Why Will Quantum Computers be Slow?

28 Jan 2018

Large error corrected quantum computers are many years away. Despite that, we know the factors that will decide how fast (or slow) these hypothetical machines will run. They are: being able to hold the computation in the first place, distilling states fast enough to feed the computation, waiting for measurements one after another, and routing within the available spacetime volume. In this post, I want to discuss each of those bottlenecks in more detail.

Bottleneck #0: Qubit count

Today, there's really only one bottleneck faced by quantum computers: qubit count, qubit count, qubit count. (Also qubit quality.)

Consider Shor's quantum factoring algorithm. To run this algorithm on a quantum computer, you need to store the number you want to factor. If you want to factor a 2048-bit number, you need 2048 qubits to store it. On top of that, you'll need a spare 2048-qubit register to do arithmetic with. Also, there are other overheads. For simplicity, let's say you need an even 5000 qubits.

Given that everybody seems to be announcing 50 qubit chips lately, you could be forgiven for thinking we're about $\frac{50}{5000} = 1\%$ of the way to the 5000 qubits I just said were needed to factor modern RSA keys. But actually all the current announcements are about *noisy qubits*. Shor's algorithm needs *error corrected qubits*. You can turn noisy qubits into error corrected qubits with an error correcting code, but each error corrected qubit will need a thousand, maybe even ten thousand, noisy qubits. So, in terms of qubit count, we're actually only about $\frac{50}{5000 \cdot 1000} = 0.001\%$ of the way to factoring modern RSA keys. If the qubit count doubles every year, it'll be 17 years before we get to 100%.

That 17 year gap is daunting. 50 to 100 noisy qubits should be enough to do something *hard*, and 100 to 200 logical qubits should be enough to do something *useful*, but at the moment it's not known if there's much in between. If there really is nothing on the decade-long road from 100 noisy qubits to 100 error-corrected qubits, the whole field of quantum computing could stall on a built-in trough of disillusionment. So the qubit count bottleneck is *kind of a big deal*.

But, supposing we do reach the error corrected regime and large-scale quantum computation becomes a thing, we stop caring about *possible* and start caring about *fast*. Meaning factors beyond qubit count start to matter.

Bottleneck #1: T count

In order to perform non-trivial operations, error corrected quantum computers must prepare and consume special resource states. In particular, a quantum computer based on the surface code is not able to natively perform 45 degree phase rotations ("T gates"). Instead, a secondary error correcting code is used to prepare and refine copies of the state $|T\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\pi/4}|1\rangle)$. For each $|T\rangle$ state produced, it is possible to perform one error corrected T gate.

Producing $|T\rangle$ states isn't cheap. It takes time, and it takes space. To produce $|T\rangle$ states, you need to annex some of your available qubits into "T factories". It's hard to give any solid estimates on the productivity or footprint of T factories, because they're the subject of ongoing research. Fortunately for you, I was able to lock Austin Fowler in a small dark room and poke him with sticks until he was willing to give order of magnitude estimates. I won't bore you with the many relevant caveats he gave; suffice it to say that we're going to assume that a T factory zone covering N (error corrected) qubits will produce $|T\rangle$ states at a rate of $100N$ per second.

For scale, suppose that factoring a 2048-bit number requires on the order of $4 \cdot n^3 \approx 30 \text{ billion}$ T gates. If you have a T factory zone made out of a thousand qubits, it will take four days to produce enough $|T\rangle$ states to finish this factoring computation. But if you had a hundred thousand qubits zoned as T factories, you could finish the factoring computation in an hour.

As long as a quantum computation is bottlenecked on $|T\rangle$ state production, making quantum computers larger will make the computation faster (because there will be more qubits available for use in T factories). The speedup from adding more qubits will be particularly significant when computations just barely fit. If a generation-X quantum computer can spare 10% of its qubit capacity for T factories, then a generation-(X+1) quantum computer with twice as many qubits could dedicate 55% of its capacity to T factories and would perform the same computation over an order of magnitude faster!

We could easily spend a decade making quantum computers faster merely by having more and more qubits to annex into T factories. But, eventually, we'll be making $|T\rangle$ states so fast that other bottlenecks become relevant.

Bottleneck #2: Measurement depth

Measurements are inherently serial. Most operations native to the surface code can be deformed in ways that allow them to be parallelized, so that they consume space instead of time. But if a measurement determines whether or not an operation is present, and that operation can affect a later measurement, there is no way to avoid waiting for the first measurement result before starting the later measurement.

The amount of time you have to wait for a measurement result is another number that is the subject of ongoing research, and of course it depends a lot on the architecture of your quantum computer. But, by ignoring even more of Austin's caveats, we can assume it takes about 10 microseconds to perform, decode, and react to an error corrected measurement. So an algorithm that performed a million measurements, one after another, would take about 10 seconds to finish.

Because most quantum algorithms only really have to measure once at the end, a bottleneck on the speed of measurements doesn't seem so bad. The problem is that, in the context of error correcting codes, many operations actually hide a measurement. In particular, performing the quantum equivalent of an AND operation within the surface code requires several measurements. And computing ANDs is really common. The measurements for a single AND can be arranged into one parallel layer, so that you're waiting for all of them at the same time instead of one after another, but still.

Going back to our factoring example, suppose the measurement depth of factoring a 2048-bit number is on the order of $2 \cdot n^3 \approx 15 \text{ billion}$. It would take two hours to wait for all those measurements. That doesn't matter when we have 1000 qubits worth of T factory and it takes four days to produce enough $|T\rangle$ states, but it does matter when we have a T factory zone covering a hundred thousand qubits and can make all the $|T\rangle$ states within an hour. Once we're making $|T\rangle$ states faster than we can use them, because we keep waiting for measurements, we have to start making tradeoffs.

As an example tradeoff, note that the dominant cost in Shor's algorithm is addition and that we have a choice of what type of adder to use. We can use ripple-carry adders, which require $4n + O(1)$ T gates but have $2n + O(1)$ measurement depth, or we can use carry-lookahead adders which require $12n + O(\lg n)$ T gates but have $O(\lg n)$ measurement depth. When we're bottlenecked on T count, we prefer the ripple-carry adders because they use a third as many T gates. When we're bottlenecked on measurement depth, we prefer the carry-lookahead adders because they're exponentially shallower. And when we're bottlenecked on T count and measurement depth simultaneously, we'd use a mix of both types of adder in order to exactly balance the two bottlenecks.

In other words, there will be an extended period of time where algorithms are bottlenecked by both measurement depth and by T count. Saving time on one will cost time on the other, and this back-and-forth balancing could be a defining feature of yet another decade of quantum computers getting faster as qubit counts rise. Yet, as the number of T factories gets ridiculously high, and circuits get ridiculously wide to save on depth, another factor can limit speed.

Bottleneck #3: Spacetime volume

Suppose you have a T-hungry computation packed into a $d \times d$ patch of qubits. The computation needs $100 \cdot d^2$ $|T\rangle$ states to finish, and we have $\Theta(d^2)$ factories to feed it with. Clearly we have enough T factories for the computation to finish in $O(1)$ time, but... does it? The answer is no.

The problem is that the perimeter of the computationally-relevant area is $4d$, and the perimeter determines how quickly $|T\rangle$ states can enter. So, as d increases, the rate that $|T\rangle$ states can enter grows like $\Theta(d)$ instead of like $\Theta(d^2)$. The $\Theta(d^2)$ T factories are producing enough $|T\rangle$ states to finish in constant time, but it simply doesn't matter: there's not enough room to route all those $|T\rangle$ states into the computationally-relevant area all at once. Consequently, the computation will take $\Omega(d)$ time instead of $O(1)$ time. It's like building really absurdly tall buildings: make the building tall enough, and keeping the higher floors in use and supplied forces the lower floors to be nothing but elevators.

This example is indicative of a more general problem: every operation in the surface code has to happen somewhere and somewhen. There is great flexibility in exactly where and exactly when, but ultimately you must allocate *some* minimum amount of "spacetime volume" from your computation to performing the operation. So, even though qubits in the surface code can be moved from place to place arbitrarily fast by placing a "horizontal tube" between the qubit's current position and the target position, there *may not be room for the horizontal tube when you need it*. The tubes from other qubits being moved and operated on could form a wall, forcing you to wait to move the qubit or to reschedule other operations in order to make a gap.

Unlike the other bottlenecks, spacetime volume doesn't have a uniform cost. It's not like with T gates, where every additional operation contributes the same amount of overall time. Instead, the cost of using spacetime volume goes up and down depending on where you are in the algorithm. If there's a lot of routing happening, spacetime volume may be at a premium. If not, it may be plentiful and free to use. But keep in mind that any under-utilized spacetime volume can be used for T factories; there will always be pressure to use less volume.

Summary

To run a quantum computation really fast, you need:

1. Enough qubits to fit the computation in the first place.
2. Then enough T factories to perform T gates as fast as the computation needs them.
3. Then wider circuits that do measurements in parallel instead of serially.
4. Then efficient strategies for routing qubits and packing braiding operations.

I like to imagine that each bottleneck will be the "most relevant" for about a decade. We'll naturally transition from one to the next because, funnily enough, every single one of these bottlenecks falls to adding more qubits. More qubits means more space to fit your computation. More qubits means more space for T factories. More qubits means more space for wide parallel circuit constructions. More qubits means more spacetime volume to route through. More qubits, more qubits, more qubits!

...More qubits.

Discuss on reddit

« A Classical Delayed Choice Experiment

Canonicalizing H+S+CNOT+T Circuits »

by: Craig Gidney

more: All Posts, Posts Feed

meta: About the Author/Blog