VQE_Screening_2

September 18, 2020

1 VQE Screening 2

```
In [1]: scaffold_codeBell = """

        // Ref[1] https://arxiv.org/pdf/1907.13623.pdf

        const double alpha0 = 3.14159265359;

        module initialRotations(qbit reg[2]) {
          Rx(reg[0], alpha0);
          CNOT(reg[0], reg[1]);
          H(reg[0]);

        }

        module entangler(qbit reg[2]) {
          H(reg[0]);
          CNOT(reg[0], reg[1]);

          H(reg[1]);
          CNOT(reg[1], reg[0]);
        }


        module prepareAnsatz(qbit reg[2]) {
          initialRotations(reg);
          entangler(reg);
        }

        module measure(qbit reg[2], cbit result[2]) {
          CNOT(reg[0], reg[1]); // Fig. 7 of Ref[1]
          H(reg[0]); // Fig. 7 of Ref[1]
          result[0] = MeasZ(reg[0]);
          result[1] = MeasZ(reg[1]);
        }

        int main() {
          qbit reg[2];
          cbit result[2];

          prepareAnsatz(reg);
          measure(reg, result);

          return 0;
        }


        """
```

---

2 Executing it!

```
In [2]: # Compile the Scaffold to OpenQASM
        from scaffcc_interface import ScaffCC
        openqasmBell = ScaffCC(scaffold_codeBell).get_openqasm()
        print(openqasmBell)
```

```
OPENQASM 2.0;
include "qelib1.inc";
qreg reg[2];
creg result[2];
rx(3.141593e+00) reg[0];
cx reg[0],reg[1];
h reg[0];
h reg[0];
cx reg[0],reg[1];
h reg[1];
cx reg[1],reg[0];
cx reg[0],reg[1];
h reg[0];
measure reg[0] -> result[0];
measure reg[1] -> result[1];
```

2.0.1 Execute on a Simulator

```
In [3]: from qiskit import Aer,QuantumCircuit, execute
        Aer.backends()
```

```
Out[3]: [<QasmSimulator('qasm_simulator') from AerProvider()>,
         <StatevectorSimulator('statevector_simulator') from AerProvider()>,
         <UnitarySimulator('unitary_simulator') from AerProvider()>]
```

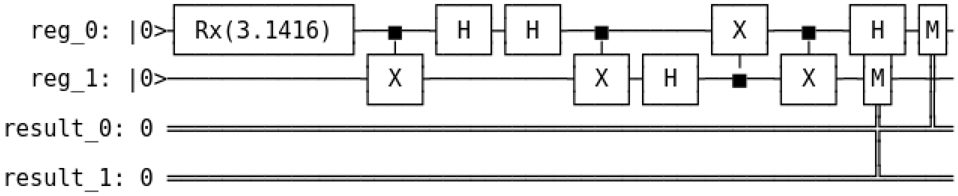
```
In [4]: simulator = Aer.get_backend('qasm_simulator')
        vqe_circBell = QuantumCircuit.from_qasm_str(openqasmBell)
        num_shots = 1000000
        sim_resultBell = execute(vqe_circBell, simulator, shots=num_shots).result()

        countsBell = sim_resultBell.get_counts()

        expected_valueBellXX = (+countsBell.get('00', 0) - countsBell.get('01', 0) + countsBell.
        expected_valueBellYY = (-countsBell.get('00', 0) + countsBell.get('01', 0) + countsBell.
        expected_valueBellZZ = (+countsBell.get('00', 0) + countsBell.get('01', 0) - countsBell.

        expected_value = 0.5 - 0.5 * expected_valueBellXX - 0.5 * expected_valueBellYY + 0.5 * e
        print('The lowest eigenvalue is the expected value, which is : %s' % expected_value)

        #print(countsBell.get('00', 0))
        #print(countsBell.get('01', 0))
        #print(countsBell.get('10', 0))
        #print(countsBell.get('11', 0))

        #print(expected_valueBellXX)
        #print(expected_valueBellYY)
        #print(expected_valueBellZZ)
```

```
The lowest eigenvalue is the expected value, which is : -1.0
```

3 Circuit Visualization

```
In [5]: from qiskit.tools.visualization import circuit_drawer
        circuit_drawer(vqe_circBell, scale=.4)
```

```
Out[5]: <qiskit.visualization.text.TextDrawing at 0x7f10fbae55c0>
```



vqe_circBell