Quantum Computing with Python and Qiskit

@ Python deverloper group Apr. 25, 2019

Jane (Hsiu-Chuan) Hsu

hchsu888@gmail.com
https://github.com/hchsu/quantumcomp

Quantum vs. Classical computing

EX: Factorization

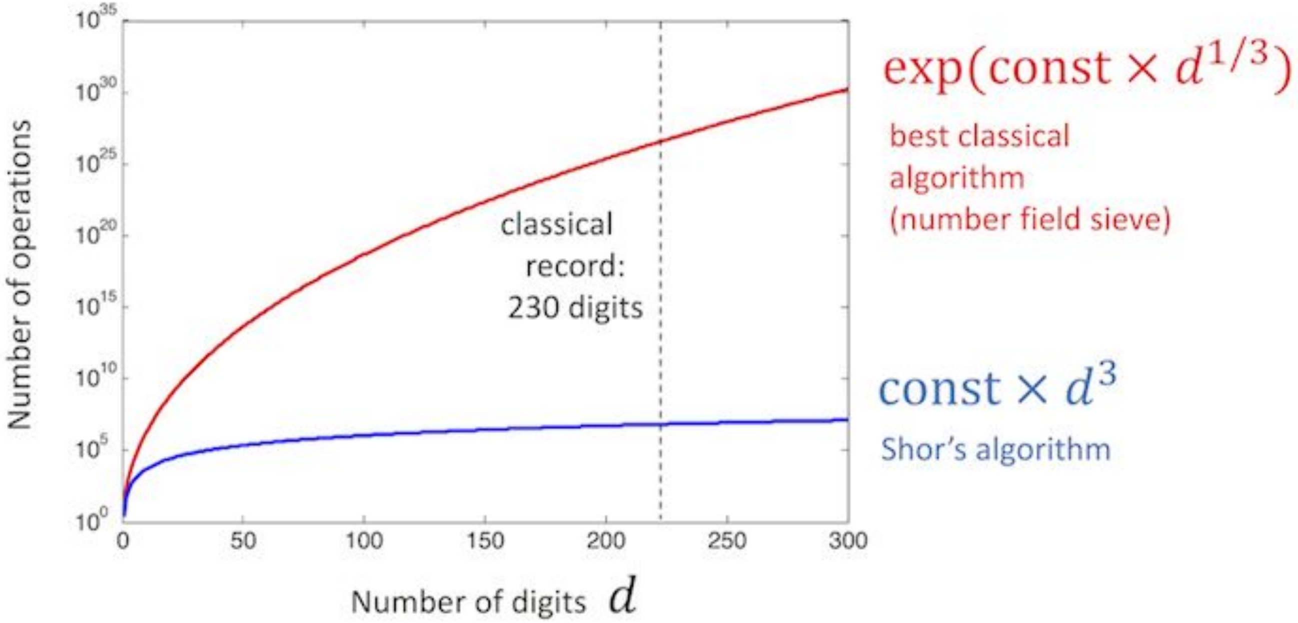


Image Source: IBMQ experience/user guide/Quantum Algorithms

What is Qiskit ?

Quantum Information Science Kit

Python interface for cloud quantum computing

```python
import qiskit
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, Aer, execute
from qiskit.tools.monitor import job_monitor, backend_monitor, backend_overview
from qiskit.tools.visualization import plot_histogram
from qiskit import IBMQ
# Only need to do once
IBMQ.save_account('3b49c9887da9c1d964ddae87e0940b1489f64341d7c4bddc76...')
```

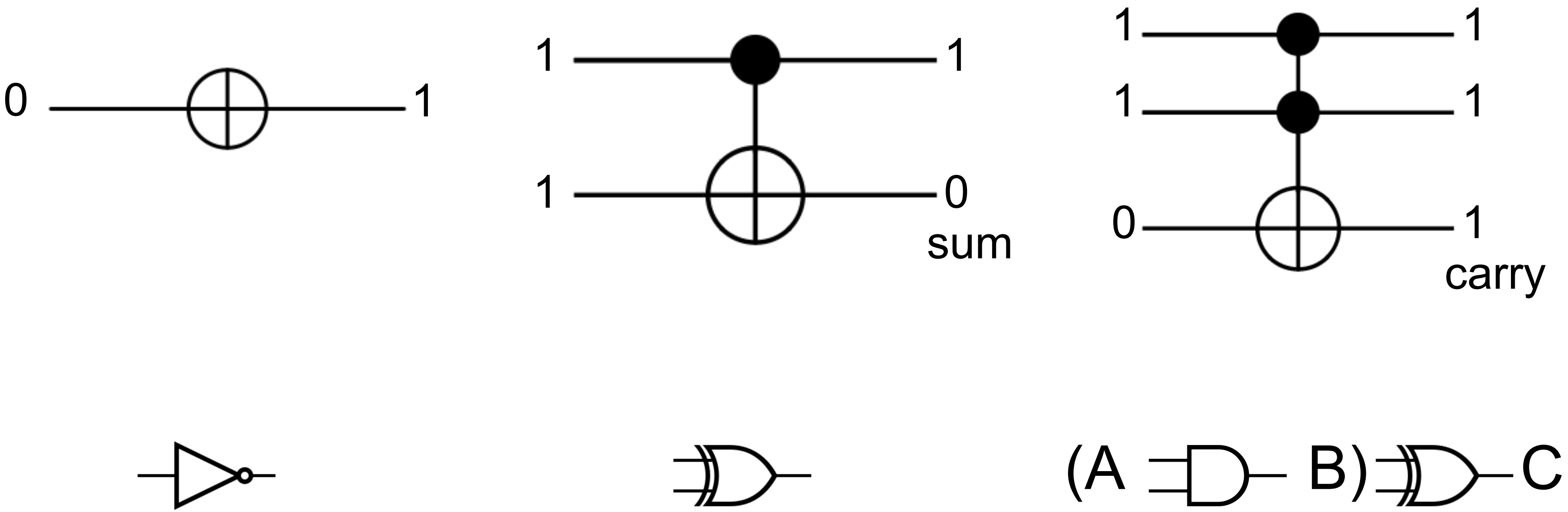
```python
IBMQ.load_accounts()
```

```python
IBMQ.backends()
```

```
[<IBMQBackend('ibmqx4') from IBMQ()>,
 <IBMQBackend('ibmqx2') from IBMQ()>,
 <IBMQBackend('ibmq_16_melbourne') from IBMQ()>,
 <IBMQBackend('ibmq_qasm_simulator') from IBMQ()>]
```

Example : Addition

Basic gates : x, cx, ccx



Example : Addition

3+1

Carry		1	0
Int1		1	1
Int2		0	1
	1	0	0

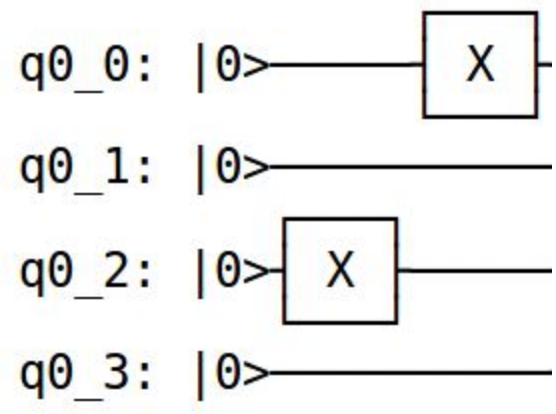
```python
# this circuit model calculates single bit addition
def fulladd(int1=1, int2=1, cin=1):
    q=QuantumRegister(4)
    c=ClassicalRegister(2)
    circuit=QuantumCircuit(q,c)
    # prepare the init state
    if int1: circuit.x(q[0])
    if int2: circuit.x(q[1])
    if cin: circuit.x(q[2])

    # apply gate
    circuit.ccx(q[0],q[1],q[3])
    circuit.cx(q[0],q[1])
    circuit.ccx(q[1],q[2],q[3])
    circuit.cx(q[1],q[2])
    circuit.cx(q[0],q[1])

    # measure
    circuit.measure(q[2],c[0])
    circuit.measure(q[3],c[1])
    return circuit
```

Example : Addition

3+1

```
cir.draw()
```

Carry		1	0
Int1		1	1
Int2		0	1
	1	0	0

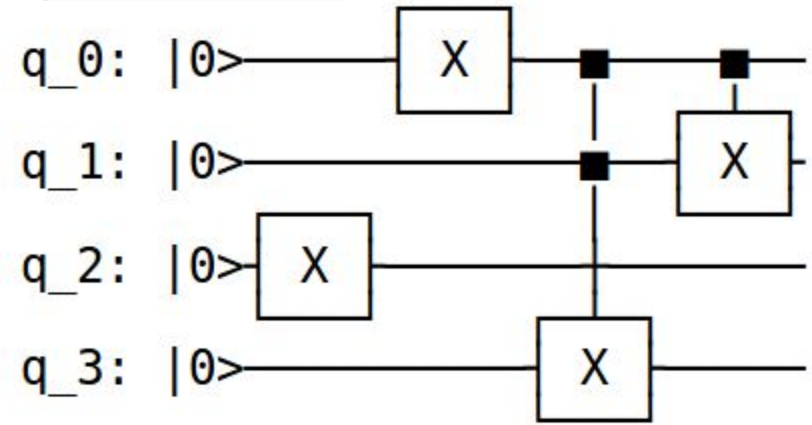


Example : Addition

3+1

```
cir.draw()
```

Carry		1	0
Int1		1	1
Int2		0	1
	1	0	0

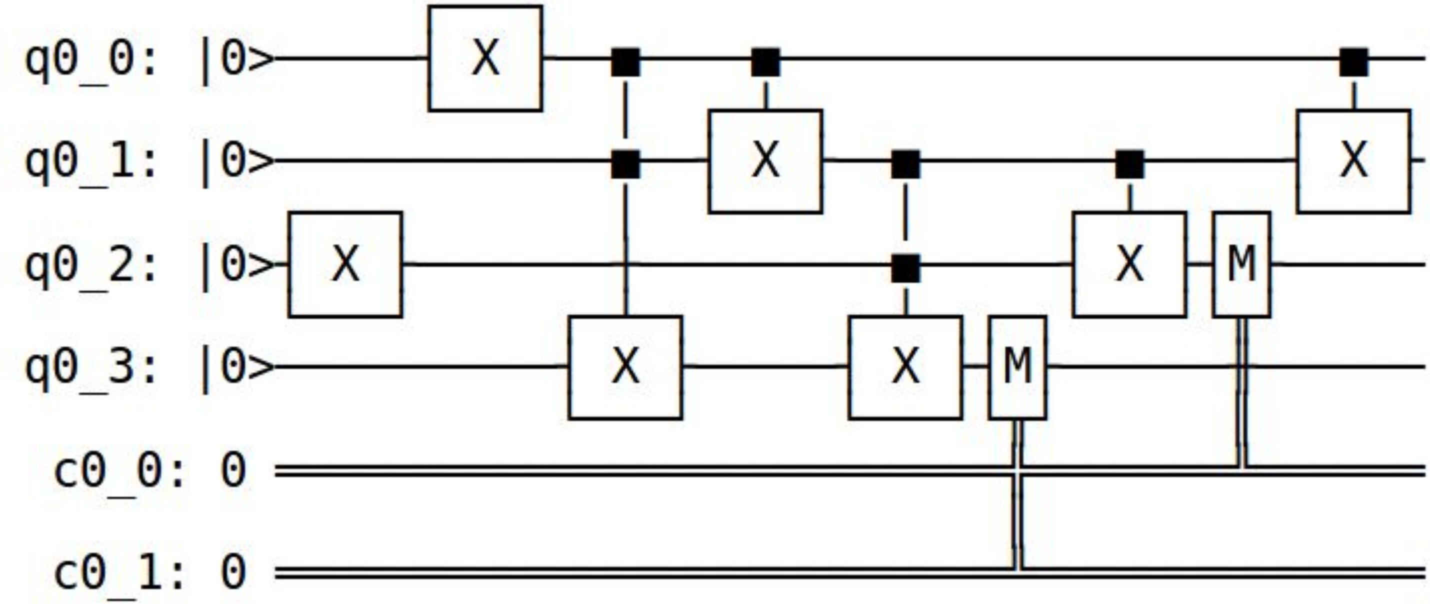


Example : Addition

3+1

Carry		1	0
Int1		1	1
Int2		0	1
	1	0	0

```
cir.draw()
```



Example : Addition

```python
#assign values to int1, int2 qubit
for i in range(len(bitstring1)):
    if int(bitstring1[::-1][i])==1:
        cir.x(int1[i])

for i in range(len(bitstring2)):
    if int(bitstring2[::-1][i])==1:
        cir.x(int2[i])

# calculate carry
for i in range(maxbitlen):
        cir.ccx(int1[i],int2[i],s[i+1])

# calculate sum
for i in range(maxbitlen):
    cir.cx(int1[i],int2[i])

# calculate sum + carry
for i in range(maxbitlen):
    cir.ccx(int2[i],s[i],carry[i+1])
    cir.cx(int2[i],s[i])

#measurement
cir.measure(s,c_s)
cir.measure(carry,c_carry)
```

```python
result=job.result()
bit=get_most_prob_bitstring(result.get_counts())
ans=bin2decimal(bit)
```

```
{'101 000': 16}
Quantum computation of addition: 2 + 3 = 5

{'000011 100000': 16}
Quantum computation of addition: 11 + 24 = 35

{'10100 00000': 16}
Quantum computation of addition: 8 + 12 = 20
```

Thank you for your attention
Stay tuned

https://github.com/hchsu/quantumcomp

- Apply quantum Fourier transform for arithmetic algorithm design.
- Noise model for the qubit computation.
- Benchmark the quantum computing with classical computing.