Quantum Computing

Dr. Andrés Gómez
Andres.gomez.tato@cesga.es
Jan. 2020
Lecture 2

- QPU
- Topologies of QPU
- Programming
- Simulators.
- Introduction to ProjectQ
- My first program with ProjectQ.
Universal Quantum Computer

Quantum Gates

**ONE QUBIT GATES**

- Hadamard:
  \[
  H = \frac{1}{\sqrt{2}} \begin{bmatrix}
  1 & 1 \\
  1 & -1
  \end{bmatrix}
  \]

- Pauli-X:
  \[
  X = \begin{bmatrix}
  0 & 1 \\
  1 & 0
  \end{bmatrix}
  \]

- Pauli-Y:
  \[
  Y = \begin{bmatrix}
  0 & -i \\
  i & 0
  \end{bmatrix}
  \]

- Pauli-Z:
  \[
  Z = \begin{bmatrix}
  1 & 0 \\
  0 & 1
  \end{bmatrix}
  \]

- Phase:
  \[
  S = \begin{bmatrix}
  1 & 0 \\
  0 & i
  \end{bmatrix}
  \]

- \(\pi/8\):
  \[
  T = \begin{bmatrix}
  1 & 0 \\
  0 & e^{i\pi/4}
  \end{bmatrix}
  \]

**TWO QUBIT GATES**

- **controlled-NOT**

- **swap**

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

**Measurement**

**Projection onto \(|0\rangle\) and \(|1\rangle\)**

\(\{H,S,T,CNOT\} = \text{Universal Quantum Gates: can approximate ANY unitary operation}\)

Quantum register initialized to |0> for a finite bit string 0 (n bits = nqubits)

Manipulated by a set of primitives one or two qubit unitary operations (quantum gates)

Measured into a classical bases labeled as a bit string of length n.

A program is provided by a set of gates to be applied onto QuBits.

 Ion Trap (I)

- $^{171}\text{Yb}^+$ trapped ions
- Information stored in the hyperfine ‘clock’ states:
  - $|0\rangle \equiv |F = 0; m_F = 0 >$
  - $|1\rangle \equiv |F = 1; m_F = 0 >$ of the $^2S_{1/2}$ electronic ground level
  - $F, m_F$ quantum numbers associated to total atomic angular momentum and its projection along the quantization axes defined by an applied magnetic field of 5.2G
- Coherence time $> 0.5\text{s}$
- Confined in a linear radio-frequency Paul trap
- Laser-cooled
- 5 µm for 5 qubits

Ion Trap (II): Universal Gates

Implemented gates (using Raman transition):

- single qubit $R_\phi(\theta) = R(\theta, \phi)$
- two - qubits $XX$

Universal Gates derived as:

- $H = R_x(-\pi)R_y\left(\frac{\pi}{2}\right)$
- $Z = R_y(-\pi/2)R_x(\theta)R_y\left(\frac{\pi}{2}\right)$
- Controlled-NOT (CNOT)
- Controlled-Phase (cP)

State preparation (single qubits rotations)

Total gates[1]:
- $5 \text{ H} = 10 \text{ R}$
- $10 \text{ CP} = 10 \left(6\text{ R} + 1 \text{ X}\right)$
- Total = 80 gates

Optimised Version [2]:
- 22 single-qubit gates
- 10 two-qubits gates
- Total: 32 gates
- Time: 2669 $\mu$s


IBM QX

Hardware Gates:
- Frame Change (FC, 0ns),
- Gaussian Derivative (GD, 80ns),
- Gaussian Flattop (GF, 170-384ns)
- Buffer after each pulse: 10ns

Limits on:
- Available gates,
- Different times for each operations
- Restrictions on 2 qubits operations
- Restrictions on measurement

https://github.com/QISKit/ibmqx-backend-information/tree/master/backends/ibmqx3
Quantum Algorithm

✓ A sequence of gates applied on a set of qubits

✓ Some well-known techniques:
  ➢ Amplify amplitude (Grover’s)
  ➢ Estimate phase

✓ More than 400 quantum algorithm papers on Quantum Zoo

✓ Many are unitary transformation on n-qubits plus measurement
  ➢ How to make them as a set of universal gates?
Quantum Programing

- Define one or several quantum registers
- Apply high level methods, a set of gates or general unitary transformations
- Decompose them in basic gates
- Transform to real operations of the QPU
- Or for simulators

# Modern Quantum Languages (I)

<table>
<thead>
<tr>
<th>Core Language</th>
<th>Unlimited Qubits</th>
<th>Simulator</th>
<th>Optimizer/decomp.</th>
<th>CESGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProjectQ Python/C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIQUi</td>
<td>&gt; F# (*</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quipper HasKell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ScaffCC Scaffold</td>
<td></td>
<td>QX</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Microsoft Quantum Development Kit (2018) Q# (**)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QisKit Python</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Xacc C++/python</td>
<td></td>
<td>TNQVM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(*) At least in the public versión  
(**) 30 locally, 40 in Azure
## Modern Quantum Languages (II)

<table>
<thead>
<tr>
<th></th>
<th>Core Language</th>
<th>Unlimited Qubits</th>
<th>Simulator</th>
<th>Optimizer/decomp.</th>
<th>CESGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quest</td>
<td>Python/C</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Red" /></td>
<td><img src="#" alt="Green" /></td>
</tr>
<tr>
<td>Circ (Google)</td>
<td>Python</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="??" /></td>
<td><img src="#" alt="??" /></td>
</tr>
<tr>
<td>Quil (Rigetti)</td>
<td>Python</td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="Green" /></td>
<td><img src="#" alt="??" /></td>
<td><img src="#" alt="??" /></td>
</tr>
</tbody>
</table>
**MainEngine**  
The MainEngine class provides all functionality of the main compiler engine.

**CommandModifier**  
CommandModifier is a compiler engine which applies a function to all incoming commands, sending on the resulting command instead of the original one.

**IBMCNOTMapper**  
CNOT mapper for the IBM backend.

**ManualMapper**  
Manual Mapper which adds QubitPlacementTags to Allocate gate commands according to a user-specified mapping.

**LocalOptimizer**  
LocalOptimizer is a compiler engine which optimizes locally (merging rotations, cancelling gates with their inverse) in a local window of user-defined size.

**AutoReplacer**  
The AutoReplacer is a compiler engine which uses `engine.is_available` in order to determine which commands need to be replaced/decomposed/compiled further.

**TagRemover**  
TagRemover is a compiler engine which removes temporary command tags.

```python
eng = MainEngine(backend=Simulator(),
                 engine_list=[AutoReplacer(rule_set), TagRemover(), LocalOptimizer(3)])
```


### ProjectQ: Backends

<table>
<thead>
<tr>
<th>Backend</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CommandPrinter</td>
<td>CommandPrinter is a compiler engine which prints commands to stdout prior to sending them on to the next compiler engine.</td>
</tr>
<tr>
<td>CircuitDrawer</td>
<td>CircuitDrawer is a compiler engine which generates TikZ code for drawing quantum circuits.</td>
</tr>
<tr>
<td>Simulator</td>
<td>Simulator is a compiler engine which simulates a quantum computer using C++-based kernels.</td>
</tr>
<tr>
<td>ClassicalSimulator</td>
<td>A simple introspective simulator that only permits classical operations.</td>
</tr>
<tr>
<td>ResourceCounter</td>
<td>ResourceCounter is a compiler engine which counts the number of gates and max.</td>
</tr>
<tr>
<td>IBMBackend</td>
<td>The IBM Backend class, which stores the circuit, transforms it to JSON QASM, and sends the circuit through the IBM API.</td>
</tr>
</tbody>
</table>


The setups package contains a collection of setups which can be loaded by the MainEngine.
Each setup then loads its own set of decomposition rules and default compiler engines.
MainEngine(setup=projectq.setups.ibm)

<table>
<thead>
<tr>
<th>default</th>
<th>Defines the default setup which provides an engine_list for the MainEngine</th>
</tr>
</thead>
<tbody>
<tr>
<td>ibm</td>
<td>Defines a setup useful for the IBM QE chip with 5 qubits.</td>
</tr>
<tr>
<td>ibm16</td>
<td>Defines a setup useful for the IBM QE chip with 16 qubits.</td>
</tr>
</tbody>
</table>


All common gates: X, Y, Z, T, S, CNOT, SWAP, etc.
Additional useful gates: Rx, Ry, Rz, R, Ph, Toffoli (ControledControledNOT/ ControledControledX)
Advanced operations:
- Quantum Fourier Transform, QFT
- Hamiltonian Time Evolution, TimeEvolution
- A set of gates to be applied on a set of qubits, QubitOperator
- Create an entangled state: Entangle
- Methods to create multi-qubit controlled gates, ControledGate
- Measurement: Measure

Syntax: <Operation> | <qubits>.
Example: H | qureg[0]
Qiskit

https://qiskit.org/documentation/the_elements.html
Qiskit

- All common gates: X, Y, Z, T, S, CNOT, SWAP, etc.
- Additional useful gates: Rx, Ry, Rz, R, Toffoli
- Specific gates for IBM hardware: u1, cu1, u2, cu2, u3, cu3
- Advanced operations:
  - Quantum Fourier Transform, QFT
  - Hamiltonian Time Evolution, TimeEvolution
  - A set of gates to be applied on a set of qubits, Operator
- Methods to create multi-qubit controlled gates
- Measurement: Measure

- Syntax: gates are methods of the circuit class:
- Example: circuit.h(0) means add a H gate to qubit 0

Qiskit. Backends

- AER.
  - Local simulator
  - Login not needed
  - Three simulators:
    - `Unitary_simulator`: produces the result operator of a circuit
    - `Statevector_simulator`: One shot simulator that returns the state vector
    - `Qasm_simulator`: simulates the real hardware including noise. Return the set of simulated measurements.

- Real backends:
  - Allow execution of circuits on real noisy hardware or advanced simulator
  - Need login
  - Remote execution using queues
Exercise: My First ProjectQ Program

OPEN PROJECTQ/DENSE_CODING NOTEBOOK
Exercise: My First Qiskit Program

OPEN QISKIT/DENSE_CODING NOTEBOOK
Some Basic Concepts with 2 qubits
Phase Kickback

\[ \frac{1}{\sqrt{2}} (|0\rangle + e^{i\varphi} |1\rangle) \]
Toffoli Gate

\[ |a> \quad |a> \\
|b> \quad |b> \\
|c> \quad |c \oplus ab> \]
Inverse Gate?
Exercise: Some Basic Gates with ProjectQ

OPEN PROJECTQ/SOME_BASICS NOTEBOOK
Exercise: Some Basic Gates with Qiskit

OPEN QISKIT/SOME_BASICS NOTEBOOK
Thanks!
Questions?