DECOMPOSING A MULTI-CONTROLLED TOFFOLI GATE

Sasanka Dowarah, Saeed Rahmanian Koshkaki, Michael H. Kolodrubetz

June 3, 2022

The problem

We want to decompose a MCX gate with 14 control qubits and a maximum of 5 ancilla qubits into a circuit of minimal depth.

Solution

Our approach is to write the MCX gates with 14 control qubits in terms of other MCX gates with fewer control qubits. Then using the function `transpile` of qiskit these gates are decomposed into single and double qubit gates. This can be done in several ways, however, in order to achieve the minimal circuit depth using 5 ancilla qubits in our circuit, the optimal solution is to use ten 3-control qubits gates and one 4-control qubits gates as shown in figure 1 and 2. The decomposition using `transpile` results in a circuit of depth 130 with 198 U gates and 176 CX gates.

The 5 ancilla qubits are prepared in the state $|0\rangle$.

The code

The following is the code in python to solve the problem.

```
from qiskit import QuantumCircuit , transpile
```

To use gates with 3 and 4- control qubits gates in our circuit, our first approach was to create the circuit in qiskit and use `print(qc.qasm(formatted = True))` to get the corresponding QASM code, but there appear to be some error in this inbuilt function for multi-controlled gates with 4-qubit gate. We overcome this problem by defining two custom gates mc3x and mc4x. We start with creating the multi-controlled gates with 3 and 4 qubits in qiskit, then using the function `transpile` to get the 1 and 2-qubit decompositions of these gates, which is then used in the definition of the two gates in QASM code below.

```
qasm_str = """

OPENQASM 2.0;
```

```
include "qelib1.inc";
gate m3cx q0,q1,q2,q3 { h q3; p(pi/8) q0; p(pi/8) q1; p(pi/8) q2; p(pi/8) q3; cx q0,q1; p(-pi/8) q1; cx q0,q1; cx q1,q2; p(-pi/8) q2; cx q0,q2; p(pi/8) q2; cx q1,q2; p(-pi/8) q2; cx q0,q2; cx q2,q3; p(-pi/8) q3; cx q1,q3; p(pi/8) q3; cx q2,q3; p(-pi/8) q3; cx q0,q3; p(pi/8) q3; cx q2,q3; p(-pi/8) q3; cx q1,q3; p(pi/8) q3; cx q2,q3; p(-pi/8) q3; cx q0,q3; h q3; }

gate m4cx q0,q1,q2,q3,q4 {h q4; p(pi/4) q3; cx q3,q4; p(-pi/4) q4; cx q3,q4; h q3; p(pi/4) q4; p(pi/4) q3; cx q2,q3; p(-pi/4) q3; h q3; cx q0,q3; p(pi/4) q3; cx q1,q3; p(-pi/4) q3;
cx q0,q3; p(pi/4) q3; cx q1,q3; p(-pi/4) q3; h q3; p(pi/4) q3; cx q2,q3; p(-pi/4) q3; h q3;
p(-pi/4) q3; cx q3,q4; p(pi/4) q4; cx q3,q4; h q3; p(-pi/4) q4; p(-7*pi/4) q3; cx q2,q3; p(-pi/4) q3; h q3; p(-7*pi/4) q3; cx q1,q3; p(-pi/4) q3; cx q0,q3; p(pi/4) q3; cx q1,q3; p(-pi/4) q3;
cx q0,q3; p(pi/16) q0; h q3; p(-7*pi/4) q3; cx q0,q4; cx q2,q3; p(-pi/16) q4; p(-pi/4) q3;
cx q0,q4; cx q0,q1; h q3; p(pi/16) q4; p(-pi/16) q1; cx q1,q4; p(pi/16) q4; cx q1,q4;
cx q0,q1; p(-pi/16) q4; p(pi/16) q1; cx q1,q4; p(-pi/16) q4; cx q1,q4; cx q1,q2; p(pi/16) q4;
p(-pi/16) q2; cx q2,q4; p(pi/16) q4; cx q2,q4; cx q0,q2; p(-pi/16) q4; p(pi/16) q2;
cx q2,q4; p(-pi/16) q4; cx q2,q4; cx q1,q2; p(-pi/16) q2; cx q2,q4; p(pi/16) q4; cx q2,q4;
cx q0,q2; p(-pi/16) q4; p(pi/16) q2; cx q2,q4; p(-pi/16) q4; cx q2,q4; p(pi/16) q4; h q4; }

qreg q[20];
m3cx q[0],q[1],q[2],q[15];
m3cx q[3],q[4],q[5],q[16];
m3cx q[6],q[7],q[8],q[17];
m3cx q[9],q[10],q[11],q[18];
m3cx q[12],q[13],q[15],q[19];

m4cx q[16],q[17],q[18],q[19],q[14];

m3cx q[12],q[13],q[15],q[19];
m3cx q[9],q[10],q[11],q[18];
m3cx q[6],q[7],q[8],q[17];
m3cx q[3],q[4],q[5],q[16];
m3cx q[0],q[1],q[2],q[15];

   """
```

We then get the circuit in qiskit from the QASM code using the following

```
qc = QuantumCircuit.from_qasm_str(qasm_str)
qc.draw('mpl')
```

Using the function `transpile` of qiskit, we then decompose the above circuit into single and double qubit gates. We have chosen the basis set as {U,CX}, as it gives the minimum circuit depth.

```
qc = transpile(qc, basis_gates=['u', 'cx'])
print("Circuit depth = ",qc.depth())
print("Number of 1 and 2-qubit gates = ",qc.count_ops())
```

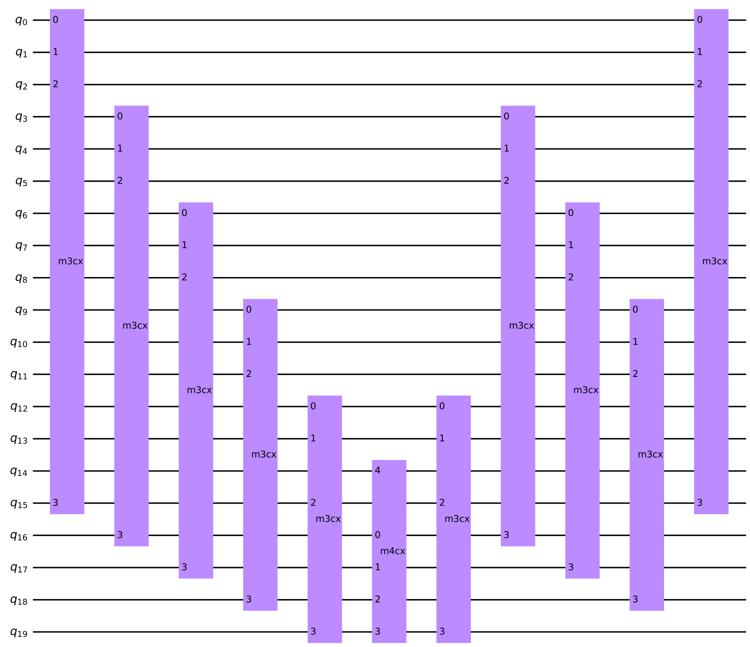

Figure 1: The MCX gate with 14 control qubits and 5 auxiliary qubits is split into ten 3-qubit gates and one 4-qubit gates.

Appendix : Circuit in qiskit

We present below the code where we draw the circuit qiskit.

```
import time
from qiskit import QuantumRegister, ClassicalRegister,
    QuantumCircuit, transpile
```

We set up the circuit with 14 control qubits and 5 ancilla qubits.

```
qr = QuantumRegister(14, 'c') # 14 control qubits.
target_qubit = QuantumRegister(1,'t') # target qubit.
anc = QuantumRegister(5, 'ancilla') # ancilla qubits.
qc = QuantumCircuit(qr,target_qubit, anc)
```

Five 3-control qubit gates.

```
for i in range(4):
    qc.mcx(qr[3*i:3*i+3], anc[i])
```

```

qc.mcx([qr[12],qr[13],anc[0]],anc[4])
```

One 4-control qubit gate.

```
qc.mcx(anc[1:5],target_qubit[0])
```

We apply the 3-control qubit gates in reverse to reset the qubits to their initial states.

```
qc.mcx([qr[12],qr[13],anc[0]],anc[4])

for i in range(3,-1,-1):
    qc.mcx(qr[3*i:3*i+3], anc[i])
```

```
qc.draw('mpl')
```

The following is the circuit in qiskit

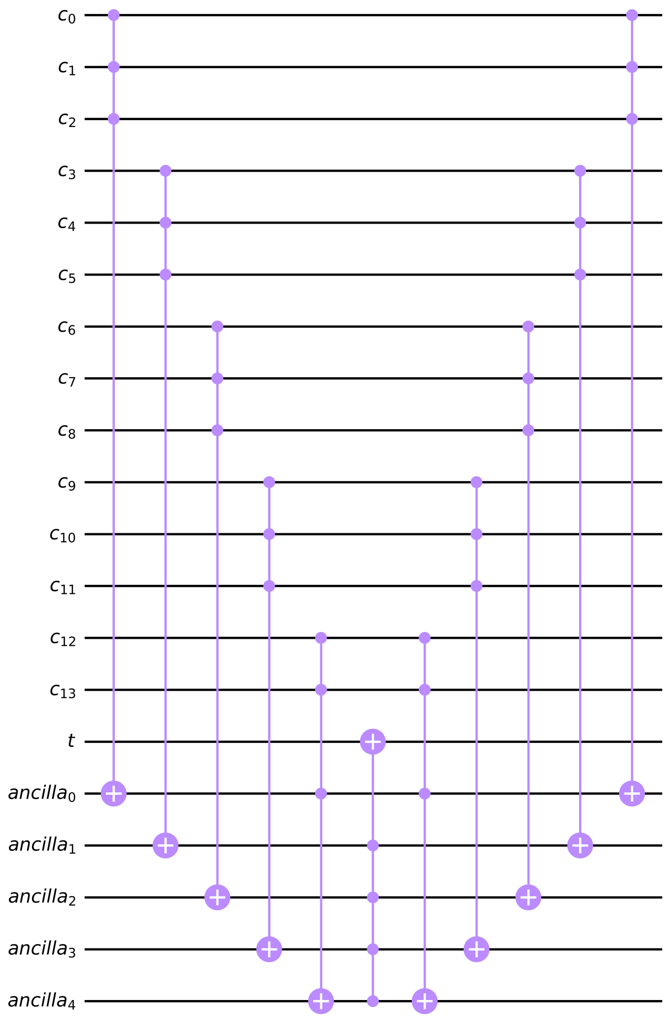

Figure 2: The MCX gate with 14 control qubits and 5 auxiliary qubits is split into ten 3-qubit gates and one 4-qubit gates.