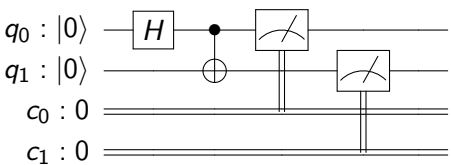
Building a Compiler for Quantum Computers

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

January 17, 2020

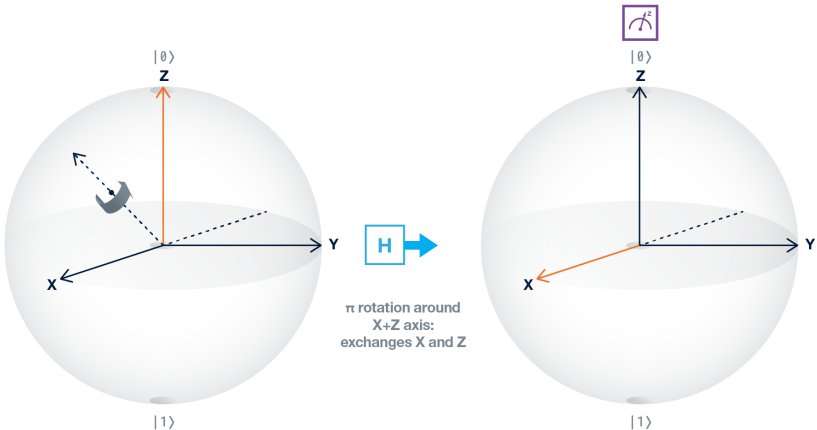
Quantum Circuits

- Used to describe operations on a quantum computer
- Each row represents a bit (either classical or quantum)
- Shows dependency of operation

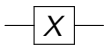
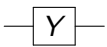
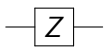
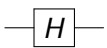


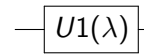
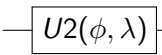
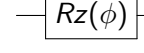
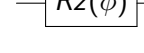
Quantum Gates

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



Quantum Gates

Gate	Symbol	Unitary
Pauli X		$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$
Pauli Y		$\begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}$
Pauli Z		$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$
Hadamard		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$
CNOT		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$

Gate	Symbol	Unitary
U1		$\begin{bmatrix} 1 & 0 \\ 0 & e^{i\lambda} \end{bmatrix}$
U2		$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -e^{i\lambda} \\ e^{i\phi} & e^{i(\phi+\lambda} \end{bmatrix}$
U3		$U(\theta, \phi, \lambda)$
Z Rotation		$\begin{bmatrix} e^{-\frac{i\phi}{2}} & 0 \\ 0 & e^{\frac{i\phi}{2}} \end{bmatrix}$
SWAP		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

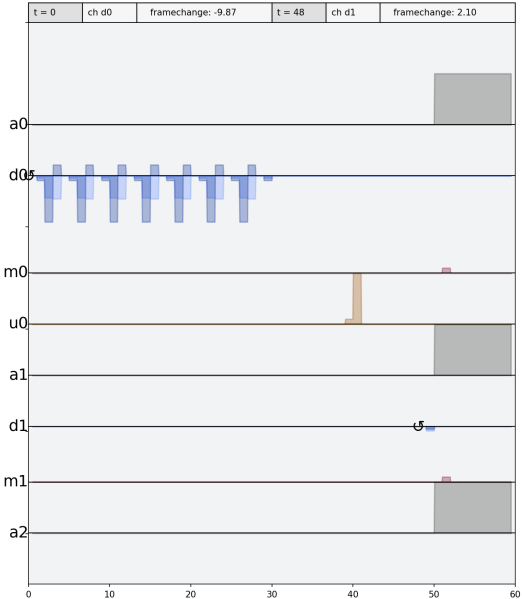
OpenQASM[1][2]

```
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
- Can be used to write circuits
- Mostly used as a transport layer to save circuits or as a transport

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

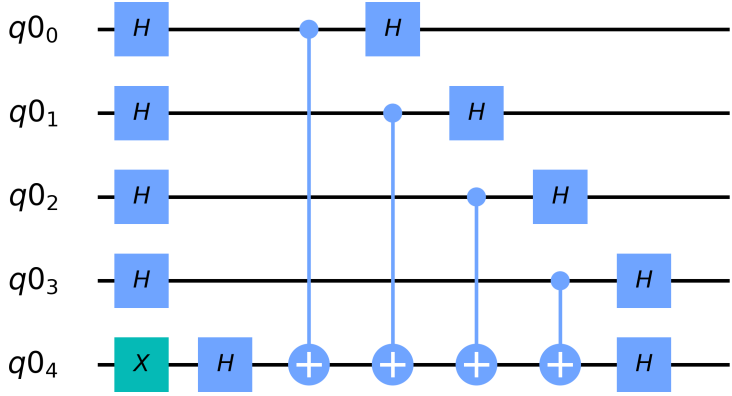
Pulse Level Programming



- 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

Why do we need compilers?

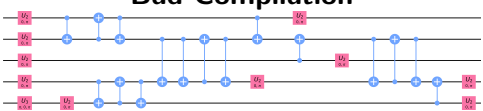
- Despite programming at a low (per bit) level it still is abstracted from the hardware.
- The compiler is used to enable running logical abstract circuit on an actual device
- NISQ devices have a number of limitations



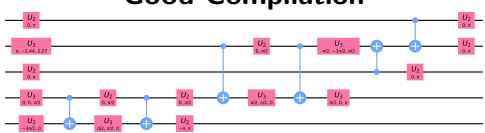
Why do we need compilers?

- Despite programming at a low (per bit) level it still is abstracted from the hardware.
- The compiler is used to enable running logical abstract circuit on an actual device
- NISQ devices have a number of limitations

Bad Compilation



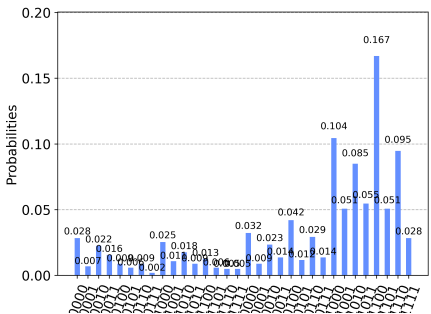
Good Compilation



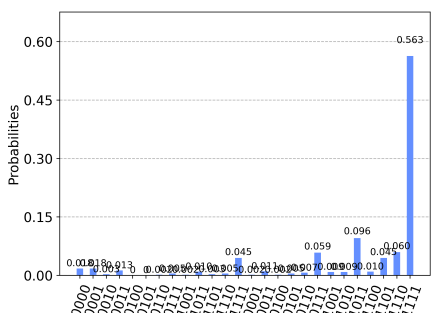
Why do we need compilers?

- Despite programming at a low (per bit) level it still is abstracted from the hardware.
- The compiler is used to enable running logical abstract circuit on an actual device
- NISQ devices have a number of limitations

Bad Compilation



Good Compilation



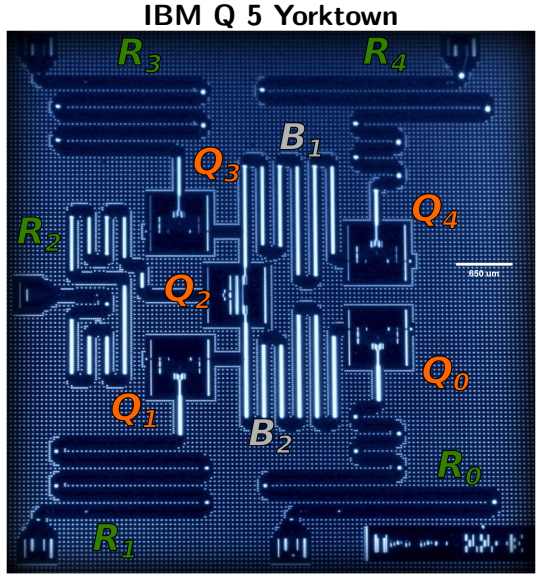
Basis Gates

- Each Quantum Computer only supports a small set of basis gates
- These gates can be used to implement any other
- For circuits to be executable by a quantum computer they must be expressed in terms of their basis gates

something to demonstrate basis

Qubit Connectivity

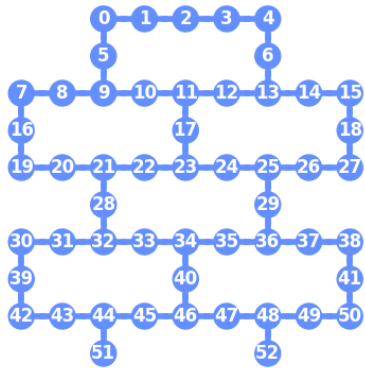
- Qubits in a device have limited connectivity
- For multi-qubit gates this means we can only run them between those qubits



Qubit Connectivity

- Qubits in a device have limited connectivity
- For multi-qubit gates this means we can only run them between those qubits

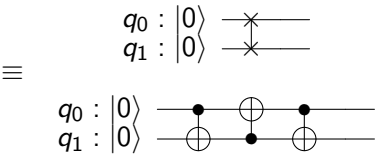
IBM Q 53 Rochester Coupling Map



Not Enough Connectivity

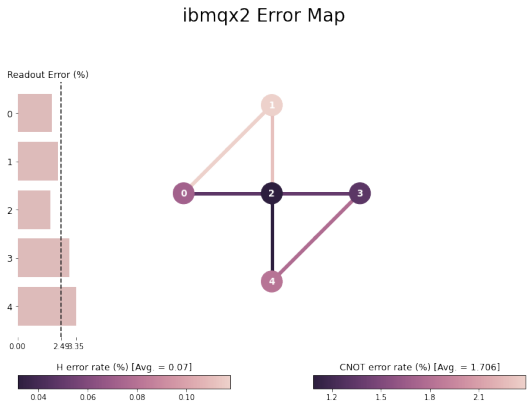
- If there
- Use SWAP gate to move state around

SWAP Gate



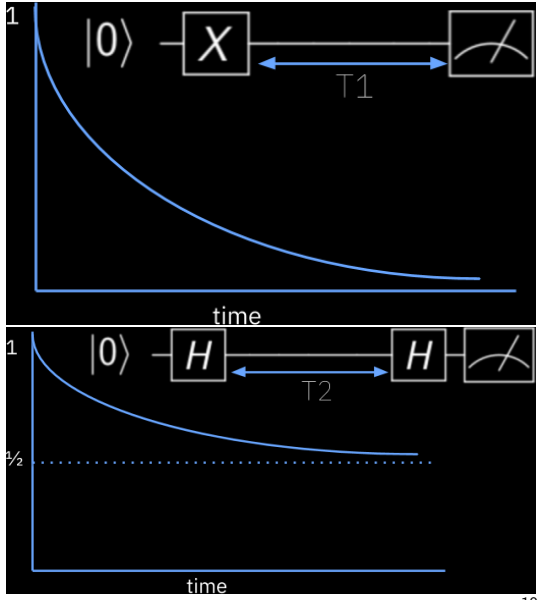
Noise

- Gate Errors
  - Single Qubit Errors
  - Multiqubit Errors
- Decoherence
  - T1: Energy relaxation, the time for a qubit at $|1\rangle$ to decay to ground state $|0\rangle$
  - T2: dephasing of a qubit in superposition state
- Readout Error



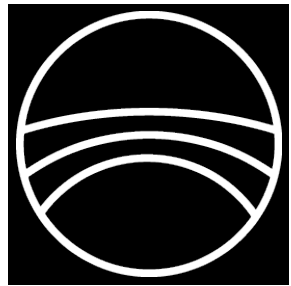
Noise

- Gate Errors
  - Single Qubit Errors
  - Multiqubit Errors
- Decoherence
  - T1: Energy relaxation, the time for a qubit at $|1\rangle$ to decay to ground state $|0\rangle$
  - T2: dephasing of a qubit in superposition state
- Readout Error



Qiskit Terra[1]

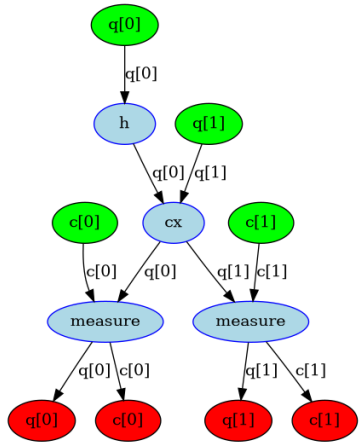
- Is the base layer for working with quantum computers provides interface to hardware and simulators
- Provides an SDK for working with quantum circuits
- Compiles circuits to run on different backends
- Designed to be backend agnostic and work with any quantum hardware or simulator
- Written in Python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

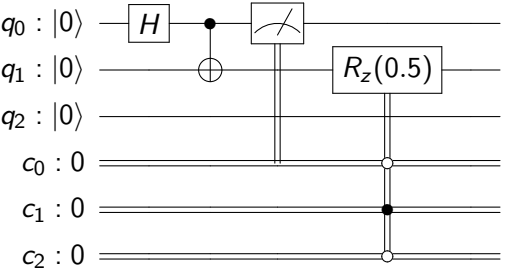


[1] https://github.com/Qiskit/qiskit-terra

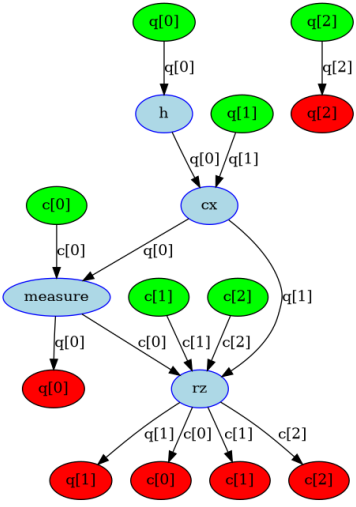
The DAG

- 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

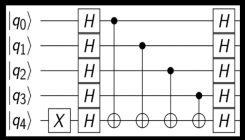



$q_0 : |0\rangle$
$q_1 : |0\rangle$
$q_2 : |0\rangle$
$c_0 : 0$
$c_1 : 0$
$c_2 : 0$
H
$R_z(0.5)$

$\rightarrow$


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



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

Pass 1 — Unroll to basis

Pass 2 — Cancel redundant gates

Pass 3 — Map to device by swapping

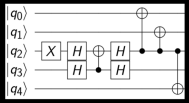
Pass 4 — Cancel redundant gates

Pass Manager

Global Property Set

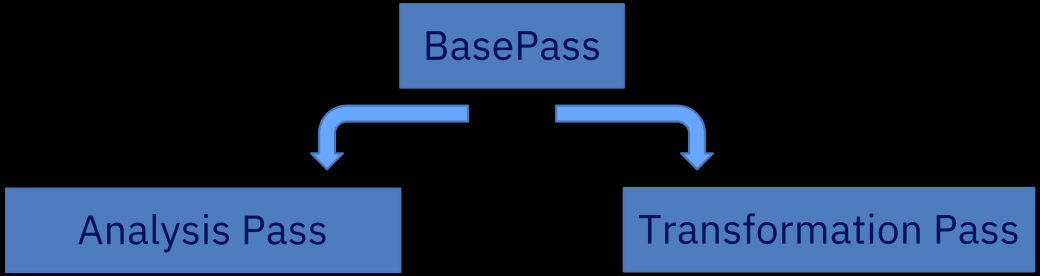
- Depth
- Commuting Sets
- Layout
- Fixed Point Condition

Each pass does one small, well-defined task.



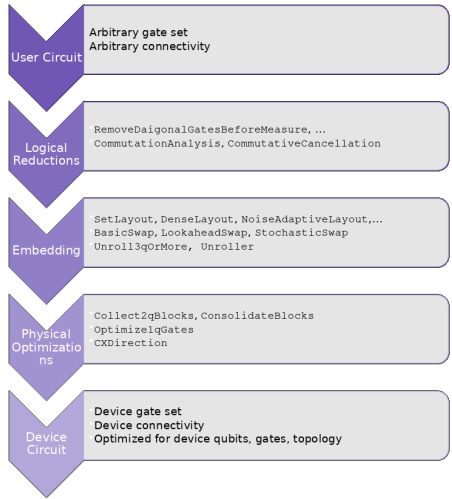
The pass manager maintains a global context about the circuit as it runs through the pipleine

Transpiler Architecture



Passmanager Stages

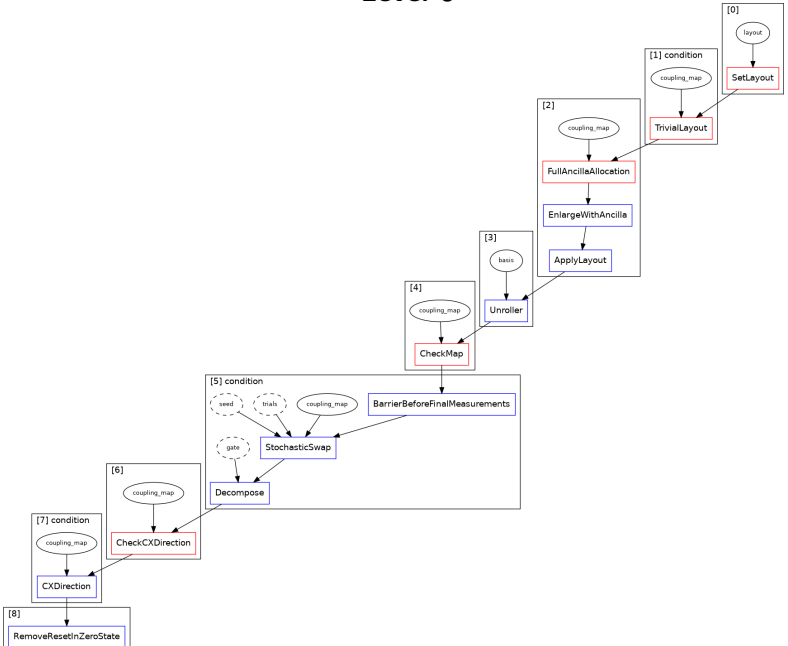
- Stages
  - Logical Reductions
  - Embedding
    - Layout
    - Mapping/Routing
    - Unrolling
  - Physical Reductions
- Pluggable and extensible



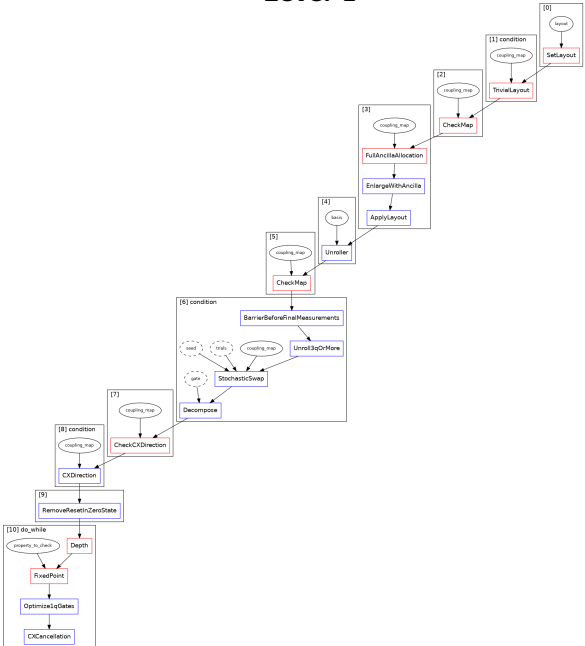
Preset Pass Managers

- Out of the box 4 optimization levels (0-3) with preset pass managers
- Increasing levels of optimization vs execute time

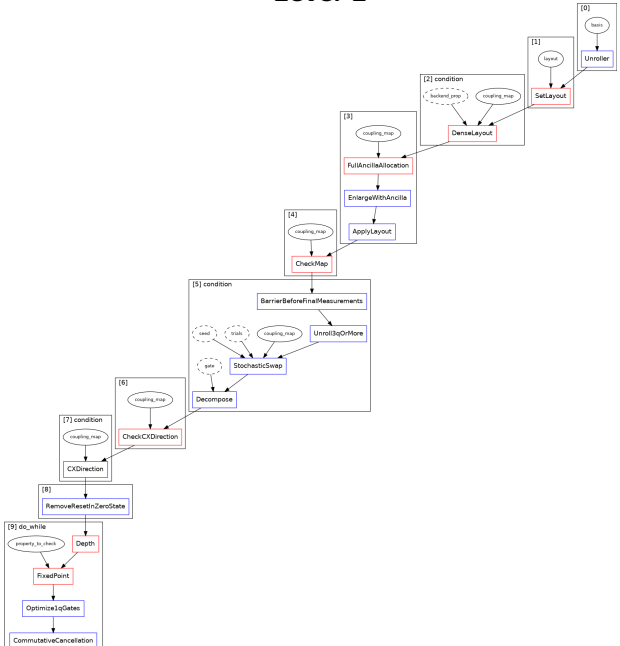
Level 0


[0]
layout
SetLayout
[1] condition
coupling_map
TrivialLayout
[2]
coupling_map
FullAncillaAllocation
EnlargeWithAncilla
ApplyLayout
[3]
basis
Unroller
[4]
coupling_map
CheckMap
[5] condition
seed
trials
coupling_map
BarrierBeforeFinalMeasurements
gate
StochasticSwap
Decompose
[6]
coupling_map
CheckCXDirection
[7] condition
coupling_map
CXDirection
[8]
RemoveResetInZeroState

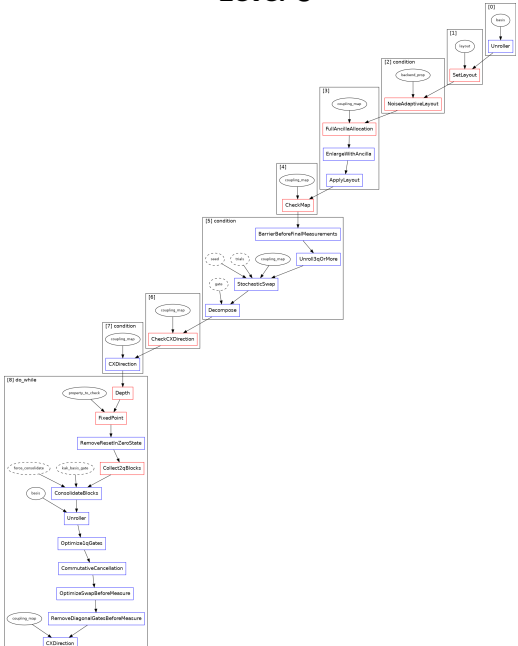
Level 1


[0]
layout
SetLayout
[1] condition
coupling_map
TrivialLayout
[2]
coupling_map
CheckMap
[3]
coupling_map
FullAncillaAllocation
EnlargeWithAncilla
ApplyLayout
[4]
basis
Unroller
[5]
coupling_map
CheckMap
[6] condition
BarrierBeforeFinalMeasurements
seed
trials
coupling_map
Unroll3qOrMore
gate
StochasticSwap
Decompose
[7]
coupling_map
CheckCXDirection
[8] condition
coupling_map
CXDirection
[9]
RemoveResetInZeroState
[10] do_while
property_to_check
Depth
FixedPoint
Optimize1qGates
CXCancellation

Level 2

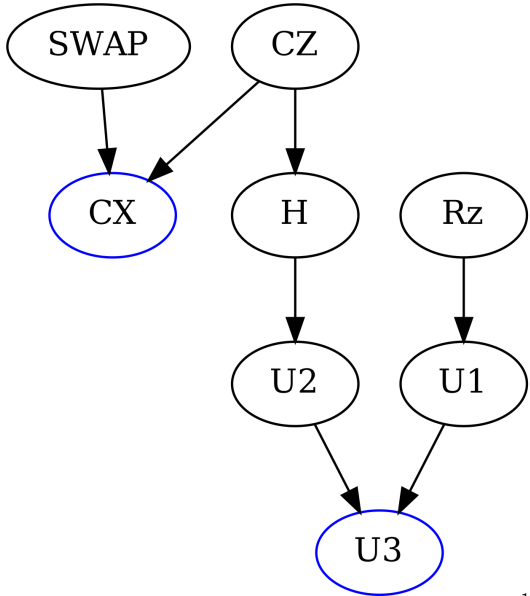

[0]
basis
Unroller
[1]
layout
SetLayout
[2] condition
backend_prop
coupling_map
DenseLayout
[3]
coupling_map
FullAncillaAllocation
EnlargeWithAncilla
ApplyLayout
[4]
coupling_map
CheckMap
[5] condition
BarrierBeforeFinalMeasurements
seed
trials
coupling_map
Unroll3qOrMore
gate
StochasticSwap
[6]
coupling_map
Decompose
[7] condition
coupling_map
CheckCXDirection
CXDirection
[8]
RemoveResetInZeroState
[9] do_while
property_to_check
Depth
FixedPoint
Optimize1qGates
CommutativeCancellation

Level 3


[0]
Unroller
[1]
SetLayout
[2] condition
NoiseAdaptiveLayout
[3]
FullAncillaAllocation
EnlargeWithAncilla
[4]
ApplyLayout
CheckMap
[5] condition
BarrierBeforeFinalMeasurements
Unroll3qOrMore
StochasticSwap
Decompose
[6]
CheckCXDirection
[7] condition
CXDirection
[8] do_while
Depth
FixedPoint
RemoveResetInZeroState
Collect2qBlocks
ConsolidateBlocks
Unroller
Optimize1qGates
CommutativeCancellation
OptimizeSwapBeforeMeasure
RemoveDiagonalGatesBeforeMeasure
CXDirection

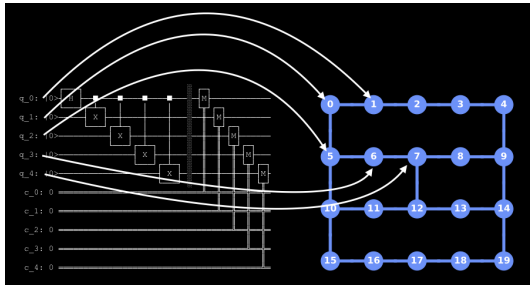
The Unroller

- The first stage of any compilation is to unroll the gates to the basis set
- Currently this is done using a descent tree
- Mainly works for superconducting qubit basis
- Other types of devices support is hardcoded extra path

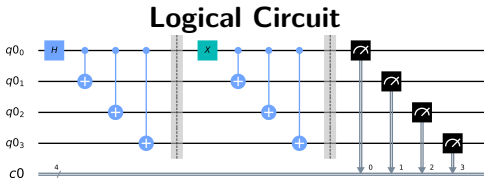


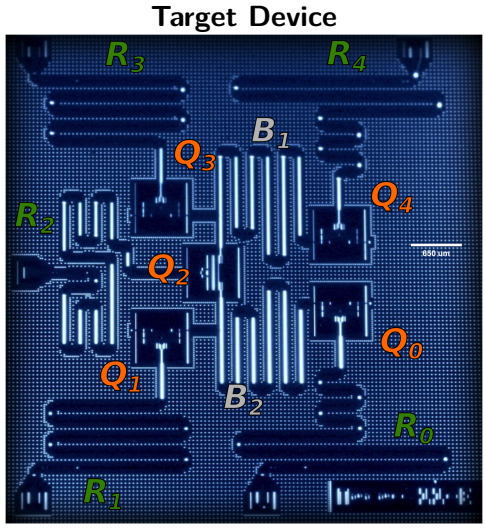
Layout

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

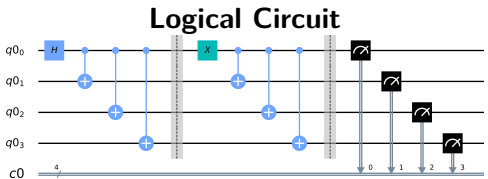


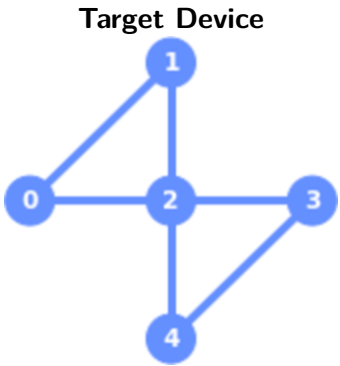
Layout Example



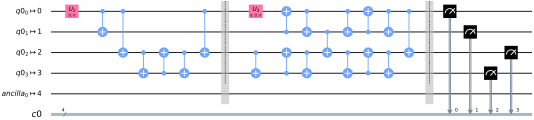


Layout Example

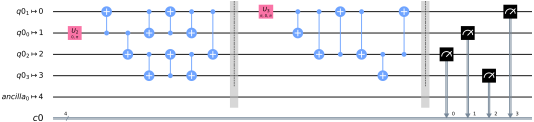




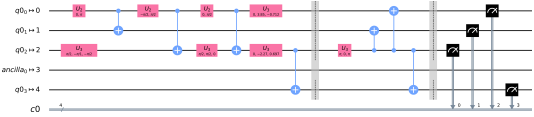
optimization_level=1
TrivialLayout



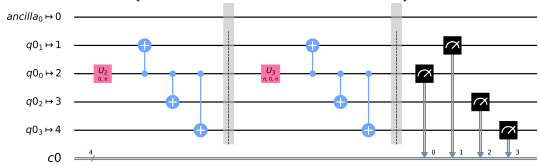
optimization_level=2
DenseLayout



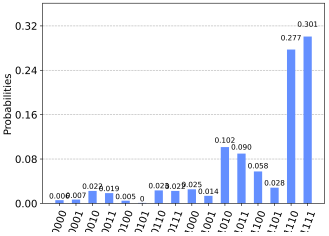
optimization_level=3
NoiseAdaptiveLayout



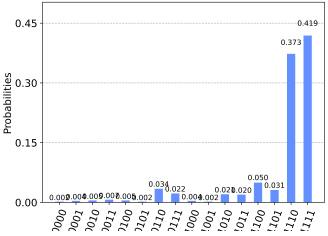
Custom Layout
(optimization_level=1)



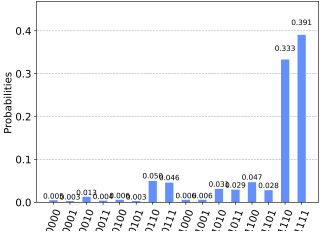
optimization_level=1
TrivialLayout



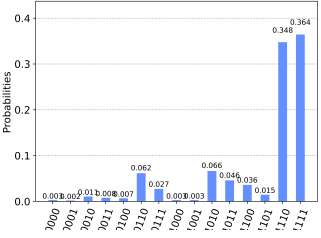
optimization_level=3
NoiseAdaptiveLayout



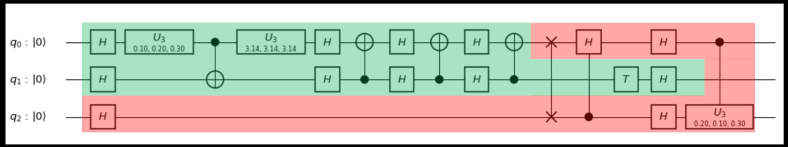
optimization_level=2
DenseLayout



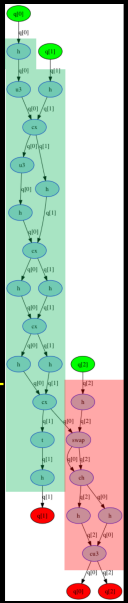
Custom Layout
(optimization_level=1)



2-qubit Block Collection

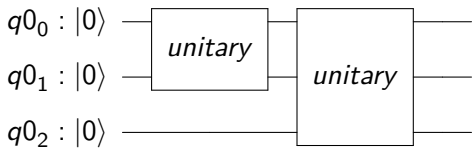


Efficient graph traversal on the DAG.
For each CNOT, collect ancestors and predecessors until a branch.



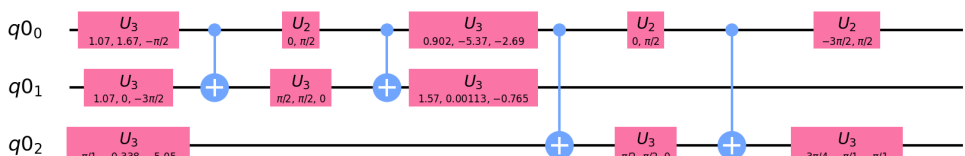
Consolidate Blocks

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



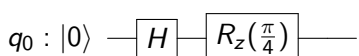
Consolidate Blocks

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



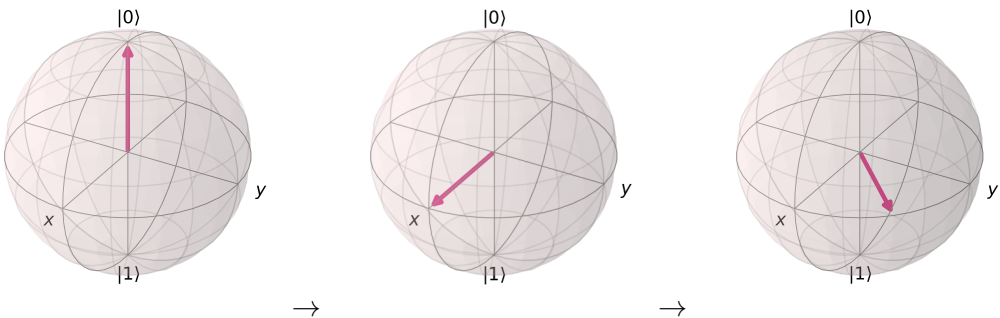
Optimize 1Q Operations

1. Find all runs of 1Q operations on each Qubit in the dag
2. For each run calculate the end state rotation
3. Replace all 1Q operations with a single rotation gate to that end state



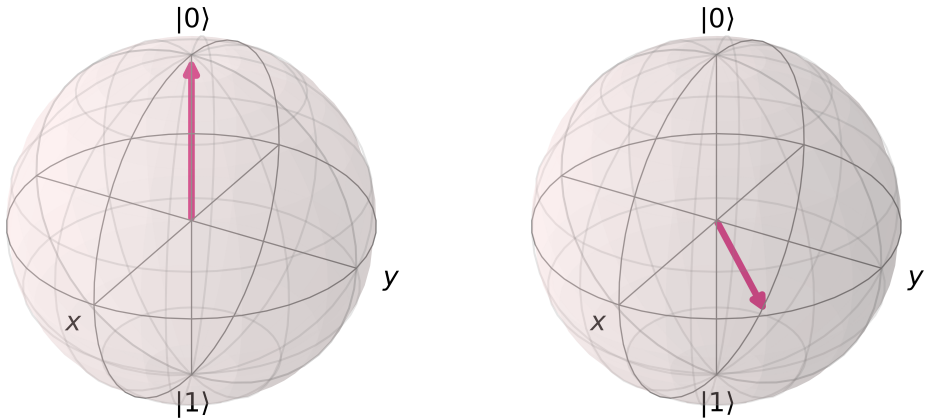
Optimize 1Q Operations

1. Find all runs of 1Q operations on each Qubit in the dag
2. For each run calculate the end state rotation
3. Replace all 1Q operations with a single rotation gate to that end state



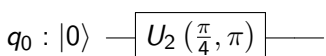
Optimize 1Q Operations

1. Find all runs of 1Q operations on each Qubit in the dag
2. For each run calculate the end state rotation
3. Replace all 1Q operations with a single rotation gate to that end state



Optimize 1Q Operations

1. Find all runs of 1Q operations on each Qubit in the dag
2. For each run calculate the end state rotation
3. Replace all 1Q operations with a single rotation gate to that end state



Swap Mapping/Routing

- After the layout step we need to ensure that
- 3 Algorithms included in Qiskit
  - BasicSwap
  - LookaheadSwap
  - StochasticSwap
- StochasticSwap is used in all presets

Insert Diagram of Swap Mapping

Quantum Volume[1]

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

Conclusions

- Techniques used are similar to a classical compiler
- Implementation details are different because of uniqueness of quantum computing
- A good compiler can be the difference between a functional program and one that doesn't work

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 (sign up to get access to public quantum computers): https://quantum-computing.ibm.com
- Tutorial on using Passmanager: https://github.com/Qiskit/qiskit-iqx-tutorials/blob/master/qiskit/advanced/terra/4_transpiler_passes_and_passmanager.ipynb
- Qiskit Textbook: https://qiskit.org/textbook/