COMPARATIVE ANALYSIS OF THE CLASSICAL APPROACH AND QUANTUM APPROACH TO SIMULATING A DIE

Mohammed Alli Tar-Mahomed, Njabulo Zwelihle Dlamini, Shehwar Faisal Finger

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

1. INTRODUCTION

The simulation of a dice between a quantum or classical computers differs and their results also differ. Three approaches to simulating a dice are done and there results are shown. A classical approach, quantum approach and a joint classical-quantum approach are simulated, and the results are shown and discussed. The approaches are compared, and the best approach is chosen.

2. BACKGROUND

In classical computers we have bits which can be either a 1 or a 0 and measuring a result is instant. In quantum computers however we have qubits which can be in a superposition state of both a 1 and a 0. The state of a quantum qubit can be represented as a vector in Hilbert space (1). Upon measuring a qubit, the value of the qubit will collapse into either a 1 or a 0. We have operations or gates that can be used on qubits like the logic gates found in classical computer circuits. Linear algebra is used for calculations in quantum circuits.

2.1. Logic Gates

2.1.1. Hadamard Gate

The Hadamard gate has a single but very important purpose, which is of placing a qubit of either |1> or |0> into superposition. When a qubit is in superposition it essentially exists in a state anywhere between |1> and |0>. This 'superpositioned' qubit, when measured will either collapse to |1> or |0>. The probability of whether a qubit will collapse to |1> or |0> is always 50% for each state. Hence when introduced to a circuit, a Hadamard gate introduces pure unbiased randomness to a circuit as the qubit has an equal probability of collapsing into |1> or |0>. Making it perfect for our application.

The Hadamard gate is represented by the following matrix:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}$$

When applying this gate to a qubit in state |0> we get the following matrix:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1\\ 1\end{bmatrix}$$

To calculate the probability, we square this result:

$$Probability = \left(\frac{1}{\sqrt{2}}\right)^2 = \frac{1}{2}$$

As shown above the probability of a collapse into |0> or |1> is 50%.

2.1.2. CNOT Gate

In quantum computing, a CNOT (Controlled-NOT) gate is a two-qubit gate that flips a quantum bit if the control qubit is in the state |1> and leaves the target qubit unaltered if the control qubit is in the state |0>. To put it another way, the CNOT gate modifies a two-qubit state as follows:

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

The CNOT gate is a key component of numerous quantum algorithms and is employed in a variety of methods to control qubits and carry out quantum calculations. For many quantum methods, for instance, the CNOT gate can be used to combine two qubits. Algorithms such as Quantum error correction codes (2), which are required for constructing massive quantum computers capable of beneficial calculations, can also be implemented using the CNOT gate.

2.2. Quantum Circuits

Qubits generally start in the |0> state. A horizontal line represents the qubit which can be read from left to right. Different operations or gates can act on the qubits (1). In the figure below there is an example of a quantum circuit. A Hadamard gate is operated on the q_0, and a X-gate is operated on q_1. Thereafter a CNOT gate is acted on q_2 with the control being q_0. Thereafter, measurements of all 3 qubits are taken and the states of each qubit collapse into either 0 or 1.

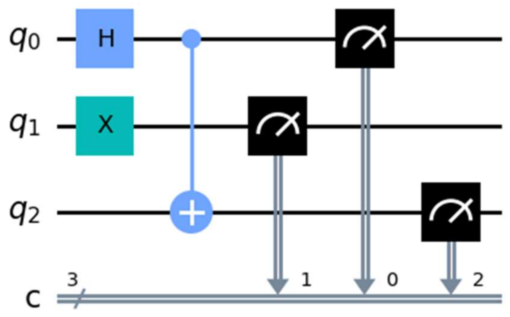

Figure 1: Example of a quantum circuit

3. DISCUSSION

3.1. Classical Approach

A program on a classical computer, using python, that models two dice being thrown is implemented best using a random number generator function. The approach is to generate random integers using the imported random python package. Then create a code that mimics the throw of two dice and stores the total of the results in an empty dictionary the counts the number of times each result occurs. The code simulates the roll of the two dice many times (e.g., 1,000 or 1,000,000). It calculates the total of the two dice for each throw, then edit the appropriate entry in the dictionary. The dictionary's entries are then printed out or plotted on a histogram for better analysis. Two different methods of implementation (See Appendix A-1&2) yielded the same output results.

3.2. Quantum Approach

A purely quantum way of simulating the rolling of 2 6-sided dice is very difficult as the value of qubits range between 0 to 2^n-1 where n is the number of qubits. Therefore using 3 qubits gives us a range between 0 to 7. Creating a circuit to be able to remove the probabilities of a 0 or a 7 is very difficult to do while keeping the rest of the outcomes with the same probability. An attempt was made, and the circuit can be found in the appendix B. This circuit uses 4 qubits where the first 3 generate the random number between 0 and 7. The fourth qubit acts as a control qubit that can randomly change the other qubits when the values of the first 3 qubits is a 0 or a 7. This helps randomising the value after already getting a 0 or 7 but there still is a probability of getting a 0 or 7. A NOT gate is then used to change the value of a single qubit if our measurement still gives us a 0 or a 7.

3.3. An amalgamation of the Quantum and Classical approaches

The amalgamated approach to the problem at hand involves using a quantum circuit along with a classically designed algorithm. Having seen both pros and cons of singular approaches, it was decided that an amalgamated approach would yield the best results. A three-qubit quantum circuit, found in figure 2, was designed with Hadamard gates for each qubit. Measurement was taken from each output to a classical wire. This produced our three-qubit output of values from 0-7. Since a 6-sided dice was being modelled the output had to be restricted from 1-6. Making use of the if statement, while loop and for loop circuitry provided by classical computing (circuitry that wasn't available in quantum circuit design due to restrictions on what was possible through QISKit alone) the output was restricted. To ensure that by restricting the output the probabilities weren't skewed, the if statement and while loops were utilised to execute the circuit each time an unwanted output was produced. This new correct output was counted, and the unwanted output was discarded. If classical computing was not used the output would've had equal probabilities for values from 0-7, with the probabilities being 1/8. After restricting the output and re-running the circuit every time, the incorrect output is produced the output probabilities adjust to exactly 1/6 to values from 1-6.

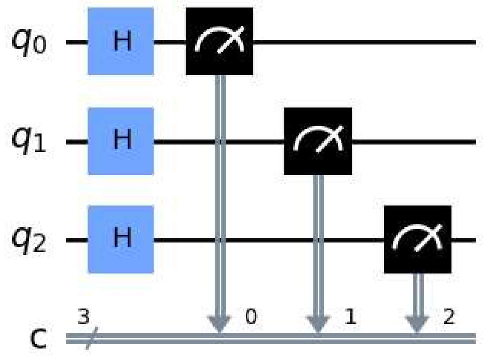

Figure 2: Quantum circuit used.

4. RESULTS

4.1. Results of the Classical approach

The random function is a program built in python that has a set of numbers which chooses a "random" number from this set. These are called pseudorandom integers. Pseudo-random numbers show many of the same statistical characteristics as truly random numbers even though they are not truly random because they are produced by a deterministic algorithm (3). They are frequently used in simulations and programs where unpredictability is required which is why it works best for the task of. A pseudorandom number generator (abbreviated PRNG) is used by the random function to produce random integers. The Mersenne Twister algorithm, a renowned PRNG known for its lengthy lifespan and high-quality unpredictability, is the foundation of the PRNG used by the random function in Python (4).

The resulting count of the simulation in a classical computer system for 1,000,000 (a million) tosses of the 2 dice is as follows:

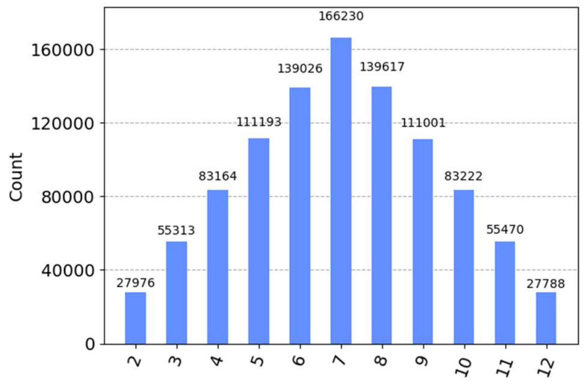

Figure 3: Graphical representation of the count of each value from the simulation.

This shows that the program is accurate because it models a normal distribution of two dice and in terms of the frequency of the different resulting values because the values with the most combinations from dice 1 and 2 have a higher probability of appearing. For example, the value '2' can only result from dice 1 and 2 being '1' & '1' while '7' can be obtained from the combinations; '1' & '6', '2' & '5', and '3' & '4' and the reverse order of these combinations. Hence the count of the '7' should be higher than that of the value '2'. The values with the same number of possible combinations yielded approximately the same probabilities. i.e., '4' and '10' both have two combinations and their probabilities are,

$$P('4') = 0.0831 \,\&\, P('10') = 0.0832$$

$$P('4') \approx P('10')$$

The difference between the graph of two 6-sided dice and a single 12-sided die is that the graph of the latter would have a similar count for each value (including '1') since the probability is equiprobable provided the 12-sided die is unbiased. The probability of each side is,

$$Probability\ P = 1/12$$

4.2. *Results of the Quantum approach*

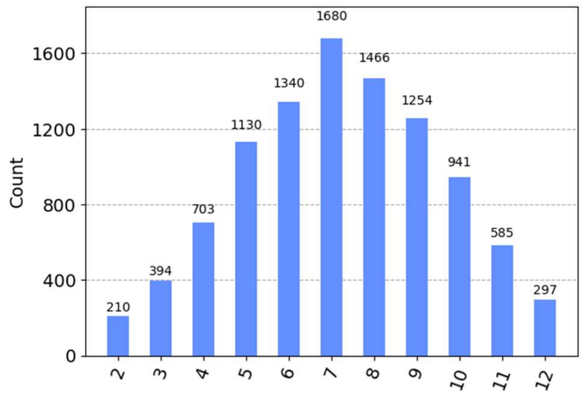

Figure 4: Graph representing the simulation of the 2 dice roll for Quantum approach.

As can be seen from figure 4 above, the distribution of the results of the 2 dice being rolled is like what is expected but is not the same. This is because the quantum circuit does not produce equal probabilities for a single dice roll. From the figure above it can be seen that the higher dice rolls occur more frequently than the lower dice rolls which is not what is required. These differences can be seen by comparing figure 4 with figure 3. This is not a very good model of rolling a dice. These results were simulated and not run on an actual quantum computer because when attempting to run on a quantum computer an error occurred. This quantum model of a dice is not ideal and there is no better model is available.

4.3. *Results of the amalgamation of Quantum and Classical approaches*

As can be seen in figure 3 and 4 the outputs produce graphs that have a similar shape to a normally distributed graph. Unlike the quantum-only approach, the output of this approach has a lot less bias as can be seen from the shape of the results. The bias towards the right-hand side in figure 3 is not present in this approaches results. This shows that an amalgamated approach seems to produce results with the least known bias. It should be known that these results were simulated and not run on a quantum computer as the waiting periods in running a job on a quantum computer made it difficult to run this approach in a feasible time. If it had run on a quantum computer, the results would've shown an even more equal approach due to the absence of pseudo randomness that came into the system by simulation on a classical computer.

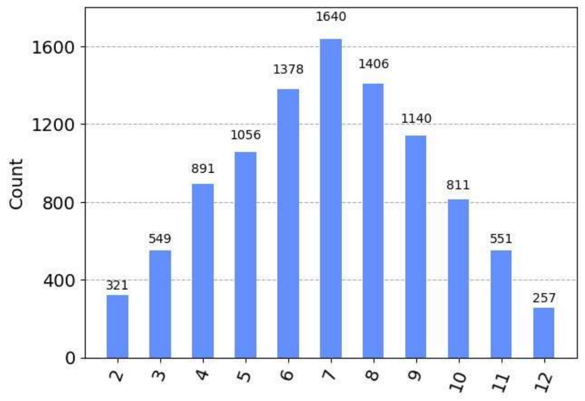

Figure 5: Graph representing the results of amalgamation of classic and quantum.

5. IDENTIFYING THE OPTIMAL APPROACH

The classical approach is more optimal when designing a program that models two 6-sided dice with unbiased characteristics because its set of outputs from each die are {1,6} and the resulting sum {2,12} from each roll. In the case of a quantum program, the set of outputs from each die includes 0 and 7 which mean that extra conditions are necessary to cater for the undesired values.

The quantum approach is not very good as can be seen from the graphs. The probability of a single die is not equal which skews the graph to the right. Thus, the quantum approach is the worst from the three discussed.

The amalgamation of both quantum and classical computing gives very nice results as the graphs are not biased and the probabilities are correct. This approach is the best if the goal is to simulate a dice with perfect randomness. A classical computer does however simulate a dice well except that it is not completely random but is pseudo-random. Many situations will favour the use of classical computers for the simulation as quantum computers are very expensive to use and is not worth the cost for complete randomness. Thus the best approach depends on the situation, is the cost of running a quantum computer worth the complete randomness, which will hardly be the case.

6. CONCLUSION

Three different approaches to simulating the rolling of 2 6-sided dice were successfully designed and simulated. The results show that quantum alone is not enough to simulate a dice without changing its probabilities. The classical approach simulates a dice with the correct probabilities but is pseudo-random. The amalgamation approach gives the best results as the probabilities are correct to that of 2 dice and are completely random. There could possibly be a way to create a quantum approach that works well but quantum algorithms are still very difficult to create.

7. REFERENCES

1. **Harkins, Frank.** CNOT. [Online] [Cited: 28 March 2023.] https://cnot.io.
2. **Steane, Andrew M.** *A Tutorial on Quantum Error Correction.* Oxford : Department of Physics, University of Oxford, 2006.
3. **Python Software Foundation.** 3.11.2 Documentation: random — Generate. *docs.python.org.* [Online] 2001-2023. [Cited: 26 March 2023.] https://docs.python.org/3/library/random.html.
4. **Tan, Ying.** Chapter 10 - GPU-Based Random Number Generators. *GPU-based Parallel Implementation of Swarm Intelligence Algorithms.* s.l. : Elsevier Inc, 2016, pp. 147-165.

Appendix A

1. First python program classical computer

```
import random


# Initialize a 2D array/matrix to keep track of the frequency of each value
frequency = [[0, 0], [0, 0], [0, 0], [0, 0], [0, 0], [0, 0]]
# Generate 1000 random numbers between 1 and 6 and update the frequency
matrix
for i in range(1000):
    # Generate a random integer between 1 and 6, inclusive
    random_number = random.randint(1, 6)


    # Update the frequency matrix based on the random number
    frequency[random_number-1][1] += 1


# Print the frequency of each value
for i in range(6):
    print(f"Number {i+1} appeared {frequency[i][1]} times.")
```

2. Final python program on classical computer (Chosen implementation)

```
import random
from qiskit.visualization import plot_histogram

results_array = {2:0, 3:0, 4:0, 5:0, 6:0, 7:0, 8:0, 9:0, 10:0, 11:0, 12:0}
for ix in range(0,1000000):
    d1 = random.randint(1, 6)
    d2 = random.randint(1, 6)
    sum = d1 + d2
    results_array[sum] = results_array[sum] + 1
plot_histogram(results_array)
```

Appendix B

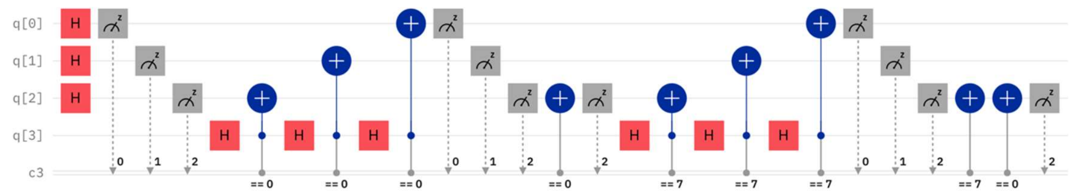

Figure 6: Quantum circuit for the quantum approach to simulate a die.