Quantum Computers

Peter Shor
MIT

What is the difference between a computer and a physics experiment?

One answer:

A computer answers mathematical questions.
A physics experiment answers physical questions.

Another answer:

A physics experiment is a big, custom-built, finicky, piece of apparatus.
A computer is a little box that sits on your desk (or in your briefcase).

A third answer:

You don't need to build a new computer for each mathematical question you want answered.

The mathematical theory of computing started in the 1930's
(before computers)

After Gödel proved his famous incompleteness theorem, it was followed by four papers giving a distinction between computable and uncomputable functions
(Church, Turing, Kleene, Post, ca. 1936)

These papers contained three definitions of computable functions which looked quite different.

Universality of computers I.

This led Church and Turing to propose

Church-Turing thesis:

A Turing machine can perform any computation that any (physical) device can perform.

(Turing, Church, ca. 1936).

Until recently, it was not generally realized this is a statement about physics, and not about mathematics.

With the development of practical computers, the distinction between uncomputable and computable become much too coarse.

To be practical, a program must return an answer in a reasonable amount of time (minutes? days? years?).

Theoretical computer scientists consider an algorithm *efficient* if its running time is a polynomial function of the size n of its input. (n^2, n^3, n^4, etc.)

The class of problems solvable with polynomial-time algorithms is called P

This is a reasonable compromise between theory and practice.

For the definition of P (polynomial-time solvable problems) to be meaningful, you need to know that it doesn't depend on the exact type of computer you use.

Universality of computers II.

This led various computer scientists to propose the

Quantitative Church's Thesis

A Turing machine can perform *efficiently* any computation that any (physical) device can perform efficiently.

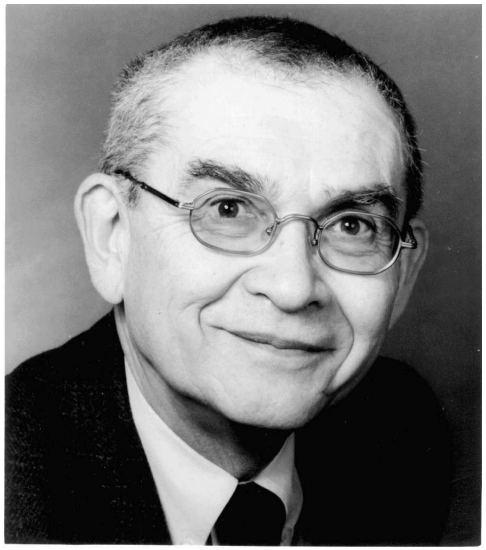
(ca. 1970).

If quantum computers can be built, this would imply this "folk thesis" is probably not true.

Misconceptions about Quantum Computers

False: Quantum computers would be able to speed up all computations.

Quantum computers are not just faster versions of classical computers, but use a different paradigm for computation. They would speed up some problems by large factors and other problems not at all.



Richard Feynman Yuri Manin

- Simulating physics using a digital computer seems inherently exponentially inefficient. (R. P. Poplavskii, 1975; Feynman, 1982)
- A "quantum computer" might be able to get around this problem. (Manin, 1980; Feynman, 1982)

David Deutsch

In 1985, he asked whether quantum computers might speed up computation.

Problems with potential speed-ups by quantum computers were found by:

David Deutsch and Richard Jozsa (1992)

André Berthiaume and Gilles Brassard (1992)

Ethan Bernstein and Umesh Vazirani (1993)

Dan Simon (1993)

Peter Shor (1994)

Lov Grover (1996)

What do we know quantum computers are good for?

- Simulating/exploring quantum mechanical systems efficiently.
  [Richard Feynman/Yuri Manin]
  Estimating zeros of certain Riemann zeta functions [van Dam]

- Finding periodicity.
  Simon's problem [Dan Simon]
  Factoring large integers and finding discrete logarithms efficiently [PWS]
  Pell's equation and class groups [Sean Hallgren].

- Searching large solution spaces more efficiently [Lov Grover]
  Amplifying the success probably of (quantum) algorithms with small success probabilities.
  Finding solutions by walking on large graphs more efficiently

Searching

Quantum computers give a quadratic speed-up for exhastive search problems (Lov Grover). Looking through N possibilities takes

- expected time $N/2$ on a classical computer.
- expected time $\frac{\pi}{4}\sqrt{N}$ on a quantum computer.

Factoring

Quantum computers give an exponential speed up for factoring large integers.

Given a number N, find A,$B < N$ so

$$A * B = N$$

Factoring an L-bit number

Best classical method is the number field sieve (Pollard)
time: $\exp(cL^{1/3}(\log L)^{2/3})$.

Quantum factoring: theoretical asymptotic time bound
$cL^2(\log L)(\log\log L)$

Practical implications

Security on the Internet is based on public key cryptography.

The most widely used (and most trusted) public key cryptosystems are based on the difficulty of factoring and of finding discrete logarithms.

Both of these are vulnerable to attacks by a quantum computer.

What are the fundamental physical principles on which a quantum computer operates?

- The superposition principle
- High dimensionality of quantum state spaces
- Quantum interference
- Quantum entanglement

Models for quantum computation

To compute, we need to

- Put the input into the computer.
- Change the state of the computer.
- Get the output out of the computer.

There are several mathematical models for describing quantum computation. As is the case classically, they are all polynomially equivalent.

- Quantum circuit model (described below).
- Quantum Turing machine.
- Quantum cellular automata.
- Quantum adiabatic computing.
- Modular functors.
- Braiding of anyons in topological quantum field theories.

Input (for quantum circuit model)

Start the computer in the state corresponding to the input in binary, e.g.

$$|\,100101101\rangle\,.$$

We may need extra workspace for the algorithm. We then need to add 0s to the starting configuration.

$$|\,100101101\rangle \otimes |\,0000000000\rangle\,.$$

Output (for quantum circuit model) At the end of the computation, the computer is in some state

$$\sum_{i=0}^{2^k-1} \alpha_i \mid i\rangle$$

We can NOT measure the state completely, because of the Heisenberg uncertainty principle.

We assume that we measure in the canonical basis. We observe the output i with probability $|\alpha_i|^2$.

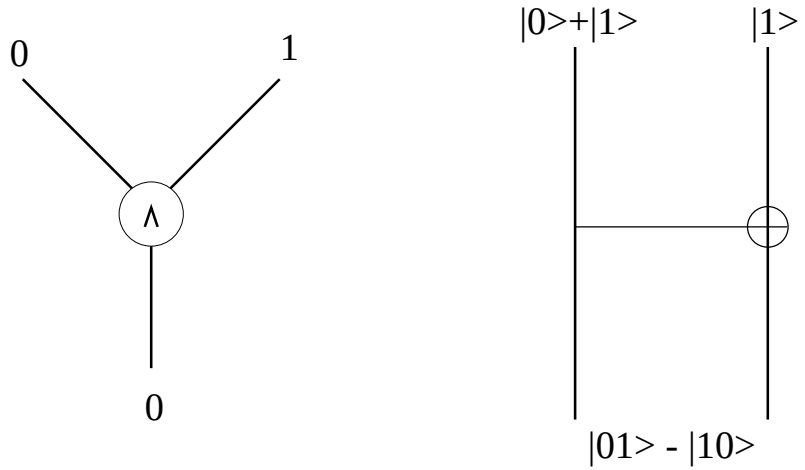
Output

When we observe the computer, we get a sample from a probability distribution.

Because of quantum mechanics, this is inherently a probabilistic process. We say that the computer computes a function correctly if we are able to get output that gives us the right answer with high probability.

Computation (in quantum circuit model)

Apply transformations to qubits two at a time.



Classical Gate Quantum Gate

By linearity of quantum mechanics, a one-qubit gate is a 2×2 unitary matrix and a two-qubit gate is a 4×4 unitary matrix.

A computation (program) is a sequence of quantum gates applied to one or two qubits at a time.

Why two?
Three doesn't give any more power, and seems more complicated experimentally.
Arbitrary many-qubit gates are hopeless, and theoretically realizable ones don't seem to add any extra computational power.

A quantum gate is thus a linear transformation on a 2-dimensional (1-qubit) or 4-dimensional (2-qubit) vector space.

It is thus a 2×2 or 4×4 matrix.

In order to preserve probabilities, it must take unit vectors to unit vectors. This means the matrix is *unitary*. That is, if G is the gate, $G^\dagger = G^{-1}$.

As in classical computing, you only need a finite set of quantum gates to perform arbitrary computation.

These gates will not let you simulate computations with an arbitrary set of gates exactly, but will let you approximate them arbitrarily closely.

How can you use quantum mechanical systems for computing?

You need 2^k numbers to describe the state of k quantum bits.

The Heisenberg uncertainty principle says you cannot measure the complete state of a quantum system.

It turns out you can only get k classical bits of information out of k quantum bits.
Therefore, does not increase the capacity of information storage.

Idea behind fast quantum computer algorithms:

Arrange the algorithm to make all the computational paths that produce the wrong answer destructively interfere, and the computational paths that produce the right answer constructively interfere, so as to greatly increase the probability of obtaining the right answer.

Simon's Problem

Given a function f mapping strings of length n to strings of length n, such that f is 2 to 1, and

$$\exists c \text{ such that } f(x) = f(x \oplus c) \forall x$$

Find c.

Classically, if f is a random such function, you have to query values of f until you have two such that $f(a) = f(b)$. Then,

$$c = a \oplus b.$$

This takes on average $\approx 2^{n/2}$ queries.

Hadamard gate

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

The amplitude for going from $|\, a\rangle$ to $|\, b\rangle$ [by this I mean the matrix element M_{ba}] is $(-1)^{ab} \frac{1}{\sqrt{2}}$.
The tensor product of n Hadamard gates gives an amplitude going from $|\, a\rangle$ to $|\, b\rangle$ of $(-1)^{a \cdot b} 2^{-n/2}$, where a and b are binary strings of length n.

Thus,

$$H^{\otimes n} \,|\, a\rangle = \frac{1}{2^{n/2}} \sum_{k=0}^{2^n - 1} (-1)^{a \cdot b} \,|\, b\rangle$$

Simon's Algorithm: Start with

$$2^{-n/2} \sum_{k=0}^{2^n-1} |\, k\rangle \,|\, 0\rangle$$

Compute $|\, f(k)\rangle$ in second register

$$2^{-n/2} \sum_{k=0}^{2^n-1} |\, k\rangle \,|\, f(k)\rangle$$

Take Hadamard transform of first register

$$2^{-n} \sum_{k=0}^{2^n-1} \sum_{l=0}^{2^n-1} (-1)^{k \cdot l} \,|\, l\rangle \,|\, f(k)\rangle$$

Observe state.

We had state

$$2^{-n} \sum_{k=0}^{2^n-1} \sum_{l=0}^{2^n-1} (-1)^{k \cdot l} \, | \, l \rangle \, | \, f(k) \rangle$$

A particular value of $f(k)$ will come from both k and $k \oplus c$. The chance you observe a particular value $| \, l \rangle \, | \, f(k \rangle)$ is

$$2^{-2n} \left((-1)^{k \cdot l} + (-1)^{(k+c) \cdot l} \right)^2$$

This is 0 unless $c \cdot l = 0$. Thus, the algorithm returns only values of l that are perpendicular to c, and it returns a random such value. If you have $n - 1$ such values of l, you can uniquely identify c.

This takes $O(n)$ queries, and $O(n^2)$ non-query steps.

Idea Behind All Fast Factoring Algorithms

To factor a large number N, Find numbers a and b so that

$$a^2 = b^2 \mod \text{N}$$

$$a \neq \pm b \mod \text{N}$$

Then

$$a^2 - b^2 = (a+b)(a-b) = cN$$

We now extract one factor from $a+b$ and another from $a-b$.

Example: Factoring 33

Take the numbers $a = 7$ and $b = 4$. Then 49 divided by 33 has remainder 16, so

$$7^2 = 4^2 \mod 33$$

Then

$$7^2 - 4^2 = (7 + 4)(7 - 4) = 33$$

and we find $33 = 3 * 11$.

Quantum Factoring Idea

To factor a large number N:

Find the smallest $r > 0$ such that $x^r \equiv 1 \pmod{N}$.

$$(x^{r/2} + 1)(x^{r/2} - 1) \equiv 0 \pmod{N}.$$

We now get two factors by taking the greatest common divisors

$$\gcd(x^{r/2} + 1,\ N)$$

$$\gcd(x^{r/2} - 1,\ N)$$

We can show this gives a non-trivial factor for at least half of the residues $x \pmod{N}$.

How do we find r with

$$x^r \equiv 1 \pmod{N}?$$

Find the period r of the sequence $x^a \pmod{N}$.

Example: Factoring 33

Take $x = 5$. Then (mod 33) we get

1	5	5^2	5^3	5^4	5^5	5^6	5^7	5^8	5^9	5^{10}	5^{11}
1	5	25	26	31	23	16	14	4	20	1	5

The period r is 10, and

$$x^{r/2} = 5^5 = 23 \mod 33.$$

Then

$$33 \text{ divides } (23+1)(23-1) = 24 * 22$$

Taking greatest common divisors, 24 gives us the factor 3, and 22 gives us the factor 11, and we find $33 = 3 * 11$.

Need to find the period of $x^a \pmod{N}$.

Idea: Use the Fourier transform

Problem: The sequence has an exponentially long period

Solution: Use the exponentially large state space of a quantum computer to take an exponentially large Fourier transform efficiently.

Factoring L-bit numbers

We will work with quantum superpositions of two registers

Register 1	Register 2
$2L$ bits	L bits

We will not give the fine details of the algorithms.

These involve more workspace
($3L$ is easy, $o(L)$ is possible).

Quantum Fourier Transform over Z_{2^k}

Have k qubits

$$|\, x\rangle \rightarrow \frac{1}{2^{k/2}} \sum_{y=0}^{2^k-1} \exp(\frac{2\pi i x y}{2^k}) \,|\, y\rangle$$

This is easily seen to be a unitary transformation on the 2^k-dimensional space of k qubits.

In order to implement this on a quantum computer, we need to break this into a series of 2-qubit gates.

The Cooley-Tukey FFT easily adapts to give $O(k^2)$ steps.

Reversible Computation

We can do classical computations on a quantum computer as long as we can do these classical computations *reversibly*. That is, with gates each of whose possible outputs maps uniquely back to the inputs.

Any classical computation can be made reversible as long as we keep the input around.

The 3-bit *Toffoli gate* is a universal gate for reversible computation

$$\Big(x, y, z\Big) \rightarrow \Big(x, y, z \oplus (x \wedge y)\Big)$$

This Toffoli gate can be implemented as a sequence of 2-qubit quantum gates.

$$|\,0\rangle\,|\,0\rangle$$

$$\downarrow \quad \approx L \text{ steps}$$

$$\frac{1}{2^L}\sum_{a=0}^{2^L} |\,a\rangle\,|\,0\rangle$$

$$\downarrow \quad \approx L^2 \log L \log\log L \text{ steps}$$

$$\frac{1}{2^L}\sum_{a=0}^{2^L} |\,a\rangle\,|\,x^a \pmod N\rangle$$

$$\downarrow \quad \approx L^2 \text{ steps}$$

$$\frac{1}{2^L}\sum_{a=0}^{2^L}\sum_{c=0}^{2^L} |\,c\rangle\,|\,x^a \pmod N\rangle \;\; e^{2\pi i ac/2^{2L}}$$

Observe computer.

We need to find the probability amplitude on

$$|\, c\rangle\, |\, x^a \pmod N\rangle$$

in the superposition

$$\frac{1}{2^L} \sum_{a=0}^{2^L} \sum_{c=0}^{2^L} |\, c\rangle\, |\, x^a \pmod N\rangle \;\; e^{2\pi i a c / 2^{2L}}$$

Many different values of a give the same value of $x^a \pmod N$.

We have to add the coefficients $e^{2\pi i a c / 2^{2L}}$ on all of them.

Let a_0 be the smallest non-negative integer such that

$$x^{a_0} \equiv x^a (\text{mod } N).$$

Then x^{a_0}, x^{a_0+r}, x^{a_0+2r}, ... are all equal (mod N).

Each contributes to the amplitude on

$$|\, c\rangle \, |\, x^a \ (\text{mod } N)\rangle$$

with the coefficient $e^{2\pi i(a_0+br)c/2^{2L}}$.

We observe $|\, c\rangle$ with probability proporitional to

$$\left| \sum_{b=0}^{\approx 2^{2L/r}} e^{2\pi i brc/2^{2L}} \right|^2$$

This is a geometric sum which is close to 0 unless, for some integer d.

$$\frac{rc}{2^{2L}} = d + O(r/2^{2L})$$

We know

$$\frac{rc}{2^{2L}} = d + O(r/2^{2L}).$$

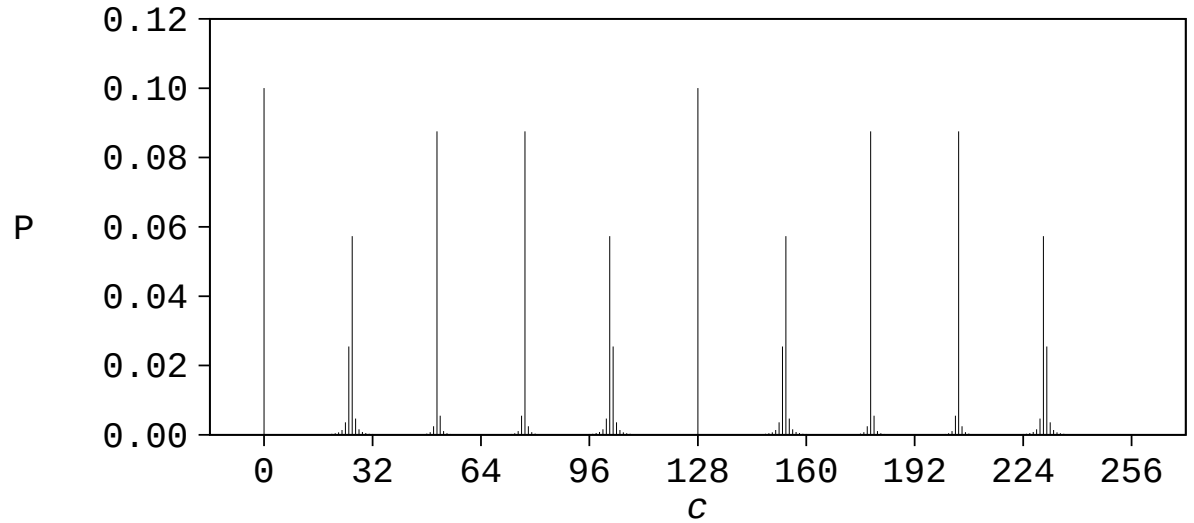
Thus

$$\frac{c}{2^{2L}} = \frac{d}{r} + O\left(\frac{1}{N^2}\right).$$

with $r < N$.

$\frac{d}{r}$ will be one of the closest fractions to $\frac{c}{2^{2L}}$ with numerator and denominator less than N.

We can use continued fractions to find $\frac{d}{r}$, and then use r to factor N.



Example: Factoring 33

The period r is 10.

Proposals for building quantum computers:

- Ion traps
  (Cirac and Zoller, 1995)
- NMR (nuclear magnetic resonance)
  (Cory et al., Gershenfeld et al., 1997)
- Nuclear spins on silicon chips
  (Kane, 1998)
- Superconducting states on silicon chips
  (Devoret, 2002)

NMR Proposal for Quantum Computers

The qubits are the spins of the nuclei of atoms in a large molecule.

These can be manipulated using standard NMR (Nuclear Magnetic Resonance) techniques.

The computing is done with 10^{20} computers (molecules) all at once, all doing the same computation.

Difficulty with NMR Proposal

Currently, NMR quantum computers are initialized by using a thermal state. The computer is put it in its initial state by using the fact that in a strong magnetic field, very slightly more of the nuclear spins are pointing down.

This initialization method means that the signal strength goes down by a factor of two each time you add a qubit. This method will break down somewhere between 10 and 20 qubits. To improve this, you need some good way of cooling these nuclear spins, so most are pointing in the same direction.

Similarly, to correct errors during computation, you need some way of getting the incorrect qubits out of the system. This essentially requires cooling some nuclear spins during the computation.

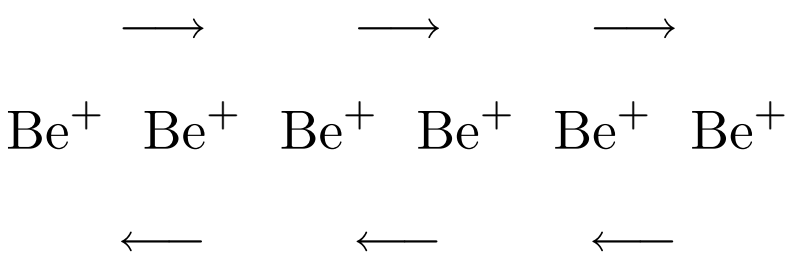
Ion Trap Proposal for Quantum Computers

Experimentalists can use electromagnetic fields to trap a line of ions in a low-temperature vacuum.

$$\mathrm{Be}^+ \quad \mathrm{Be}^+ \quad \mathrm{Be}^+ \quad \mathrm{Be}^+ \quad \mathrm{Be}^+ \quad \mathrm{Be}^+$$

Proposal: Use the ground state and an excited state of the ions for qubits in a quantum computer.
The ions should be far enough apart so you can manipulate their states by shining laser pulses on individual ions. This lets you do one-qubit gates.

The ions can be made to interact, giving two-qubit gates, by using a shared vibrational mode of the trap.



Difficulty with ion trap proposal:

It is very difficult to make ion traps do what you want.

So far, at NIST, David Wineland has managed to put an ion trap containing four ions into the ground state, and do a few computations.

Conclusions

- Quantum computers have the potential for enormous speed-up of certain problems.
- Need good new idea to build large (100- to 1000- qubit) quantum computers.
- Good new ideas appear to be coming regularly.