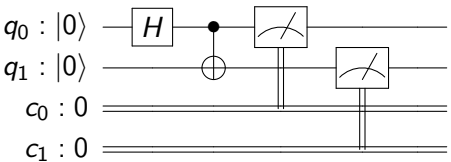
Building a Compiler for Quantum Computers

Matthew Treinish
Software Engineer - IBM Research
mtreinish@kortar.org
`mtreinish on Freenode`
https://github.com/mtreinish/quantum-compilers

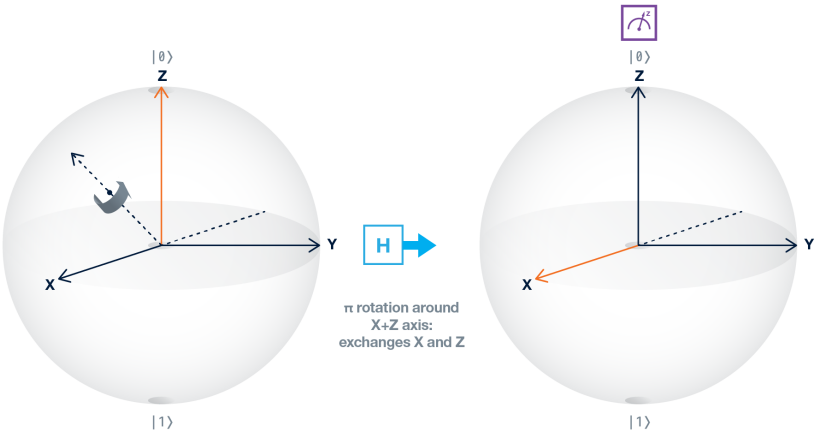
January 17, 2020

Quantum Circuits



Quantum Gates

- Quantum gates
- Gates are reversible
- Each gate can be represented as a unitary matrix



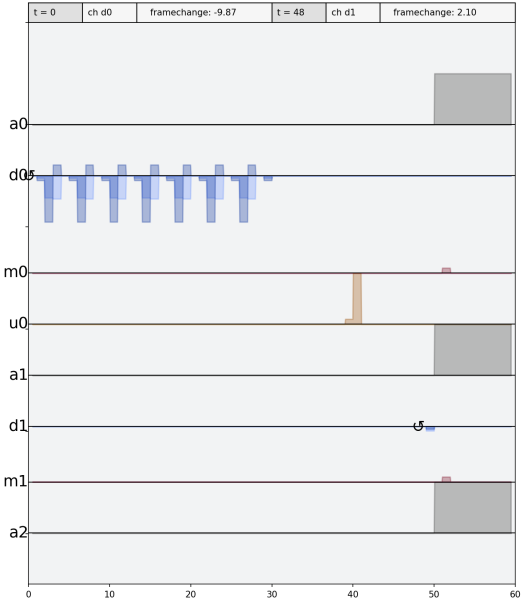
OpenQASM[1][2]

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[2];
creg c[2];
h q[0];
cx q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

- Open Quantum Assembly Language
- An option
- Mostly used as a transport layer to save circuits or share circuits

[1] https://arxiv.org/abs/1707.03429
[2] https://github.com/Qiskit/openqasm

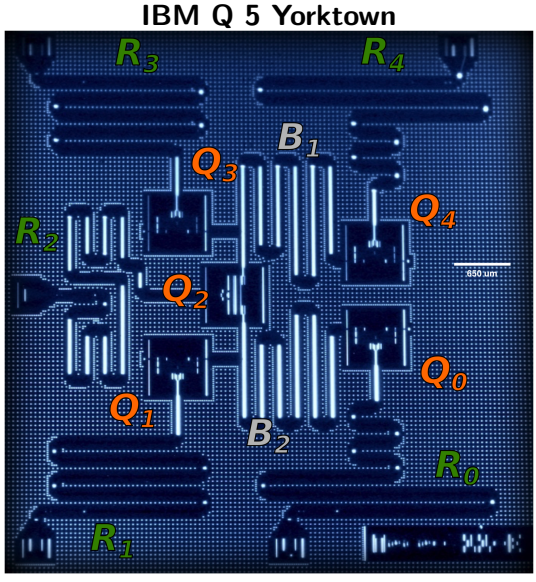
Pulse Level Programming



- A layer below the circuit model is to use pulses
- Each quantum gate is defined as a pulse that gets applied to the qubits
- Enables you to tweak the definition of gates
- Mostly used for physics research or hardware characterization

Qubit Connectivity

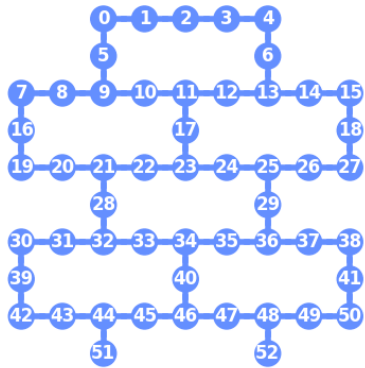
- Qubits in a device have limited connectivity
- For multi-qubit gates this means we can only run them between those qubits
- Need to use SWAP gates to move state around



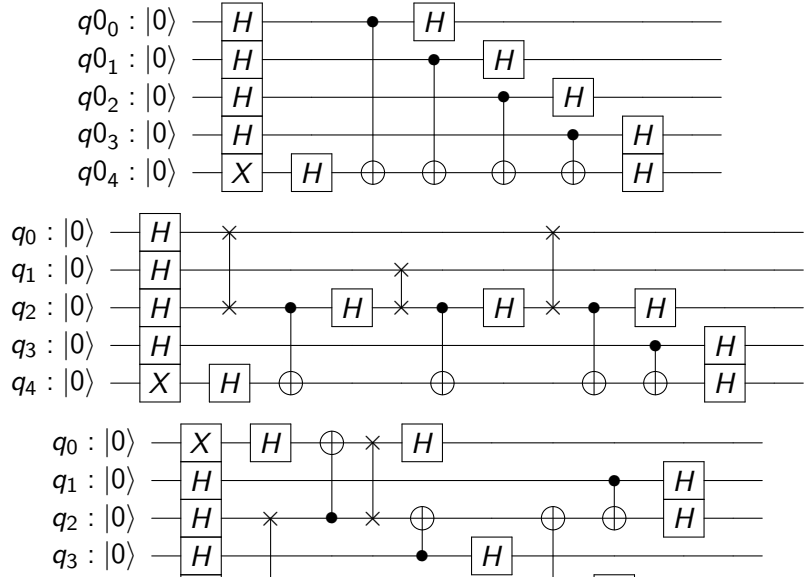
Qubit Connectivity

- Qubits in a device have limited connectivity
- For multi-qubit gates this means we can only run them between those qubits
- Need to use SWAP gates to move state around

IBM Q 53 Rochester Coupling Map



Mapping

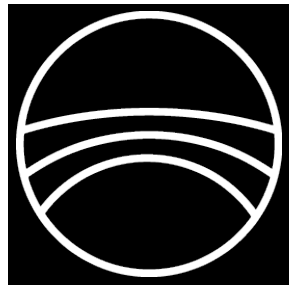

$q0_0 : |0\rangle$
$q0_1 : |0\rangle$
$q0_2 : |0\rangle$
$q0_3 : |0\rangle$
$q0_4 : |0\rangle$
$q_0 : |0\rangle$
$q_1 : |0\rangle$
$q_2 : |0\rangle$
$q_3 : |0\rangle$
$q_4 : |0\rangle$
$q_0 : |0\rangle$
$q_1 : |0\rangle$
$q_2 : |0\rangle$
$q_3 : |0\rangle$

Basis Gates

Unrolling

Qiskit Terra[1]

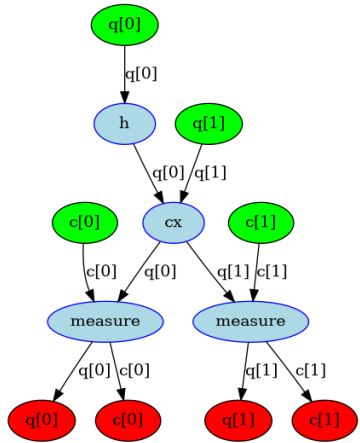
- Is the base layer for working with quantum computers provides interface to hardware and simulators
- Provides an SDK for working with quantum circuits
- Compiles circuits to run on different backends
- Designed to be backend agnostic and work with any quantum hardware or simulator
- Written in Python
- Apache 2.0 Licensed

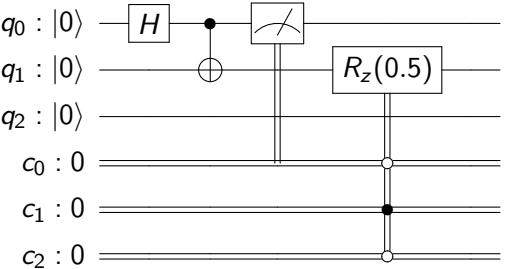


[1] https://github.com/Qiskit/qiskit-terra

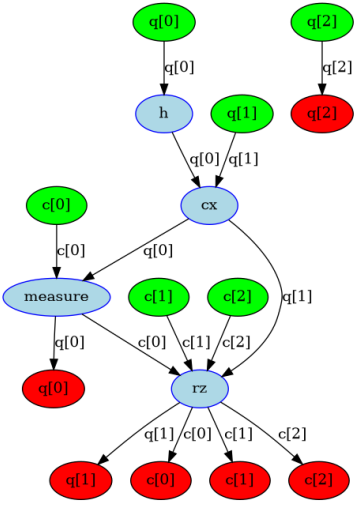
The DAG

- Compiler represents circuits as a DAG
- Each node in the DAG is an operation, an input, or an output
- Each edge indicates data flow between nodes
- Makes flow of information between operations explicit

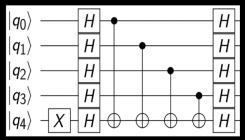



$q_0 : |0\rangle$
$q_1 : |0\rangle$
$q_2 : |0\rangle$
$c_0 : 0$
$c_1 : 0$
$c_2 : 0$
H
$R_z(0.5)$

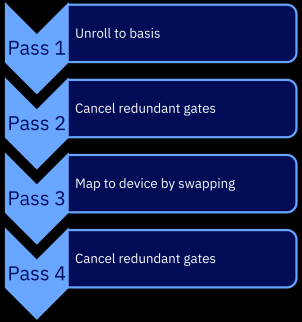
$\rightarrow$

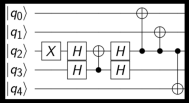

q[0]
q[2]
q[0]
q[2]
h
q[1]
q[2]
q[0]
q[1]
c[0]
cx
c[0]
q[0]
measure
c[1]
c[2]
q[1]
q[0]
c[0]
c[1]
c[2]
q[0]
rz
q[1]
c[0]
c[1]
c[2]
q[1]
c[0]
c[1]
c[2]

Transpiler Architecture

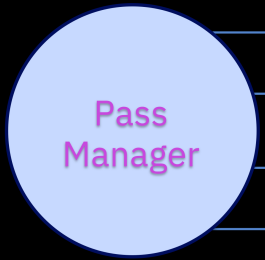


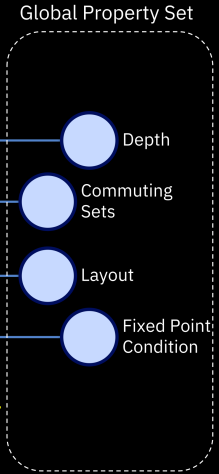
Each pass does one small, well-defined task.





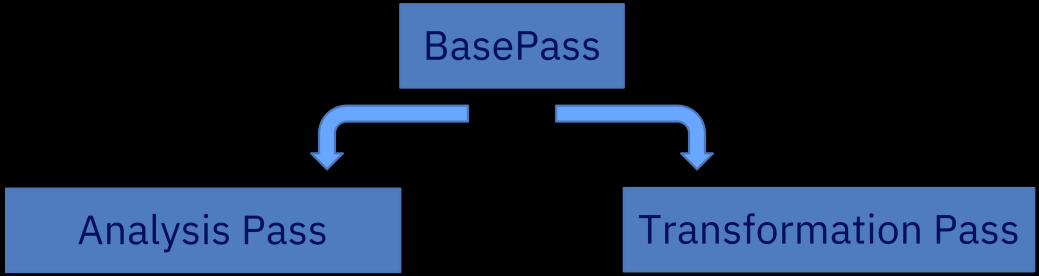
The complexity of scheduling is solely in the pass manager, to keep passes simple





The pass manager maintains a global context about the circuit as it runs through the pipleine

Transpiler Architecture



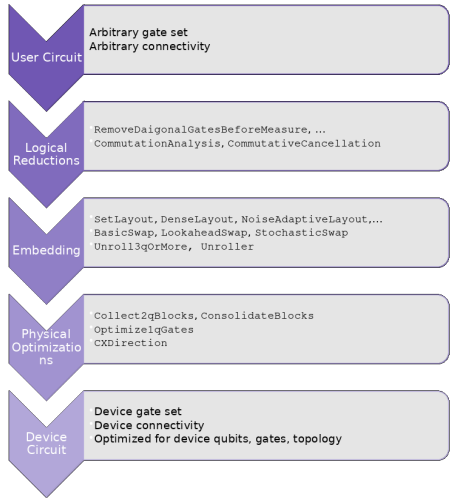
- Read-only Access to DAG
- Write Access to Property Set

- Write Access to DAG
- Read-only Access to Property Set

Example (commutation relationships):
Pass A is an Analysis Pass that inspects the graph and finds nodes that can commute.
Pass B is a Transformation Pass that dissolves those edges that commute (are false dependencies).
Pass B requires Pass A, and indicates this to the Pass Manager

Passmanager Stages

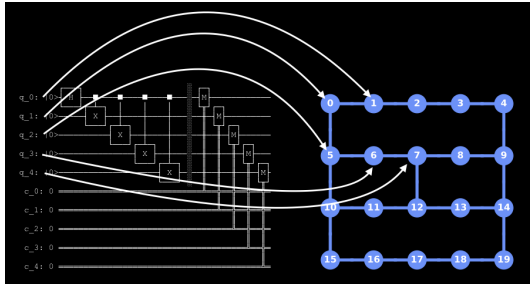
- Stages
  - Logical Reductions
  - Embedding
    - Layout
    - Mapping/Routing
    - Unrolling
  - Physical Reductions
- Pluggable and extensible



The Unroller

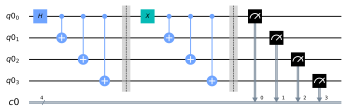
Layout

- Initial mapping to physical qubits
- Multiple options included
  - TrivialLayout
  - DenseLayout
  - NoiseAdaptiveLayout

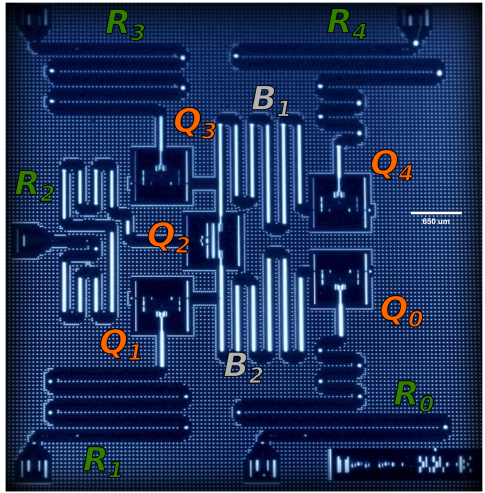


Layout Example

Logical Circuit

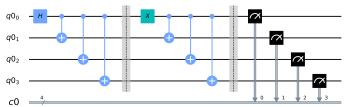


Target Device

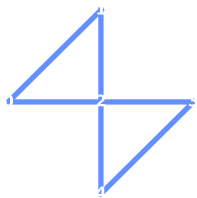


Layout Example

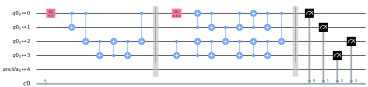
Logical Circuit



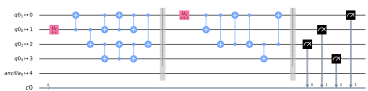
Target Device



optimization_level=1
TrivialLayout

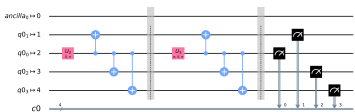


optimization_level=2
DenseLayout

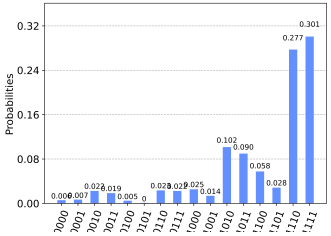


optimization_level=3
NoiseAdaptiveLayout

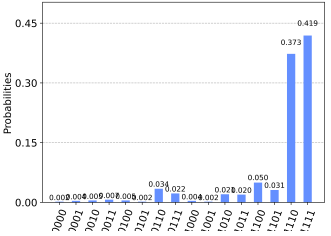
Custom Layout
(optimization_level=1)



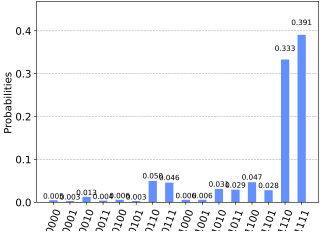
optimization_level=1
TrivialLayout



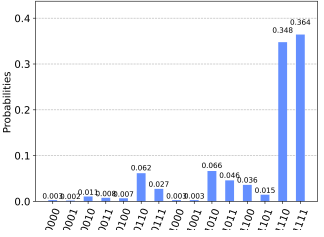
optimization_level=3
NoiseAdaptiveLayout



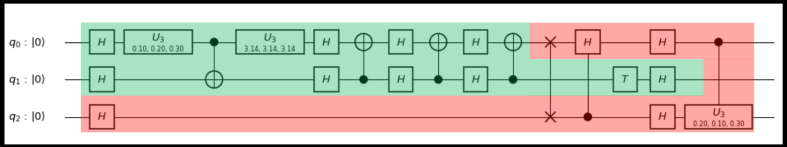
optimization_level=2
DenseLayout



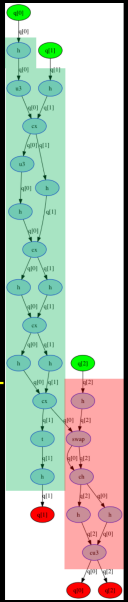
Custom Layout
(optimization_level=1)



2-qubit Block Collection

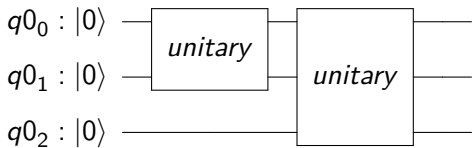


Efficient graph traversal on the DAG.
For each CNOT, collect ancestors and predecessors until a branch.



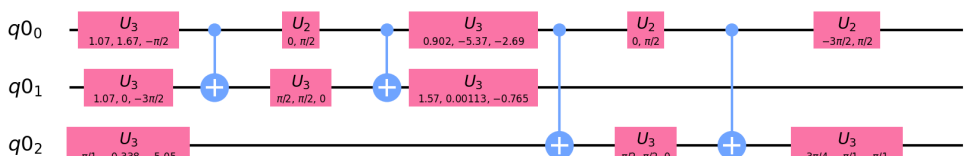
Consolidate Blocks

1. Loop over 2Q blocks collecteds from analysis pass
2. Run a simulation to find unitary matrix of each block
3. Replace block with unitary matrix



Consolidate Blocks

1. Loop over 2Q blocks collecteds from analysis pass
2. Run a simulation to find unitary matrix of each block
3. Replace block with unitary matrix



Optimize 1Q Operations

- For each run of single qubit gates find end state of rotations

Quantum Volume[1]

[1] https://arxiv.org/abs/1811.12926

Conclusions

- similar but different

Where to get more information

- These Slides: https://github.com/mtreinish/quantum-compilers
- Qiskit: https://qiskit.org/
- Qiskit Terra on Github: https://github.com/Qiskit/qiskit-terra
- IBM Q Experience: https://quantum-computing.ibm.com
- Tutorials on Quantum Computing and Qiskit: https://github.com/Qiskit/qiskit-tutorials

BACKUP SLIDES