Quantum Searching: Grover's algorithm

Presented by: Crînganu Denis-Florin (343), Cristea Petru-Theodor (343), Petre Vasile-Eduard (343), Totolici Alexandru-Gabriel (343)

I. Introduction

Quantum searching refers to the process of finding a particular item or solution within a **search space** using principles from **quantum mechanics**. It's most famously exemplified by **Grover's algorithm**, proposed by **Lov Grover** in **1996**.

Traditional search algorithms, like the ones used in classical computing, typically have a time complexity that grows **linearly** with the size of the dataset. For example, to verify If a certain item is present within a space with **N items**, classical computing requires an algorithm that loops through all items and verifies them. Because the algorithm verifies each item individually, the total complexity will be *O(N)*, furthermore, if the algorithm loops through **K elements**, where K ≤ N, then the total complexity will be *O(K),* with a probability of finding the item of $\frac{N}{K}$.

In contrast, **Grover's algorithm** can search an unsorted database of *N* items in roughly $\sqrt{N}$ time. This **quadratic speedup** makes it exponentially faster than classical algorithms for large spaces. Although this algorithm requires $\sqrt{N}$ steps in order to find the element, these results come with a **certain probability**, more exactly $\frac{1}{2}$. Because this approach is quadratic faster than a linear algorithm, running Grover's algorithm multiple times **increases the probability** of finding the element. The expected number of times required to run this algorithm is on average **2**, which will give a total complexity of $O(\sqrt{N})$.

Grover's algorithm make use of **quantum parallelism** and **interference**, by considers all possible solutions **simultaneously** and uses interference to amplify the probability amplitude of the correct solution, while reducing the amplitudes of incorrect solutions.

For a better visualization of this algorithm, instead of looping through all elements to find a specific item, this approach makes use of **quantum physics properties** and can explore all the possibilities (elements) simultaneously.

II. Quantum Computing

Quantum computers are machines that use the properties of **quantum physics** to store data and perform computations. Classical computers use **bits** as the smallest unit of data represented as either *0* or *1*, instead **quantum computers** use quantum bits, also named **qubits**.

In quantum computing, a **qubit** is a basic unit of **quantum information**, the quantum version of the classic binary **bit**. A qubit is a **two-state quantum-mechanical system**, meaning that a qubit can exist in a **superposition** of both states simultaneously. Examples include the spin of the electron, or the polarization of a single photon, both of these examples can have two states.

Qubits can represent numerous possible combinations of *1* and *0* at the same time. This ability to simultaneously be in multiple states is called **superposition**. Because of this counterintuitive phenomenon, a quantum computer with several qubits in superposition can crunch through a vast number of potential outcomes simultaneously. The final result of a calculation emerges only once the qubits are measured, which immediately causes their quantum state to **collapse** to either *1* or *0*.

By generating pairs of qubits that are **entangled**, the performance of a quantum computer can be improved **exponentially** with the number of qubits. **Entanglement** is a phenomenon in quantum mechanics where the quantum states of two or more particles (in quantum computing case: qubit) become **correlated** in such a way that the state of one particle cannot be described independently of the state of the others, even when they are separated by large distances. This propriety implies that measuring the value of one qubit, will **instantaneously** affect the state of the other qubits.

The interaction of qubits with their environment in ways that cause their quantum behavior to decay and ultimately disappear is called **decoherence**. Their quantum state is **extremely fragile**. The slightest vibration or change in temperature, disturbances known as **noise** in **quantum-speak**, can cause them to tumble out of superposition before their job has been properly done.

III. Grover's algorithm

Let $\Sigma = \{0, 1\}$ denote the **binary alphabet**, and let's suppose that:

$$f : \Sigma^n \rightarrow \Sigma$$

is a function from binary strings of length n to bits. We'll assume that we can compute this function efficiently. Let's consider that there are $N = 2^n$ strings in Σ^n.

What **Grover's algorithm** does is to search for a string $x \in \Sigma^n$, for which $f(x)=1$. We'll refer to strings like this as **solutions** to the searching problem. If there are multiple solutions, then any one of them is considered to be a correct output, and if there are no solutions, then a correct answer requires that we report that there are no solutions.

Grover's algorithm solves the **unstructured search problem** with **high probability**, and required just $O(\sqrt{N})$ evaluations of f, where this function evaluations must happen in **superposition**.

Grover's algorithm takes an iterative approach: it evaluates f on superpositions of input strings and intersperses these evaluations with other operations that have the effect of creating interference patterns, leading to a solution with high probability (if one exists) after $O(\sqrt{N})$ iterations.

To formalize, let's assume that we have access to the function $f : \Sigma^n \rightarrow \Sigma$ through a query gate defined as:

$$U_f(|a\rangle|x\rangle) = |a \oplus f(x)\rangle|x\rangle$$

for every $x \in \Sigma^n$ and $a \in \Sigma$.

Grover's algorithm makes use of operations known as **phase query gates**. In contrast to an ordinary query gate U_f, defined for a given function f in the usual way described above, a phase query gate for the function f is defined as:

$$Z_f|x\rangle=(-1)^{f(x)}|x\rangle$$

for every $x \in \Sigma^n$.

In addition to the operation Z_f, we will also make use of a phase query gate for the n-bit **OR function**, which is defined as follows for each string $x \in \Sigma^n$.

$$OR(x) = 0, \text{ if } x = 0^n$$

$$OR(x) = 1, \text{ if } x \neq 0^n$$

Explicitly, the phase query gate for the n-bit OR function operates like this:

$$Z_{OR}|x\rangle = |x\rangle, \text{ if } x = 0^n$$

$$Z_{OR}|x\rangle = -|x\rangle, \text{ if } x \neq 0^n$$

The operation Z_{OR} can be implemented as a **quantum circuit** by beginning with a **Boolean circuit** for the OR function, then constructing a U_{OR} operation, and finally a Z_{OR} operation using the **phase kickback phenomenon**.

The algorithm is performed by using a number *t*, which is the **number of iterations** it performs, as well as the number of queries to the function *f* it requires. Number *t* isn't specified by Grover's algorithm, instead is chosen considering the **optimal** number of iterations with **high probability**.

Grover's algorithm works as follow:

1. Initialize an n qubit register **Q** to the all-zero state $|0^n\rangle$ and then apply a **Hamadard operation** to each qubit of **Q**.
2. Apply *t* times the unitary operation $G = H^{\oplus n} * Z_{OR} * H^{\oplus n} * Z_f$ to the register **Q**
3. Measure the qubits of **Q** with respect to standard basis measurements and output the resulting string.

The operation $G = H^{\oplus n} * Z_{OR} * H^{\oplus n} * Z_f$ is called **Grover operation**. Hamadard operation applied on each qubit works as follow:

$$H|0\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

Grover's algorithm can ve applied to the Search problem as follows:

- Choose the number t
- Run Grover's algorithm on the function *f* , using chosen t, to obtain a string $x \in \Sigma^n$
- Query the function *f* on the string x to see if it's a valid solution:
    - If *f(x) = 1,* then we have found a solution, so we can stop and output x
    - Otherwise, if *f(x) = 1,* then we can either run the procedure again or we can decide to give up and output "no solution".

By taking t = $O(\sqrt{N})$ we'll obtain a solution to our search problem (if one exists) with high probability.

IV. Implementation

We used **IBM Quantum Computing** in order to run the code on a quantum machine. The code is written in **Python** and we used APIs provided by **Qiskit** library, an open-source SDK for working with quantum computers at the level of extended quantum circuits, operators, and primitives.

Grover's algorithm requires an oracle that specifies one or more marked computational basis states. A controlled-Z gate, or its multi-controlled generalization over N qubits, marks the 2^N-1 state ('1' * N bit-string). Marking basis states with one or more '0' in the binary representation requires applying X-gates on the corresponding qubits before and after the controlled-Z gate. Here's our implementation by following IBM tutorial:

```python
def grover_oracle(marked_states):
    """Build a Grover oracle for multiple marked states

    Here we assume all input marked states have the same number of bits

    Parameters:
        marked_states (str or list): Marked states of oracle

    Returns:
        QuantumCircuit: Quantum circuit representing Grover oracle
    """
    if not isinstance(marked_states, list):
        marked_states = [marked_states]

    num_qubits = len(marked_states[0])

    qc = QuantumCircuit(num_qubits)

    for target in marked_states:
        rev_target = target[::-1]
        zero_inds = [ind for ind in range(num_qubits) if rev_target.startswith("0", ind)]
        qc.x(zero_inds)
        qc.compose(MCMT(ZGate(), num_qubits - 1, 1), inplace=True)
        qc.x(zero_inds)

    return qc
```

The built-in **Qiskit GroverOperato**r takes an oracle circuit and returns a circuit that is composed of the oracle circuit itself and a circuit that amplifies the states marked by the oracle.

```
marked_states = random_el_to_search
oracle = grover_oracle(marked_states)

grover_op = GroverOperator(oracle)
```

Then calculate the optimal number of iterations – number ***t*** – that is calculated using the formula:

$$t = \left\lfloor \frac{\pi}{4 \cdot \sin^{-1}\left(\sqrt{\frac{K}{2^N}}\right)} \right\rfloor$$

where K is the number of marked states and N is the number of qubits.

```
optimal_num_iterations = math.floor(
    math.pi / (4 * math.asin(math.sqrt(len(marked_states) / 2**grover_op.num_qubits)))
)
```

A complete Grover experiment starts with a **Hadamard gate** on each qubit, creating an even superposition of all computational basis states, followed the **Grover operator** repeated the optimal number of times. Then we make use of the **QuantumCircuit.power(INT)** method to repeatedly apply the Grover operator.

```
qc = QuantumCircuit(grover_op.num_qubits)
qc.h(range(grover_op.num_qubits))

qc.compose(grover_op.power(optimal_num_iterations), inplace=True)

qc.measure_all()
```

The steps required by Grover's algorithm are now complete, but we still need to run this code on a quantum machine. As we presented earlier, we used **IBM Quantum Computing**, the first thing we need to do is to connect to a Quantum Machine using an API Key, and then run our Quantum Circuit on that machine.

Here's the code that allows us to get control of a Quantum Machine:

```
    service = QiskitRuntimeService(channel="ibm_quantum", token="f14
    backend = service.least_busy(operational=True, simulator=False)
    backend.name
```

```
'ibm_kyoto'
```

Then we need to run the circuit on the machine and get the results.

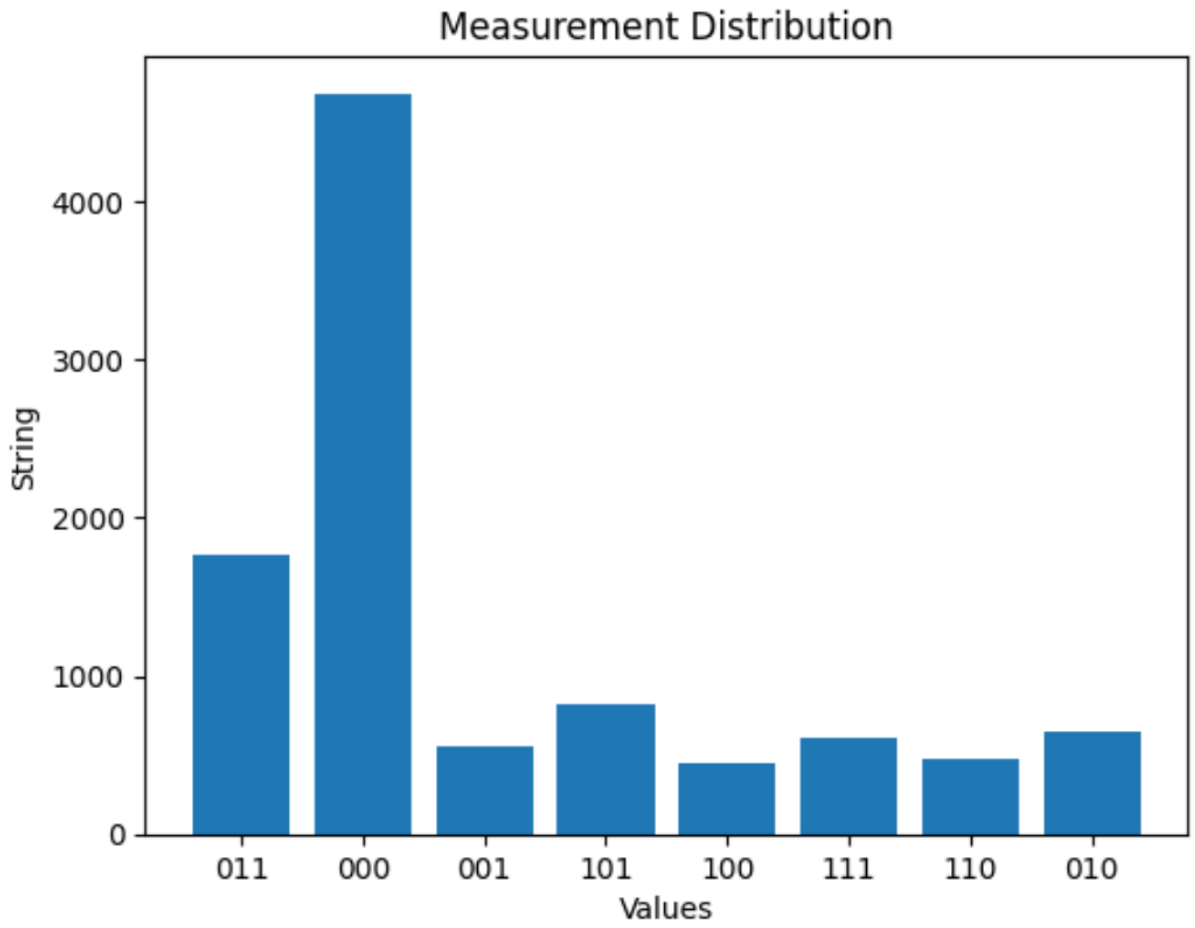
```
target = backend.target
pm = generate_preset_pass_manager(target=target, optimization_level=3)

circuit_isa = pm.run(qc)

sampler = Sampler(backend=backend)
sampler.options.default_shots = 10_000
result = sampler.run([circuit_isa]).result()
dist = result[0].data.meas.get_counts()
```

V. Experiment

Let $\Sigma = \{0, 1\}$ denote the **binary alphabet,** and let be Σ^N the set of all strings composed from 0s and 1s.

We ran the experiment on N = 3, and marked the states = [**“011”**, **“000”**].

The results are:



As we can see, the values for **"000"** and **"011"** are **the largest** in the entire distribution, so we successfully found them.

VI. Conclusion

Grover's algorithm offers a remarkable improvement in search efficiency compared to classical algorithms. By making use of quantum parallelism and interference, it can search an unsorted database quadratically faster, providing a significant advantage for large-scale search problems.

Although this algorithm offers a quadratically faster approach, at this moment in terms of technology is impractical, because of challenges that can occur while working at atomic scale such as **qubit quality and error correction**, **hardware scalability**, **algorithmic optimizations**, **mentability**, and **cost**.

VII. References

1. https://learning.quantum.ibm.com/tutorial/grovers-algorithm
2. https://learning.quantum.ibm.com/course/fundamentals-of-quantum-algorithms/grovers-algorithm
3. https://en.wikipedia.org/wiki/Grover%27s_algorithm

4. https://en.wikipedia.org/wiki/Quantum_computing
5. https://en.wikipedia.org/wiki/Qubit
6. https://en.wikipedia.org/wiki/Two-state_quantum_system
7. https://www.ibm.com/topics/quantum-computing
8. https://www.newscientist.com/question/what-is-a-quantum-computer/
9. https://www.technologyreview.com/2019/01/29/66141/what-is-quantum-computing/
10. https://en.wikipedia.org/wiki/Hadamard_transform