rigetti

Shallow pyQuil circuits

/tbabej

@tomasbabej

Tomas Babej & Mark Fingerhuth

ProteinQure Inc.

/m-fingerhuth

@mark_fingerhuth

Creative Destruction Lab
Toronto, CA
July 12, 2018

ProteinQure

pyQuil: High-level quantum programming

Code examples | Classical control flow | Quantum state preparation | Grove

pyQuil: QVM/QPU Connection

Code examples | Classical control flow | Quantum state preparation | Grove

First step: Establish connection to Rigetti's server

```
from pyquil.api import QVMConnection

qvm = QVMConnection()
```

pyQuil: Program

Code examples | Classical control flow | Quantum state preparation | Grove

Second step: Define an empty pyQuil program

```
from pyquil.api import QVMConnection
from pyquil.quil import Program

qvm = QVMConnection()
p = Program()
```

pyQuil: Quantum circuit

Third step: Define a quantum circuit by modifying the Program()

```
from pyquil.api import QVMConnection
from pyquil.quil import Program
from pyquil.gates import X, H, CNOT

qvm = QVMConnection()
p = Program()

p.inst(H(0), H(1))
```

pyQuil: Quantum circuit

Code examples | Classical control flow | Quantum state preparation | Grove

There is **many different ways** of defining programs!

```
p.inst(H(0), H(1))
```
1. In one .inst() call

```
p.inst(H(0)).inst(H(1))
```
2. in multiple .inst() calls

```
quil = """
H 0
H 1
"""
p = Program(quil)
```
3. First define Quil then load into Program()

```
p = Program("H 0\nH 1")
```
4. In one-line as Quil

```
p.inst([H(0), H(1)])
```
5. With a list of gates

```
p.inst(H(i) for i in range(2))
```
6. Using a generator (so cool!)

pyQuil: Quantum circuit

Fourth step: Don't forget to measure!

```
from pyquil.api import QVMConnection
from pyquil.quil import Program
from pyquil.gates import X, H, CNOT, MEASURE

qvm = QVMConnection()
p = Program()

p.inst(H(0), H(1))
p.inst(MEASURE(0, 0))
```

pyQuil: Quantum circuit

Code examples | Classical control flow | Quantum state preparation | Grove

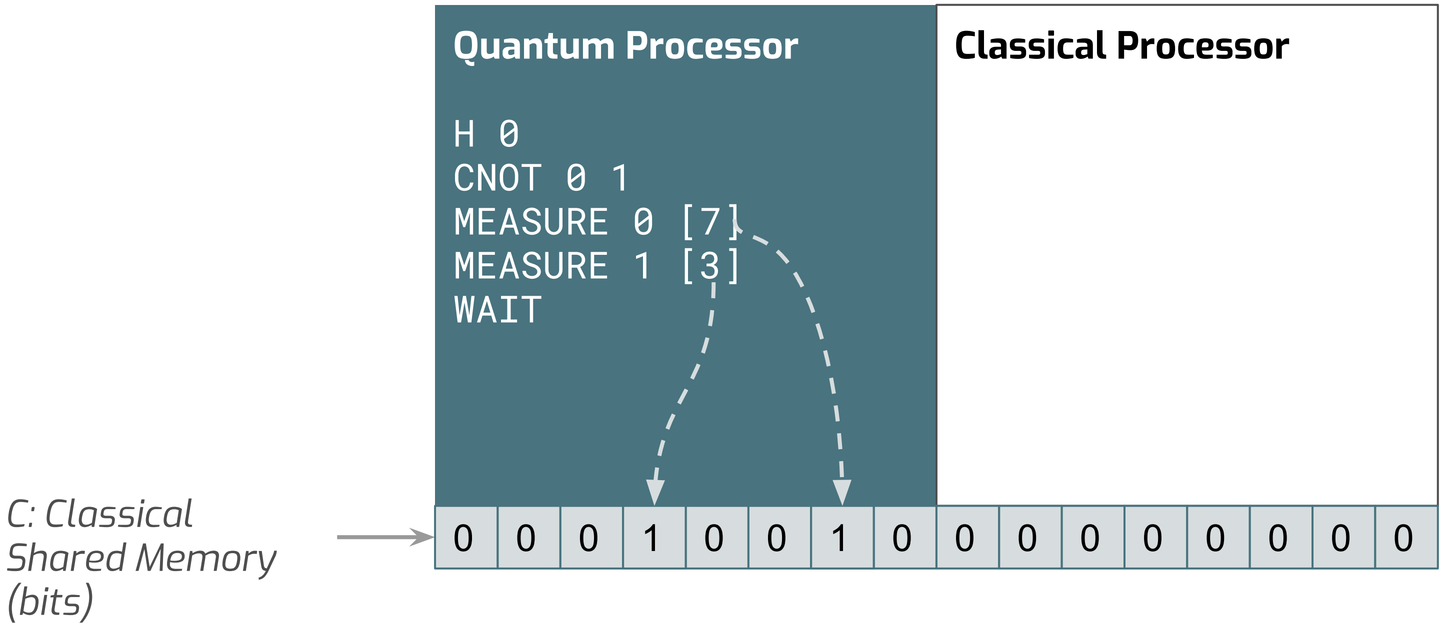
Fourth step: Don't forget to measure!

```
from pyquil.api import QVMConnection
from pyquil.quil import Program
from pyquil.gates import X, H, CNOT, MEASURE

qvm = QVMConnection()
p = Program()

p.inst(H(0), H(1))
p.inst(MEASURE(0, 0))
```

Quantum register   Classical register

pyQuil: Recap on shared classical memory

Code examples | Classical control flow | Quantum state preparation | Grove



pyQuil: Quantum circuit

Code examples | Classical control flow | Quantum state preparation | Grove

There is **many different ways** of measuring!

```
p.inst(MEASURE(0, 0))
```

→ 1. In a .inst() call

```
p.measure(0, 0)
```

→ 2. With a „measure() call

```
quil = """
H 0
H 1
MEASURE 0 [0]
"""
p = Program(quil)
```

→ 3. Defining it in Quil

```
p = Program("H 0\nH 1\nMEASURE 0 [0]")
```

→ 4. In a one-liner as Quil

pyQuil: Quantum circuit

Code examples | Classical control flow | Quantum state preparation | Grove

Fifth step: Run the circuit on the QVM!

```
from pyquil.api import QVMConnection
from pyquil.quil import Program
from pyquil.gates import X, H, CNOT, MEASURE

qvm = QVMConnection()
p = Program()

p.inst(H(0), H(1))
p.inst(MEASURE(0, 0))
p.inst(MEASURE(1, 1))

results = qvm.run(p, trials=10)
```

Don't forget measuring qubit 1!

If you use **qvm.run()** you will have to **define explicit measurements.**

pyQuil: Quantum circuit

Code examples | Classical control flow | Quantum state preparation | Grove

QVM: run() vs. run_and_measure()

```
p.inst(H(0), H(1))
p.inst(MEASURE(0, 0))
p.inst(MEASURE(1, 1))

results = qvm.run(p, trials=10)
```

Results:

[[1, 0], [0, 0], [1, 0], [1, 0], [1, 1],
[1, 1], [1, 0], [0, 0], [0, 0], [0, 0]]

✓

```
p.inst(H(0), H(1))
p.inst(MEASURE(0, 0))
p.inst(MEASURE(1, 1))

results = qvm.run_and_measure(p, [0, 1], trials=10)
```

Results:

[[0, 1], [0, 1], [0, 1], [0, 1], [0, 1],
[0, 1], [0, 1], [0, 1], [0, 1], [0, 1]]

?

If you use **qvm.run()** you will sample from the distribution
but using **qvm.run_and_measure()** will always give you the same result.

pyQuil: Quantum circuit

Code examples | Classical control flow | Quantum state preparation | Grove

QVM: run() vs. run_and_measure()

```
p.inst(H(0), H(1))
p.inst(MEASURE(0, 0))
p.inst(MEASURE(1, 1))

results = qvm.run(p, trials=10)
```

Results:

[[1, 0], [0, 0], [1, 0], [1, 0], [1, 1],
[1, 1], [1, 0], [0, 0], [0, 0], [0, 0]] ✓

```
p.inst(H(0), H(1))
p.inst(MEASURE(0, 0))
p.inst(MEASURE(1, 1))

results = qvm.run_and_measure(p, [0, 1], trials=10)
```

Results:

[[0, 1], [0, 1], [0, 1], [0, 1], [0, 1],
[0, 1], [0, 1], [0, 1], [0, 1], [0, 1]] ?

The reason being that **qvm.run_and_measure() determines the final wavefunction once** and then samples from it.

pyQuil: Quantum circuit

Code examples | Classical control flow | Quantum state preparation | Grove

QVM: run() vs. run_and_measure()

```
p.inst(H(0), H(1))
p.inst(MEASURE(0, 0))
p.inst(MEASURE(1, 1))

results = qvm.run(p, trials=10)
```

Results:

[[1, 0], [0, 0], [1, 0], [1, 0], [1, 1],
[1, 1], [1, 0], [0, 0], [0, 0], [0, 0]] ✓

Classical addresses in shared memory

```
p.inst(H(0), H(1))
~~p.inst(MEASURE(0, 0))~~
~~p.inst(MEASURE(1, 1))~~

results = qvm.run_and_measure(p,[0, 1], trials=10)
```

Results:

[[1, 0], [0, 0], [1, 0], [1, 0], [1, 1],
[1, 1], [1, 0], [0, 0], [0, 0], [0, 0]] ✓

Do NOT include measurements into the Program()
if using **qvm.run_and_measure()!**

pyQuil: Classical control flow

When programming classical computers we often use **if/else** statements. For example:

```
heads, tails = 0, 1
state = lambda: random.randint(0,1)

if state is heads:
    state = tails
else:
    pass

print('It's tails!')
```

How can we implement this with pyQuil?

pyQuil: Classical control flow

Code examples | **Classical control flow** | Quantum state preparation | Grove

Classical

```
state = lambda: random.randint(0,1)


if state is heads:
    state = tails
else:
    pass


print('It's tails!')
```

Quantum

```
state_register  = 0
branching_prog = Program(H(0))
branching_prog.measure(0,  state_register)
```

Quantum coin flip

pyQuil: Classical control flow

Classical

```
state = lambda: random.randint(0,1)

if state is heads:
    state = tails
else:
    pass

print('It's tails!')
```

Quantum

```
state_register  = 0
branching_prog = Program(H(0))
branching_prog.measure(0,  state_register)

then_branch = Program(X(0))
else_branch = Program()
branching_prog.if_then(state_register,
then_branch, else_branch)

branching_prog.measure(0,  state_register)

print('It's tails!')
```

if/else statement

pyQuil: Classical control flow

The **if_then()** statement in **pyQuil** and **QUIL**

```
state_register  = 0
branching_prog = Program(H(0))
branching_prog.measure(0,  state_register)


then_branch = Program(X(0))
else_branch = Program()
branching_prog.if_then(state_register,   then_branch,
else_branch)

branching_prog.measure(0,  state_register)

print('It's tails!')
```

```
H 0
MEASURE 0 [0]
JUMP-WHEN @THEN3 [0]
JUMP @END4
LABEL @THEN3
X 0
LABEL @END4
MEASURE 0 [0]
```

pyQuil: Classical control flow

When programming classical computers we often use **while** statements. For example:

Coin flip

```
wait = True
check_if_go_time = lambda: random.randint(0,1)

while wait:
    wait = check_if_go_time()

print('Go!')
```

How can we implement this with pyQuil?

pyQuil: Classical control flow

Code examples | **Classical control flow** | Quantum state preparation | Grove

Classical

```
wait = True

check_if_go_time = lambda:
random.randint(0,1)

while wait:
    wait = check_if_go_time()

print('Go!')
```

Quantum

```
wait = 2
init_register = Program(TRUE([wait]))
```

Keeps initializing the classical register *wait* to True

pyQuil: Classical control flow

Code examples | **Classical control flow** | Quantum state preparation | Grove

Classical

```
wait = True

check_if_go_time = lambda:
random.randint(0,1)

while wait:
    wait = check_if_go_time()

print('Go!')
```

Quantum

```
wait = 2
init_register = Program(TRUE([wait]))

check_if_go_time = Program(H(0)).measure(0, wait)
```

Quantum coin flip

pyQuil: Classical control flow

Code examples | **Classical control flow** | Quantum state preparation | Grove

Classical

```
wait = True

check_if_go_time = lambda:
random.randint(0,1)

while wait:
    wait = check_if_go_time()

print('Go!')
```

Quantum

```
wait = 2
init_register = Program(TRUE([wait]))

check_if_go_time = Program(H(0)).measure(0, wait)

loop_prog = init_register.while_do(wait,
check_if_go_time)
qvm.run(loop_prog)

print('Go!')
```

A quantum while loop!

pyQuil: Classical control flow

The **while_do()** statement in **pyQuil** and **QUIL**

```
wait = 2

init_register = Program(TRUE([wait]))
check_if_go_time = Program(H(0)).measure(0, wait)

loop_prog = init_register.while_do(wait,  check_if_go_time)

qvm.run(loop_prog)

print('Go!')
```

```
TRUE [2]
LABEL @START1
JUMP-UNLESS @END2 [2]
H 0
MEASURE 0 [2]
JUMP @START1
LABEL @END2
```

Quantum state preparation

For many quantum algorithms you need to
load data into a quantum computer:

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \Rightarrow |\Psi\rangle = \sum_{i=0}^{N-1} \frac{a_i}{|\mathbf{a}|} |i\rangle$$

Quantum state preparation

This is a **non-trivial problem** and still an **active field of research**.
In the near future, you might still have to prepare quantum states manually.

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \dots \\ a_N \end{bmatrix} \Rightarrow |\Psi\rangle = \sum_{i=0}^{N-1} \frac{a_i}{|\mathbf{a}|} |i\rangle$$

Grove

Code examples | Classical control flow | Quantum state preparation | **Grove**

Grove

Grove is a library of quantum algorithms implemented in **pyQuil**:

- Variational Quantum Eigensolver (**VQE**)
- Quantum Approximate Optimization Algorithm (**QAOA**)
- Quantum Fourier Transform (**QFT**)
- **Phase Estimation** Algorithm
- **Grover** Search
- Arbitrary **State Generation**

In **tomorrow's tutorial** you will explore VQE and QAOA in more depth.

rigetti

Now, please start working through the exercises in the Jupyter Notebook for

Tutorial 3

ProteinQure