rigetti

Programming the QPU

in /tbabej
@tomasbabej

Tomas Babej & Mark Fingerhuth
ProteinQure Inc.

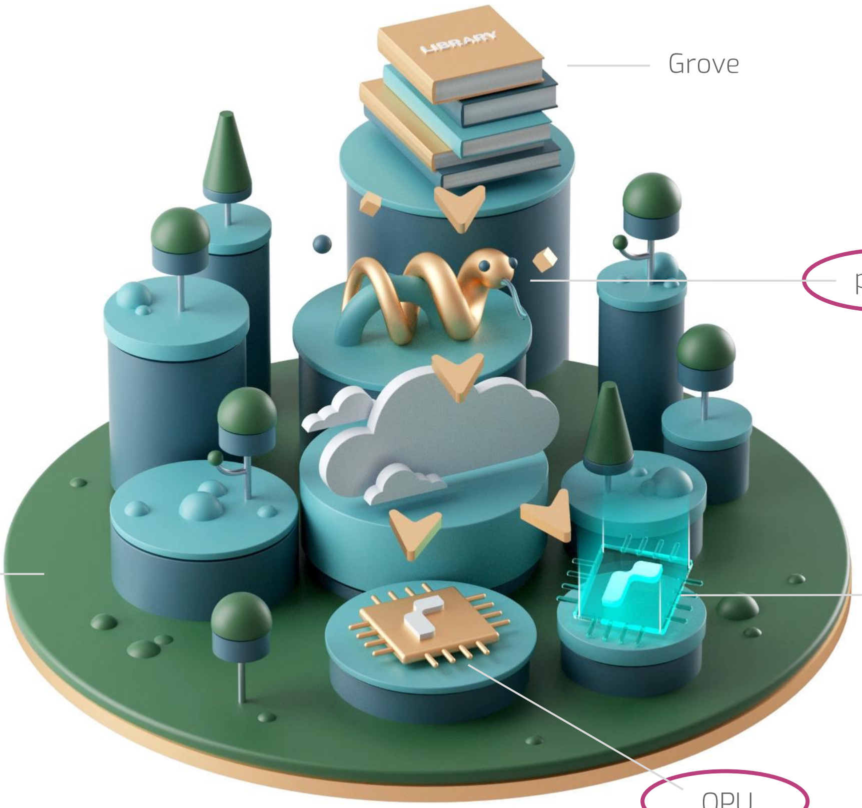
/m-fingerhuth in
@mark_fingerhuth

Creative Destruction Lab
Toronto, CA
July 12, 2018

ProteinQure

QPU: Accessing the real device

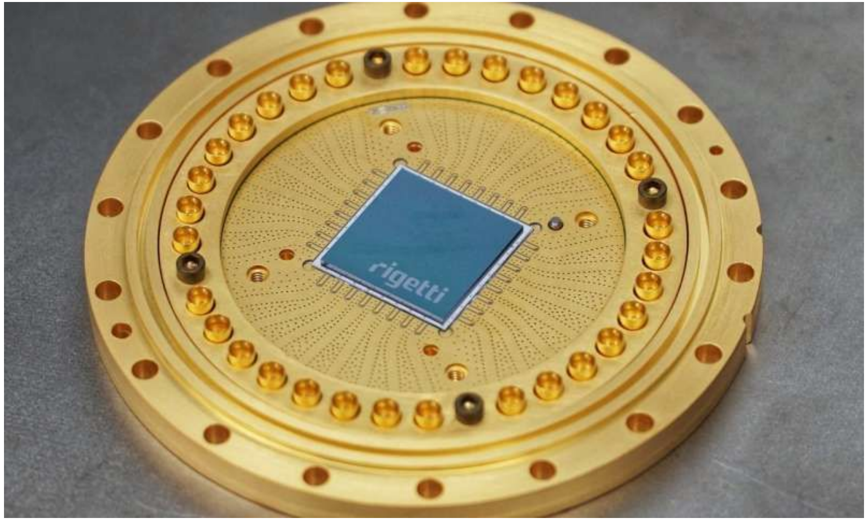
QPU | API | Decoherence & Noise | Compilation



Rigetti 19Q-Acorn

Latest generation QPU
(currently offline)

19 superconducting qubits

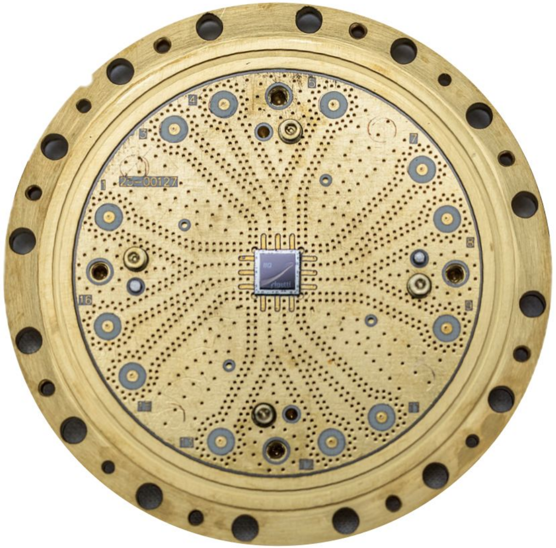


Rigetti 8Q-Agave

QPU | API | Decoherence & Noise | Compilation

Previous generation QPU
(currently online)

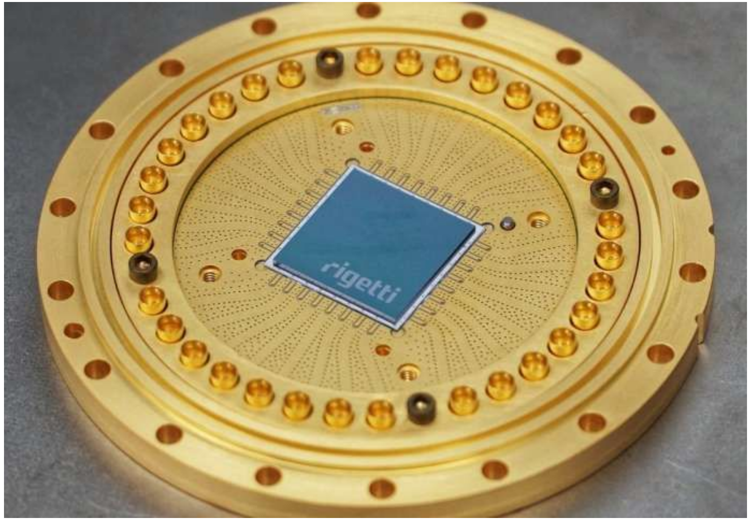
8 superconducting qubits

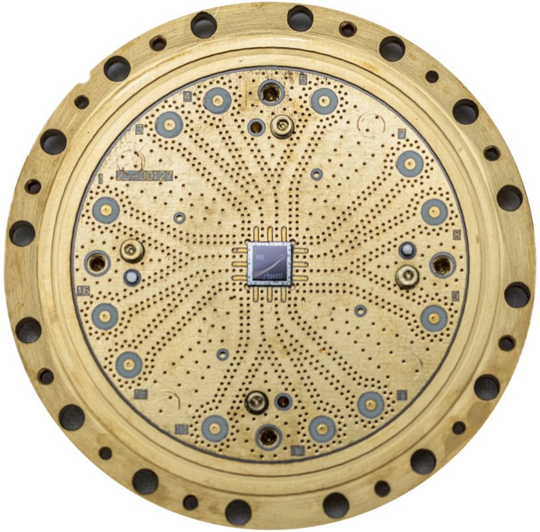


Metrics of a QPU

QPU | API | Decoherence & Noise | Compilation

How does QPUs differ from one another? It's not just about the **qubits**..





Metrics of a QPU

Table 1 | Rigetti 8Q performance

	$\omega_{\mathrm{r}}^{\max}/2\pi$	$\omega_{01}^{\max}/2\pi$	T_1	T_2^*	F_{1q}	F_{RO}
	MHz	MHz	μs	μs		
0	5863	4586	10.72	10.6	0.957	0.784
1	5293	3909	10.04	9.2	0.951	0.910
2	5713	4524	15.52	12.5	0.982	0.943
3	5411	4054	14.17	18.5	0.970	0.912
4	5620	4660	14.58	26.2	0.969	0.678
5	5171	4081	14.86	12.8	0.962	0.832
6	5751	4760	14.17	12.9	0.969	0.753
7	5454	4110	13.19	17.7	0.932	0.895

Metrics of a QPU

QPU | API | Decoherence & Noise | Compilation

T1/T2 relaxation time

Table 1 | Rigetti 8Q performance

	$\omega_r^{max}/2\pi$	$\omega_{01}^{max}/2\pi$	T_1	T_2^*	F_{1q}	F_{RO}
	MHz	MHz	μs	μs		
0	5863	4586	10.72	10.6	0.957	0.784
1	5293	3909	10.04	9.2	0.951	0.910
2	5713	4524	15.52	12.5	0.982	0.943
3	5411	4054	14.17	18.5	0.970	0.912
4	5620	4660	14.58	26.2	0.969	0.678
5	5171	4081	14.86	12.8	0.962	0.832
6	5751	4760	14.17	12.9	0.969	0.753
7	5454	4110	13.19	17.7	0.932	0.895

Metrics of a QPU

QPU | API | Decoherence & Noise | Compilation

Gate fidelities

Table 1 | Rigetti 8Q performance

	$\omega_r^{\max}/2\pi$	$\omega_{01}^{\max}/2\pi$	T_1	T_2^*	F_{1q}	F_{RO}
	MHz	MHz	μs	μs		
0	5863	4586	10.72	10.6	0.957	0.784
1	5293	3909	10.04	9.2	0.951	0.910
2	5713	4524	15.52	12.5	0.982	0.943
3	5411	4054	14.17	18.5	0.970	0.912
4	5620	4660	14.58	26.2	0.969	0.678
5	5171	4081	14.86	12.8	0.962	0.832
6	5751	4760	14.17	12.9	0.969	0.753
7	5454	4110	13.19	17.7	0.932	0.895

Metrics of a QPU

QPU | API | Decoherence & Noise | Compilation

Readout fidelities

Table 1 | Rigetti 8Q performance

	$\omega_r^{max}/2\pi$	$\omega_{01}^{max}/2\pi$	T_1	T_2^*	F_{1q}	F_{RO}
	MHz	MHz	μs	μs		
0	5863	4586	10.72	10.6	0.957	0.784
1	5293	3909	10.04	9.2	0.951	0.910
2	5713	4524	15.52	12.5	0.982	0.943
3	5411	4054	14.17	18.5	0.970	0.912
4	5620	4660	14.58	26.2	0.969	0.678
5	5171	4081	14.86	12.8	0.962	0.832
6	5751	4760	14.17	12.9	0.969	0.753
7	5454	4110	13.19	17.7	0.932	0.895

Metrics of a QPU

QPU | API | Decoherence & Noise | Compilation

Table 3 | Rigetti 19Q performance

	$\omega_r^{max}/2\pi$	$\omega_{01}^{max}/2\pi$	$\eta/2\pi$	T_1	T_2^*	F_{1q}	F_{RO}
	MHz	MHz	MHz	μs	μs		
0	5592	4386	-208	**15.2** $\pm$ 2.5	**7.2** $\pm$ 0.7	0.9815	0.938
1	5703	4292	-210	**17.6** $\pm$ 1.7	**7.7** $\pm$ 1.4	0.9907	0.958
2	5599	4221	-142	**18.2** $\pm$ 1.1	**10.8** $\pm$ 0.6	0.9813	0.97
3	5708	3829	-224	**31.0** $\pm$ 2.6	**16.8** $\pm$ 0.8	0.9908	0.886
4	5633	4372	-220	**23.0** $\pm$ 0.5	**5.2** $\pm$ 0.2	0.9887	0.953

Notice the substantial **improvement in all the metrics** compared to previous generation.

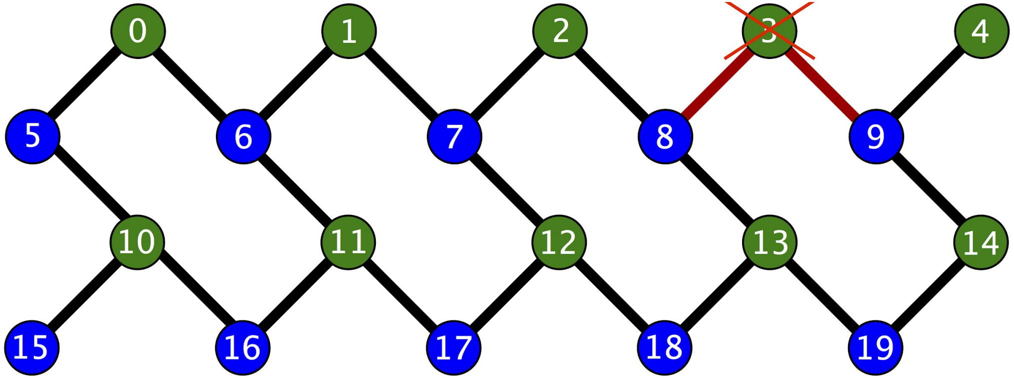
Metrics of a QPU

QPU | API | Decoherence & Noise | Compilation

Not just the number of qubits, but also:

- Decoherence times
- Readout fidelities
- Gate fidelities

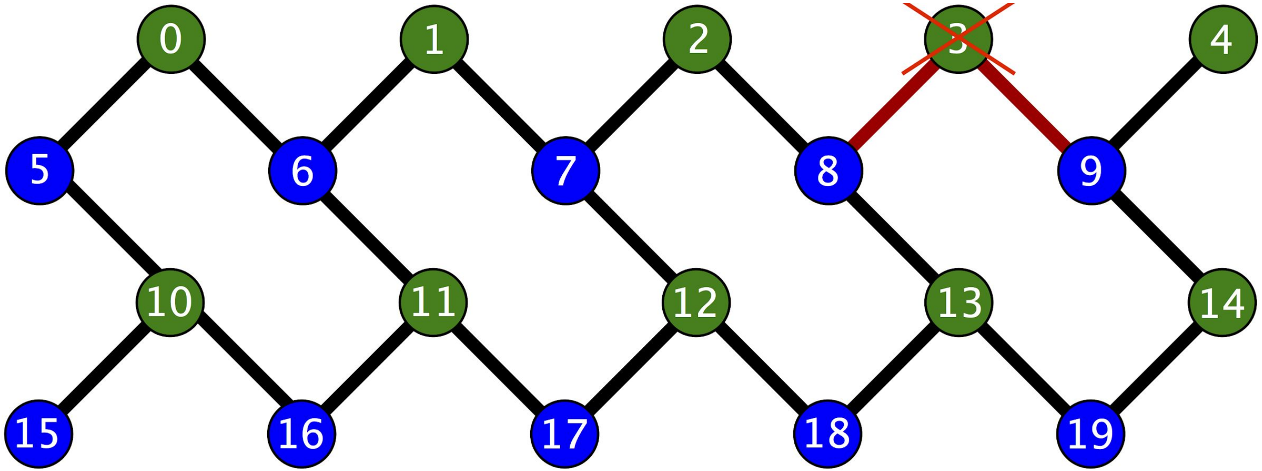
Circuit depth



Chip topology

Chip topology

QPU | API | Decoherence & Noise | Compilation



Not fully connected -> might require **topological SWAPs**

Working with the QPU

QPU | **API** | Decoherence & Noise | Compilation

Just replace QVMConnection with **QPUConnection**!

```
>>> from pyquil.api import QVMConnection()
>>> qvm = QVMConnection()
```

↓

```
>>> from pyquil.api import QPUConnection, get_devices
>>> acorn = get_devices(as_dict=True)['19Q-Acorn']
>>> qpu = QPUConnection(device=acorn)
```

Simulating the QPU

QPU | **API** | Decoherence & Noise | Compilation

Specify the **device** you want the QVM to simulate

```
>>> from pyquil.api import QVMConnection()
>>> qvm = QVMConnection()
```

↓

```
>>> from pyquil.api import QVMConnection, get_devices
>>> acorn = get_devices(as_dict=True)['19Q-Acorn']
>>> qvm = QVMConnection(device=acorn)
```

Simulating the QPU

QPU | API | **Decoherence & Noise** | Compilation

```
>>> from pyquil.api import QVMConnection, get_devices
>>> from pyquil.gates import H, MEASURE
>>> from pyquil.quil import Program

>>> acorn = get_devices(as_dict=True)['19Q-Acorn']
>>> qvm = QVMConnection(device=acorn)

>>> program = Program([H(0), H(0), MEASURE(0, 0)])
>>> qvm.run(program, trials=10)
[[0], [0], [0], [0], [0], [0], [0], [0], [1], [0]]
```

Simulating the QPU

QPU | API | **Decoherence & Noise** | Compilation

```
>>> from pyquil.api import QVMConnection, get_devices
>>> acorn = get_devices(as_dict=True)['19Q-Acorn']
>>> qvm = QVMConnection(device=acorn)
```

Provides a way of **estimating the performance** of your circuit

- Simulates readout error
- Uses simulated noisy gates to implement gate fidelities
- Has no notion of time

Compiling code to the QPU

QPU | API | Decoherence & Noise | **Compilation**

pyQuil

```
H 0
CNOT 0 1
MEASURE 0 [0]
MEASURE 1 [1]
```

compilation →

```
RZ(pi/2) 0
RX(pi/2) 0
RZ(-pi/2) 1
RX(pi/2) 1
CZ 1 0
RZ(-pi/2) 0
RX(-pi/2) 1
RZ(pi/2) 1
MEASURE 0 [0]
MEASURE 1 [1]
```

QPU

Compiling code to the QPU

QPU | API | Decoherence & Noise | **Compilation**

pyQuil

```
H 0
CNOT 0 1
MEASURE 0 [0]
MEASURE 1 [1]
```

compilation →

```
RZ(pi/2) 0
RX(pi/2) 0
RZ(-pi/2) 1
RX(pi/2) 1
CZ 1 0
RZ(-pi/2) 0
RX(-pi/2) 1
RZ(pi/2) 1
MEASURE 0 [0]
MEASURE 1 [1]
```

QPU

Why make things more complex?

Compiling code to the QPU

QPU | API | Decoherence & Noise | **Compilation**

QPUs implements only a **restricted set of quantum gates** (by definition, since there are infinitely many!)

RZ(θ), **RX(kπ/2)** and **CZ**

Still universal, but **gate decomposition is required.**

Problems with compilation

Gate decomposition **introduces overhead** in your circuit

4 instr.

```
H 0
CNOT 0 1
MEASURE 0 [0]
MEASURE 1 [1]
```

compilation →

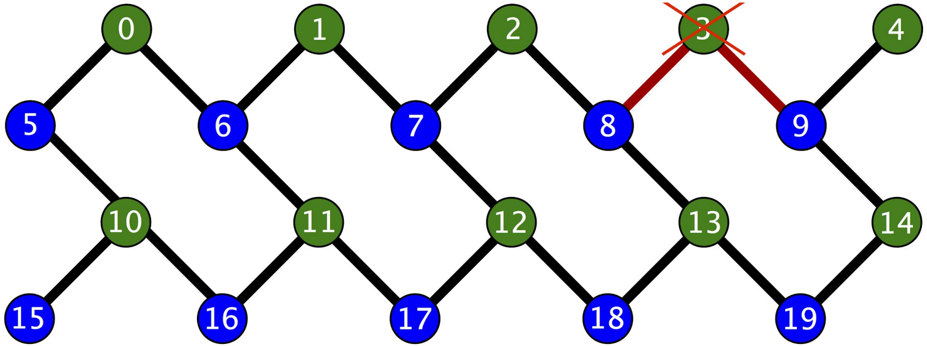
```
RZ(pi/2) 0
RX(pi/2) 0
RZ(-pi/2) 1
RX(pi/2) 1
CZ 1 0
RZ(-pi/2) 0
RX(-pi/2) 1
RZ(pi/2) 1
MEASURE 0 [0]
MEASURE 1 [1]
```

10 instr.

Problems with compilation

QPU | API | Decoherence & Noise | **Compilation**

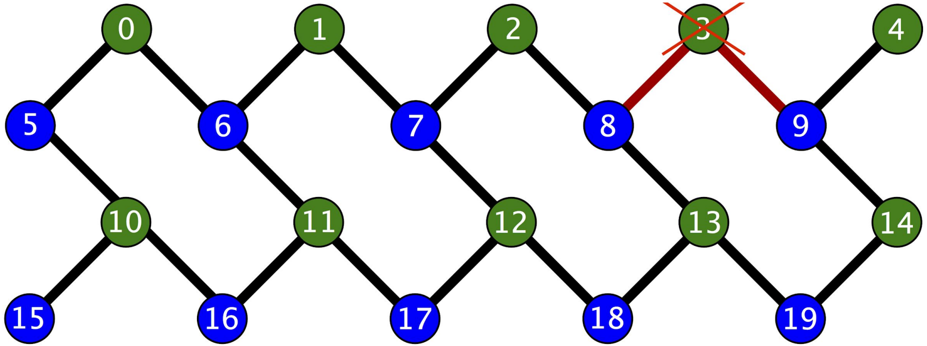
QPU instructions **must respect** graph topology



~~CZ 1 0~~

Problems with compilation

QPU instructions **must respect** graph topology



~~CZ 1 0~~ → Qubits 1 and 0 **are not connected**

Compiling code to the QPU

QPU | API | Decoherence & Noise | **Compilation**

```
>>> from pyquil.api import CompilerConnection, get_devices
>>> from pyquil.gates import H, CNOT, MEASURE
>>> from pyquil.quil import Program

>>> program = Program([H(0), CNOT(0, 1), MEASURE(0, 0), MEASURE(1, 1)])

>>> agave = get_devices(as_dict=True)['8Q-Agave']
>>> compiler = CompilerConnection(device=agave)
>>> compiled = compiler.compile(program)
```

Compiling code to the QPU

QPU | API | Decoherence & Noise | **Compilation**

```
>>> from pyquil.api import CompilerConnection, get_devices
>>> from pyquil.gates import H, CNOT, MEASURE
>>> from pyquil.quil import Program

>>> program = Program([H(0), CNOT(0, 1), MEASURE(0, 0), MEASURE(1, 1)])

>>> agave = get_devices(as_dict=True)['8Q-Agave']
>>> compiler = CompilerConnection(device=agave)
>>> compiled = compiler.compile(program)
```

Why compile manually?

Benefits of manual compilation

QPU | API | Decoherence & Noise | **Compilation**

Compilation usually happens **behind the scenes**

```
>>> program = Program([H(0), H(0), MEASURE(0, 0)])
>>> qpu = QPUConnection(device=acorn)
>>> qpu.run(program)
```

Program is **first compiled**, then run on QPU

Compiled version enables you to:

- Properly **debug** actual circuit run on the device
- Accurately **observe** the circuit depth
- **Appreciate** all layers of complexity that make your quantum code work

rigetti

Now, please start working through the exercises in the Jupyter Notebook for **Tutorial 4**

Brought to you by

/tbabej

@tomasbabej

/m-fingerhuth

@mark_fingerhuth