this form. We can write down what we need to output for each such function as follows.

$f(0)$	$f(1)$	Required output
0	0	0
0	1	1
1	0	1
1	1	0

So this is just the “exclusive or” function $f(0) \oplus f(1)$ from the previous section. Our quantum algorithm for this problem will compute $f(0) \oplus f(1)$ with one use of f , and will use two qubits. For readability in what follows, we'll define $\bar{x} = \text{NOT}(x) = 1 \oplus x$, where x is 0 or 1.

3.1 The quantum algorithm

In words, the algorithm is described by the following steps. It's easiest to understand what's going on by following each step with pen and paper.

1. **Action:** Initialise the first qubit to 0, and the second to 1.

State of computer: $|0\rangle|1\rangle$

2. **Action:** Apply Hadamard gates to both qubits.

New state: $\frac{1}{2}(|0\rangle + |1\rangle)(|0\rangle - |1\rangle) = \frac{1}{2}(|0\rangle|0\rangle - |0\rangle|1\rangle + |1\rangle|0\rangle - |1\rangle|1\rangle)$

3. **Action:** Evaluate the function f using the gate F of the previous section.

New state: $\frac{1}{2}(|0\rangle|f(0)\rangle - |0\rangle|\overline{f(0)}\rangle + |1\rangle|f(1)\rangle - |1\rangle|\overline{f(1)}\rangle)$

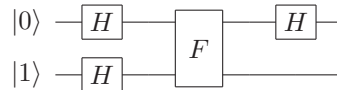
4. **Action:** Apply a Hadamard gate to the first qubit.

New state:

$$\begin{aligned} & \frac{1}{2\sqrt{2}} \left((|0\rangle + |1\rangle) |f(0)\rangle - (|0\rangle + |1\rangle) |\overline{f(0)}\rangle + (|0\rangle - |1\rangle) |f(1)\rangle - (|0\rangle - |1\rangle) |\overline{f(1)}\rangle \right) \\ &= \frac{1}{2\sqrt{2}} \left(|0\rangle \left(|f(0)\rangle - |\overline{f(0)}\rangle + |f(1)\rangle - |\overline{f(1)}\rangle \right) + |1\rangle \left(|f(0)\rangle - |\overline{f(0)}\rangle - |f(1)\rangle + |\overline{f(1)}\rangle \right) \right) \end{aligned}$$

5. **Action:** Measure the first qubit, and output the measurement result.

The following quantum circuit diagram is another way to describe the algorithm.



The claim is that the output is equal to $f(0) \oplus f(1)$ with certainty. Why is this true? If $f(0) \oplus f(1) = 0$, then $f(0) = f(1)$. This means that the sum

$$|f(0)\rangle - |\overline{f(0)}\rangle - |f(1)\rangle + |\overline{f(1)}\rangle = 0,$$

so the component of the state that has a $|1\rangle$ on the first qubit disappears completely. This means that the probability of getting a result of $\underline{1}$ when we measure the first qubit is zero! On the other hand, if $f(0) \oplus f(1) = 1$, then $f(0) = \overline{f(1)}$. This means that

$$|f(0)\rangle - |\overline{f(0)}\rangle + |f(1)\rangle - |\overline{f(1)}\rangle = 0,$$

so the probability of getting a measurement result of 0 is zero. We have therefore successfully evaluated $f(0) \oplus f(1)$ with certainty using only one call of f . This means that we can beat *Meal or No Meal* every time, and a billion dollars is ours.

This algorithm is known as *Deutsch's algorithm* [1]. Although it's simple, it illustrates a key property of quantum algorithms: we can design an algorithm that cancels out computational paths, leaving a desired result. An important extension of this algorithm, known as the Deutsch-Jozsa algorithm [2], was actually the first quantum algorithm to demonstrate an *exponential speed-up* over classical computers. This extended algorithm solves the problem of determining whether a function f from n bits to 1 bit is constant or balanced – i.e. takes the value 1 on exactly half of its inputs – in time of order n , while it is easy to see that no classical algorithm can solve the problem with certainty with fewer than 2^{n-1} queries to f .

4 So that's it?

You may have been left underwhelmed by the power of quantum computing so far. It's worth stressing, though, how surprising the result we've seen is: using the bizarre rules of quantum mechanics, we've achieved something that's provably impossible for computers that use only the physical principles by which we intuitively assume the world works.

However, you may be looking for some more practical applications of a quantum computer. The following quantum algorithms are examples of what is known so far; as more research is done, these will hopefully turn out to be the tip of the iceberg.

- Quantum computers can find the prime factors of a large integer quickly. Given an n -digit number $N = pq$, where p and q are prime, Shor's quantum algorithm can find p and q in time proportional to n^3 [10]. The best known classical algorithm, on the other hand, needs time proportional to about $2^{n^{1/3}}$ – considerably longer even for small n .

This may appear to be an esoteric problem, but in fact many important cryptographic protocols, such as the RSA encryption scheme that underlies Internet security [8], are based around the assumption that factoring large integers is hard. So, for example, a thief with a large-scale quantum computer could commit credit card fraud on a grand scale.

- Consider the following basic problem. Given an *unsorted* list of n names, check whether Joe Bloggs appears in the list. It seems clear that all n of the names have to be checked to say with certainty whether Joe is on the list. However, a famous result known as Grover's algorithm [4] says that a quantum computer can find Joe's name by looking at only about \sqrt{n} of the names. This algorithm extends to a vast number of applications: given any problem where there are n possible solutions to check, a quantum computer can find a solution with about \sqrt{n} tests.
- One of the first killer applications for a quantum computer is likely to be the simulation of quantum mechanical systems, which is becoming an essential part of modern solid-state physics, computational chemistry, and drug design. It is believed that a general-purpose quantum computer could simulate quantum systems exponentially more efficiently than a classical computer could.

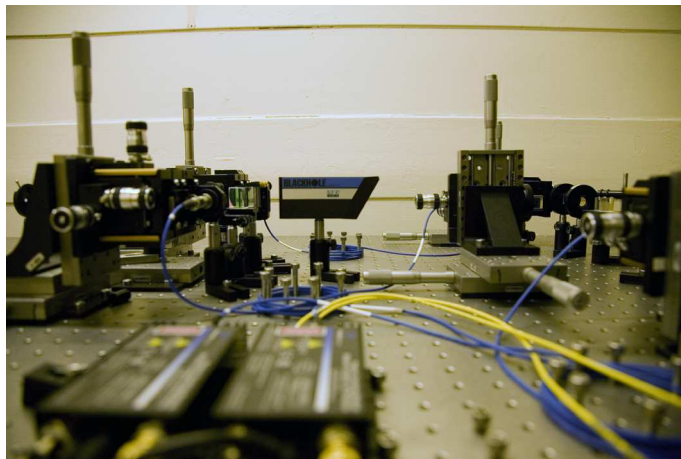
5 When will I have a quantum computer on my desk?

The field of quantum computing is in its infancy, and it's safe to say that nobody in the field knows for certain when – or even if – we'll have large-scale quantum computers. Predictions range from 10 years, to 50, to never. It's also unclear whether they will be general-purpose machines or only useful for specialised applications. Finally, it's not even known what physical form the components of a large-scale quantum computer will take. Possibilities include superconducting “quantum dots”, trapped ions, photons in optical lattices and even “nitrogen vacancy” centres in diamond [7].

However, there have been some encouraging small-scale experiments that suggest that the barriers to building a large quantum computer are engineering challenges rather than fundamental scientific impediments. Control has been demonstrated over small numbers of qubits (the record at the time of writing is 13 qubits), and both Shor's and Grover's algorithms have been demonstrated on tiny problem sizes (for example, the number 15 has been successfully factored as 3×5 ... a shocking revelation).

There's still a long way to go, but it's wise never to bet against technological progress. With this in mind, we'll finish with a relevant quotation:

“Computers in the future may weigh no more than 1.5 tons.”
– Popular Mechanics, 1949.



An optical quantum computing experiment at the University of Bristol.
(Photo: Carmel King, www.carmelking.com)

References

- [1] Wikipedia: Deutsch's algorithm
Available at http://en.wikipedia.org/wiki/Deutsch's_algorithm.
- [2] Wikipedia: Deutsch-Jozsa algorithm
Available at http://en.wikipedia.org/wiki/Deutsch-Jozsa_algorithm.
- [3] Wikipedia: Quantum entanglement
Available at http://en.wikipedia.org/wiki/Quantum_entanglement.
- [4] Wikipedia: Grover's algorithm
Available at http://en.wikipedia.org/wiki/Grover's_algorithm.
- [5] Wikipedia: Hadamard transform
Available at http://en.wikipedia.org/wiki/Hadamard_transform.
- [6] Wikipedia: Interference.
Available at <http://en.wikipedia.org/wiki/Interference>.
- [7] Wikipedia: Quantum computing candidates
Available at http://en.wikipedia.org/wiki/Quantum_computer#Candidates.
- [8] Wikipedia: RSA
Available at <http://en.wikipedia.org/wiki/RSA>.
- [9] Wikipedia: Schrodinger's cat
Available at http://en.wikipedia.org/wiki/Schrodinger's_cat.
- [10] Wikipedia: Shor's algorithm
Available at http://en.wikipedia.org/wiki/Shor's_algorithm.
- [11] Wikipedia: Toffoli gate
Available at http://en.wikipedia.org/wiki/Toffoli_gate.
- [12] Wikipedia: Unitary matrix
Available at http://en.wikipedia.org/wiki/Unitary_matrix.

Where next?

If you enjoyed this introduction to quantum computing, there are many other resources freely available on the Internet which you might find interesting. Though beware that popular science treatments of quantum computing are frequently inaccurate and overhyped!

- The main Wikipedia page about quantum computing is a good place to start:

http://en.wikipedia.org/wiki/Quantum_computer

- For the slightly braver reader, the lecture notes from the 4th year course “Quantum Computation” at Bristol University are freely available online:

<http://www.cs.bris.ac.uk/Teaching/Resources/COMSM0214/>

- Scott Aaronson's series of lecture notes “Quantum Computing since Democritus” is a good informal introduction to some of the philosophical issues around quantum computing.

<http://www.scottaaronson.com/democritus/>

- One of the most appealing aspects of quantum computing research is that almost all significant papers on the subject are freely available via the pre-print server arxiv.org. For example, the original papers introducing the famous algorithms of Shor and Grover are available. Start searching at

<http://arxiv.org/find/quant-ph>

- There isn't too much mathematical background required to understand quantum computing. But one subject well worth learning is *linear algebra*. A free linear algebra textbook is available online at

<http://joshua.smcvt.edu/linearalgebra/>

Frequently Asked Questions (FAQ)

This document represents one part of a larger series which attempts to answer the question “What Is Computer Science?”. The intended readership isn’t people who already know the answer to this question !

Why are you doing this ? Recruiting students to do Computer Science degrees is quite hard work. Partly this is because school subjects such as ICT often give the wrong impression of what Computer Science is about at University. This document is an attempt to explain what Computer Science is and motivate you to be interested in it. Of course, summing up such a broad subject is difficult; our approach is to showcase specific topics, demonstrating how they relate to other subjects (such as Mathematics) and how they solve real-world problems.

I’ve got a question or comment. We’d welcome any feedback, experience or comment on these documents both as a way of improving and extending them; to get in contact, email

`uga@cs.bris.ac.uk`

Can I use these documents for something ? We are distributing these documents under the Creative Commons Attribution License

<http://creativecommons.org/licenses/by/2.0/uk/>

This is a fancy way to say you can basically do whatever you want with them (i.e. use, copy and distribute it however you see fit) **as long as** correct attribution of the original source is maintained.

Why are all your references to Wikipedia ? Our goal is to give an easily accessible overview of a topic, so it didn’t seem to make sense to reference lots of research papers. There are two reasons why: research papers are often written in a way which makes them hard to read, and although many research papers are available on the Internet, many are not (or have to be paid for). So although there are some valid criticisms of Wikipedia, for introductory material on Computer Science the articles seem a good place to start.

Why don’t the examples include much programming ? Our goal is to focus on interesting concepts rather than the mechanics of programming. So even when we include example programs, the idea is to do so in a way where the meaning is fairly clear. For example we’d rather use pseudo-code algorithms or reuse existing software tools than complicate a description by including pages of C program. Even more attractive are interactive environments, for example BASH and Python shells, which allow easy demonstration of intermediate results of small programming steps rather than the monolithic result of a large program.

Some material in this document duplicates others from the series ! This is true, but isn’t because we are lazy ! Each document is intended to be fairly self contained; obviously there will be some overlap between topics so there will inevitably be some repetition.