Quantum Disadvantage

Or, simulating IBM's 'quantum utility' experiment with a Commodore 64

Anonymous
quantum.disadvantage@proton.me
https://quantum-disadvantage.co.uk

March 29, 2024

Abstract

IBM recently performed an experiment measuring expectation values of a Trotterized Ising model on their 127-qubit 'Eagle' quantum computer, along with a claim that their results were unlikely to be replicable on a classical computer. In the spirit of friendly academic shitposting, we implement a classical simulation of the same experiment using a Commodore 64, responding to a challenge posted by Dulwich Quantum. Our simulation requires less than 15kB of memory and approximately four minutes of wall-clock time per data point. To accomplish this we use a variant of the sparse Pauli dynamics (SPD) method recently developed by Begušić and Chan. We show that aggressive truncation combined with a shallow depth-first search avoids the prohibitive (for a C64) memory cost of storing the truncated Pauli basis in SPD, while maintaining sufficient accuracy to match the error-mitigated results obtained from the quantum device.

Disclaimer: *Given the current debate about the significance of IBM's experiment, I wish to clarify that this work is not intended as any disrespect to IBM's team or their results - their experiment is a compelling case study of error mitigation for meaningful NISQ computation. My view is that they were unlucky in the specific structure of the circuits used in the experiment, and that with a few tweaks it would no longer be vulnerable to many forms of classical simulation. That said, I couldn't resist the challenge...*

1 Introduction

This project is a joke, and like all jokes, it's only funny if you have to explain it. Therefore, we will provide some brief introduction and motivation. Let's consider a hypothetical – suppose you are minding your own business, reading Louis H. Kauffman's book 'Knots and Physics'. You can feel in your bones that you're on the verge of a breakthrough: the idea isn't quite fully formed in your mind, but you're sure that you're about to make a revolutionary discovery in knot theory that will simultaneously solve world hunger and climate change, and will inevitably lead the human race to a new era ruled by a Platonic philosopher-king who speaks only in category theory. Suddenly, you see it – a single mosquito on the other side of the room emits a high-pitched buzz, threatening to break your focus. Of course, you know what you have to do – after all, the fate of the human race is at stake – so you calmly take your trusty M20 super bazooka, and snipe that pissant arthropod out of the sky, cremating it where it stands. All is now well in the world, and you can return to your research.

Suppose for the sake of argument that this was funny. Why was it funny? Of course, apart from the primal lizard brain instinct of "big thing goes boom, hur hur 😏"[1] we can see that it forms a kind of strange loop: we wanted to do something simple (swat a mosquito) but decided to use something vastly overpowered to solve the task (a rocket launcher) which ought to make the task easier, but instead made the task much harder than it needed to be (have you tried to snipe a mosquito??). In his seminal work [10], Tom Murphy VII refers to this as an *improper hierarchy*, and he defines it to be a type of joke, so of course it must be funny.

Quantum computers have been widely touted as a new computational tool with great potential to enable computations that were previously infeasible. As explained by the leading expert Michio Kaku, they can solve difficult problems by trying all possible answers at once in quantum superposition and then returning the correct one[2]. In this work we are going to take an experiment that was run on a quantum computer, and run it on several devices that are decidedly not quantum computers (cuantum qomputers, if you will) and take ourselves far too seriously while doing it. This is an improper hierarchy, but of the opposite flavor to the one we had before: we are going to take something that is ostensibly a hard problem (approximating the expectation values of an Ising model), solve it with a vastly under-powered device, and in so doing, make it much easier[3].

2 Heisenberg's Cat

> Did you ever hear the tragedy of Heisenberg's cat? No? I thought not, it's not a story Nielsen & Chuang would tell you – it's an Everettian legend.
>
> *Richard Feynman [2]*

Quantum computers are just a bunch of atoms in a trench coat pretending to be a computer, except the trench coat is an electromagnetic field, and we're the ones doing the pretend-

[1] This is intended to be vaguely euphemistic in a way that makes the reader uncomfortable and confused.
[2] To Michio Kaku: this is satire, please don't sue me. To Scott Aaronson: you can relax, I do actual quantum algorithms for my day job :)
[3] This was still plenty hard, but easier than building a quantum computer.

ing. That is, they are carefully-controlled systems of physical objects which obey the laws of quantum mechanics. Mathematically, we describe the state of a system by some high-dimensional complex-valued vector v. The laws of quantum mechanics assert two things:

1. When something happens to the quantum system, this corresponds to multiplying v by some matrix U, so that the new v' is given by $v' = Uv$.

2. Any quantity we want to measure about the system, an *observable*, can be represented by a matrix O, and measuring that observable gives you (approximately) the value $v^\dagger O v$ (note $v^\dagger = \overline{v^T}$ where $\overline{x}$ is the complex conjugate).

To perform quantum computations, we initialize the vector v to some known state, and then perform a sequence of operations U_i, and measure some final observable O. The operations U_i correspond directly to physical actions on the system, for instance shooting lasers at trapped ions or applying electrical signals to points on a semiconductor. This gives us the quantity

$$\langle O \rangle = v^\dagger U_1^\dagger U_2^\dagger \cdots U_n^\dagger O U_n \cdots U_2 U_1 v$$

which, if you're lucky, will be a number that you care about.

There are two ways to think about this process: we can see $\langle O \rangle$ as measuring a fixed observable O of a sequence of states v_i that evolve forward in time as $v_{i+1} = U_i v_i$ with $v_1 = v$, so that:

$$\langle O \rangle = v_{n+1}^\dagger O v_{n+1}$$

This is called the Schrödinger picture. Alternatively, and more useful for this project, we can imagine measuring a fixed state v against a sequence of observables O_i that evolve backwards in time as $O_i = U_i^\dagger O_{i+1} U_i$ with $O_{n+1} = O$, so that:

$$\langle O \rangle = v^\dagger O_1 v$$

This is called the Heisenberg picture, which is what we will work with. At first, it might seem that quantum computers are somewhat trivial, since they just do linear algebra, which we can already do with regular (also called 'classical') computers. However, the vector v is so large (2^{127} dimensions for the system we will look at here), that we could never hope to directly calculate it. The ability to work with such high-dimensional spaces natively is where the power of quantum computers over classical computers comes from (if there is any), and gently coaxing $\langle O \rangle$ to be an interesting and commercially useful number by carefully choosing each U_i is the delicate art of quantum algorithm design.

2.1 'The Experiment'

On 14th June 2023, a team at IBM (Kim et al) published results from an experiment where they calculated $\langle O \rangle$ on a real quantum device, configured to approximate the behavior of a system known as the Ising model [6]. This is a model that is used by physicists to study the behavior of ferromagnetic materials. The team claimed that this computation would be too difficult to perform on a classical computer to an acceptable accuracy, using the leading approximation techniques, and hence that it showed their quantum device could be useful. They published directly in *Nature*, without first releasing a preprint on arXiv (which is customary in the field). This made a lot of people very unhappy and was widely regarded as a *bad move*.

You see, a team at Google had made similar claims in 2019 [1]. After much research it turns out that it is probably possible to simulate their experiment classically, given access to a large cluster of GPUs for a significant amount of time [7]. However, in IBM's case they had unwittingly chosen a task that was much easier to approximate classically, and indeed only *five days* after the results were released, a preprint appeared on arXiv showing that the experiment could be simulated in minutes on a laptop [9]. Two days later another preprint appeared obtaining similar results by a different method [3, 4] - if I had to guess, I'd say this is probably the shortest time-to-refutation of any paper published in Nature. Naturally, the situation immediately devolved into a Twitter war and ended up with IBM's VP for quantum fighting it out in the comments of Linkedin posts. This paper is a response to a challenge set by Twitter user Dulwich Quantum to simulate IBM's experiment using a Commodore 64.

2.2 Okay, so like how are we doing this?

I'm so glad you asked. We will use the *sparse Pauli dynamics* technique developed by Begušić, Hejazi, and Chan [5, 3]. The idea is as follows: the vector v lives in a 2^{127}-dimensional space, and correspondingly the matrices U_i and O are of size $2^{127} \times 2^{127}$, which is clearly too big to calculate directly. However, we can approximate these matrices as the sum of matrices that have a special form. Let I, X, Y, and Z represent the matrices:

$$I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad Y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix} \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

We call matrices of the form $P_1 \otimes P_2 \otimes \cdots \otimes P_n$ *Pauli strings*, where $\otimes$ is the Kronecker product that maps an $n \times n$ matrix and a $k \times k$ matrix to an $nk \times nk$ matrix according to:

$$(A \otimes B)_{ak+c, bk+d} = A_{a,b} B_{c,d}$$

We will represent them compactly by the string $P_1 P_2 \cdots P_n$ - for example, $IXZI$. Note also that since the set I, X, Y, and Z is closed under multiplication, then so are Pauli strings. In this way, a Pauli string of length 127 represents a $2^{127} \times 2^{127}$ matrix. These have the useful property that given Pauli strings P and Q, we have:

$$(e^{i\theta Q})^\dagger P e^{i\theta Q} = \begin{cases} \cos(\theta) P + \sin(\theta) QP & \text{if } PQ \neq QP \\ P & \text{if } PQ = QP \end{cases}$$

Note that any U_i as defined in the previous section can be written as some sequence $e^{i\theta_1 Q_1} \cdots e^{i\theta_m Q_m}$ - therefore, we can write the whole computation as:

$$\langle O \rangle = v^\dagger (e^{i\theta_1 Q_1})^\dagger \cdots (e^{i\theta_n Q_n})^\dagger O e^{i\theta_n Q_n} \cdots e^{i\theta_1 Q_1} v$$

We wish to approximate the sequence of observables O_i in the Heisenberg picture as a linear combination of Pauli strings. Let

$$O_{i+1} = \sum_{j=0}^{n} \alpha_{i+1,j} P_{i+1,j}$$

then we can see from the above property of Pauli strings that

$$O_i = (e^{i\theta_i Q_i})^\dagger O_{i+1} e^{i\theta_i Q_i} = \sum_{j=1}^{n} \alpha_{i+1,j} (e^{i\theta_i Q_i})^\dagger P_{i+1,j} e^{i\theta_i Q_i}$$
$$= \sum_{j=1}^{n} \alpha_{i+1,j} (\beta_{i,j} P_{i+1,j} + \sqrt{1-\beta_{i,j}^2} Q_i P_{i+1,j})$$
$$= \sum_{j=1}^{m} \alpha_{i,j} P_{i,j}$$

where $m \leq 2n$ and $\beta_{i,j} = 1$ if $P_{i+1,j} Q_i = Q_i P_{i+1,j}$ and $\cos\theta_i$ otherwise. Thus, this says that if O_{i+1} can be written as a sum of n Pauli strings, then O_i can be written as a sum of at most $2n$ Pauli strings.

We will assume that the observable $O_{n+1} = O$ is given by a single Pauli string, and that the vector v is given by:

$$v = \begin{pmatrix} 1 & 0 & \cdots & 0 \end{pmatrix}^T$$

This is convenient because we have that $v^\dagger P v$ is equal to 0 whenever P contains an X, and is 1 otherwise. Putting this all together, as $O_{n+1} = \sum_{j=1}^{1} \alpha_{n+1,j} P_{n+1,j}$ with $\alpha_{n+1,1} = 1$ and $P_{n+1,1} = O$, we can use the method above to calculate the value $\langle O \rangle$ as

$$\langle O \rangle = v^\dagger O_1 v^\dagger = \sum_{j=0}^{m} \alpha_{1,j} v^\dagger P_{1,j} v = \sum_{\substack{j=0 \\ X \notin P_{1,j}}}^{m} \alpha_{1,j}$$

where the values $\alpha_{i,j}$ and $P_{i,j}$ are determined iteratively from $\alpha_{i+1,j}$ and $P_{i+1,j}$ until we reach $i = 1$.

However, this is still too costly to calculate directly - since each step may double the amount of Pauli strings in the expansion, we may have as many as 2^n by the end. Various truncation schemes have been proposed to deal with this. We will use a particularly simple scheme: *we fix a limit k to the total number of terms*. At each step, we will calculate all the terms in the new expansion, combining any like terms into one. We then keep all of the terms with Pauli strings that appeared in the previous step - if this is less than k, we add as many new terms as possible so that the total is at most k. If any terms remain, they are discarded. In this way, we can approximate $\langle O \rangle$ as a sum of at most k terms. This process is efficient - if we have n steps, this takes $O(nk)$ time and $O(k)$ memory - and should converge to the correct value as $k \to \infty$.

Since we are going to be implementing this on a device with very little memory we need to keep k small and so this will not be very accurate. To improve the accuracy, we can perform a *shallow depth-first search*. Suppose we perform d steps of the process given above (without a limit k). Then we have an expansion of O_{n+1-d} as

$$O_{n+1-d} = \sum_{j=1}^{2^d} \alpha_{n+1-d,j} P_{n+1-d,j}$$

and in fact by linearity we can rewrite $\langle O \rangle$ as

$$\langle O \rangle = \sum_{j=1}^{2^d} \alpha_{n+1-d,j} \langle P_{n+1-d,j} \rangle_d$$

where $\langle P \rangle_d$ is the observable with respect to the computation of the first $n-d$ steps only:

$$\langle P \rangle_d = v^\dagger (e^{i\theta_1 Q_1})^\dagger \cdots (e^{i\theta_{n-d} Q_{n-d}})^\dagger P e^{i\theta_{n-d} Q_{n-d}} \cdots e^{i\theta_1 Q_1} v$$

Therefore, if we compute each of $\langle P_{n+1-d,j} \rangle_d$ separately one after the other using some small truncation limit k, we can combine them together to yield $\langle O \rangle$, and this is more accurate than using a fixed k to estimate $\langle O \rangle$ directly[4]. This procedure scales like $O(2^d nk)$ in time but only $O(k+2^d)$ in memory, which is advantageous for small d.

3 Qommodore 64

The Commodore 64 is a computer first manufactured in 1982. It is based on the MOS Technology 6510 processor, which is a variant on the extremely popular MOS 6502 processor, introduced in 1975. The 6502 is an 8-bit processor with a 16-bit memory bus, that was widely used in early 8-bit home computers (e.g the Apple II, C64, BBC Micro) and video game consoles (e.g the NES, various Atari machines). In fact, modern variants are still in production. The Commodore 64 runs at 1 MHz and saturates the 16-bit memory bus with 64kB of RAM, some of which is also memory-mapped to various peripherals.

3.1 Implementation

We chose to implement the algorithm described above in 6502 assembly language for performance reasons. While implementations of other languages such as BASIC and FORTH are available for the C64, they are prohibitively slow for intensive arithmetic calculations.

In our implementation we set $k = 128$ and $d = 2$. It is structured as follows:

- We pre-calculate the branches of the depth-first search. Since $d = 2$ is small, we do this by hand for each observable. In the code, we assume that we do no search - each branch of the search is run separately and combined by hand afterwards.
- We are given an observable O as a Pauli string, and initialize the expansion. At each step of the computation, we update the expansion and truncate it if it becomes too large.

We keep track of the following data:

- A list of 256 Pauli strings $P_{i,j}$, along with their coefficients $\alpha_{i,j}$. The first 128 of these are terms in the expansion, the rest is used as scratch space. The Pauli strings are stored as a pair of bitstrings of length 128: each pair of bits represents the Pauli matrix in that position of the string (00 for I, 01 for X, 10 for Z and 11 for Y). The coefficients are represented as fixed-point numbers with a sign bit, a 1-bit integer part and a 14-bit fractional part (this can represent the numbers -2 to 2 with four digits of precision).

[4] We call this a shallow depth-first search because, if we think of every term in the expansion as a node of a tree which is split into at most two nodes at each step, then this corresponds to a depth-limited depth-first traversal of the tree.

- A hash table that maps each Pauli string to its position in the list. It is implemented using an open addressing scheme with linear probing (we do not need the delete operation, which makes this scheme particularly simple to implement). Hashes of each Pauli string in the list are generated by Zobrist hashing and updated whenever the strings are modified.

This occupies about 15kB of data total for the given parameters - see Figure 1 for the layout. Each step of the computation proceeds as follows:

1. For each Pauli string in the expansion, we check if $P_{i+1,j}Q_i \neq Q_iP_{i+1,j}$. When this is true, we multiply its coefficient by $\cos(\theta_i)$ and then copy this term to the scratch space at the end of the list.
2. For every Pauli string in the scratch space, we multiply it by Q_i (updating its hash) and multiply its coefficient by $\sin(\theta_i)$. These now represent the $\sqrt{1-\beta_{i,j}^2}Q_iP_{i+1,j}$ terms.
3. Using the hash table, we check if any Pauli strings in the scratch space correspond to terms of the expansion. If so, we merge the corresponding coefficients and remove the terms from the scratch space.
4. If the number of terms in the expansion is less than 128, we copy as many terms as possible from the scratch space back to the list.

The scratch space is cleared at the end of each iteration. At the end of the computation, we check which Pauli strings do not contain X matrices, and sum up their coefficients.

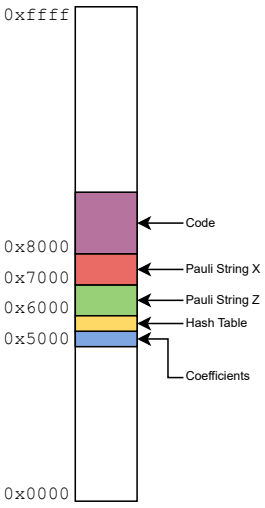


Figure 1: The memory map of the implementation, as set within the address space of the Commodore 64 - about 15kB of the accessible 64kB RAM is used. 8kB of this is reserved for code, although most of this is unused. Each of the two bitstrings for each Pauli string is stored separately (labeled as Pauli String X/Z) for more efficient addressing.

In the case that $|\sin(\theta)| > |\cos(\theta)|$, it is beneficial for accuracy to favor the $\sqrt{1-\beta_{i,j}^2}Q_iP_{i+1,j}$ terms in the expansion rather than the $\beta_{i,j}P_{i+1,j}$ terms when considering the truncation. In this case, we swap the roles of these terms in step one above: if $P_{i+1,j}Q_i \neq Q_iP_{i+1,j}$, the term is copied to the scratch space, and then the *original* is multiplied by Q_i and its coefficient by $\sin(\theta_i)$. In step two, for each term in the scratch space, the string is left unchanged and the coefficient is multiplied by $\cos(\theta_i)$. This effectively keeps all the $\sqrt{1-\beta_{i,j}^2}Q_iP_{i+1,j}$ at each step and only includes $\beta_{i,j}P_{i+1,j}$ when there is space left over.

It is worth noting that it would be possible to increase k, there is enough remaining RAM for k as large as ≈ 1500. However, since the 6502 processor is 8-bit it is much more convenient to work with numbers that are at most 256. Additionally, this would make the whole thing much slower, and it seems that more accuracy is not required, as we will see in the next section.

3.2 Results

Now that we've explained how it was implemented, let's see how well it works. Experiments were run on an original 'bread-bin'-style Commodore 64 manufactured in 1984 that the author paid surprisingly little money for on eBay – see Figure 2 for the experimental setup. However, we still have to actually get our code on to the device. The Commodore 64 has four primary means of interacting with the outside world: the microphone port, the IEC serial-bus port, the keyboard, and the expansion port.

These first two would allow connecting a floppy drive or cassette player to the machine, but since the author didn't have either of these to hand and didn't feel like spending any more money on this project, we are left with two options: the keyboard or the expansion port. Typing programs in to the machine directly was relatively popular, especially for short programs written in BASIC which were printed in magazines. However, the final implementation of this project is about 2500 lines of 6502 assembly, so this would not be very fun. Therefore, we opted to manufacture a cartridge that plugs into the expansion port.

Figure 2: The experimental setup – a Commodore 64 is connected to a monitor through a composite video to HDMI converter, with the code cartridge inserted into the expansion port.

The expansion port provides a way to connect a device directly to the memory bus of the machine. Conveniently, all the timing works out to connect this directly to a standard ROM chip, so that the data inside can be accessed (indeed, many games were historically distributed in this format). We programmed one of these with the code for this project and attached it to an adapter PCB designed by the

OpenC64Cart16K project [8] – see Figure 3. Figure 4 shows what it looks like in action.

(a) **(b)**

Figure 3: The cartridge constructed to house the code for this project. (a) Inside is a PCB designed by Github user SukkoPera [8], with an Atmel AT28C256 ROM chip holding the code. (b) The housing is 3D-printed to resemble original Commodore 64 cartridges.

The computation performed in Kim et al's experiment has a special structure: all $\theta_i = \theta$ are identical, and the list of Q_i can be organized into cycles of four layers, each layer containing 127 steps. In the original experiment, various observables were measured with between 5 and 20 cycles. The number of cycles is referred to as d in the figures referenced below. Some observables were given an extra 'half-cycle' consisting of only one layer, we represent this with the variable e.

We replicate computations of the five different observables given by Kim et al [6] and Begušić and Chan [3]. We calculate their value for a range of θ values from 0 to $\frac{\pi}{2}$. To obtain the datapoint for each value, we run the program to compute the trajectory for the given observable for each branch of the search, as described above, and sum them.

The first three observables, shown in Figure 5, were used by Kim et al to benchmark the accuracy of their experiment, since these can be calculated exactly on a classical computer using tensor network methods, and they find good agreement with the experimental data. For two of these observables, we see that the numbers produced by the Commodore 64 fall within the error bars of the experiment. However for the observable labeled $\langle M_z \rangle$ we do not see good agreement for intermediate values of θ – this issue does go away when increasing the search depth from two to five, but this increases the time required by a factor of eight.

The last two observables, shown in Figure 6, were supposed by Kim et al to be too difficult to approximate by a classical computer, but we can see that we achieve a good match with both the experimental data and with Begušić and Chan's results. The first of these has since been calculated exactly [9], showing good agreement with sparse Pauli dynamics, and hence our results as well.

To verify that our implementation was correct, we also implemented the same algorithm on a modern laptop and we get the same results (except it's 300,000x faster – roughly 800μs per datapoint, as opposed to 4 minutes). Of course, science is only valuable if it is replicable, so to support any replication studies of this work, source code will be available upon reasonable request to the author. In the spirit of SIGBOVIK, you may request either a copy handwritten on papyrus, a slideshow of blurry screenshots recorded on a VHS tape, or that I dictate it to you personally over the phone.

4 Discussion

To conclude we should rate how good of a cuantum qomputer the Qommodore 64 is - first objectively: it is faster than the quantum device datapoint-for-datapoint (although whether this will stay this way is debatable), it is much more energy efficient (superconducting quantum computers need to be cooled by an extremely power-hungry dilution refrigerator), and it is decently accurate on this problem (although it misses some fine structure). On the other hand, it probably won't work on almost any other problem (but then again, neither do quantum computers right now).

So I'd say objectively its an okay cuantum qomputer. But more importantly, how good is it in terms of memeability? I think quite high - I'm not aware of any other simulation technique which could get away with this little memory, and I think this is just about the oldest and slowest device that would do the job - I'm sure there are fancy high-tech toasters nowadays that have faster CPUs. I'll only be truly impressed if someone does this calculation by hand (I bet it wouldn't even take that much paper), and livestreams the whole thing on Twitch.

References

[1] Frank Arute et al. "Quantum supremacy using a programmable superconducting processor". In: *Nature* 574.7779 (2019), pp. 505–510.

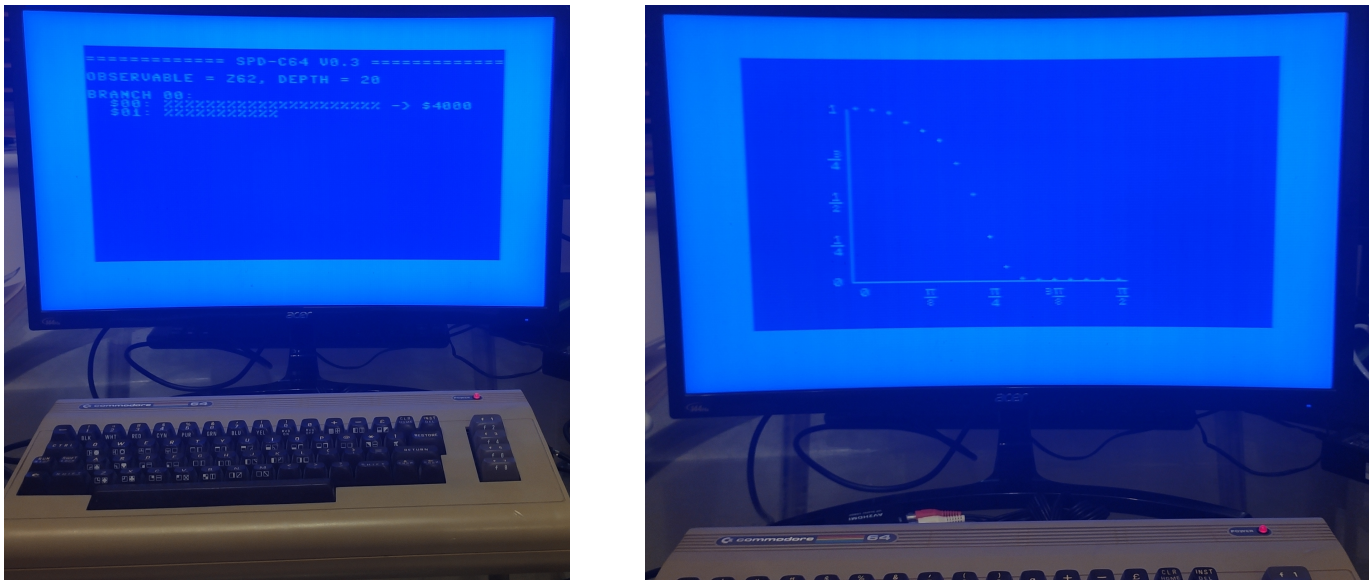
[2] Rick Astley. Never Gonna Give You Up (Official Music Video). `https://www.youtube.com/watch?v=dQw4w9WgXcQ`. Accessed: 2023-12-12. 2009.

[3] Tomislav Begušić and Garnet Kin-Lic Chan. *Fast classical simulation of evidence for the utility of quantum computing before fault tolerance*. 2023. arXiv: `2306.16372 [quant-ph]`.

[4] Tomislav Begušić, Johnnie Gray, and Garnet Kin-Lic Chan. "Fast and converged classical simulations of evidence for the utility of quantum computing before fault tolerance". In: *Science Advances* 10.3 (Jan. 2024). ISSN: 2375-2548. DOI: `10.1126/sciadv.adk4321`. URL: `http://dx.doi.org/10.1126/sciadv.adk4321`.

[5] Tomislav Begušić, Kasra Hejazi, and Garnet Kin-Lic Chan. *Simulating quantum circuit expectation values by Clifford perturbation theory*. 2023. arXiv: `2306.04797 [quant-ph]`.

[6] Youngseok Kim et al. "Evidence for the utility of quantum computing before fault tolerance". In: *Nature* 618.7965 (2023), pp. 500–505.



(a) **(b)**

Figure 4: (a) The program running on the device – in this example, the program is computing the first search branch (out of four) for the $\langle Z_{62} \rangle$ observable with a depth of twenty cycles and $\theta = \frac{i}{32}\pi$ for $0 \leq i \leq 16$. Data can be read off the machine from the right-hand column as fixed-point numbers in hexadecimal format. (b) When it is done running, the calculated values are plotted against θ. This example took about sixteen minutes in total to complete.

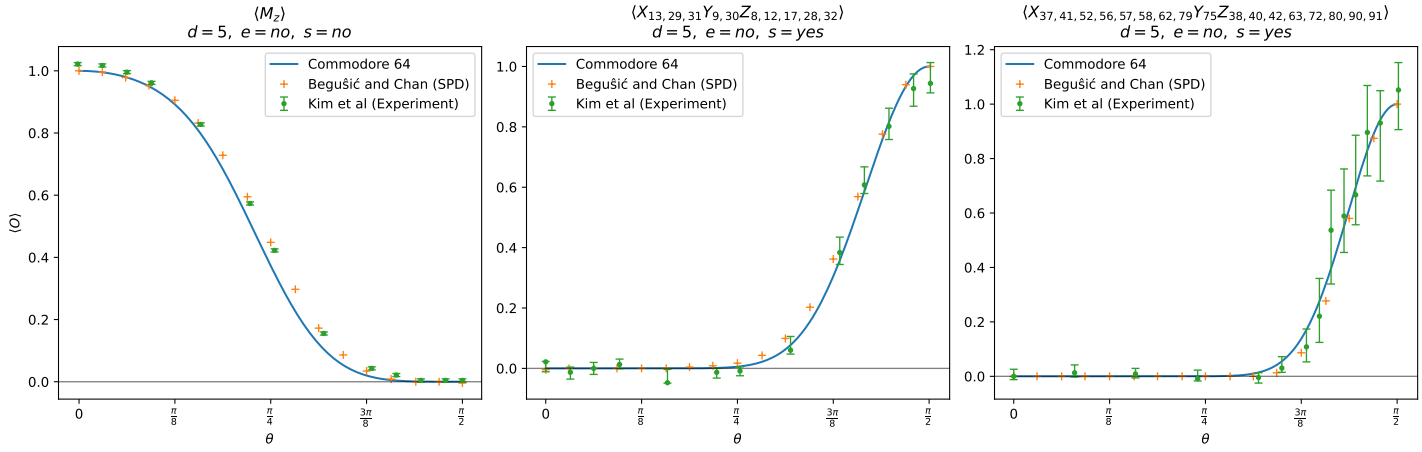


Figure 5: The results of the simulation are plotted for three different observables with $\theta = \frac{i}{256}\pi$ for $0 \leq i \leq 128$. These correspond to Figure 3 of Kim et al [6]. The variables d, e, and s represent the parameters of the simulation: d is the number of cycles, e is marked 'yes' if an extra half-cycle is applied, and s is marked *yes* if the role of the $\cos(\theta)$ and $\sin(\theta)$ terms are swapped in the truncation. The observable $\langle M_z \rangle$ is given by the average of the observables $\langle Z_i \rangle$ for $1 \leq i \leq 127$. Note that both our results and the results from Begušić and Chan do not show some features of the exact solution calculated by Kim et al (for instance small negative values of $\langle X_{13,29,31}Y_{9,30}Z_{8,12,17,28,32} \rangle$ with $\theta < \frac{\pi}{4}$).

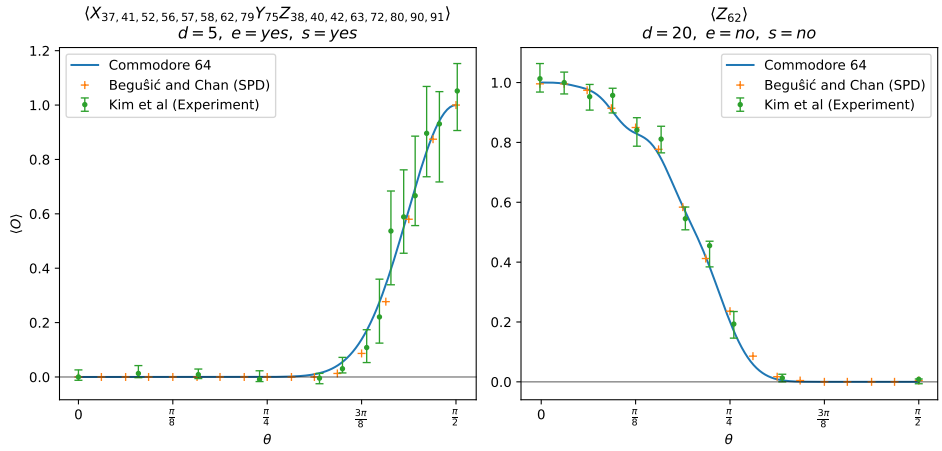


Figure 6: The results of the simulation for two observables with $\theta = \frac{i}{256}\pi$ for $0 \leq i \leq 128$. These correspond to Figure 4 of Kim et al [6]. The variables d, e, and s have the same meanings as in Figure 5 – note that the left-hand plot is given for five cycles with an extra half-cycle, unlike all other plots. Each computation was carried out with a search depth of two, and took approximately 4 minutes for each observable and each value of θ, which is faster than the quantum device described by Kim et al (although they report that the device could be sped up significantly).

[7] Feng Pan, Keyang Chen, and Pan Zhang. "Solving the Sampling Problem of the Sycamore Quantum Circuits". In: *Physical Review Letters* 129.9 (Aug. 2022). ISSN: 1079-7114. DOI: 10.1103/physrevlett.129.090502. URL: http://dx.doi.org/10.1103/PhysRevLett.129.090502.

[8] SukkoPera. *OpenC64Cart16K*. 2017. URL: https://github.com/SukkoPera/OpenC64Cart16K.

[9] Joseph Tindall et al. "Efficient Tensor Network Simulation of IBM's Eagle Kicked Ising Experiment". In: *PRX Quantum* 5 (1 Jan. 2024), p. 010308. DOI: 10.1103/PRXQuantum.5.010308. URL: https://link.aps.org/doi/10.1103/PRXQuantum.5.010308.

[10] Tom Murphy VII. *Reverse emulating the NES to give it SUPER POWERS!* https://www.youtube.com/watch?v=ar9WRwCiSr0. Accessed: 2023-12-12. 2018.