Why classical computers need exponentially more time and memory to simulate quantum computers, according to the size of the quantum computer

Ronald Landheer-Cieslak

3 November 2019

1 Introduction

If you're a bit like me, you get annoyed by the over-simplified explanations of quantum computers that have been going around since Google demonstrated quantum supremacy[1]. One of the things that those explanations *always* gloss over is how it's so much harder for a classical computer to simulate a quantum computer running what is basically linear algebra, than it is for a quantum computer to just run it. The answer to that is *quantum entanglement*, and in this paper I will try to explain how it works.

I should point out that this means either math or meth will be involved in understanding what I'm about to write. The second option being temporary for understanding and permanent for negative effects, I recommend the first.

In order to explain why classical computers need exponentially more time and memory to simulate quantum computers according to the size of the quantum computer (i.e. $2^Q N$ bits to simulate Q qubits in $2^{2Q}T$ time, so there's a linear *and* an exponential component), I first need to explain a few things about quantum computers.

First, I'll explain what a qubit is, how it compares to a classical bit (2). Second, I will show how we represent qubits in the familiar notation for vectors from linear algebra, and in a less familiar notation called the *Dirac braket* notation (3). Third, I will show how operations on classical bits and operations on qubits compare, and how linear algebra comes into play (4). Fourth, I will show how to represent a pair of qubits in the vector and Dirac notations, and show how operations on qubits scale (5). Finally, I will show how quantum entanglement comes into play, showing why those notations were important, and why a classical computer needs all that extra space and time (6).

2 Qubits

Qubits are the basic unit of data storage and manipulation in quantum computers, like classical bits, or *cbit*s are in classical computing. The fundamental difference between a qubit and a classical bit is the way they behave: a classical bit can only have a value of 0 or 1. For any other value, they need to be combined with other bits. To encode the value “3”, for example, you need two bits (both set to 1). To represent the value $1/\sqrt{2}$ you need a whole bunch of bits and IEEE Std™ 754-2019.

Qubits don’t hold individual values: they hold a value “somewhere between 0 and 1” with a *probability of collapsing* to either 0 or 1 when they are measured. As long as a qubit isn’t measured, it is in a *superposition* of 0 and 1. The value the qubit actually has is *undecided* until it is measured[1]. The superposition can be *represented* as described below, but the value of the qubit *is not* that representation. As any model, it is a simplified representation of reality. In this case, reality may be, for example, the polarization of a photon or the energy state of an electron.

This got weird in a hurry. That is mostly because nature has been shown to be fundamentally probabilistic. Quantum computers are a real-world *engineering* application of the probabilistic nature of the universe, exploiting this aspect of nature for sheer, raw, computing power.

Fundamentally, you cannot know what the value of any given qubit is at any time without measuring it, and when you measure it, you’re almost never 100% sure of the value you’re going to get. What you can know is the probability that you’re going to get a 0 or a 1 for any given measurement, with a margin of error, but which one you’re going to get remains *purely* random.

Mathematically, a qubit may be represented as a vector from the origin to any point on a unit sphere, so while a classical bit has just the one dimension, a qubit is a vector in a two-dimensional complex vector space $\mathbb{C}^2$ called a *Hilbert space*, and can be represented as a vector in a unit sphere, as shown in Figure 1.
A *vector* is a value that has both a direction and a magnitude. In this case, the magnitude is always 1, but the direction can be anything.

3 Notation

Any vector in this unit sphere can be represented with the usual notation for vectors. Remember that in math, you only need two numbers to represent a two-dimensional complex vector space, even if there are three axes: if you use complex numbers, which is what we do here, the vector $\vec{q}$ in the image can be

[1] Proof that the value is, indeed, undecided and not hidden is outside the scope of this paper. Suffice it to say that this is indeed the case. Exactly *why* this is the case or how it works is the subject of much research.

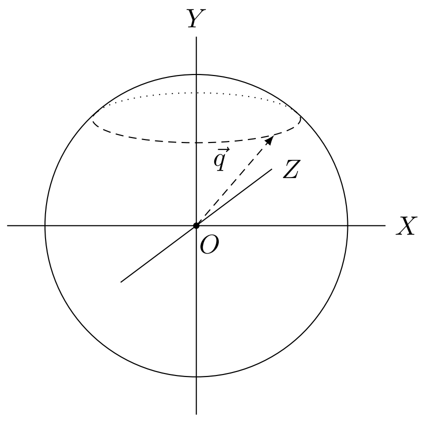

Figure 1: A qubit, represented as a vector in a unit sphere

represented as:

$$q = \begin{pmatrix} \frac{1}{\sqrt{2}} e^{-i\frac{\pi}{6}} \\ \frac{1}{\sqrt{2}} e^{-i\frac{\pi}{6}} \end{pmatrix}$$

Note that there is nothing special about this particular vector $\vec{q}$, but I wanted it to point out of the page, so I needed a complex component. Any complex number can be represented as either a real part plus an imaginary part $(a + bi)$, or as an amplitude and an angle, $re^{i\theta}$. I chose the second notation here because it is both more compact and allows me to easily set the amplitude to $\frac{1}{\sqrt{2}}$, thus keeping my vector's magnitude at 1.

Another way to represent a qubit is the Dirac braket notation. It's basically a shorthand for qubit-typed variables, so our $\vec{q}$ would be noted as:

$$|q\rangle = \frac{1}{\sqrt{2}} e^{-i\frac{\pi}{6}} |0\rangle + \frac{1}{\sqrt{2}} e^{-i\frac{\pi}{6}} |1\rangle$$

By convention, classical bits are represented in qubits as $|0\rangle$ and $|1\rangle$, which correspond, respectively, to $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \end{pmatrix}$, which gets us to what these numbers actually mean.

In the vector notation $\begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix}$, α_0 is the *probability amplitude* if the qubit collapsing to 0 when measured, and α_1 is the probability amplitude of the qubit collapsing to 1 when it is measured. To find the actual probability, this number has to be multiplied with its complex conjugate.

The complex conjugate of a complex number $p + qi$ is $p - qi$, so multiplying a number with its own complex conjugate is

$$(p + qi)(p - qi) = p^2 + pqi - pqi - q^2 = p^2 - q^2$$

When written as $c = re^{i\theta}$, the complex conjugate of c is $\overline{c} = re^{-i\theta}$, which make the probability, with a probability amplitude of c: $c\overline{c} = re^{i\theta}re^{-i\theta} = r^2$.

So, for a real number, this means just taking the square of the probability amplitude to find the probability.

This makes intuitive sense, if you consider that the probability of a qubit collapsing to 0 must be governed by the projection of the vector of that qubit onto the X axis, and the probability of it collapsing to 1 must be governed by its projection onto the Y axis. However, why it maps so cleanly to the probability is one of the mysteries of nature.

4 Operations on qubits

There are four basic operations one can perform on a single classical bit: *set*, *clear*, *identity*, and *not*. The first two operations are inherently destructive, whereas the second two are not. In this context, being *destructive* means that it is impossible to reverse the operation because *using only the output* you cannot know what the value of the bit was before the operation.

Classical computing has many destructive operations: the two "universal" gates[2] in classical computing are `NAND` and `NOR`, both of which are destructive.

The NAND gate takes two bits as input and outputs a single bit that is set unless both of the input bits are set. Looking at the truth table for NAND (Table 1) you can see that three out of the four states have the same output. It is impossible therefore to reverse this gate, because there is no way to know which of the three inputs we had if we get an output 1.

a	b	r
0	0	1
0	1	1
1	0	1
1	1	0

Table 1: Truth table for a classic NAND gate

In quantum computing, there are two destructive operations that can be performed on a single qubit:*clear* and *measure*. The *clear* operation sets the qubit to a fixed value of $|0\rangle$. Measuring a qubit collapses the quantum superposition of that qubit *and any qubit it is entangled with*, and is therefore also destructive. As useful circuits usually end up entangling many qubits together, measuring a qubit can collapse the entire system. These are the only destructive operations, and they're the only destructive operations we're allowed to have.

Any operation U performed on a set of qubits must therefore be reversible: for every U there must be a $U^\dagger$ such that $I = UU^\dagger = U^\dagger U$. This means that

[2]Universal gates in this context are logic gates that, when combined with themselves, can be used to create any other gate, and thus any operation, and thus any function.

there can be no loss of information for any operation performed on a set of qubits by a quantum computer.

This does not mean we cannot implement classical operations with quantum computers: it just means that there's a bit of work to do to get there, and we need to keep more state around.

First, the fact that our quantum version of the gate has to be reversible means we cannot lose any information. It also means we need to have the same number of inputs as we do outputs. In the case of NAND, this means we cannot implement it with just two inputs and one output: we need three inputs and three outputs (looking at the truth table for NAND tells you this). Now, imagine our three inputs are $|a\rangle$, $|b\rangle$ and $|r\rangle$, and that we can guarantee $|r\rangle$ to be $|1\rangle$ on input. We need a gate that flips $|r\rangle$ if both $|a\rangle$ and $|b\rangle$ are true, but leaves it alone if they aren't. Now, remember, again, that $|a\rangle$ and $|b\rangle$ may both be anywhere between $|0\rangle$ and $|1\rangle$, we actually need the probability of flipping $|r\rangle$ to be the probability that both $|a\rangle$ and $|b\rangle$ are true.

This is where we remind ourselves that these are vectors. The way we operate on vectors is to multiply them with matrices. This allows us to do all kinds of nifty things to them, such as rotating them around an axis. So, if we can represent qubits as vectors, we can represent operations on qubits as matrices that we multiply with those vectors.

Let's remember how the product of a matrix with a vector works:

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ax + by + cz \\ dx + ey + fz \\ hx + hy + iz \end{pmatrix}$$

Now let's look at the two simplest operations we can do with a qubit: *Identity* and *NOT*.

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \; NOT = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

Applying these two operators to a single qubit we can see how they work:

$$I\,|0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \times 1 + 0 \times 0 \\ 0 \times 1 + 1 \times 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

$$I\,|1\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \times 0 + 0 \times 1 \\ 0 \times 0 + 1 \times 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

$$I\,|\psi\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} 1 \times \alpha_0 + 0 \times \alpha_1 \\ 0 \times \alpha_0 + 1 \times \alpha_1 \end{pmatrix} = \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = |\psi\rangle$$

$$NOT\,|0\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \times 1 + 1 \times 0 \\ 1 \times 1 + 0 \times 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = |1\rangle$$

$$NOT\,|1\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \times 0 + 1 \times 1 \\ 1 \times 0 + 0 \times 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} = |0\rangle$$

$$NOT\,|\psi\rangle = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} 0 \times \alpha_0 + 1 \times \alpha_1 \\ 1 \times \alpha_0 + 0 \times \alpha_1 \end{pmatrix} = \begin{pmatrix} \alpha_1 \\ \alpha_0 \end{pmatrix} = |\psi'\rangle$$

So, applying an operator to a qubit is basically the same as applying a matrix multiplication to a vector.

As software engineers know, a classical computer can be programmed to do matrix multiplications in papallel, but as you can see, the matrix is 2×2 for an 2-dimensional vector, so there are still $2^2 = 4$ multiplications to be done.

These operators, and quite a few others, allow us to work with a single qubit. The operator we're looking for needs to allow us to work with *three* qubits. How do we do that?

5 More dimensions

The Dirac braket notation can be extended to any number of qubits, pretty much the same way as we extend binary notation for classical bits. Bits can be concatenated from right to left, so $b_0 | b_1 \rightarrow b_1 b_0$. The decimal value of these bits is:

$$2^n b_n + 2^{n-1} b_{n-1} + \cdots + 2^2 b_2 + 2^1 b_1 + 2^0 b_0$$

We can do the same thing with qubits: two qubits' states can be combined into a single notation. So if $|\psi_\alpha\rangle = \alpha_0\,|0\rangle + \alpha_1\,|1\rangle$ and $|\psi_\beta\rangle = \beta_0\,|0\rangle + \beta_1\,|1\rangle$, then $|\psi_\alpha \psi_\beta\rangle = \alpha_0\beta_0\,|00\rangle + \alpha_0\beta_1\,|01\rangle + \alpha_1\beta_0\,|10\rangle + \alpha_1\beta_1\,|11\rangle$, so the vector becomes

$$\begin{pmatrix} \alpha_0\beta_0 \\ \alpha_0\beta_1 \\ \alpha_1\beta_0 \\ \alpha_1\beta_1 \end{pmatrix}$$

Note that this is a vector in a four-dimensional complex vector space, $\mathbb{C}^4$, still a Hilbert space. Sometimes you'll see these different (basis) states written as the $\alpha_0\beta_0\,|0\rangle + \alpha_0\beta_1\,|1\rangle + \alpha_1\beta_0\,|2\rangle + \alpha_1\beta_1\,|3\rangle$, which takes us back to the decimal notation we know for classical bits as well.

Now look at what this does to any operator U_χ that we might have: if U_α operates on one qubit and U_β on another, what does a U look like that operates on both? The answer is fairly easy to derive:

$$U_\alpha \left|\alpha\right\rangle = \begin{pmatrix} a & c \\ b & d \end{pmatrix} \begin{pmatrix} \alpha_0 \\ \alpha_1 \end{pmatrix} = \begin{pmatrix} a\alpha_0 + c\alpha_1 \\ b\alpha_0 + d\alpha_1 \end{pmatrix} = \left|\alpha'\right\rangle \tag{1}$$

$$U_\beta \left|\beta\right\rangle = \begin{pmatrix} e & g \\ f & h \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} = \begin{pmatrix} e\beta_0 + g\beta_1 \\ f\beta_0 + h\beta_1 \end{pmatrix} = \left|\beta'\right\rangle \tag{2}$$

$$\begin{gathered} \left|\alpha\beta\right\rangle = \alpha_0\beta_0 \left|0\right\rangle + \alpha_0\beta_1 \left|1\right\rangle + \alpha_1\beta_0 \left|2\right\rangle + \alpha_1\beta_1 \left|3\right\rangle \\ \Rightarrow \\ \left|\alpha'\beta'\right\rangle = (a\alpha_0 + c\alpha_1)(e\beta_0 + g\beta_1) \left|0\right\rangle + \\ (a\alpha_0 + c\alpha_1)(f\beta_0 + h\beta_1) \left|1\right\rangle + \\ (b\alpha_0 + d\alpha_1)(e\beta_0 + g\beta_1) \left|2\right\rangle + \\ (b\alpha_0 + d\alpha_1)(f\beta_0 + h\beta_1) \left|3\right\rangle \end{gathered} \tag{3}$$

$$U_\alpha \otimes U_\beta = \begin{pmatrix} ae & ag & ce & cg \\ af & ah & cf & ch \\ be & bg & de & dg \\ bf & bh & df & dh \end{pmatrix} = \begin{pmatrix} a\begin{pmatrix} e & g \\ f & h \end{pmatrix} & c\begin{pmatrix} e & g \\ f & h \end{pmatrix} \\ b\begin{pmatrix} e & g \\ f & h \end{pmatrix} & d\begin{pmatrix} e & g \\ f & h \end{pmatrix} \end{pmatrix} = U \tag{4}$$

In step (1), we just get the result of applying an operator U_α to a qubit $\left|\alpha\right\rangle$. In step (2), we do the same for an operator U_β applied to a qubit $\left|\beta\right\rangle$. In step (3), we remind ourselves of what $\left|\alpha\beta\right\rangle$ looks like, and derive from that what $\left|\alpha'\beta'\right\rangle$ should look like. Finally in step (4), we extract the matrix that corresponds to U from what we found in step (3) by applying the matrix in reverse to its outcome. We show how we would have gotten there using a tensor multiplication, which this turns out to be.

Note that we now have a four-by-four matrix as an operator to apply to a four-dimensional complex vector, which implies sixteen parallel multiplications for any operator applied to two qubits.

When we scale this up to three qubits, we find that we need to make our vector notations twice as large again:

$$\begin{pmatrix} \alpha_0\beta_0\gamma_0 \\ \alpha_0\beta_0\gamma_1 \\ \alpha_0\beta_1\gamma_0 \\ \alpha_0\beta_1\gamma_1 \\ \alpha_1\beta_0\gamma_0 \\ \alpha_1\beta_0\gamma_1 \\ \alpha_1\beta_1\gamma_0 \\ \alpha_1\beta_1\gamma_1 \end{pmatrix} \Leftrightarrow \alpha_0\beta_0\gamma_0 \left|000\right\rangle + \alpha_0\beta_0\gamma_1 \left|001\right\rangle + \cdots + \alpha_1\beta_1\gamma_1 \left|111\right\rangle$$

The quantum circuit we need to build to implement the equivalent of the `NAND` gate needs two input bits and one output bit, which means we need an eight-dimensional complex vector space, $\mathbb{C}^8$ to represent it. The matrix to represent it looks like this:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & u_{00} & u_{01} \\ 0 & 0 & 0 & 0 & 0 & 0 & u_{10} & u_{00} \end{pmatrix}$$

where $u_{\alpha\beta}$ are values that allow us to flip the target (output) qubit according to the values of the input qubits. [2] uses the notation $\wedge_2 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ for this matrix.

The gate is called CCNOT, also known as the Toffoli gate, and flips the output qubit iff both the control bits are set. The schematic for the circuit is shown below. If we initialize the output bit to $|1\rangle$, this effectively implements a NAND.

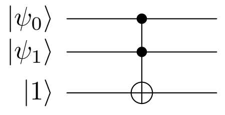

Figure 2: CCNOT Toffoli gate

Note that we now have sixty-four parallel multiplications to do to simulate this eight-dimensional, three-qubit system.

6 Flavors of entanglement

When we combine two qubits, we can make then share a four-dimensional complex state in such a way that it becomes impossible to represent the two individual qubits separately from each other. Consider, for example, the following state for a pair of qubits:

$$|\psi\rangle = \frac{1}{\sqrt{2}} |00\rangle \frac{1}{\sqrt{2}} |11\rangle$$

In this state, there's a 50% chance that both qubits, when measured, collapse to the 0 state and a 50% chance that they both collapse to the 1 state, but *no* chance that they don't both collapse to the same state! This means that $|\psi\rangle$ can no longer correspond to a state $|\alpha\beta\rangle$ where the two states can be factored out as $\alpha_0\beta_0 |0\rangle + \alpha_0\beta_1 |1\rangle + \alpha_1\beta_0 |2\rangle + \alpha_1\beta_1 |3\rangle$ because it is impossible to find a set of values $\{\alpha_0, \alpha_1, \beta_0, \beta_1\}$ such that $\alpha_0\beta_0 = \frac{1}{\sqrt{2}} \wedge \alpha_1\beta_1 = \frac{1}{\sqrt{2}} \wedge \alpha_0\beta_1 = 0 \wedge \alpha_1\beta_0 = 0$.

Creating this state is not that difficult: we need to apply a gate called the *Hadamard* gate to the control bit in a CNOT gate (which is like the CCNOT gate, but with fewer dimensions as it only has one control bit). We can apply this to two unentangled qubits, both initialized to $|0\rangle$, to obtain a maximally entangled pair of qubits:

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{5}$$

$$CNOT = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \tag{6}$$

$$H|0\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = |\psi_0\rangle \tag{7}$$

$$|\psi_1\rangle = |0\rangle \Rightarrow |\psi\rangle = |\psi_0\psi_1\rangle = \frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|10\rangle = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} \tag{8}$$

$$CNOT|\psi\rangle = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 0 \\ 0 \\ \frac{1}{\sqrt{2}} \end{pmatrix} = |\psi'\rangle \tag{9}$$

The circuit to implement this looks like this:

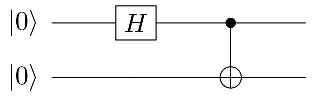

Figure 3: Simple circuit to entangle two qubits

So, while a classical computer could get around implementing NAND using sixty-four multiplications by assuming the bits are always independent, it can't get around implementing its quantum-equivalent CCNOT gate with sixty-four multiplications, and keeping a state for eight data points, because qubits may be entangled and can therefore not be assumed to be independent.

7 Conclusion

Needing $2^Q N$ bits of memory (where N is a linear factor to allow for representing complex numbers) to represent Q simulated qubits is a conservative lower bound: it does not account for the memory needed to operate on those qubits and to keep the operators in memory. Needing $2^{2Q}T$ units of time for the multiplications needed to perform the operations on the qubits is not quite a lower bound:

though it does not account for any overhead needed for efficient scheduling of those operations to allow them to run in parallel, for example, it also does not account for the fact that not all operations run on the entire state of the quantum computer.

A quantum computer uses the laws of physics that underpin the universe to perform its operations, however weird the effects of those laws turn out to be. This means that all those calculations that need to be done by a classical computer to simulate the quantum computer, are done automatically by nature in a real quantum computer. Nobody understands quite why or how it works, but it does seem to work rather nicely.

This is why Google's quantum supremacy experiment was both important, and possible: with 53 qubits, a classical computer needs $2^{53}N$ bits of memory, which is at least (at $N = 1$) 1,125,899,906,842,624 bytes, or about one petabyte of memory just to hold the possible values. Google's quantum computer never did run operations on all the available qubits at once: it could only run them on pairs of qubits, though it could run operations on multiple qubits or pairs of qubits simultaneously. So while the memory requirement would be prohibitive, the execution time requirement . . . was still prohibitive.

This does not explain why quantum computers can be *useful*. A hint for that is provided by Grover's algorithm, which reduces the search time for an item in an unsorted list from $O(n)$ to $O(\sqrt{n})$. Another hint may be found in the quantum computer you already own and carry around with you on a daily basis: your brain. Biology has a lot to gain from quantum computers, including the ability to more accurately model biological processes which are inherently quantum systems. Nature has been using quantum mechanics for things like photosynthesis since life began. Now that we are harnassing those same quantum mechanics for computing, we may be able to exploit it to learn more about nature, medicine, and many other fields.

References

[1] A. Zalcman, A. Ho, A. Korotkov, A. Vainsencher, A. Dunsworth, A. Megrant, B. Chiaro, B. Villalonga, B. Burkett, C. Neill, *et al.*, "Quantum supremacy using a programmable superconducting processor," 2019.

[2] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical review A*, vol. 52, no. 5, p. 3457, 1995.