Quantum Computing

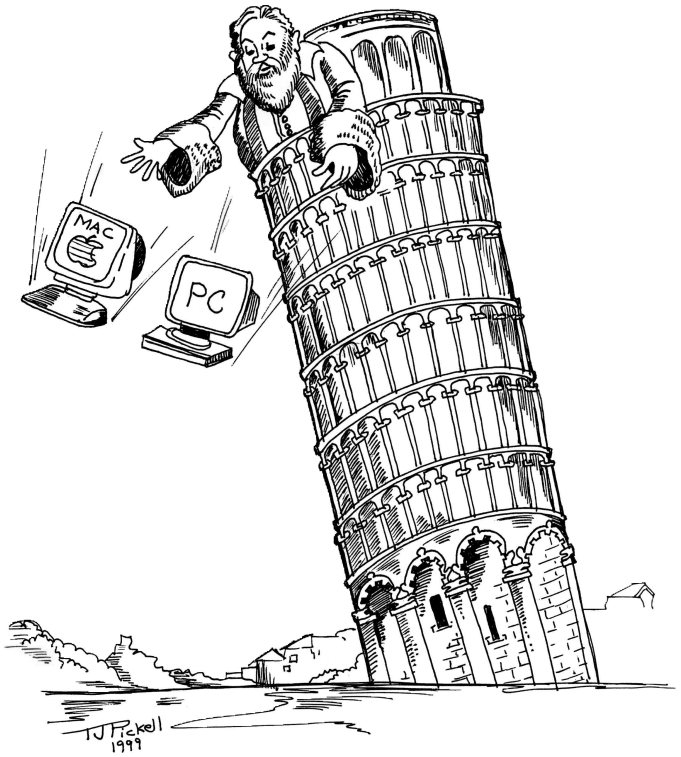
Peter Shor
M.I.T.
Cambridge, MA

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

One answer:

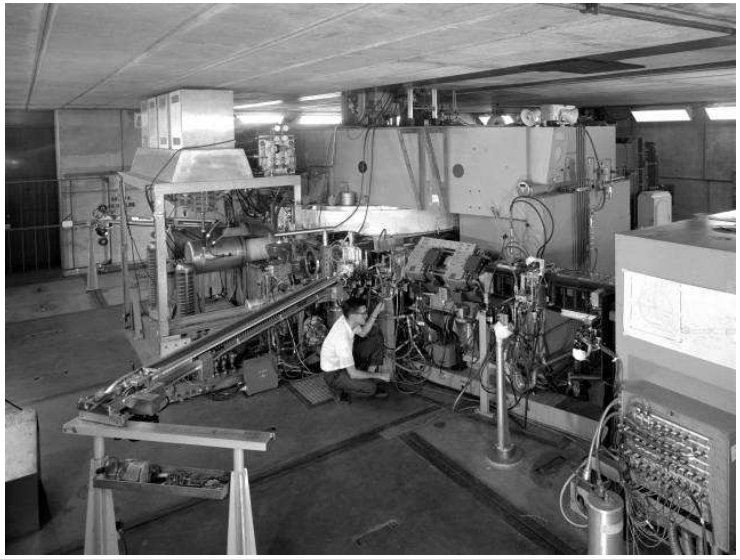
A computer answers mathematical questions.
A physics experiment answers physical questions.

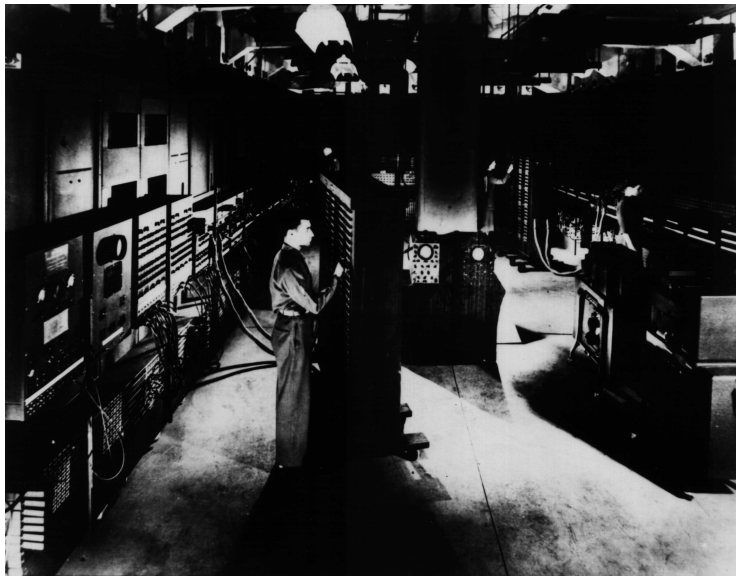
Using computers to test whether two bodies fall at the same rate



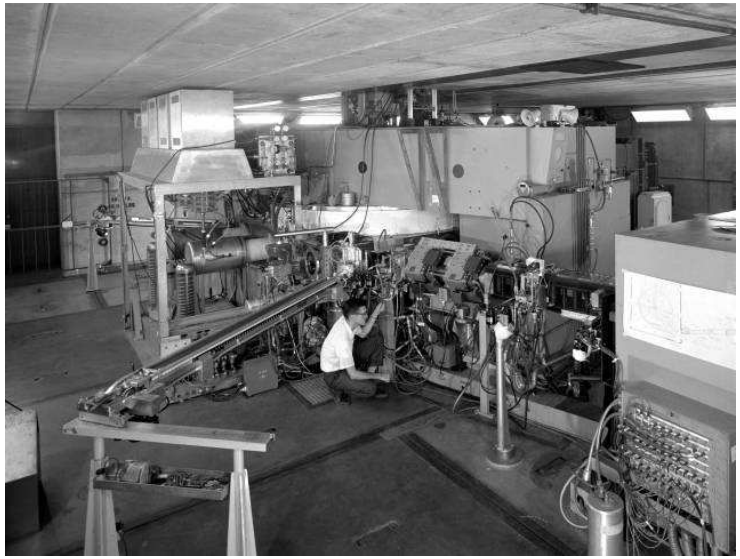
Another answer:

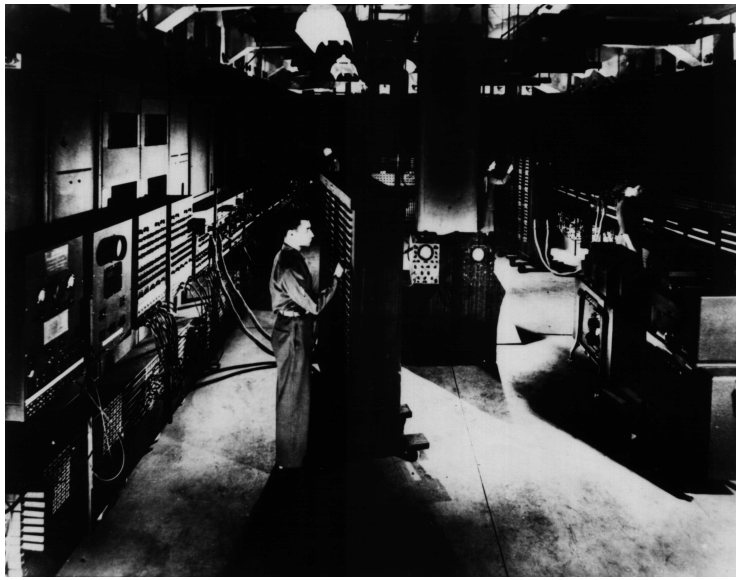
A physics experiment is a big, custom-built, finicky, piece of apparatus.
A computer is a little box that sits on your desk (or in your briefcase).





Physics experiment





Computer

Another 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 computation I.

Church-Turing thesis:

A Turing machine can perform any computation that any (physical) device can perform.
(Turing, Church, ca. 1936).

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

To be practical, a program must compute a function in a reasonable amount of time (in years, at the longest).

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 computation revisited.

This led various computer scientists to propose the

Quantitative Church's Thesis (Cobham)

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

(Various theoretical computer scientists, 1960's).

If quantum computers can be built, this would imply this "folk thesis" is 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.

They would speed up some problems by large factors and other problems not at all.

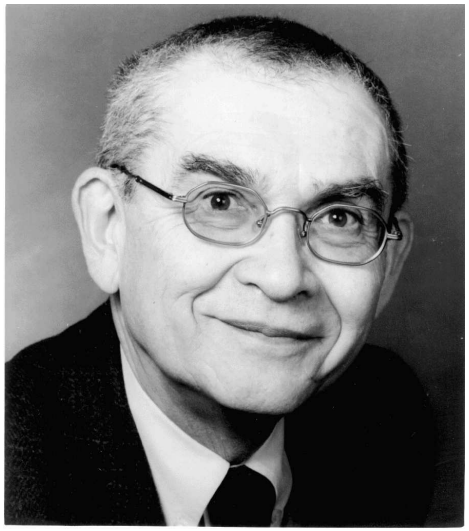
The fact that this misconception is so widespread shows that the public has absorbed the Quantitative Church's Thesis.

A single step on a quantum computer is almost certain to take longer than a single step on a classical computer. Quantum computers speed up computations by drastically reducing the number of steps needed.

Where would you look for counterexamples for the Quantitative Church-Turing thesis?

Maybe in physical systems that are hard to simulate.

Richard Feynman



Yuri Manin

- Simulating physics using a digital computer seems inherently exponentially inefficient. (Feynman, 1982, R. P. Poplavskii, 1975)

- A "quantum computer" might be able to get around this problem. (Feynman, 1982; Manin, 1980)

In 1985, David Deutsch asked whether quantum computers might speed up computation for non-quantum mechanics problems.

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)

What do we know quantum computers are good for?

- Simulating/exploring quantum mechanical systems efficiently. [Richard Feynman/Yuri Manin]

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

- Searching large solution spaces more efficiently [Lov Grover]
  Amplifying the success probability of (quantum) algorithms with small success probabilities.

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 computer (Shor)
time $cL^2(\log L)(\log\log L)$

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?

This is a difficult question, as quantum computers seem to much of the structure of quantum mechanics. They use:

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

The Superposition Principle:

If a quantum system can be in one of two mutually distinguishable states $|\,A\rangle$ and $|\,B\rangle$, it can be both these states at once. Namely, it can be in the *superposition* of states

$$\alpha\,|\,A\rangle + \beta\,|\,B\rangle$$

where α and β are complex numbers and $|\alpha|^2 + |\beta|^2 = 1$.

If you look at the system, the chance of seeing it in state $|\,A\rangle$ is $|\alpha|^2$ and in state $|\,B\rangle$ is $|\beta|^2$.

The Superposition Principle (in mathematics)

Quantum states are represented by unit vectors in a complex vector space.

Multiplying a quantum states by a unit complex phase does not change the essential quantum state.

Two quantum states are *distinguishable* if they are represented by orthogonal vectors.

A *qubit* is a quantum system with 2 distinguishable states, i.e., a 2-dimensional state space.

If you have a polarized photon, there can only be two distinguishable states, for example, vertical $|\updownarrow\rangle$ and horizontal $|\leftrightarrow\rangle$ polarizations.

All other states can be made from these two.

$$|\nearrow\rangle = \frac{1}{\sqrt{2}}|\leftrightarrow\rangle + \frac{1}{\sqrt{2}}|\updownarrow\rangle \qquad |\nwarrow\rangle = \frac{1}{\sqrt{2}}|\leftrightarrow\rangle - \frac{1}{\sqrt{2}}|\updownarrow\rangle$$

$$\left|\circlearrowright\right\rangle = \frac{1}{\sqrt{2}}|\leftrightarrow\rangle + \frac{i}{\sqrt{2}}|\updownarrow\rangle \qquad \left|\circlearrowleft\right\rangle = \frac{1}{\sqrt{2}}|\leftrightarrow\rangle - \frac{i}{\sqrt{2}}|\updownarrow\rangle$$

If you have two qubits, they can be in any superposition of the four states

$$|00\rangle \quad |01\rangle \quad |10\rangle \quad |11\rangle$$

This includes states such as

$$\frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$$

where neither qubit alone has a definite state.

Such states are called *entangled*.

If you have n qubits, their joint state can be described by a superposition of 2^n basis states.

These basis states can be taken to be:

$$|000\ldots00\rangle \quad |000\ldots01\rangle \quad \cdots \quad |111\ldots11\rangle$$

The high dimensionality of this space is one of the places where quantum computing obtains its power.

The "circuit model" 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.

Input

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.

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

(Alternatively we may permit an operation which adds additional qubits in the middle of the computation.)

Output

At the end of the computation, the computer is in some state

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

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

We assume we measure in the canonical basis $|\,000...00\rangle$, $|\,000...01\rangle$, ... $|\,111...11\rangle$

We observe the output $|\,i\rangle$ 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.

The Linearity Principle

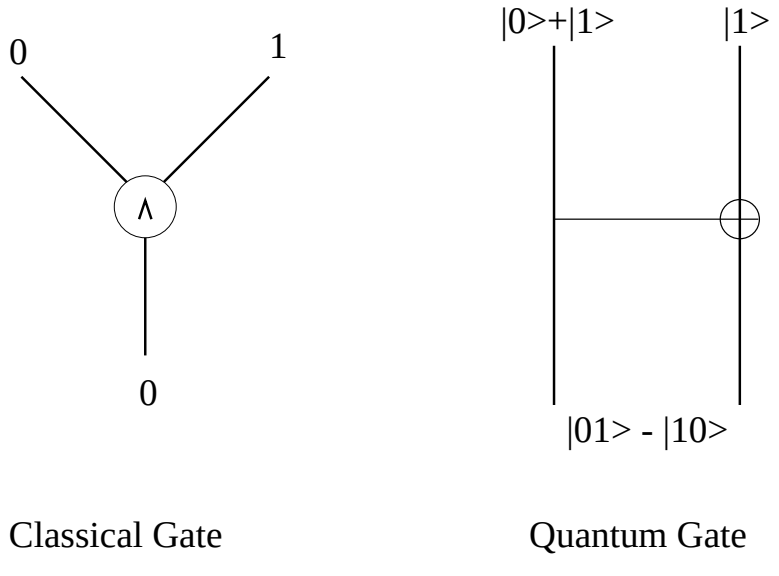
The evolution of an *isolated* quantum system is linear.

Because linear transformations can be represented as a matrix, the evolution of pure states in an isolated quantum system can be described by a matrix operating on the state space.

To preserve probabilities, the matrices must be unitary.

Computation

Apply transformations to qubits two at a time.



Classical Gate Quantum Gate

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

Why at most two qubits at a time?

If we allowed unitary gates that transformed all the qubits, then we could choose a gate that took the input to the desired output in one step, and experimental physicists would have no clue as to how to implement it. This wouldn't be helpful.

Three-qubit gates are theoretically not any more powerful than two-input gates (they reduce computation time by a constant in general), and much harder to implement experimentally.

Quantum Gates

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

It is thus a 2×2 or 4×4 matrix.

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

Quantum Gates

How does a quantum gate on two qubits (4×4 matrix) operate on the quantum state space of n qubits (a 2^n-dimensional vector).

You have to take the tensor product of the quantum gate on those two qubits with the identity matrix on the remaining qubits.

Interference

Because superpositions of states can have complex coefficients, you can make qubits interfere with themselves.

Applying the transformation

$$\begin{aligned} |\,0\rangle &\rightarrow \frac{1}{\sqrt{2}}(|\,0\rangle + |\,1\rangle) \\ |\,1\rangle &\rightarrow \frac{1}{\sqrt{2}}(|\,1\rangle - |\,0\rangle) \end{aligned}$$

twice takes $|\,0\rangle \rightarrow |\,1\rangle$ and $|\,1\rangle \rightarrow -|\,0\rangle$, since the $|\,0\rangle$ terms in the result cancel out.

This is written in matrices as $\begin{pmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}^2 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$.

Without interference, a quantum computer can be simulated by a digital computer with a random number generator.

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.

This is generally very difficult to figure out how to accomplish, and this may account for the fact that so few quantum algorithms have been discovered.

Idea Behind All Fast Factoring Algorithms

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

$$a^2 = b^2 \mod N$$

$$a \neq \pm b \mod N$$

Then

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

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

We can use Euclid's algorithm for greatest common divisors to find the factors; the greatest common divisor of $a + b$ and N will be one factor.

Example: Factoring 33

Take the numbers $a = 10$ and $b = 1$. Then 100 divided by 33 has remainder 1, so

$$10^2 = 1^2 \mod 33$$

Then

$$10^2 - 1^2 = (10 + 1)(10 - 1) = 33.$$

The first factor gives 11, since $\gcd(11, 33) = 11$;
the second gives 3, since $\gcd(9, 33) = 3$.

Thus, we find $33 = 11 * 3$.

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

$$33 \text{ divides } (5^5 + 1)(5^5 - 1) = (23 + 1)(23 - 1) = 24 * 22$$

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
($3L$ workspace is easy, $\ll L$ workspace is possible).

Quantum Fourier Transform over Z_{2^k}

Have k qubits

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

Need to break this into a series of 2-qubit gates.

The Cooley-Tukey Fast Fourier Transform algorithm can be adapted to $\approx k^2$ steps on a quantum computer.

Quantum Fourier Transform over Z_{2^k}

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

Break x and y up into bits: $x = \sum x_\beta 2^\beta$
For each pair of bits x_α and y_β, either:

If $\alpha + \beta \geq k$, then we have $\exp(-2\pi i x y \frac{2^{\alpha+\beta}}{2^k}) = 1$ and we don't have to do anything.

If $\alpha + \beta < k - 1$: then we use $\left|\, x_\alpha y_\beta \right\rangle \rightarrow \exp(-2\pi i x y \frac{2^{\alpha+\beta}}{2^k}) \left|\, x_\alpha y_\beta \right\rangle$.

If $\alpha + \beta = k - 1$, then we use $|\,x_\alpha\rangle \rightarrow \sum_{y_\beta=0}^{1} (-1)^{x_\alpha y_\beta} \left|\, y_\beta \right\rangle$

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.

Making Computations Reversible

In general, if we have a classical computation, then we can write reversibly it as

$$|\mathbf{input}\rangle\,|000\ldots0\rangle \rightarrow |\mathbf{output}\rangle\,|\mathbf{garbage}\rangle$$

where the $|\mathbf{garbage}\rangle$ is extra information that we have store to make it reversible.

We can't use it for the factoring algorithm in this form, since the garbage destroys the interference. However, we can copy the output into another register

$$|\mathbf{output}\rangle\,|\mathbf{garbage}\rangle\,|000\ldots0\rangle \rightarrow |\mathbf{output}\rangle\,|\mathbf{garbage}\rangle\,|\mathbf{output}\rangle$$

Now, we can undo the computation on the first two registers.

$$|\mathbf{output}\rangle\,|\mathbf{garbage}\rangle\,|\mathbf{output}\rangle \rightarrow |\mathbf{input}\rangle\,|000\ldots0\rangle\,|\mathbf{output}\rangle$$

Reversible Gates

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

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

We can get AND, OR, and NOT from it by putting certain constants in the input. For example, $x = 1, y = 1$ gives NOT z in the output, and $z = 0$ gives (x AND y).

Recall that AND and NOT gates are universal for classical computation.

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

Factoring Algorithm

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

$\downarrow \quad \approx L$ steps

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

$\downarrow \quad \approx L^2 \log L \log\log L$ steps

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

$\downarrow \quad \approx L^2$ steps

$$\frac{1}{2^{3L/2}}\sum_{a=0}^{2^{2L}-1}\sum_{c=0}^{2^L-1} |\,c\rangle\,|\,x^a \pmod N\rangle\ e^{-2\pi i ac/2^L}$$

Observe computer.

We need to find the probability amplitude on

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

in the superposition

$$\frac{1}{2^{3L/2}} \sum_{a=0}^{2^{2L}-1} \sum_{c=0}^{2^L-1} |\, c\rangle\, |\, x^a \pmod N\rangle \;\; e^{-2\pi i a c/2^L}$$

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

We have to add the coefficients $e^{-2\pi i a c/2^L}$ 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 $e^{-2\pi i(a_0+br)c/2^L}$.

The a_0 term can be dropped, since it just contributes a phase $e^{-2\pi i a_0 c/2^L}$ to the sum.

Our quantum computer was (before observation) in the state

$$\frac{1}{2^{3L/2}} \sum_{a=0}^{2^{2L}-1} \sum_{c=0}^{2^L-1} |\, c\rangle\, |\, x^a \pmod N\rangle \; e^{-2\pi i a c/2^L}$$

We concentrate on what happens when we observe a paticular $|\, x^{a_0} \pmod N\rangle$. Recall that $a = a_0 + br$. We can make this substitution, and remove the $e^{-2\pi i a_0 c/2^L}$ factor.

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

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

This is a geometric sum which is close to 0 unless

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

for some integer d.

We know

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

Thus

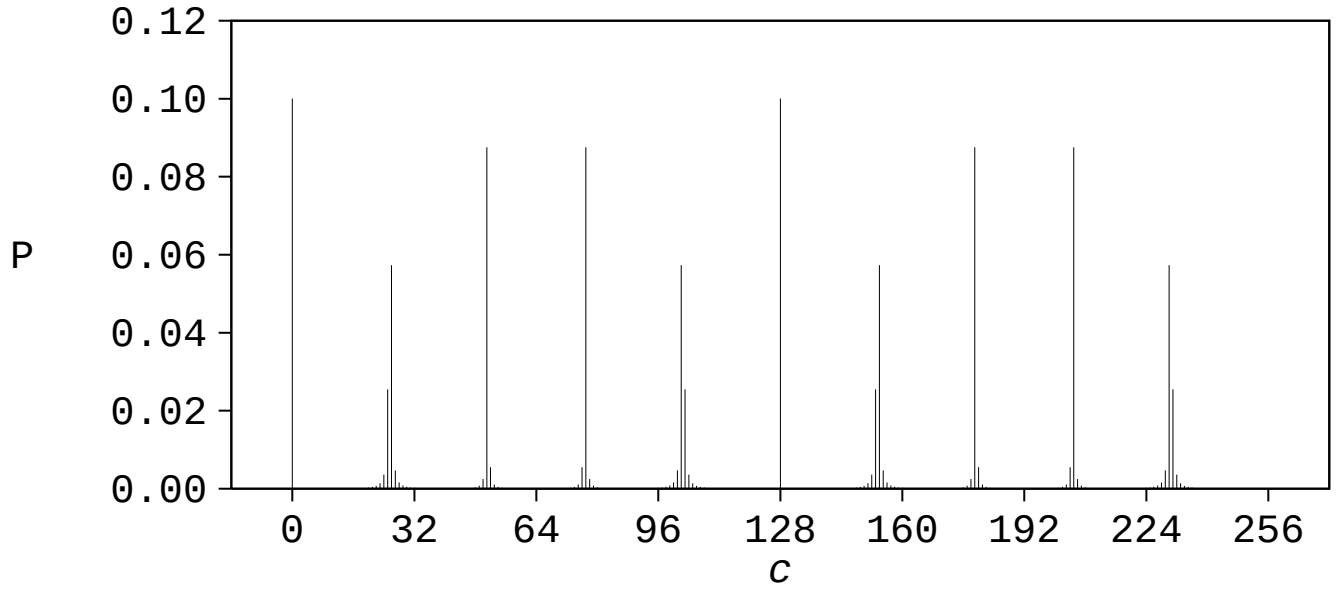
$$\frac{c}{2^L} = \frac{d}{r} + O\left(\frac{1}{2^{2L}}\right).$$

with $r < N$.

But L was the number of bits in N, so $2^{2L} \approx N^2$.

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

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



Example: Factoring 33

The period r is 10.

Difficulties of Quantum Computing

Quantum states are notoriously hard to manipulate.

To do 10^{10} steps on a quantum computer without error correction, and still come up with the right answer, you would need to perform each step with accuracy one part in 10^{10}.

The same objection was raised to scaling up classical computers in the 1950's.

Von Neumann showed that you could build reliable classical computers out of unreliable classical components.

Currently, we don't use many of these techniques because we have extremely reliable integrated circuits, so we don't need them.

Main techniques for fault-tolerance
on classical computers.

- Consistency Checks
- Checkpointing
- Error-Correcting Codes
- Massive Redundancy

Quantum Computing Difficulties

Heisenberg Uncertainty Principle:

You cannot measure a quantum state without changing it.

No-Cloning Theorem:

You cannot duplicate an unknown quantum state.

Can you use these techniques on a quantum computer?

- Consistency Checks
- Checkpointing
- Error-Correcting Codes
- Massive Redundancy

Can you use these techniques on a quantum computer?

- Consistency Checks
  Doesn't get you far on either classical or quantum computers.

- Checkpointing

- Error-Correcting Codes

- Massive Redundancy

Can you use these techniques on a quantum computer?

- Consistency Checks
  Doesn't get you far on either classical or quantum computers.

- Checkpointing
  Cannot use on a quantum computer.

- Error-Correcting Codes

- Massive Redundancy

Can you use these techniques on a quantum computer?

- Consistency Checks
  Doesn't get you far on either classical or quantum computers.

- Checkpointing
  Cannot use on a quantum computer.

- Error-Correcting Codes
  Works well quantum mechanically.

- Massive Redundancy

Can you use these techniques on a quantum computer?

- Consistency Checks
  Doesn't get you far on either classical or quantum computers.

- Checkpointing
  Cannot use on a quantum computer.

- Error-Correcting Codes
  Works well quantum mechanically.

- Massive Redundancy
  Adaptable to quantum computers, but does not work well.

Quantum error correction

Quantum error correcting codes exist.

They can be used to make quantum computers fault-tolerant. so that you only need to perform each step with accuracy approximately one part in 10^4.

How do quantum error-correcting codes get around the no-cloning theorem Heisenberg uncertainty principle?

Measuring one of the qubits gives NO information about the encoded state, so the remaining qubits can retain all the information about the encoded state without violating the non-cloning theorem.

We design the codes so that we can measure the error without measuring (or disturbing) the encoded state.

This means that in our codes, we must have all likely errors *orthogonal* to the encoded data

We can then measure and fix the error without destroying the encoded data.

Repetition Code

The simplest classical error correcting code is the repetition code.

$$\begin{aligned} 0 &\rightarrow 000 \\ 1 &\rightarrow 111 \end{aligned}$$

What about the quantum version?

$$\begin{aligned} |0\rangle &\rightarrow |000\rangle \\ |1\rangle &\rightarrow |111\rangle \end{aligned}$$

Quantum Bit Error Correcting Code

$$\begin{aligned} |0\rangle &\rightarrow |000\rangle \\ |1\rangle &\rightarrow |111\rangle \end{aligned}$$

This works against bit flips

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$\sigma_x(2): \begin{array}{lcl} |000\rangle & \rightarrow & |010\rangle \\ |111\rangle & \rightarrow & |101\rangle \end{array}$$

Can measure "which bit is different?"

Possible answers: none, bit 1, bit 2, bit 3.

Applying σ_x to incorrect bit corrects error.

Quantum Bit Error Correcting Code

This also works for superpositions of encoded $|0\rangle$ and $|1\rangle$.

$$\sigma_x(2) : \alpha\,|000\rangle + \beta\,|111\rangle \rightarrow \alpha\,|010\rangle + \beta\,|101\rangle$$

When this is measured, the result is "bit 2 is flipped," and since the measurement gives the same answer for both elements of the superposition, the superposition is not destroyed.

Thus, bit 2 can now be corrected by applying $\sigma_x(2)$.

Quantum Bit Error Correcting Code

$$\begin{aligned} |0\rangle &\rightarrow |000\rangle \\ |1\rangle &\rightarrow |111\rangle \end{aligned}$$

What about a phase flip error $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$?

$$\begin{aligned} |E_0\rangle = |000\rangle &\rightarrow \quad |000\rangle = |E_0\rangle \\ |E_1\rangle = |111\rangle &\rightarrow -|111\rangle = |E_1\rangle \end{aligned}$$

A phase flip on any qubit gives a phase flip on the encoded qubit, so phase flips are three times as likely. The same thing happens for a general phase error $\begin{pmatrix} 1 & 0 \\ 0 & e^{i\theta} \end{pmatrix}$.

Another 3-qubit code

The unitary transformation $H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$ takes

phase flips to bit flips and vice versa: $H \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} H = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

Suppose we apply H to the 3 encoding qubits and to the encoded qubit. What does this do to our code?

We get a new code

$$\begin{aligned} |0\rangle &\rightarrow \frac{1}{2}(|000\rangle + |011\rangle + |101\rangle + |110\rangle) \\ |1\rangle &\rightarrow \frac{1}{2}(|100\rangle + |010\rangle + |001\rangle + |111\rangle) \end{aligned}$$

Phase error correcting code

$$\begin{aligned} |0\rangle &\rightarrow \frac{1}{2}(|000\rangle + |011\rangle + |101\rangle + |110\rangle) \\ |1\rangle &\rightarrow \frac{1}{2}(|100\rangle + |010\rangle + |001\rangle + |111\rangle) \end{aligned}$$

A phase flip on any qubit is correctable. E.g. $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ on bit 3.

$$\sigma_z(3)\,|E_0\rangle = \frac{1}{2}(|000\rangle - |011\rangle - |101\rangle + |110\rangle)$$

This is orthogonal to $\sigma_z(a)\,|E_b\rangle$ unless $a = 3$, $b = 0$.

So we can measure "which qubit has a phase flip?" and then correct this qubit.

Phase error correcting code

$$
\begin{aligned}
|0\rangle &\rightarrow \frac{1}{2}(|000\rangle + |011\rangle + |101\rangle + |110\rangle) \\
|1\rangle &\rightarrow \frac{1}{2}(|100\rangle + |010\rangle + |001\rangle + |111\rangle)
\end{aligned}
$$

$|0\rangle$ is encoded as the superposition of states with an odd number of 0's;
$|1\rangle$ is encoded as the superposition of states with an even number of 0's.

So a bit flip on any qubit exchanges 0 and 1.

Thus a bit flip is three times as likely as on an unencoded state.

The 9-qubit code

First quantum error correcting code discovered:

$$|0\rangle \rightarrow \frac{1}{2}(|000000000\rangle + |000111111\rangle + |111000111\rangle + |111111000\rangle)$$

$$|1\rangle \rightarrow \frac{1}{2}(|111000000\rangle + |000111000\rangle + |000000111\rangle + |111111111\rangle)$$

This code will correct any error in one of the nine qubits.

It is composed of two codes which are *concatenated*: the outer one corrects phase errors, and the inner one corrects bit errors.

If you have a bit flip: $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, it is corrected by comparison with the other two qubits in its group of three.

The 9-qubit code

$$|0\rangle \rightarrow \frac{1}{2}(|000000000\rangle + |000111111\rangle + |111000111\rangle + |111111000\rangle)$$

$$|1\rangle \rightarrow \frac{1}{2}(|111000000\rangle + |000111000\rangle + |000000111\rangle + |111111111\rangle)$$

If you have a phase flip on a single qubit: $\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, it gives the same result as a phase flip on any of the other qubits in the same group of three.

The correction works via the groups of three bits exactly as it does in the three-qubit phase-correcting code.

t-error correcting codes

By repeating each qubit $2t+1$ times instead of three in the above construction, you get a t-error correcting code which maps one qubit to $(2t+1)^2$ qubits.

Theorem: If you can correct a tensor product of t of any of the following three types of error

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \quad \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

then you can fix any error restricted to t qubits.

Proof Sketch:
The identity matrix and σ_x, σ_y and σ_z for a basis for 2×2 matrices. One can thus decompose any error matrix into a sum of these four matrices. If the error only affects t qubits, it applies the identity matrix to the other qubits, so the decomposition never has more than t terms in the tensor product not equal to the identity.

Example in 3-qubit phase code

$$
\begin{aligned}
|0\rangle &\rightarrow \frac{1}{2}(|000\rangle + |011\rangle + |101\rangle + |110\rangle) \\
|1\rangle &\rightarrow \frac{1}{2}(|100\rangle + |010\rangle + |001\rangle + |111\rangle)
\end{aligned}
$$

Suppose we apply a general phase error $\begin{pmatrix} 1 & 0 \\ 0 & e^{2i\theta} \end{pmatrix}$ to qubit 1, say. Can we correct this?

Rewrite error as $\begin{pmatrix} e^{-i\theta} & 0 \\ 0 & e^{i\theta} \end{pmatrix}$

We can do this, since global phase changes are immaterial.

$$
\begin{aligned}
|E_0\rangle \rightarrow e^{-i\theta}(|000\rangle + |011\rangle) + e^{i\theta}(|101\rangle + |110\rangle) \\
= \cos\theta(|000\rangle + |011\rangle + |101\rangle + |110\rangle) \\
-i\sin\theta(|000\rangle + |011\rangle - |101\rangle - |110\rangle)
\end{aligned}
$$

Correcting an arbitrary phase error

If we had a general phase error of $\begin{pmatrix} 1 & 0 \\ 0 & e^{2i\theta} \end{pmatrix}$ on qubit 1, we got the state

$$\cos\theta(\,|000\rangle + |011\rangle + |101\rangle + |110\rangle\,)$$
$$-i\sin\theta(\,|000\rangle + |011\rangle - |101\rangle - |110\rangle\,)$$

When we measure "which bit has a phase flip," we get "bit 1" with probability $|\sin^2\theta|$ and "no error' with probability $|\cos^2\theta|$.

The state has "collapsed," so our measurement is now correct.

We have a 9-qubit code that can correct any error in 1 qubit. How can we make more general quantum codes?

Better classical codes exist than repetition codes.

The $[7, 4, 3]$ Hamming code, for example.

The codewords are the binary row space of

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

This code maps 4 bits to 7 bits. The minimum distance between two codewords is 3, so it can correct one error.

Quantum Hamming code

$$|0\rangle \rightarrow \frac{1}{\sqrt{8}} \left(\begin{array}{llll} & |0000000\rangle & + & |1110100\rangle \\ + & |0111010\rangle & + & |0011101\rangle \\ + & |1001110\rangle & + & |0100111\rangle \\ + & |1010011\rangle & + & |1101001\rangle \end{array} \right)$$

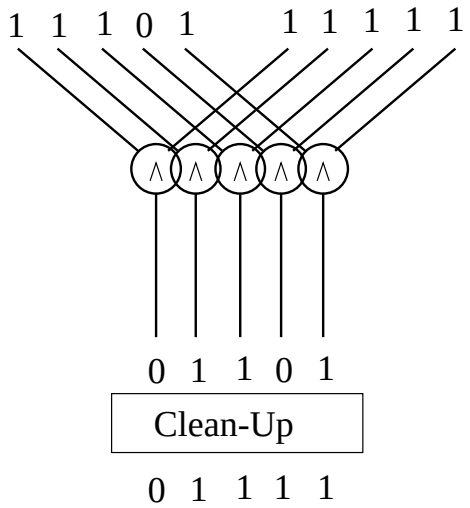
$$|1\rangle \rightarrow \frac{1}{\sqrt{8}} \left(\begin{array}{llll} & |1100010\rangle & + & |0110001\rangle \\ + & |1011000\rangle & + & |0101100\rangle \\ + & |0010110\rangle & + & |0001011\rangle \\ + & |1000101\rangle & + & |1111111\rangle \end{array} \right)$$

This code corrects one error in any qubit.

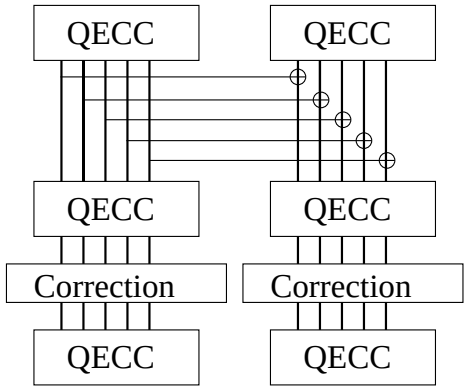
More general stabilizer codes can be constructed which encode k bits into n bits and correct t errors, for various values of (n, k, t).

Fault Tolerant Computing

Classically:



Quantum Mechanically:



Threshold Theorem

Suppose you have a circuit with n qubits. Then you can make a circuit with $O(n \log^c n)$ qubits such that it can with high probability tolerate error on a 10^{-4} fraction of the gates (or an error of size 10^{-4} on all of the gates).

The constant 10^{-4} depends on the exact architecture of your circuit, how large a blow-up in the size of the circuit you are willing to tolerate, and how clever you are.

The best ways of doing fault-tolerance may have not yet been discovered. All the ways discovered to date have fairly large overhead.

NP-complete Problems

These are a class of problems which are notoriously difficult, and which it is widely believed that a classical computer cannot solve.

A problem is in NP if, given the solution, it can easily be checked.

A problem is NP-complete if it is one of the hardest problems in NP; i.e., if it can be solved efficiently, then all NP-complete problems can be solved efficiently.

Can a quantum computer solve NP-complete problems?

We don't know. We suspect not.

The complexity classes P and NP have probabilistic and quantum analogs.

The quantum analogs contain the probabilistic analogs, which in turn contain the original classes. They are all contained in polynomial space, which is in turn contained in exponential time.

This means any computation we can do with T steps on a quantum computer can be done in 2^T steps on a classical computer.