reference frame

Some curious facts about quantum factoring

N. David Mermin

For six years before he retired last year, **David Mermin** taught quantum computation to Cornell University students of computer science. Google "CS483" for his lecture notes, where everything that he says can easily be shown is shown easily.

As explained in my Reference Frame column of April 2007 (page 8), a key to factoring the product $N = pq$ of two prime numbers, each hundreds of digits long, is being able to find the smallest power r for which $a^r \pmod N = 1$ for random integers a. (Two numbers are equal modulo N if they differ by a multiple of N.) Peter Shor's 1994 discovery that a quantum computer would be superefficient at this cryptographically crucial task underlies today's widespread interest in quantum computation. Shor's period-finding algorithm—so called because the function $f(x) = a^x \pmod N$ is periodic with period r—illustrates in striking ways the novel basis quantum mechanics provides for computation.

In a quantum computer, a nonnegative integer x less than 2^n is represented by the product state $|x\rangle_n = |x_{n-1}\rangle\cdots|x_0\rangle$ of n two-state systems (called Qbits—if you prefer the vulgar spelling *qubit*, please regard *Qbit* as an abbreviation), where $x_{n-1}, \ldots, x_0$ are the bits (0 or 1) in the binary expansion of x. Quantum-computational architecture executes a function f that takes n-bit integers to m-bit integers, with a linear subroutine $\mathbf{U}_f$ that takes an n-Qbit "input register" initially in the state $|x\rangle_n$ and an m-Qbit "output register" initially in the state $|0\rangle_m$ into the state $|x\rangle_n|f(x)\rangle_m$.

Suppose the initial state of the input register is the superposition of all possible n-Qbit inputs,

$$|\phi\rangle = (1/2^{n/2}) \sum_x |x\rangle_n.$$

(This can be constructed by starting with each Qbit in the state $|0\rangle$ and applying to each a rotation taking $|0\rangle$ into $(1/\sqrt{2})(|0\rangle + |1\rangle)$.) Since $\mathbf{U}_f$ is linear, if the initial state of input and output registers is $|\phi\rangle|0\rangle_m$ then their final joint state will be

$$|\Psi\rangle = (1/2^{n/2}) \sum_x |x\rangle_n\, |f(x)\rangle_m\,.$$

So it *appears* that just one application of the subroutine evaluates the function f for all the 2^n possible values of x. This trick is called "quantum parallelism."

But appearances can be deceptive. Given a system in a state you know nothing about, there is no way to learn that state. You can only extract information through measurement. If you immediately measure all the Qbits, you acquire a random value x_0 of x and the value f_0 of $f(x_0)$, after which the state becomes $|x_0\rangle_n|f_0\rangle_m$ from which you can learn nothing more. So you could have accomplished as much with an ordinary classical computer by feeding it a random input.

Ah, but suppose you measure only the m Qbits in the output register. When $f(x)$ is $a^x \pmod N$, you will find with equal probability any one of the r distinct values f_0 and the input register will be left in a state $|\psi\rangle$ proportional to

$$|x_0\rangle_n + |x_0 + r\rangle_n + |x_0 + 2r\rangle_n + \ldots,$$

where x_0 is that random integer less than r at which $a^{x_0} \pmod N = f_0$.

This looks promising. By learning only two of the many states $|x\rangle$ appearing in $|\psi\rangle$, you could learn a multiple of the sought-for period r and be well on the way to learning r itself. With high probability, you could learn such a multiple with two measurements on two sets of Qbits, both in the same state $|\psi\rangle$. But this route to success is thwarted by the "no-cloning" theorem, which establishes the impossibility of duplicating an unknown quantum state. Something more clever is needed.

Fourier to the rescue

The clever move is Fourier analysis, in the form of a spectacularly efficient linear subroutine $\mathbf{U}_{\mathrm{FT}}$ that takes the n-Qbit state $|x\rangle_n$ into

$$\mathbf{U}_{\mathrm{FT}}|x\rangle_n = (1/2^{n/2}) \sum_y e^{2\pi i xy/2^n}|y\rangle_n\,.$$

Here the eyes of the quantum physicist tend to glaze over. Ah yes, if you go from position to momentum states, then measurement probabilities become sharply peaked at integral multiples j/r of the inverse period, from which the period r itself can be learned. Ho-hum.

But things are not ho-hummish. To lose interest at this stage is to overlook two crucial differences from boring everyday quantum mechanics. First, x has nothing to do with the position of anything, concrete or abstract. It is an arithmetically useful numerical construction out of the states $|x_i\rangle$ of n independent two-state systems, devoid of physical content: $x = x_0 + 2x_1 + 4x_2 + \ldots + 2^{n-1}x_{n-1}$. Second, "sharply peaked" normally means sharply enough that widths are smaller than the resolution of any detectors. But here one wants an integer r that could be hundreds of digits long. An error of only one part in 10^{10} would get all but a few digits wrong. Such precision lends to the word "sharp" new meaning that no physicist ever dreamed of. All the folklore has to be reexamined.

Reexamination shows that when n Qbits in the state $\mathbf{U}_{\mathrm{FT}}|\psi\rangle$ are measured, the resulting integer $0 \le y < 2^n$ has a significant (over 40%) chance of being within ½ of—that is, as close as possible to—an integral multiple of $2^n/r$. So with a little luck, $y/2^n$ is going to be within $1/2^{n+1}$ of j/r for some random integer j. Does this pin down a unique rational number j/r? Suppose there is a second candidate $j'/r' \neq j/r$. The difference between them is $(j'r - jr')/rr'$. Since the candidates are different, the integer $j'r - jr'$ can't be zero, and since the possible periods r and r' are both less than N, the difference between the candidates is at least $1/N^2$. So if $2^n > N^2$, such an integer y does indeed determine a unique rational number j/r.

Of course, j/r determines not r, but r after any factors it has in common with the random integer j have been divided out. So what our quantum computer gives us is a 40% shot at learning a divisor r_0 of r. The odds that j and r have any really big factors in common are small, so chances are that r will be a fairly small multiple of r_0. You can easily check with a classical computer to see if $a^x = 1 \pmod N$ when $x = r_0$. If so,

$r = r_0$. If not, try $2r_0, 3r_0, \ldots$, and with a little luck one of them will work. If you get up to $1000r_0$ without success, then either you were in the unlucky 60%, or the j you got did indeed share a large factor with r. In that case try the whole thing over again. After not enormously many runs, you're quite likely to succeed. And *that* is how Shor's "factoring" algorithm actually works.

Troublesome phases

But wait a minute! Looking more closely at the crucial subroutine $\mathbf{U}_{\mathrm{FT}}$, one finds it to be cunningly constructed out of operations that apply conditional phase shifts $e^{2\pi i\varphi}$ to Qbits, where the values of φ are inverse powers of 2, ranging from 1/2 to $1/2^n$. Since 2^n must exceed N^2, and N is hundreds of digits long, most of those phase shifts are absurdly tiny—far too tiny for real hardware, with its inevitable imperfections, to produce. All a real quantum computer can execute is an approximate $\mathbf{U}_{\mathrm{FT}}$, grotesquely crude on the scale of parts in 10^{300}, the scale on which one needs to learn the period r.

When this dawned on me, I concluded that all the hoopla rested on a silly failure to notice that you can't turn fields on and off for durations you control to parts in 10^{300}. But I was the silly one. I failed to appreciate the exquisite interplay between digital and analog in a quantum computation. Subroutines like $\mathbf{U}_{\mathrm{FT}}$ depend on parameters that vary continuously, as in analog computation. But readout through measurement produces an unambiguous sequence of 0s and 1s, as in digital computation. Reading out a thousand Qbits gives you a thousand bits of a definite integer—about 300 digits of the decimal representation of that integer. You learn every one of those 300 digits. The question is whether they are the *right* 300 digits.

Those "huge" (perhaps parts in 10^4) uncontrollable phase errors lead to comparable errors in the *probability* that that 300-digit integer will be one of the ones you're looking for. So realistic phase errors might change the probability of getting what you'd like from a little over 40% to a little under 40%. They hardly matter!

A nice illustration of how quantum and classical programming styles differ is provided by the actual calculation of a^x (mod N). Start with a, square it to get a^2 (here I stop writing "(mod N)"—all multiplications are modulo N), square that to get a^4, continuing in this way to get all the powers of the form a^{2^j} for $0 < j < n$. Then to get a^x, you simply form the product of all those powers of a^{2^j} for which $x_j = 1$ in the binary expansion of x. But now there is a parting of the quantum and classical ways.

In a classical computer, the two-state systems used to represent 0 and 1, called Cbits, are cheap and time is precious. If you want to calculate a^x for 2^n different values of x, you use n groups of Cbits to make a table of all the n different a^{2^j} and you look up the various entries going into a^x for each value of x, thereby removing the need to recompute those squares each time you turn to a new value of x.

But in a quantum computer, Qbits are precious and time is cheap. The multiplication of the appropriate a^{2^j} into a^x is not applied 2^n different times to an input register in each of the states $|x\rangle_n$, but only once, to an input register in the state $|\phi\rangle$ in which *all* 2^n possible $|x\rangle_n$ are superposed. Each a^{2^j} is used just once. In some terms in the superposition it's multiplied in, and in others it isn't, depending on whether the jth bit of x is 1 or 0. After that single conditional multiplication is carried out, you can square a^{2^j} to get the next power $a^{2^{(j+1)}}$ and store the result in the same group of Qbits that formerly held a^{2^j}, at a huge saving in Qbits.

Why factoring 15 is too easy

There is another saving in Qbits to watch out for, because it is misleading. The period r of a^x (mod pq) can easily be shown to be a divisor of $(p-1)(q-1)$. So if $p-1$ and $q-1$ are both powers of 2, as with the primes 3, 5, 17, 257, . . . , then so is r. But when r is 2^m, then $2^n/r$ is itself an integer, and measuring the output of the subroutine $\mathbf{U}_{\mathrm{FT}}$ can, again easily, be shown to give an integral multiple of $2^n/r$ with probability 1, even when 2^n merely exceeds N but not N^2. Therefore factoring $N = 15 = (2+1)\times(4+1)$ with a 4-Qbit input register does not provide a serious test of the real Shor algorithm. The smallest number that demonstrates the full subtlety of the procedure is 21, which requires an input register of 9 Qbits, big enough to accommodate $(21)^2$.

Another subtlety, only recently pointed out,[1] reduces that irritating 60% chance of not getting a divisor of r. When N is the product of two distinct primes, one can (again easily) show that r is not only less than N but less than $\frac{1}{2}N$. As a result, besides getting a divisor of r when the measurement yields a y as close as possible to an integral multiple of $2^n/r$, second, third, and even fourth closest also work. This increase in the number of useful outcomes lowers the probability of failure from 60% to 10%, greatly simplifying the subsequent classical detective work.

Features of Shor's "factoring" algorithm like those mentioned here are usually buried in the technical details. By exposing them to the light, I hope to have revealed some of the subtlety and charm of that remarkable procedure.

Reference

1. E. Gerjuoy, *Am. J. Phys.* **73**, 521 (2005). ■