qLearn: Quantum Compilers and Architecture

Michael Silver - UTQC

Today's Content

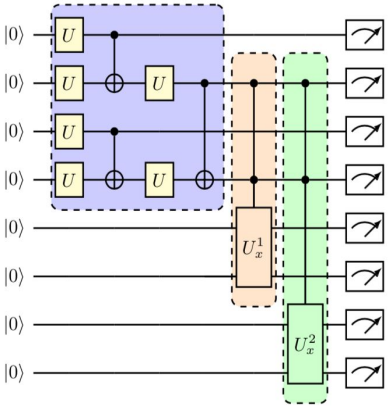
End-User Software Code

Quantum Circuit (Instructions)

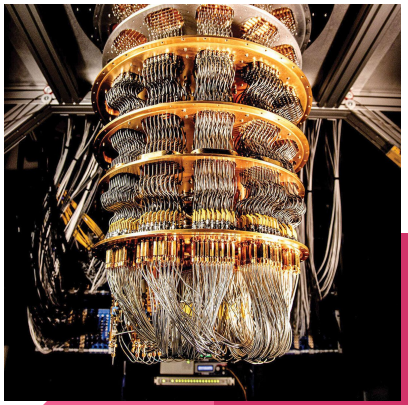
Hardware Performs Algorithm

```
import pennylane as qml
from pennylane import numpy as np

# create a quantum device
dev1 = qml.device('default.qubit', wires=1)

@qml.qnode(dev1)
def circuit(phi1, phi2):
  # a quantum node
  qml.RX(phi1, wires=0)
  qml.RY(phi2, wires=0)
  return qml.expval(qml.PauliZ(0))

def cost(x, y):
  # classical processing
  return np.sin(np.abs(circuit(x, y))) - 1

# calculate the gradient
dcost = qml.grad(cost, argnum=[0, 1]
```

???



???



In Classical Computers…

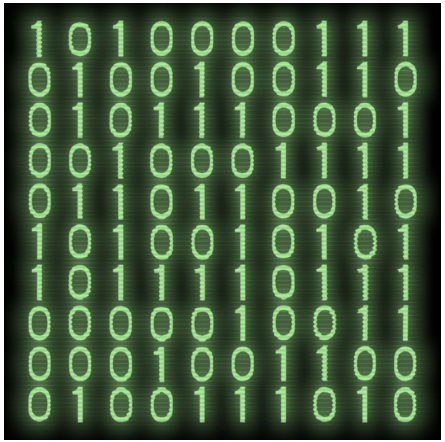
High-Level Code

```
#import the necessary modules!
import random
import string

print('hello, Welcome to Password generator!')

#input the length of password
length = int(input('\nEnter the length of password: '))

#define data
lower = string.ascii_lowercase
upper = string.ascii_uppercase
num = string.digits
symbols = string.punctuation

#combine the data
all = lower + upper + num + symbols

#use random
temp = random.sample(all,length)

#create the password
password = "".join(temp)

#print the password
print(password)
```

Low-Level (Executable) Code

```
start:
    add  $v0, $zero, $zero  # v0 = 0
    add  $t0, $zero, $zero  # t0 = 0

outer:
    sltu $t2, $t0, $a1      # t2 = (t0 < a1)
    beq  $t2, $zero, fin    # if (!t2) goto fin
    lw   $t3, 0($a0)        # t3 = mem[a0]
    addi $t4, $zero, 32     # t4 = 32

inner:
    beq  $t4, $zero, next   # if (!t4) goto next
    andi $t1, $t3, 1        # t1 = t3 & 1
    add  $v0, $v0, $t1      # v0 = v0 + t1
    srl  $t3, $t3, 1        # t3 = t3 >> 1
    subi $t4, $t4, 1        # t4 = t4 - 1
    j    inner              # goto inner

next:
    addi  $t0, $t0, 1       # t0 = t0 + 1
    addi  $a0, $a0, 4       # a0 = a0 + 4
    j     outer             # goto outer

fin:
    jr    $ra               # return to caller
```

Machine Code



Electrical Processing

Today's Content

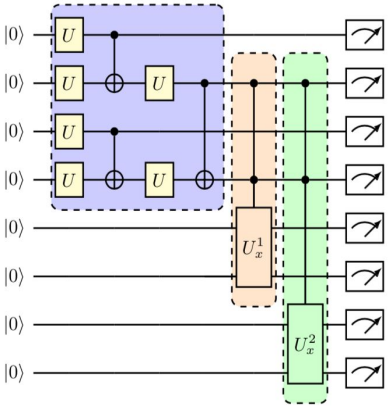
End-User Software Code

Quantum Circuit (Instructions)

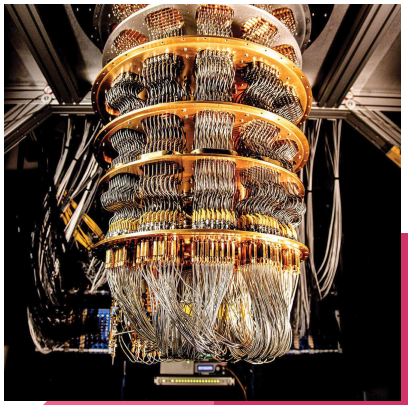
Hardware Performs Algorithm

```
import pennylane as qml
from pennylane import numpy as np

# create a quantum device
dev1 = qml.device('default.qubit', wires=1)

@qml.qnode(dev1)
def circuit(phi1, phi2):
  # a quantum node
  qml.RX(phi1, wires=0)
  qml.RY(phi2, wires=0)
  return qml.expval(qml.PauliZ(0))

def cost(x, y):
  # classical processing
  return np.sin(np.abs(circuit(x, y))) - 1

# calculate the gradient
dcost = qml.grad(cost, argnum=[0, 1]
```

???



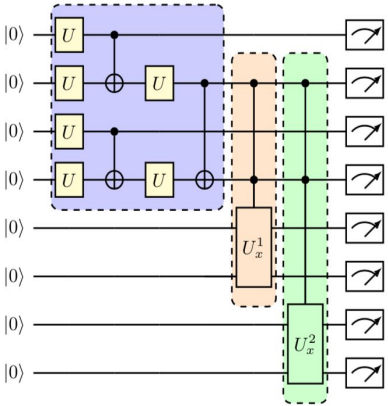
???

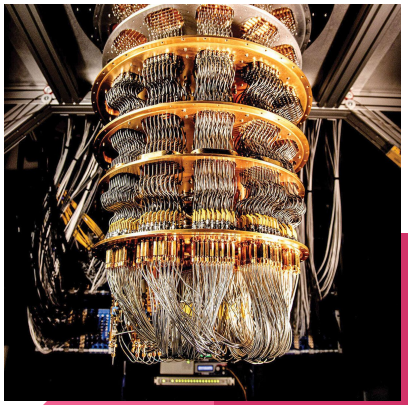


Today's Content

End-User Software Code → **Circuit Transpilation** → **Quantum Circuit (Instructions)** → **Hardware-level Compilation** → **Hardware Performs Algorithm**

```python
import pennylane as qml
from pennylane import numpy as np

# create a quantum device
dev1 = qml.device('default.qubit', wires=1)

@qml.qnode(dev1)
def circuit(phi1, phi2):
  # a quantum node
  qml.RX(phi1, wires=0)
  qml.RY(phi2, wires=0)
  return qml.expval(qml.PauliZ(0))

def cost(x, y):
  # classical processing
  return np.sin(np.abs(circuit(x, y))) - 1

# calculate the gradient
dcost = qml.grad(cost, argnum=[0, 1]
```



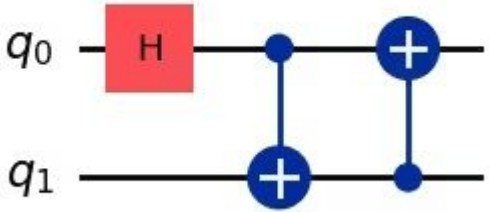


Circuit Transpilation

- **Purpose:** Adapting a high-level circuit into a form compatible with hardware
- **Key Functions**:**
  - Converting abstract gates into hardware-supported gates (Universal Gateset, etc.)
  - Mapping logical qubits to physical qubits
  - Circuit optimization (Minimization)
    - Number of gates/operations
    - Circuit Depth
    - Number of Logical Qubits
- Most hardware/companies have their own proprietary transpilation methods; large area of research (especially privately)

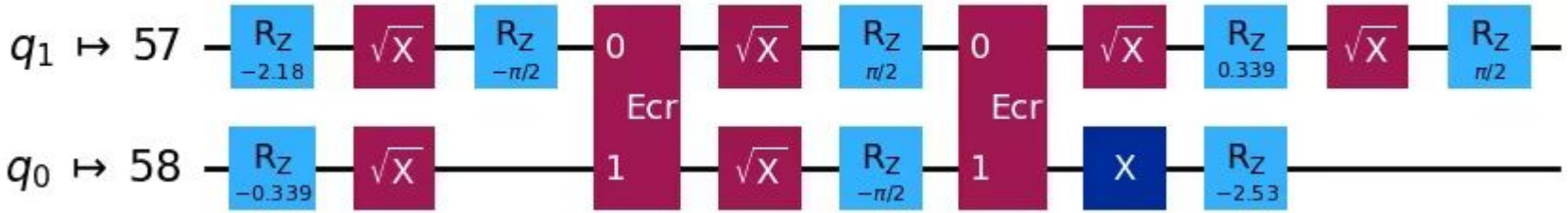
**Note, not in order

Circuit Transpilation Example



Note: Some gates in this example are proprietary IBM Gates (useful in their superconducting systems)

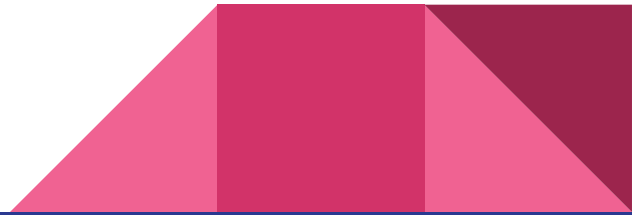
Global Phase: 2π



Cred. IBM Quantum Introduction to Transpilation:
https://docs.quantum.ibm.com/guides/transpile

Qubit Mapping

- **Purpose:** Mapping logical qubits to physical qubits in hardware system
- **Key Actions:**
  - Hardware specific: Creating/Assigning physical qubits in terms of logical qubits; accounting for hardware connectivity (e.g. limited qubit interaction in superconducting qubits due to isolation of qubits)
  - Hardware Specific: Performance Calibration; ensuring performance thresholds are met for circuit timings (e.g. coherence times, gate fidelities)
  - Gate/Operation Scheduling

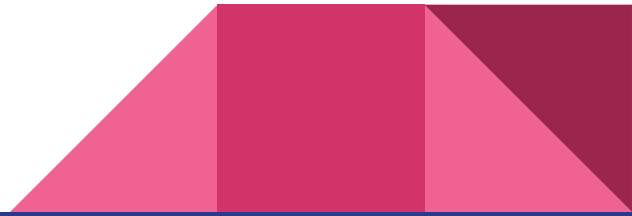


Hardware-Level Compilation

- Hardware Specific; every type of hardware has different types of signals and data it uses to perform calculations and measurements; Re. qLearn hardware section first semester
- **Purpose**: Translate quantum gates and circuits into the low-level control signals needed to manipulate physical qubits
- **Key actions:**
  - Converting gates into control signals specific to qubit type (e.g. trapped ion uses laser pulses)
  - Optimize pulse shapes and timings to minimize errors and crosstalk
  - Synchronize multi-qubit gate operations
- The most intersectional area in QC; engineers needed for signal processing and device control, physicists needed for qubit particle interactions (and everything quantum), computer scientists needed for gate and algorithmic interactions

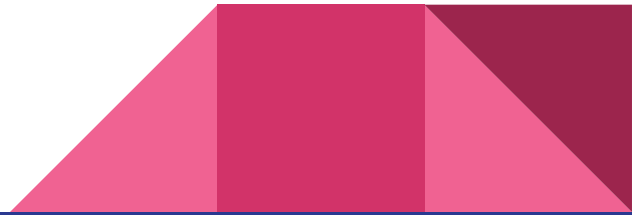
Execution on the Quantum Processing Unit (QPU)

- **Purpose**: Performing the quantum operations defined by the circuit on the physical qubits (using control signals dictated by hardware-level compilation)
- Hardware specific: Covered in Semester 1 Hardware
- E.g. microwave pulses for superconducting, laser pulses for trapped ion, etc.



Measurement and Post-Processing

- **Purpose**: Capture and interpret the results of the quantum computation performed on hardware
- **Key Actions:**
  - Readout pulses; Hardware dependent; signal processing used to extract state information (e.g. Digitizing resonator frequencies in superconducting devices)
  - Post-processing: generally classical analysis of results to derive interpretation of algorithm (mostly seen in application, not theory)



Today's Content