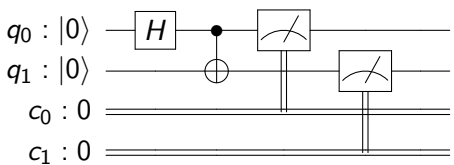
Building a Compiler for Quantum Computers

Matthew Treinish
Software Engineer - IBM Research
mtreinish@kortar.org
`mtreinish on Freenode`
https://github.com/mtreinish/quantum-compilers/tree/fossasia-2021

March 16, 2021

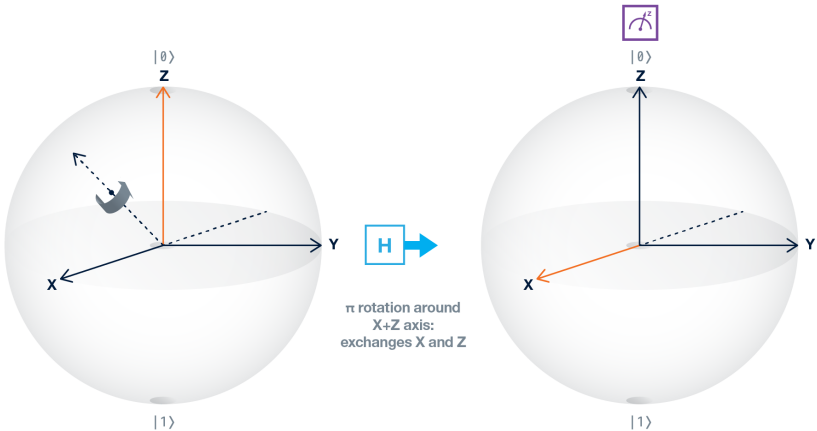
Quantum Circuits

- Used to describe a series of operations on a quantum computer
- Each row represents a bit (either classical or quantum)
- The operations in the circuit performed in order on each qubit from left to right
- Shows dependency of operation, no timing information

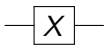
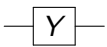
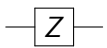
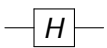


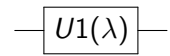
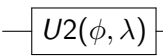
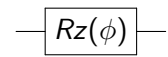
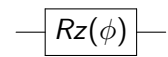
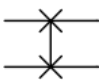
Quantum Gates

- Quantum gates are the operations performed on qubits
- Gates are reversible
- Each gate is represented as a unitary matrix



Some 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		$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

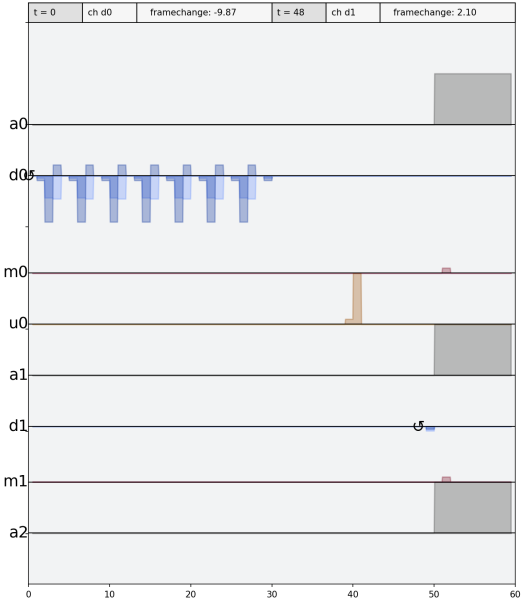
OpenQASM[1][2]

```
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 or to save circuits

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

Pulse Level Programming

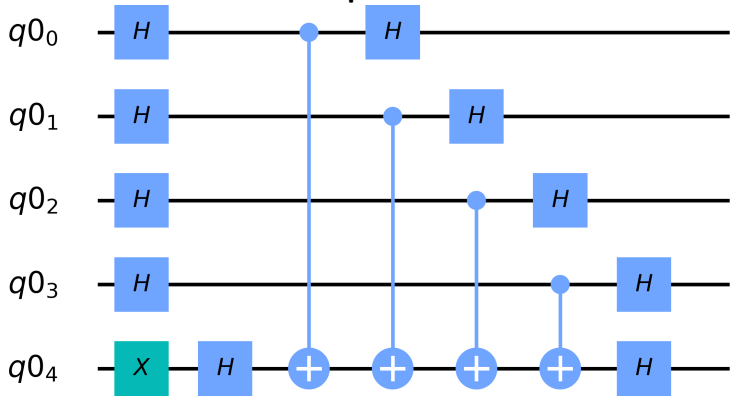


- A layer below the circuit model is to use pulses
- Each quantum gate is defined as a pulse that gets applied to the qubits
- Enables you to tweak the definition of gates for an existing circuit
- Also can just define a custom pulse schedule and run that directly
- Mostly used for physics research or hardware characterization

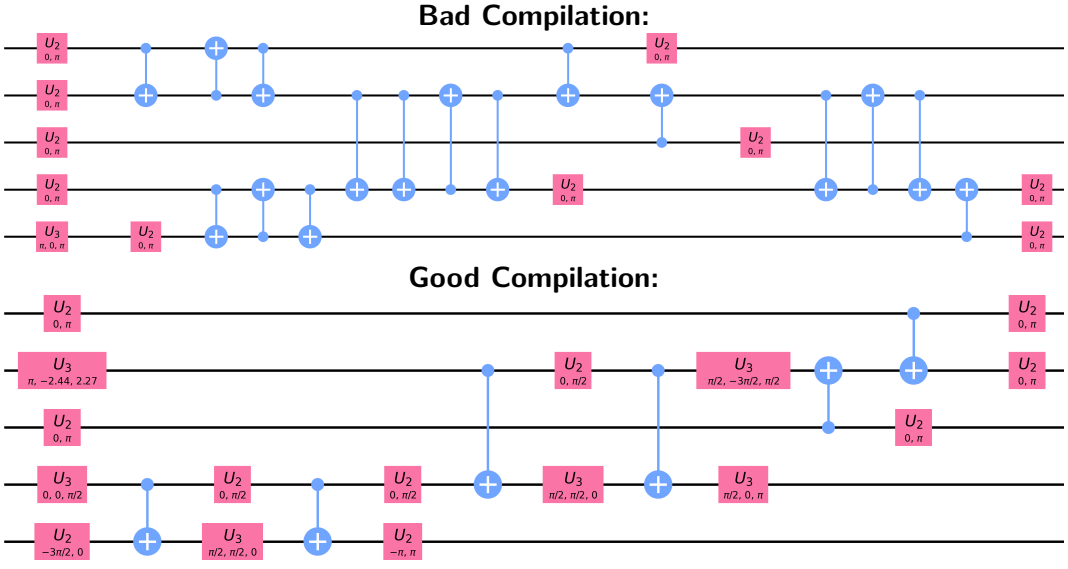
Why do we need compilers?

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

Example Circuit:

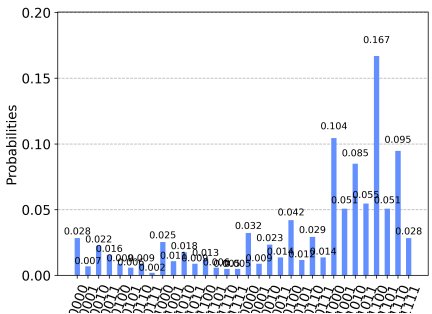


Why do we need compilers?

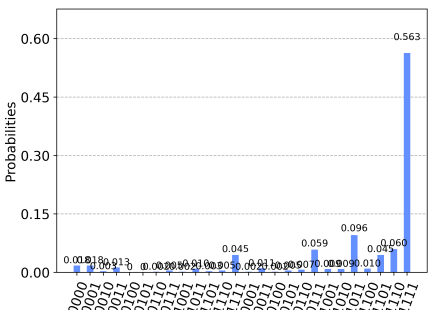


Why do we need compilers?

Bad Compilation



Good Compilation



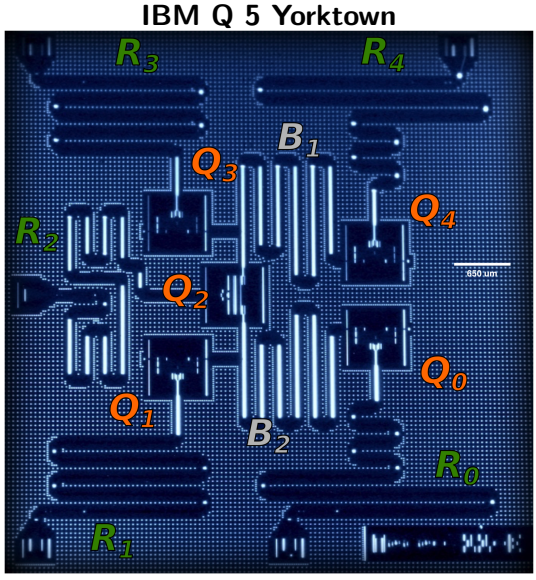
Basis Gates

- Each Quantum Computer only supports a set of basis gates
- These gates can be used to implement all other gates
- Prior to running on a quantum computer the gates must be defined using the basis set

Superconducting Qubits:
CX, Rz, $\sqrt{X}$, X, id
Trapped Ion Qubits:
Ms, Rx, Ry
Simulator:
U1, U2, U3, CX, Cz, Id, X, Y, Z, H, S, Sdg, t, tdg, swap, ccx, unitary, initialize, cu1, cu2, cu3, swap, mcx, mcz, mcu1, mcu2, mcu3, mcswap, muliplexer

Qubit Connectivity

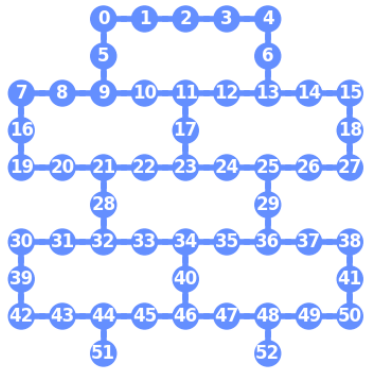
- Qubits in a device have limited connectivity
- For multi-qubit gates this means we can only run them between those qubits



Qubit Connectivity

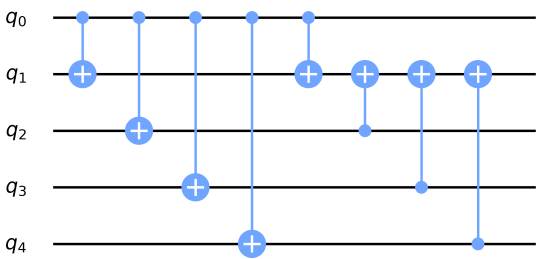
- Qubits in a device have limited connectivity
- For multi-qubit gates this means we can only run them between those qubits

IBM Q 53 Rochester Coupling Map



Not Enough Connectivity?

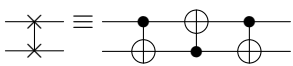
- Connectivity is not always sufficient to fully map logical circuit
- Use SWAP gate to move state around between qubits
- Adding swaps are potentially expensive from noise perspective



Not Enough Connectivity?

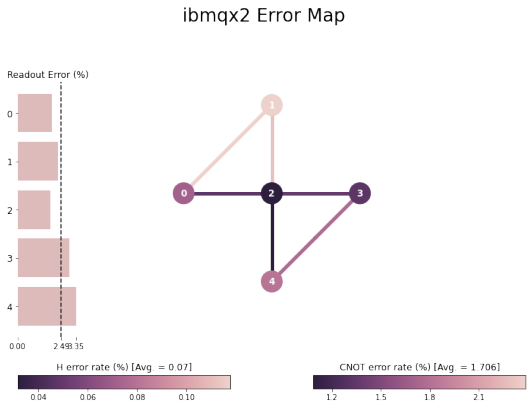
- Connectivity is not always sufficient to fully map logical circuit
- Use SWAP gate to move state around between qubits
- Adding swaps are potentially expensive from noise perspective

SWAP Gate



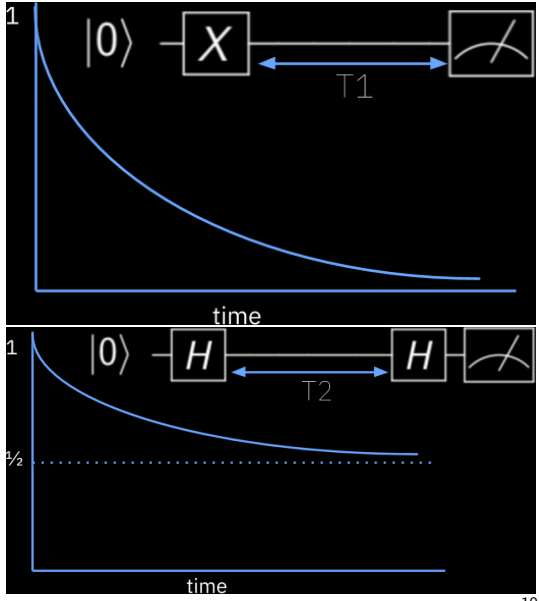
Noise

- 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



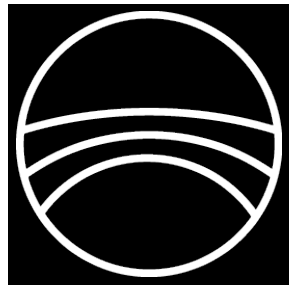
Noise

- 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]

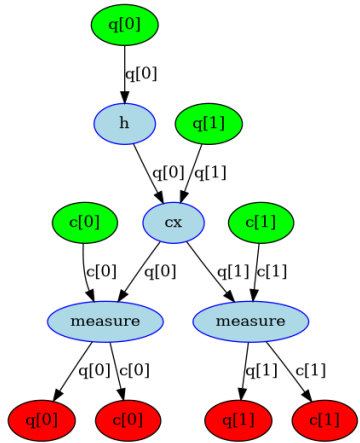
- Is the base layer for working with quantum computers provides interface to hardware and simulators
- Provides an SDK for working with quantum circuits
- Compiles circuits to run on different backends
- Designed to be backend agnostic and work with any quantum hardware or simulator
- Written in Python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



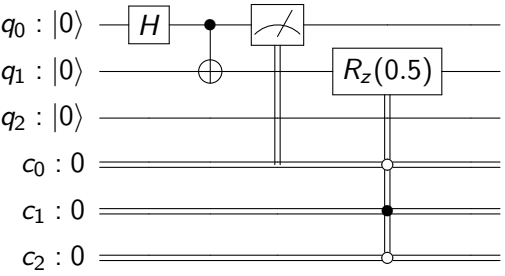
[1] https://github.com/Qiskit/qiskit-terra

The DAG

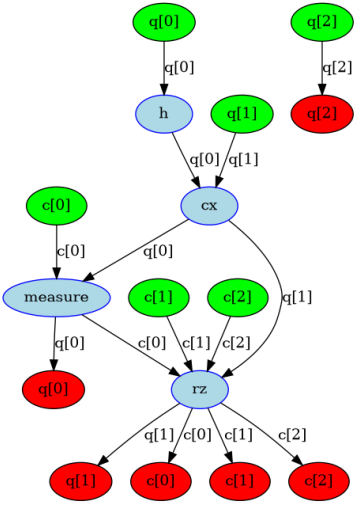
- Compiler represents circuits as a DAG
- Each node in the DAG is an operation, an input, or an output
- Each edge indicates data flow between nodes
- Makes flow of information between operations explicit



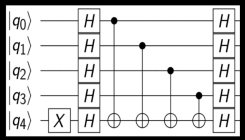
The DAG



$\rightarrow$



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

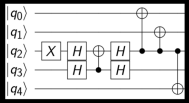
Pass 4 — Cancel redundant gates

Pass Manager

Global Property Set

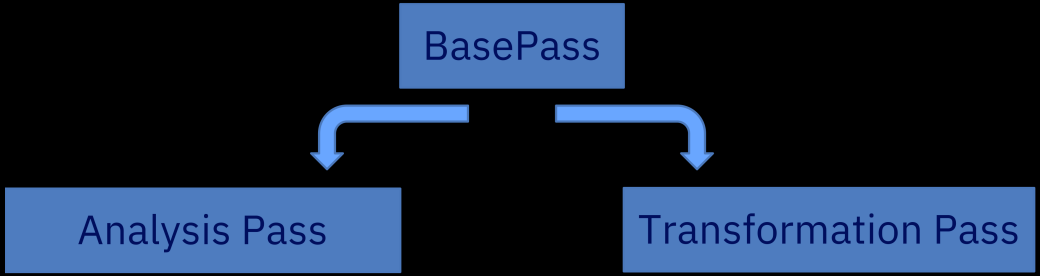
- Depth
- Commuting Sets
- Layout
- Fixed Point Condition

Each pass does one small, well-defined task.



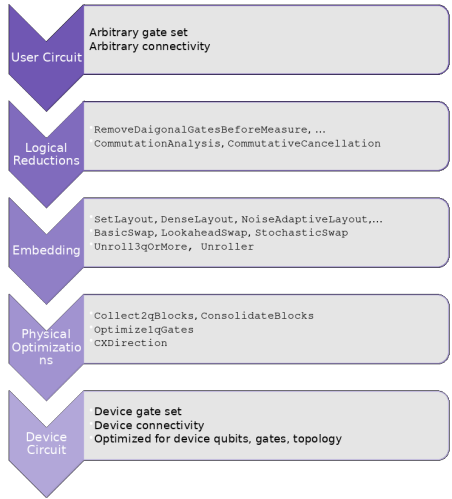
The pass manager maintains a global context about the circuit as it runs through the pipleine

Transpiler Architecture



Passmanager Stages

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

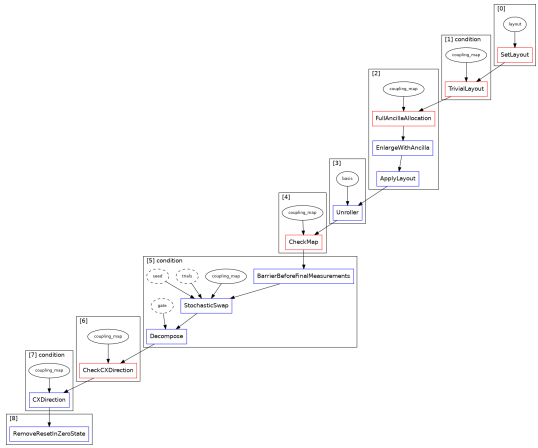


Preset Pass Managers

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

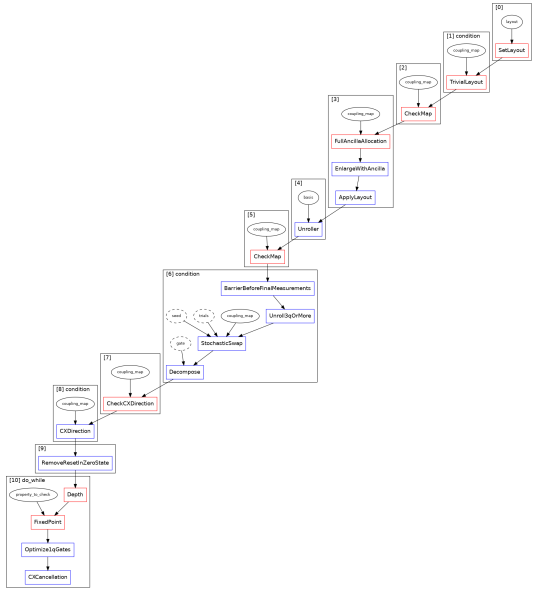
Optimization Level 0

- No Optimization
- Unroll
- Trivial Layout
- Swap Mapping



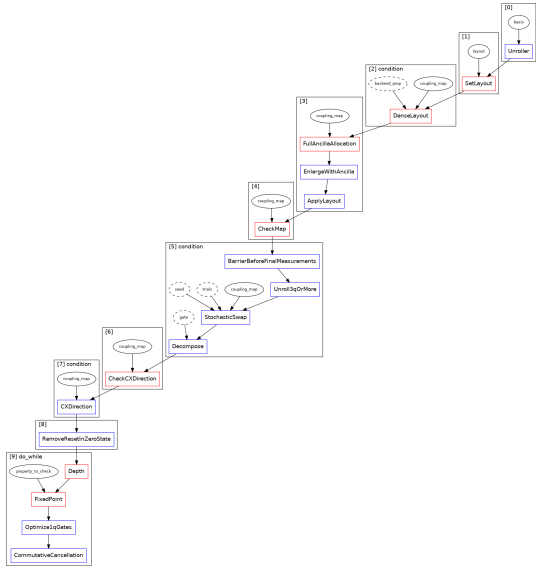
Optimization Level 1

- Unroll
- Trivial Layout
- Swap Mapping
- Optimization
  - Optimize 1Q
  - CX Cancellation



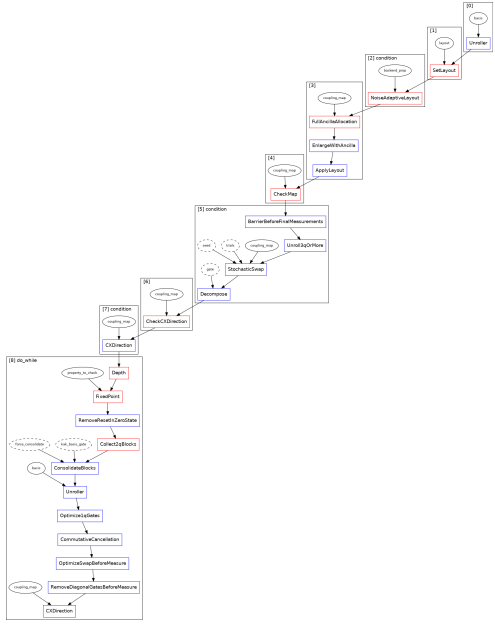
Optimization Level 2

- Unroll
- Dense Layout
- Swap Mapping
- Optimization
  - Optimize 1Q
  - Commutative Cancellation



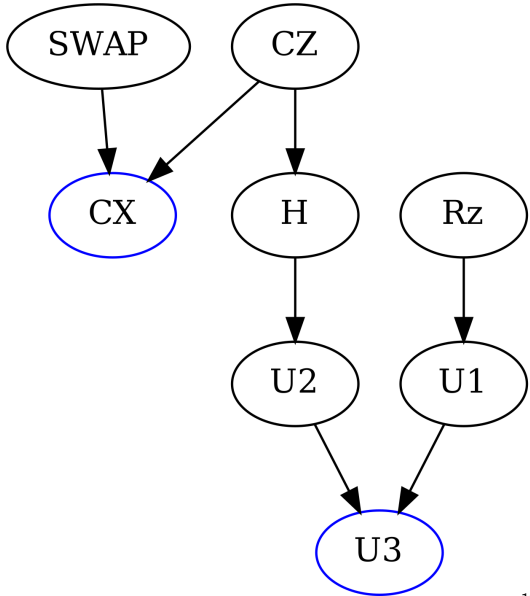
Optimization Level 3

- Unroll
- Noise Aware Layout
- Swap Mapping
- Optimization
  - Optimize 1Q
  - Commutative Cancellation
  - Consolidate Blocks
  - Remove Reset in 0 State
  - Optimize Swap before measure
  - Remove Diagonal Gates before measure



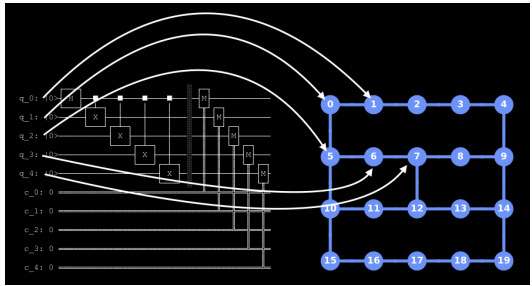
The Unroller

- Often the first step of a 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



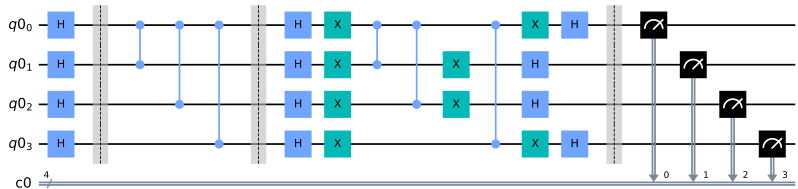
Layout

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

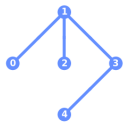


Layout Example

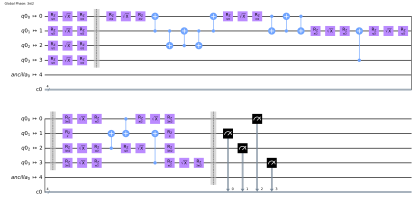
Logical Circuit



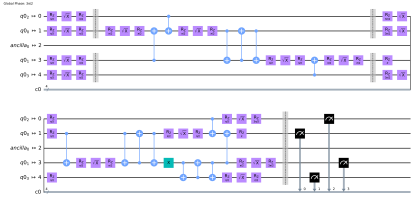
Target Device



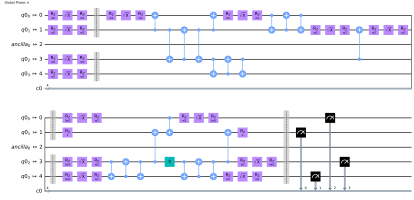
TrivialLayout



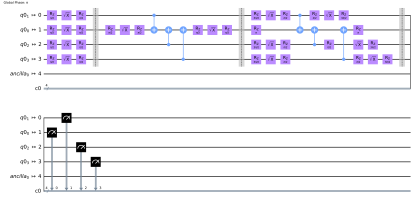
NoiseAdaptiveLayout



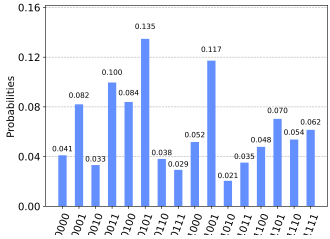
DenseLayout



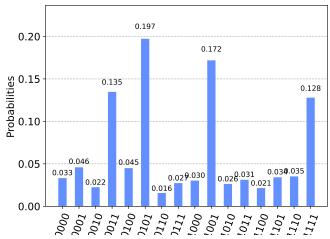
Custom Layout



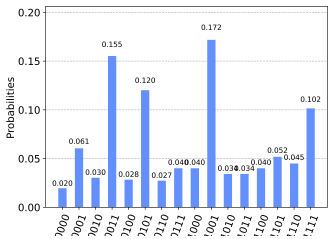
TrivialLayout



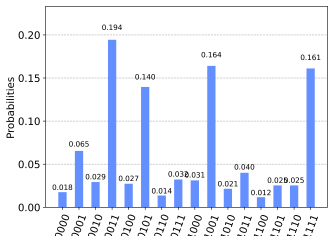
NoiseAdaptiveLayout



DenseLayout



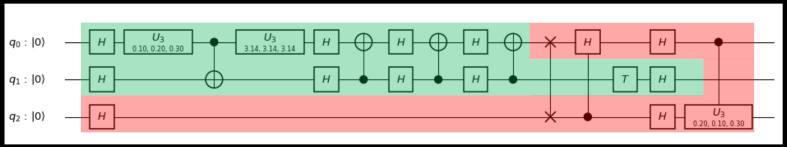
Custom Layout



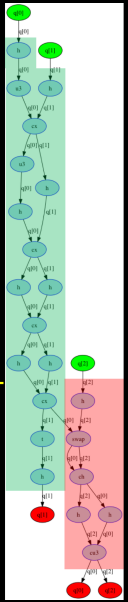
Swap Mapping/Routing

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

2-qubit Block Collection

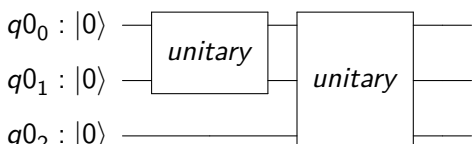


Efficient graph traversal on the DAG.
For each CNOT, collect ancestors and predecessors until a branch.



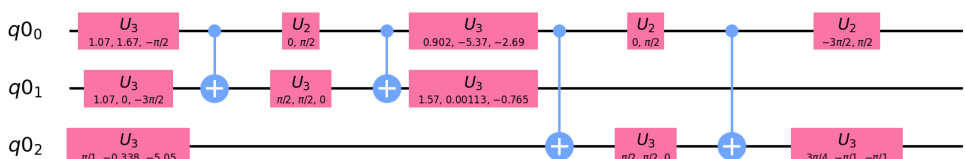
Consolidate Blocks

1. Loop over 2Q blocks collected from analysis pass
2. Run a simulation to find unitary matrix of each block
3. Replace block with unitary matrix
4. Decompose unitary to basis gates with unroller



Consolidate Blocks

1. Loop over 2Q blocks collected from analysis pass
2. Run a simulation to find unitary matrix of each block
3. Replace block with unitary matrix
4. Decompose unitary to basis gates with unroller

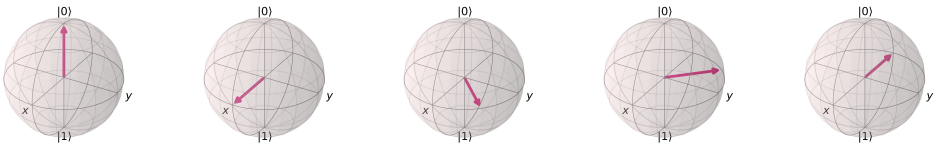


Optimize 1Q Operations

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

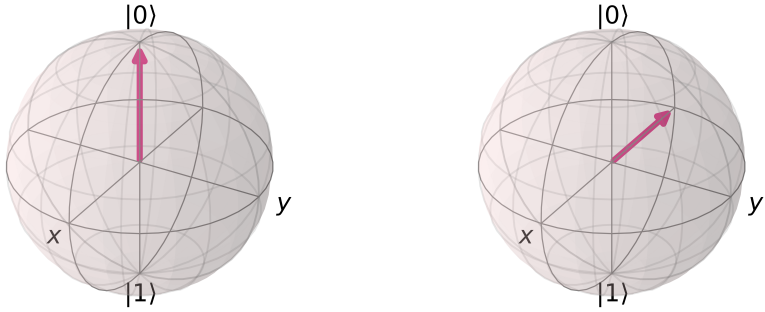
Optimize 1Q Operations

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



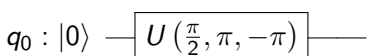
Optimize 1Q Operations

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

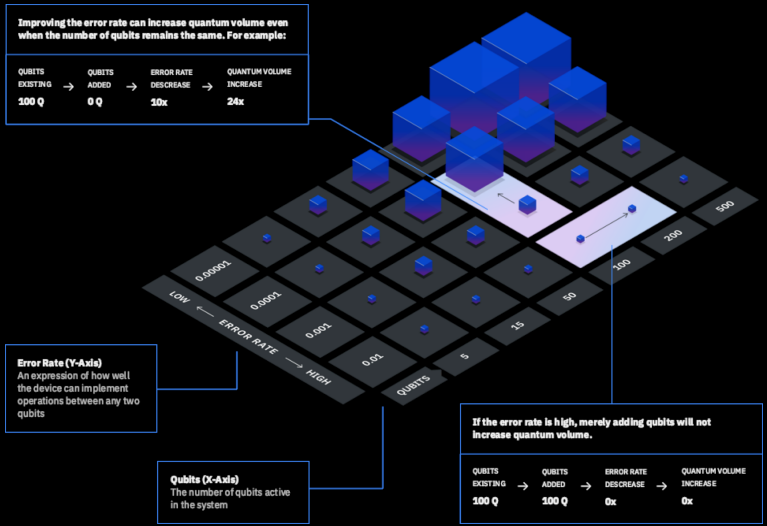


Optimize 1Q Operations

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



Quantum Volume[1]



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

Where to get more information

- These Slides: https://github.com/mtreinish/quantum-compilers/tree/fossasia-2021
- Qiskit: https://qiskit.org/
- Qiskit Terra on Github: https://github.com/Qiskit/qiskit-terra
- IBM Quantum (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/
- Talk on Open Source Quantum Computing from LCA 2019: https://www.youtube.com/watch?v=th4TDYX6xoc

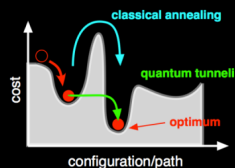
BACKUP SLIDES

Types of Quantum Computing

Quantum Annealing

Optimization Problems

- Machine learning
- Fault analysis
- Resource optimization
- etc…



Many ‘noisy’ qubits can be built;
large problem class in optimization;
amount of quantum speedup unclear

Approximate Q-Comp.

Simulation of Quantum Systems, Optimization

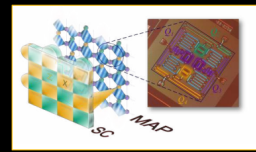
- Material discovery
- Quantum chemistry
- Optimization
  (logistics, time scheduling,…)
- Machine Learning

Hybrid quantum-classical approach;
already 50-100 “good” physical qubits
could provide quantum speedup.

Fault-tolerant Universal Q-Comp.

Execution of Arbitrary Quantum Algorithms

- Algebraic algorithms
  (machine learning, cryptography,…)
- Combinatorial optimization
- Digital simulation of quantum systems



Surface Code: Error correction in a Quantum Computer

Proven quantum speedup;
error correction requires significant qubit overhead.

Ivano Tavernelli - ita@zurich.ibm.com

IBM