Grover's Algorithm Comparison

In various languages

Parish Wolfe
Dept. of Computer Science
Vanderbilt University)
Raleigh, NC
parish.m.wolfe@vanderbilt.edu

***Abstract*—Implementations of Grover's Algorithm are compared**

***Index Terms*—quantum, grover, oracle**

I. Introduction

Grover's algorithm is a search algorithm. This is a notable algorithm because it is able to search unstructured data at O=Sqrt(n) time complexity. This is significant because it is quite a bit faster than any other search algorithm available to date. The downside of Grover's algorithm is the fact that it is resource intensive. It requires nearly as many qubits as search items to perform the search. In this paper, we will be comparing Grover's algorithm in three different quantum programming languages. Two of these languages are technically python libraries. Those libraries are IBM Qiskit and Google Cirq. The third language is an offering from Microsoft, Q#. The far and away majority of quantum languages are an extension of python. Q# is different, and simmilar to C#.

II. Qiskit versus Cirq

Qiskit and Cirq have major differences in the way that the circuit is constructed. Both languages have a circuit instantiation call first.

```
# Qiskit
grover_circuit = QuantumCircuit(n)

# Cirq
circuit = cirq.Circuit()
```

As we can see from this call, the Qiskit circuit takes a parameter of number of qubits while the cirq implementation does not. Additional differences exist in the way that the circuit is constructed.

```
# Cirq
# Create an equal superposition
# over input qubits.
circuit.append(cirq.H.on_each(*qubits))

# Put the output qubit in the |-⟩ state.
circuit.append([
    cirq.X(ancilla),
    cirq.H(ancilla)])

# Query the oracle.
circuit.append(oracle)

# Construct Grover operator.
circuit.append(cirq.H.on_each(*qubits))
circuit.append(cirq.X.on_each(*qubits))
circuit.append(cirq.H.on(qubits[1]))
circuit.append(cirq.CNOT(qubits[0], qubits[1]))
circuit.append(cirq.H.on(qubits[1]))
circuit.append(cirq.X.on_each(*qubits))
circuit.append(cirq.H.on_each(*qubits))
```

```
# Qiskit
grover_circuit = initialize_s(
    grover_circuit,
    [0,1])
grover_circuit.draw()

grover_circuit.cz(0,1) # Oracle
grover_circuit.draw()

# Diffusion operator (U_s)
grover_circuit.h([0,1])
grover_circuit.z([0,1])
grover_circuit.cz(0,1)
grover_circuit.h([0,1])
grover_circuit.draw()
```

As Evidenced here, the verbiage of the individual gates differs quite a bit.

III. Python implementations versus Q#

The Microsoft implementation is wildly different from the python ones. From the following except one can see how quantum gates are applied to the system.

```
operation ReflectAboutUniform(
    inputQubits : Qubit[]) : Unit {
    within {
        ApplyToEachA(H, inputQubits);
        ApplyToEachA(X, inputQubits);
    } apply {
        Controlled Z(Most(inputQubits),
```

```
            Tail(inputQubits));
    }
}
```

Microsoft's implementation of quantum programming may be more suitable as a next generation C compared to the python implementations.

IV. Similarities

Despite the many differences between the languages, there are many similarities between them as well. ALL three implementations of Grover's algorithm have an Oracle. All implementations use the same quantum gates, X, H, CNOT. These gates may have different syntax but they're all the same types of gates.

V. Conclusion

Grover's Algorithm is an exciting new addition to usable search algorithms. In the future when quantum hardware becomes more available, this algorithm paired with distributed computing, could produce incredible results.