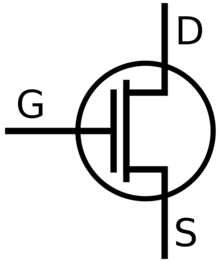
Enabling Deeper Quantum Compiler Optimizations at High Level

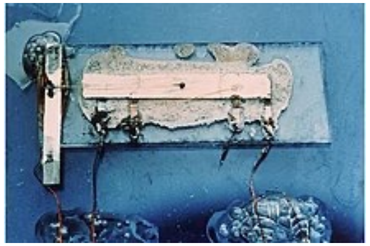
Gushu Li

11/10/2022

The Quantum Revolution

1st Quantum Revolution





Classical Computer

1900

Quantum Mechanics

1947

Transistor

1956

MOSFET

1958

Integrated Circuit

1980

2nd Quantum Revolution: the power of quantum is not fully exploited

The Quantum Revolution

2nd Quantum Revolution

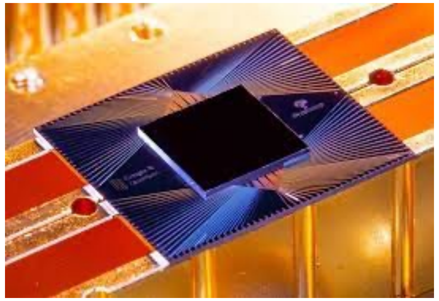
1980
Quantum Computer

1994
Cryptography

2019
Supremacy

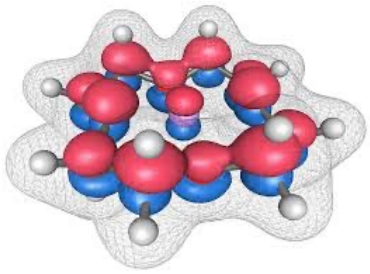
Now

Practical Quantum Computing

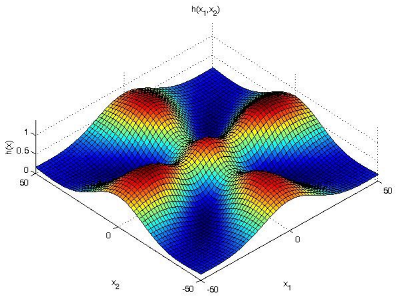


Quantum Applications

Application



Simulation



Optimization

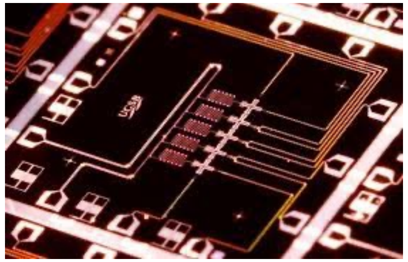
Machine Learning

Cryptography

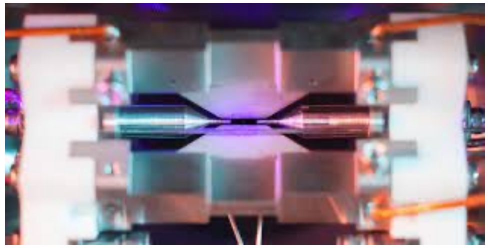
Technology Stacks

Quantum Devices

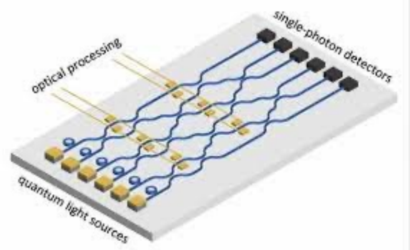
Application



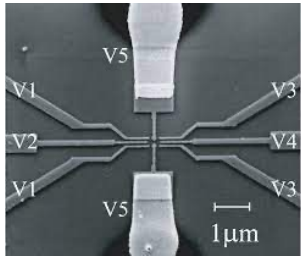
Superconducting



Ion Trap



Photonics



Quantum Dot

Device →

Technology Stacks

Quantum Computer System

Application

Language

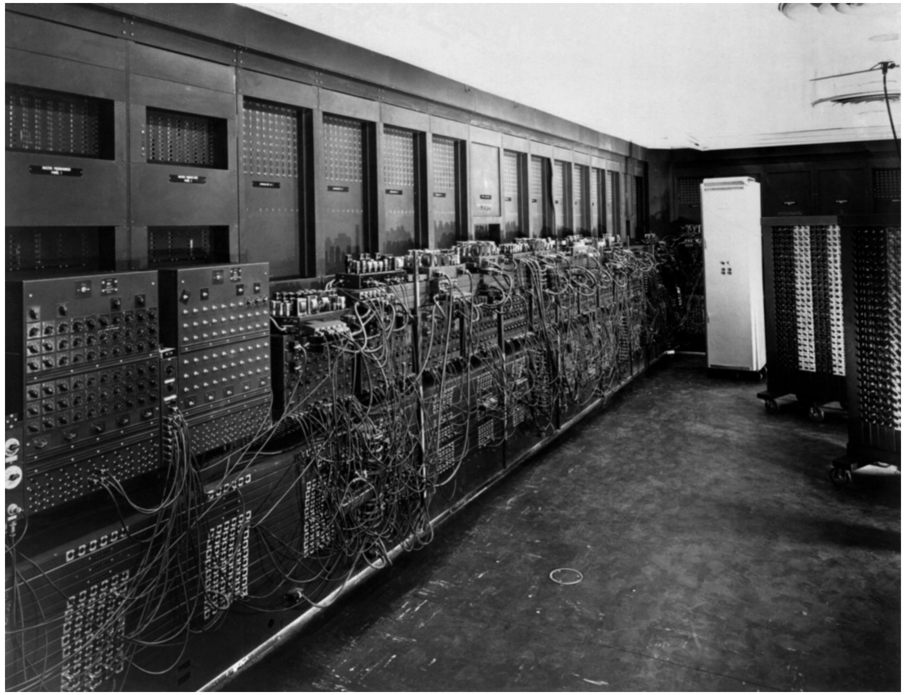
Compiler

Architecture

Device

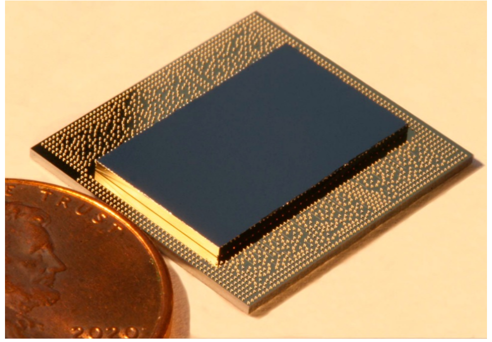
Technology Stacks

ENIAC, the first electronic general purpose digital computer, 1945



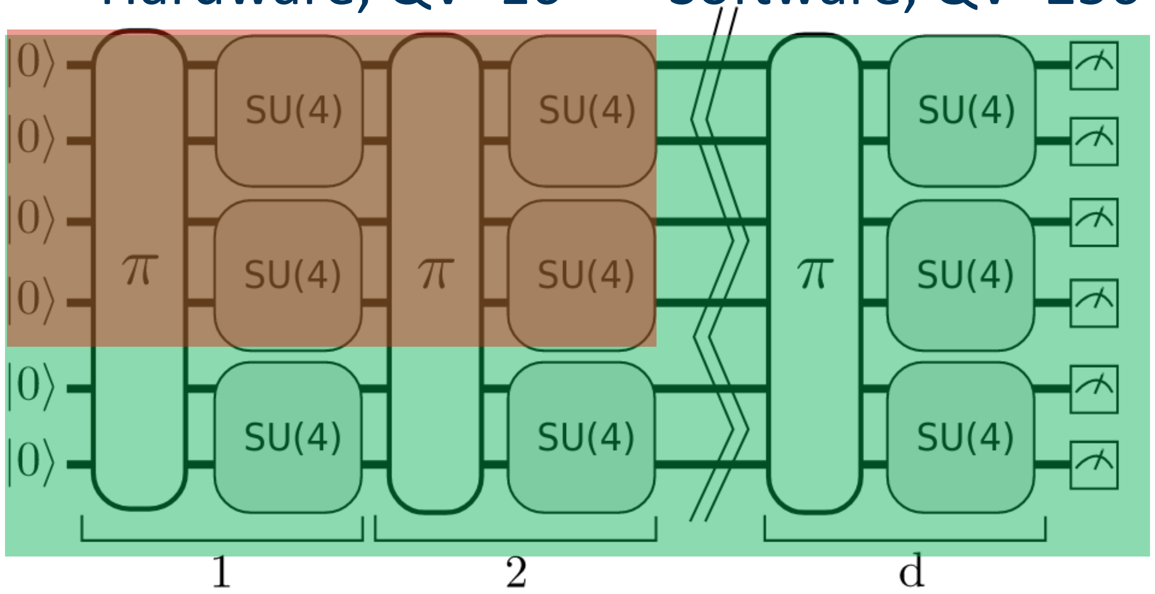
Hardware vs Software

- IBM benchmarking results



IBM Q Montreal

Hardware, QV=16 Software, QV=256



Quantum system research extends the computation capability!

And both software and hardware are important

Contribution Breakdown

1 : 1

[IBM Quantum]

Quantum Volume (QV): the size of Hilbert space that a quantum processor can explore reliably

My Research Overview

Application

Language

Compiler

Architecture

Device

Technology Stacks

Projection-Based Runtime Assertion [OOPSLA'20]

Qubit Mapping onto Superconducting Device [ASPLOS'19]

Efficient Superconducting Architecture Design [ASPLOS'20]

Paulihedral Compiler for Quantum Simulation [ASPLOS'22]

Cross-Layer Co-Design for Quantum Chemistry [ISCA'21]

Industry Adoptions & Awards

Application

Language

Compiler

Architecture

Device

Technology Stacks

Cambridge Quantum Computing

acm SIGPLAN

NSF

Projection-Based Runtime Assertion [OOPSLA'20]

Qubit Mapping onto Superconducting Device [ASPLOS'19]

Paulihedral Compiler for Quantum Simulation [ASPLOS'22]

Cross-Layer Co-Design for Quantum Chemistry [ISCA'21]

Efficient Superconducting Architecture Design [ASPLOS'20]

IBM

aws

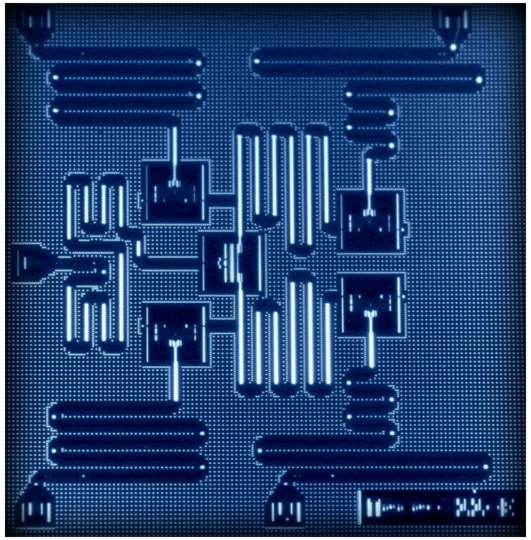
Amazon Braket

QCOR

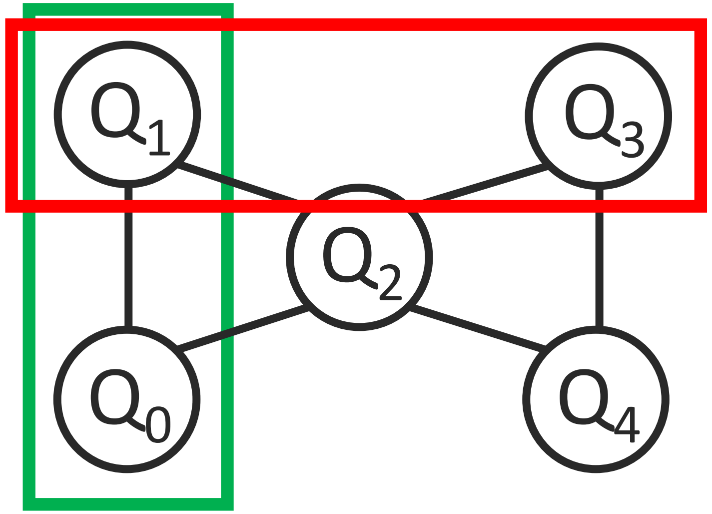
QUANTUM COMPILER

Background

- Limitation of the superconducting architecture



IBM’s 5-qubit superconducting quantum chip



Coupling graph – limited qubit connection

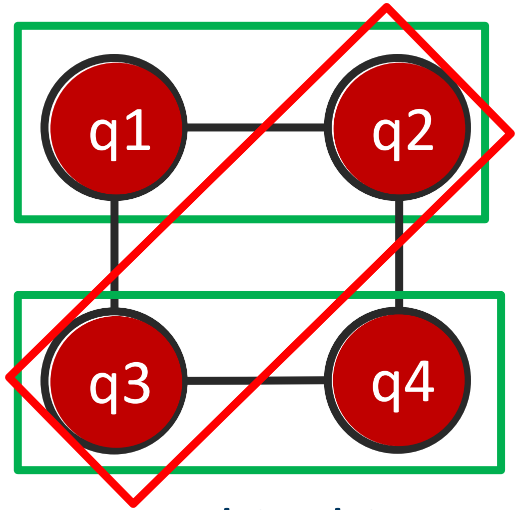
CNOT gates only allowed on the connected edges

CNOT Q0, Q1

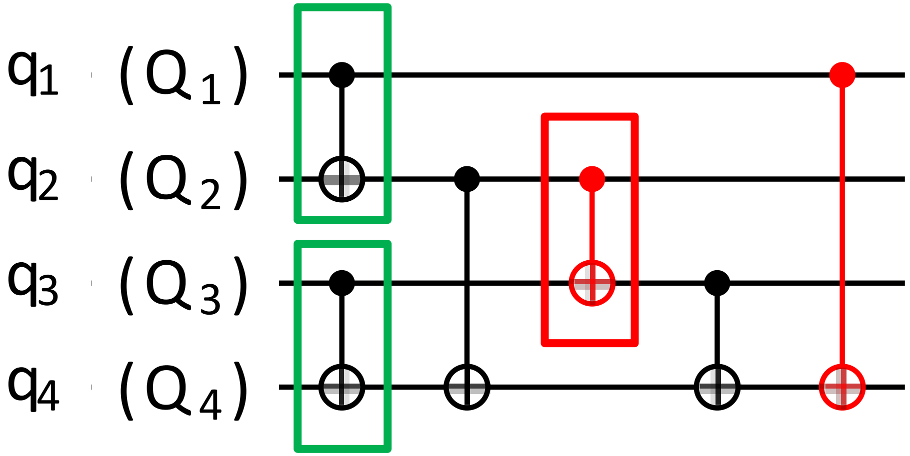
CNOT Q1, Q3

Qubit Mapping

- An example



4-qubit chip



4-qubit circuit

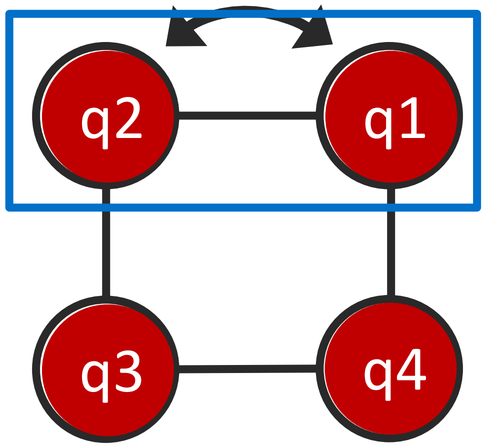
Some CNOTs are executable

But some CNOTs are not executable

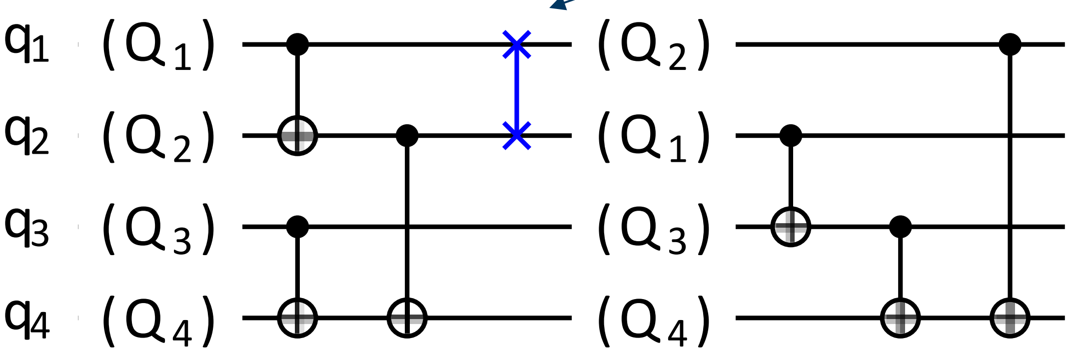
Qubit Mapping

- An example

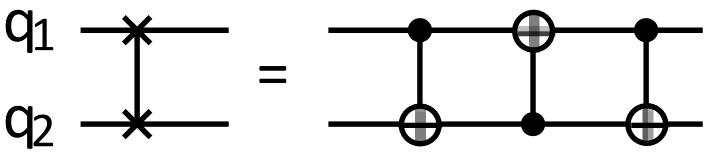
Insert additional gate SWAP: exchange the mapping



4-qubit chip



4-qubit circuit



1 SWAP = 3 CNOT

Each additional SWAP leads to more noise

Qubit Mapping

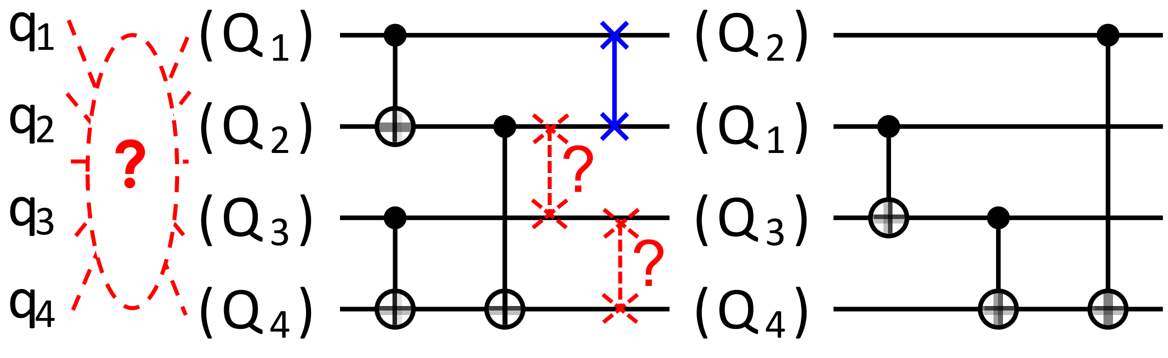
- Output:

  Initial mapping & SWAPs for remapping
- Constraint:

  Satisfy all two-qubit gate dependencies
- Optimization objectives:

  Compilation quality: Gate count, Circuit depth

  Compilation time



Quantum 'register allocation' but with different constraints

hardware-independent quantum program → Quantum Compiler Optimization → hardware-compatible quantum program

Qubit Mapping

- Output:

  Initial mapping & SWAPs for remapping
- Constraint:

  Satisfy all two-qubit gate dependencies
- Optimization objectives:

  Compilation quality: Gate count, Circuit depth

  Compilation time

Qubit Mapping onto Superconducting Device [ASPLOS’19]

- **The default mapping method in Qiskit**

IBM

QCOR QUANTUM COMPILER

hardware-independent quantum program → Quantum Compiler Optimization → hardware-compatible quantum program

Going Further

Application

Language

Compiler

Architecture

Device

Can we also further optimize it with algorithm/program information?

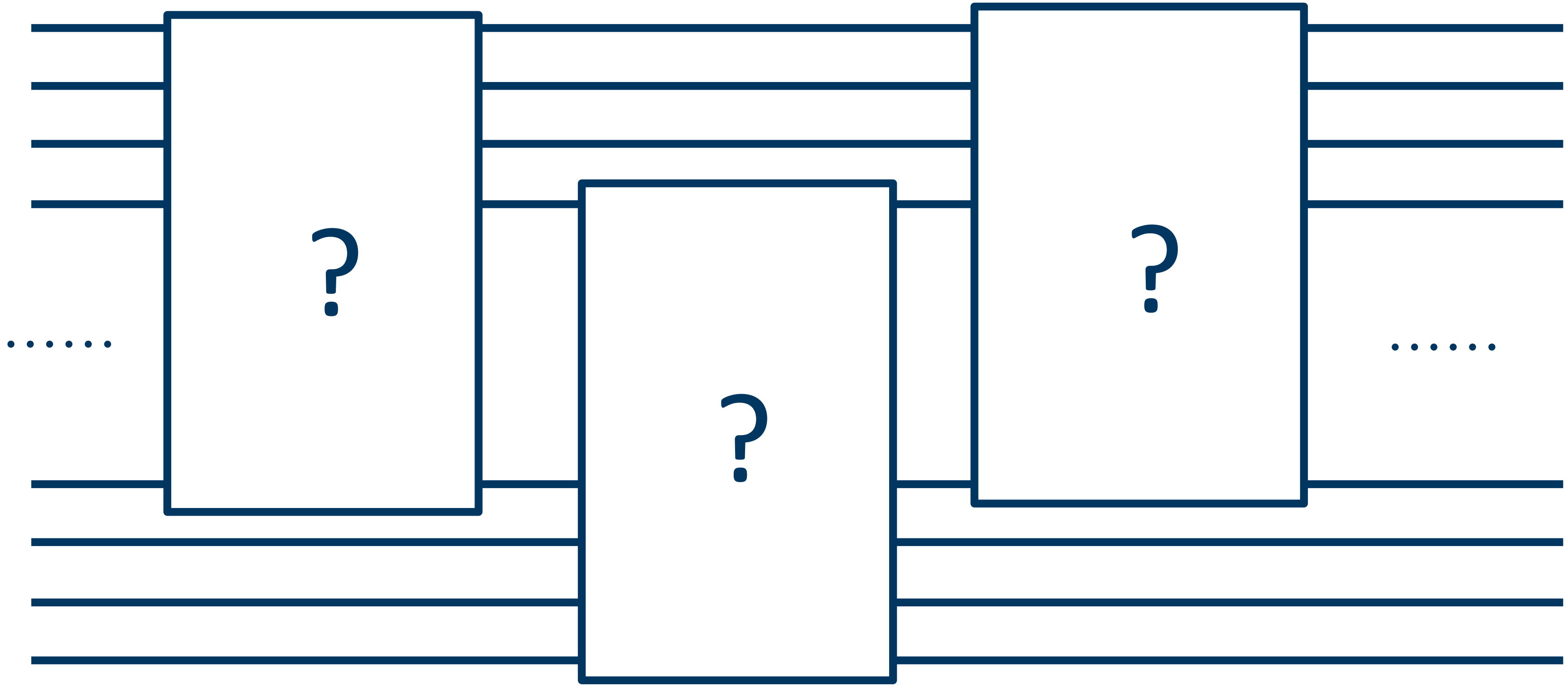
Mapping is an optimization based on hardware information

Paulihedral Compiler for Quantum Simulation [ASPLOS'22]

Technology Stacks

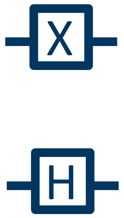
Challenge

- How to extract algorithmic information from the program

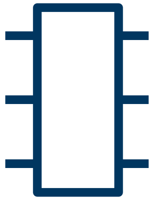


Challenge

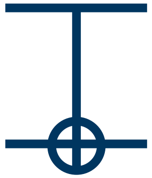
- Gate matrix size grows exponentially



2x2 matrices



8x8 matrices



4x4 matrices

n-qubit gate **2^n x 2^n matrices**

- Hard to reason about, analyze, synthesize, decompose

Challenge

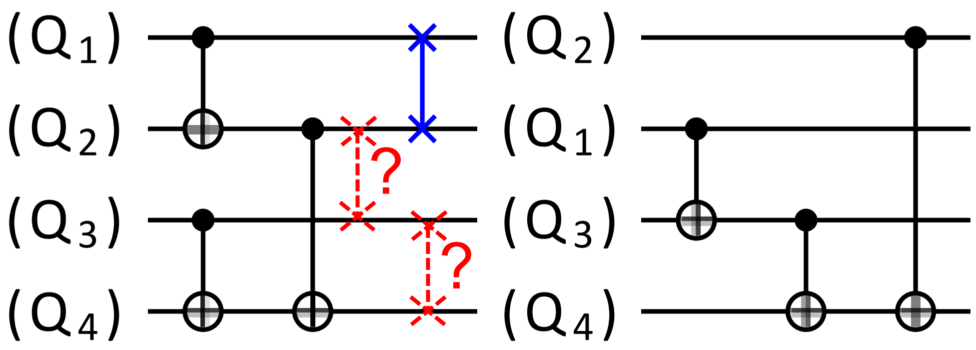
- Gate matrix size grows exponentially

trices

^n matrices

- Hard to r

pose

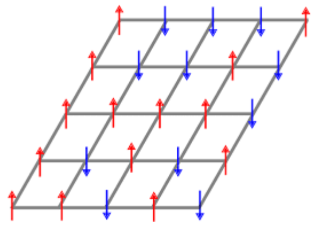


Small-scale circuit transformations only!

Opportunities at High-Level

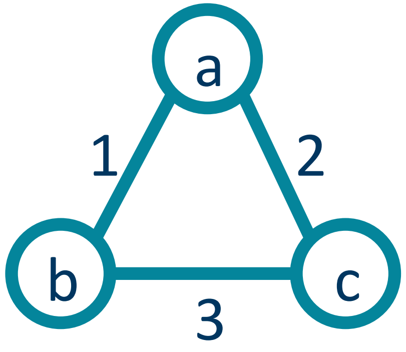
- More abstract compact form? Yes
- Simulation is widely used in quantum algorithm design

Simulation



$$H = -\sum J_{ij} Z_i Z_j - \mu \sum h_j Z_j$$

Graph Cut



$$H = \frac{1}{2}(Z_a Z_b - I) + \frac{2}{2}(Z_a Z_c - I) + \frac{3}{2}(Z_b Z_c - I)$$

And many more

Quantum Simulation Kernel

- A widely-used subroutine

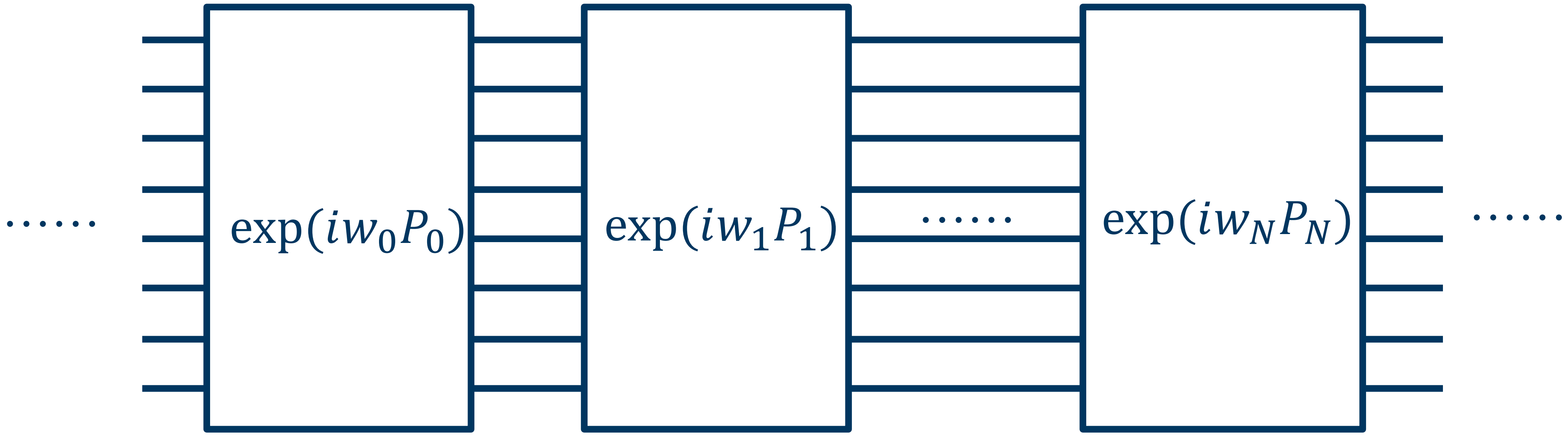
$$P = \sigma_{n-1}\sigma_{n-2} \dots \sigma_0$$

$$\sigma_i = X \mid Y \mid Z \mid I$$

$$P = Y_4 Z_3 I_2 X_1 Z_0$$

$$\exp(iHt)$$

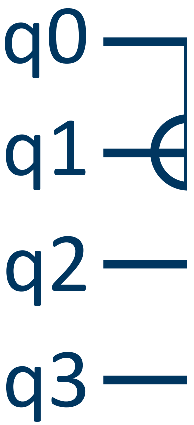
$H = \sum_i w_i P_i$, P_i is a Pauli string, $w_i \in \mathbb{R}$ is weight



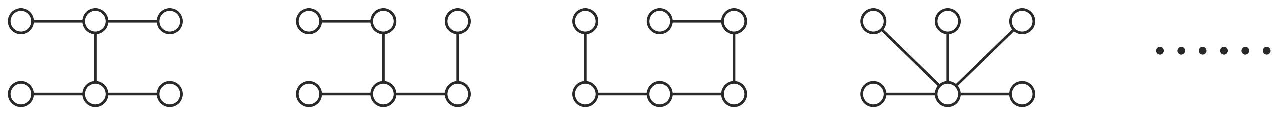
Tree Structure

- The program pattern

$$\exp(iwZ_3Z_2Z_1Z_0)$$

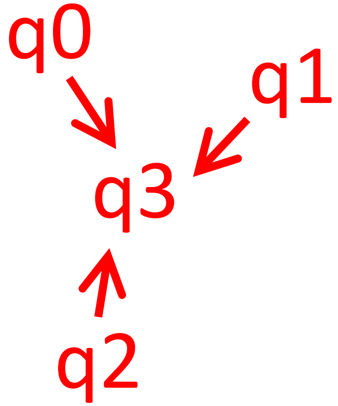


q0



$$\exp(iw\sigma_{n-1}\sigma_{n-2}\ldots\sigma_0)$$

Large-scope transformations
can be easily supported

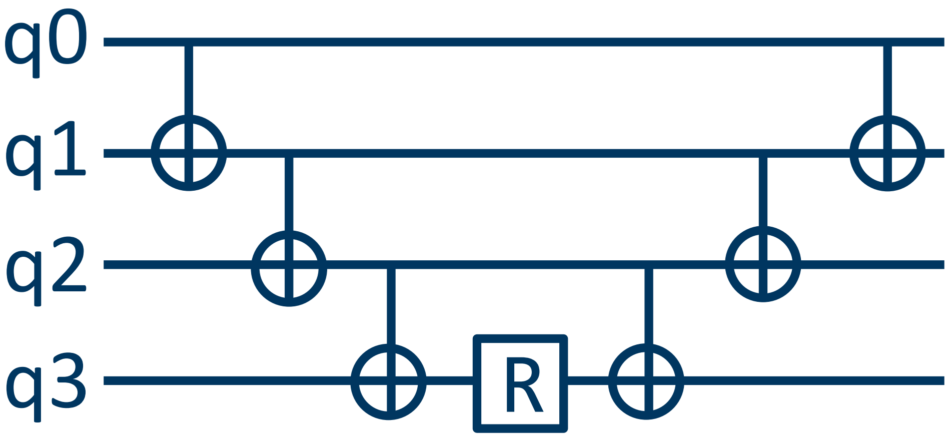


All thes
equivale
have a t

Conventional Compilation

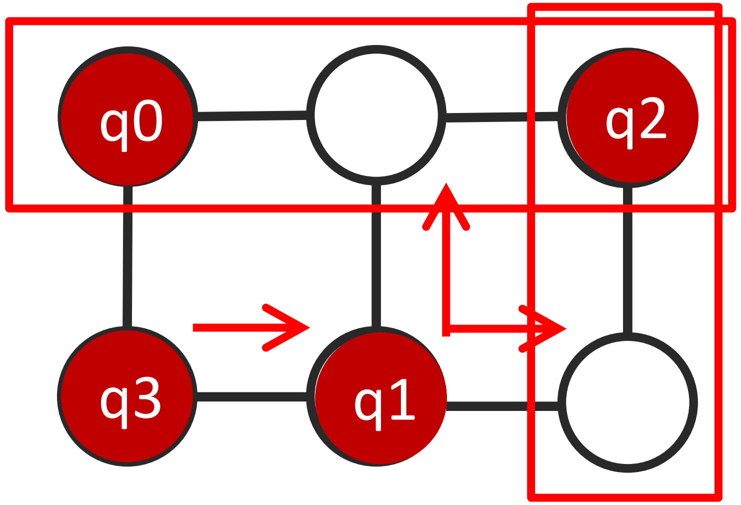
- Find SWAP in gate sequence

$$\exp(iwZ_3Z_2Z_1Z_0)$$



CNOT q0, q1
CNOT q1, q2
CNOT q2, q3

SWAP q1, …
CNOT q0, q1
CNOT q1, q2
SWAP q3, …
SWAP q3, …
CNOT q2, q3

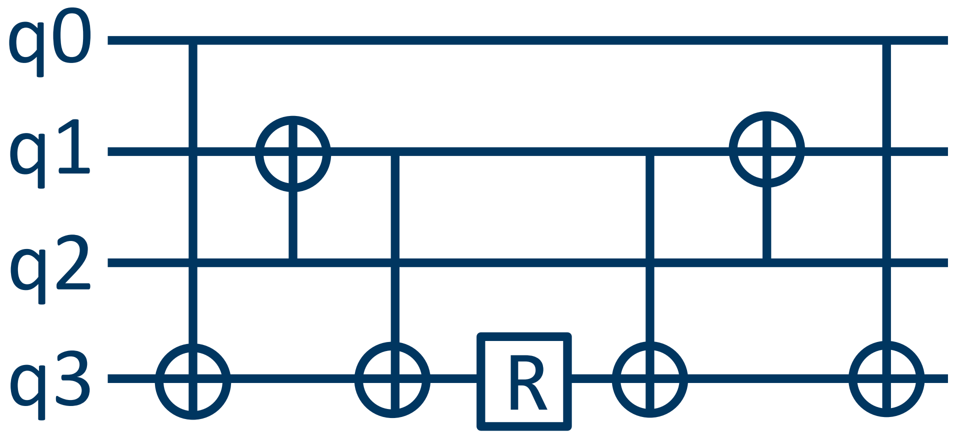


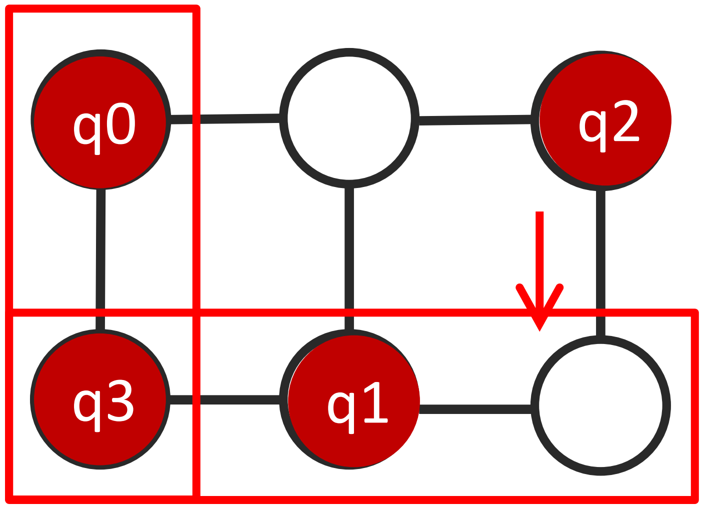
Paulihedral Compilation

- Leverage high-level information

$$\exp(iwZ_3Z_2Z_1Z_0) \longrightarrow$$

SWAP q2, …
CNOT q0, q3
CNOT q2, q1
CNOT q1, q3





Find a tree embedding, then generate the CNOT tree

Comparison

- Significant SWAP overhead reduction

$\exp(iwZ_3Z_2Z_1Z_0)$

Paulihedral

SWAP q2, …
CNOT q0, q3
CNOT q2, q1
CNOT q1, q3

One SWAP

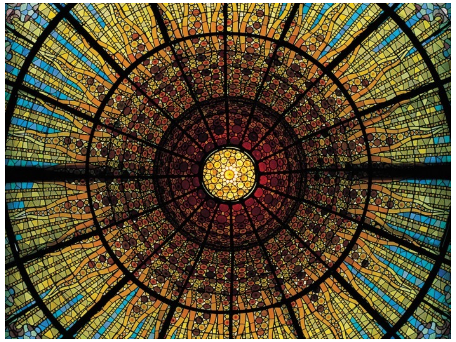
Conventional compilation

CNOT q0, q1
CNOT q1, q2
CNOT q2, q3

SWAP q1, …
CNOT q0, q1
CNOT q1, q2
SWAP q3, …
SWAP q3, …
CNOT q2, q3

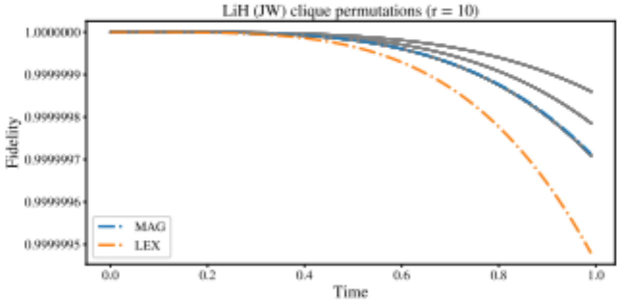
Three SWAPs

Moreover



[Livio, 2012]

symmetry preserving



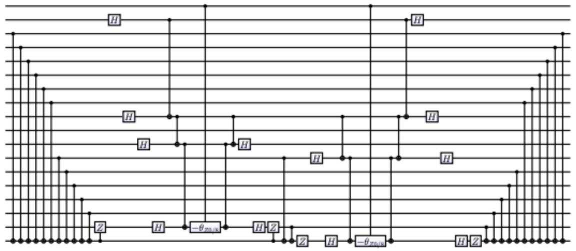
[Gui et al. 2019]

error mitigation

$$(a_2^\dagger a_0 - a_0^\dagger a_2) = \frac{i}{2}(X_2 Z_1 Y_0 - Y_2 Z_1 X_0)$$
$$(a_3^\dagger a_1 - a_1^\dagger a_3) = \frac{i}{2}(X_3 Z_2 Y_1 - Y_3 Z_2 X_1)$$
$$\begin{aligned}(a_3^\dagger a_2^\dagger a_1 a_0 - a_0^\dagger a_1^\dagger a_2 a_3) = {} \\ \frac{i}{8}(X_3Y_2X_1X_0 + Y_3X_2X_1X_0 + Y_3Y_2Y_1X_0 + Y_3Y_2X_1Y_0 \\ - X_3X_2Y_1X_0 - X_3X_2X_1Y_0 - Y_3X_2Y_1Y_0 - X_3Y_2Y_1Y_0).\end{aligned}$$

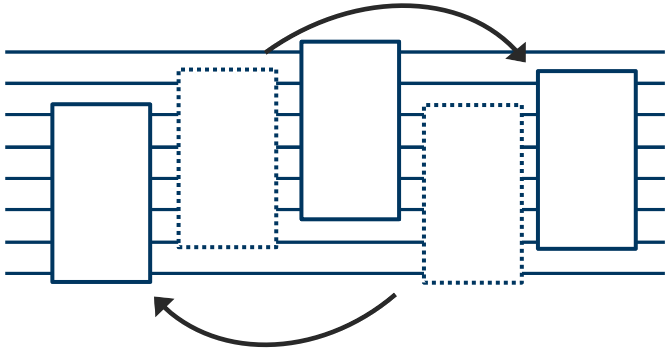
[McArdle et al. 2020]

parameter sharing

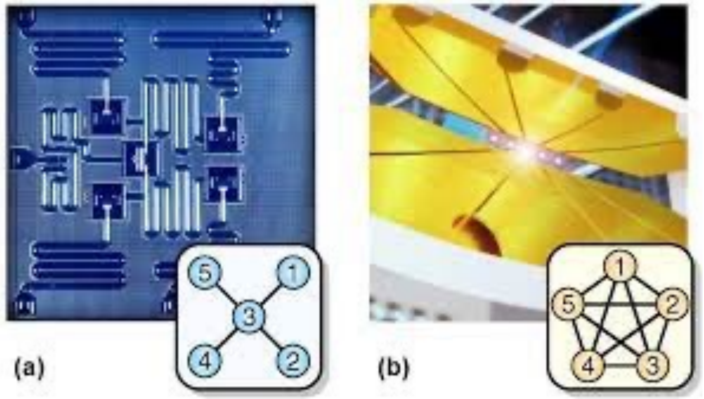


[Hastings et al. 2015]

more gate cancellation



large-scope scheduling



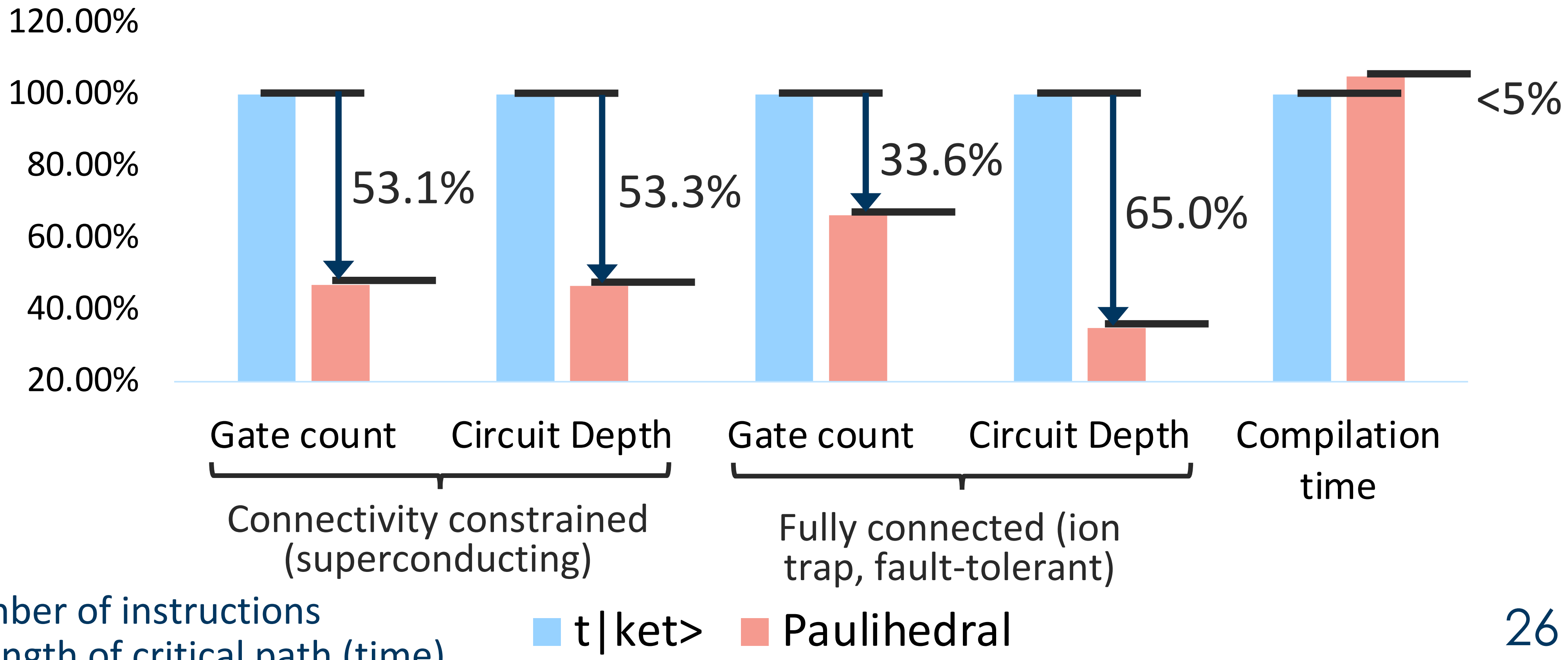
[Linke et al. 2017]

different backends

Please see paper for details

Evaluation

- Benchmarks: molecule/Ising/Heisenberg/random Hamiltonian, UCCSD/QAOA graph ansatz



gate count -> number of instructions
circuit depth -> length of critical path (time)

Changes in Qiskit Infrastructure

Before:

Application $\xrightarrow{\text{Gate sequence}}$ Qiskit gate-level compiler

After:

Application $\xrightarrow{\text{High-level IR + gate sequence}}$ Qiskit high-level compiler $\xrightarrow{\text{Gate sequence}}$ Qiskit gate-level compiler

Amazon Braket

More high-level optimizations are incoming!

High-Level Optimization in QEC

- Quantum hardware, e.g., superconducting quantum devices, is noisy.

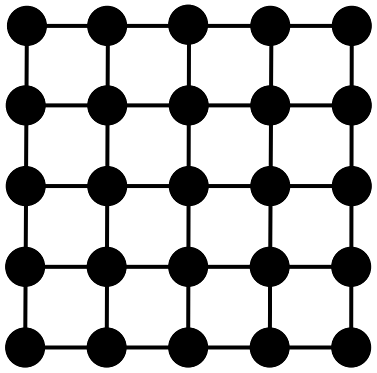
  IBM Washington. CNOT error: ~1%; 70 CNOT, fidelity < 50%.

- Quantum error correction (QEC) is important for future, fault-tolerant quantum computing.

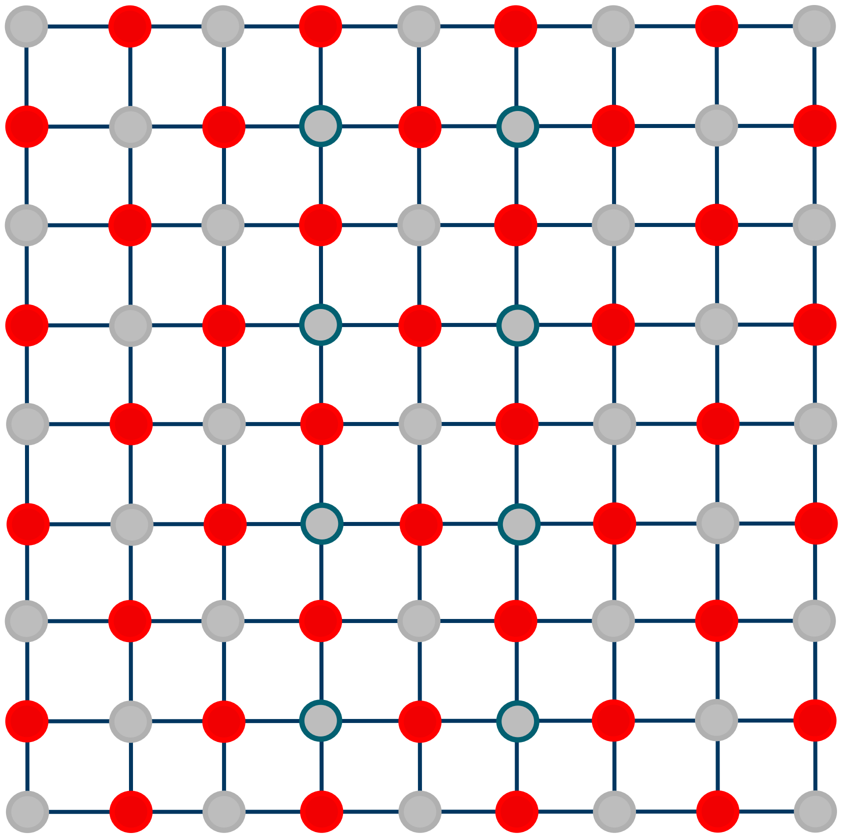
- Surface code is among the best QEC codes, with great error correction capability and mild resource requirement.

How Does Surface Code Work?

- It composes physical qubits in 2-D lattice with high error into a logic qubit with less error.

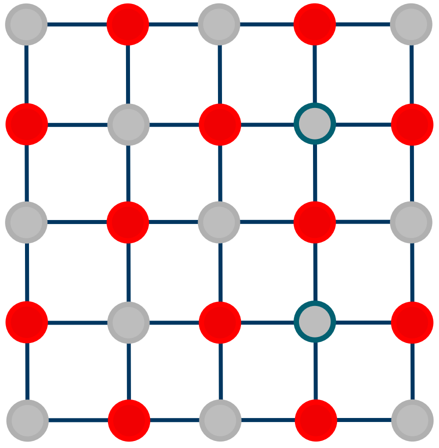


Coupling graph for Physical Qubits in 2-D lattice Space

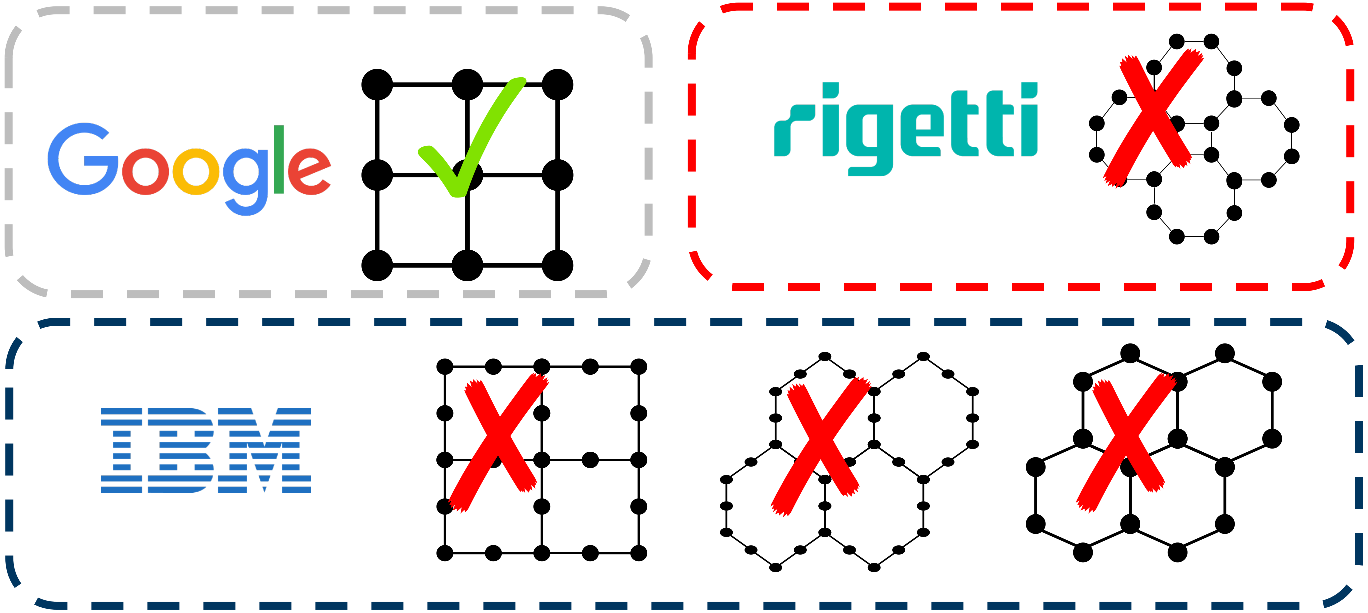


- Freedom: 13 -12 = 1

Mismatch between Surface Code and Sparse Architectures



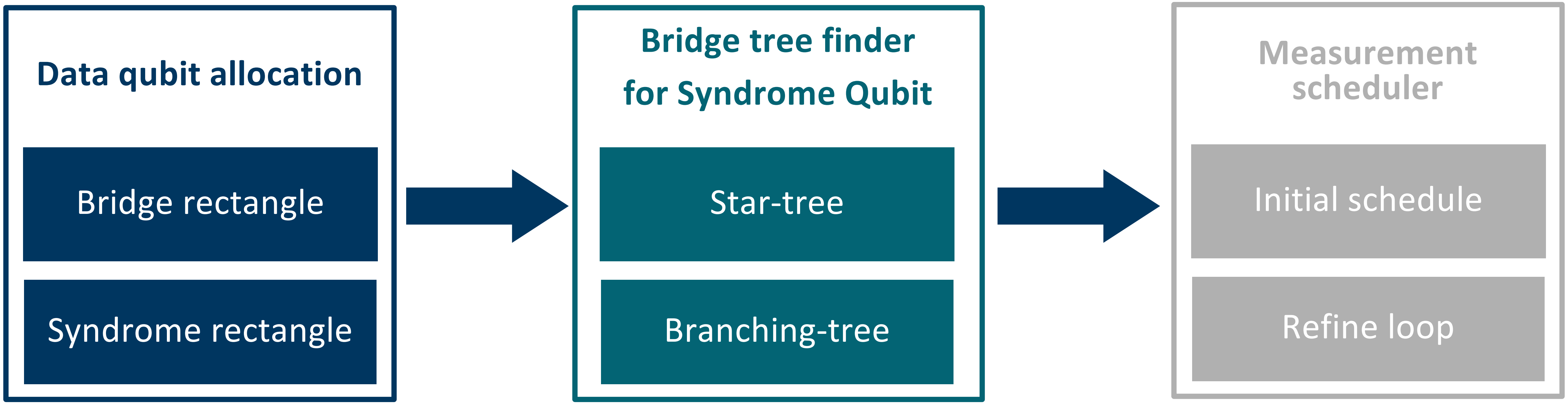
Surface Code



Some recent study **manually** designed **new** QEC codes tailored for these **sparse** architecture.

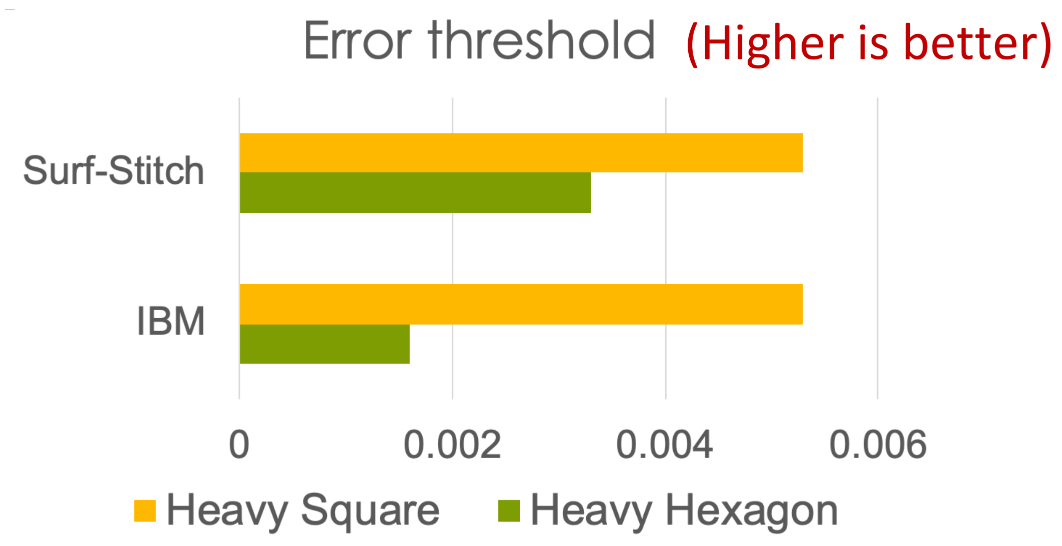
Our Surf-Stitch Compiler: Overview [ISCA'22]

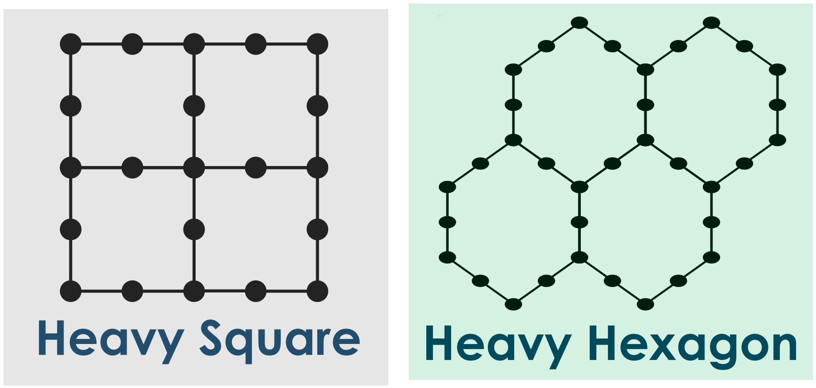
- We can systematically solve the mismatch: 1) Good abstraction (**Stablizer**); 2) Knowledge of **beneficial and legal transformations**; 3) An efficient search scheme.
- Compilation flow surface code to *different sparse quantum architectures*



More details in our paper

Performance





More results in our paper

Key Observation:

Our automatically generated QEC code is comparable or even better than IBM's manually designed QEC codes tailored for the sparse architectures.

Our Key Insight:

Automatic compiler framework is more efficient to tap the large design space of QEC design when the problem is properly **abstracted**.

High-Level Operator in Program Testing

- Projection-Based Runtime Assertion for Quantum Program Testing and Debugging [OOPSLA’20]

Quantum Programming

- Quantum programming is error-prone for programmers living in the classical world

Wrong sign on teleportation.qasm when X,Z == [1, 1] #

Open vsoftco opened this issue on 24 Aug · 0 comments

vsoftco commented on 24 Aug • edited

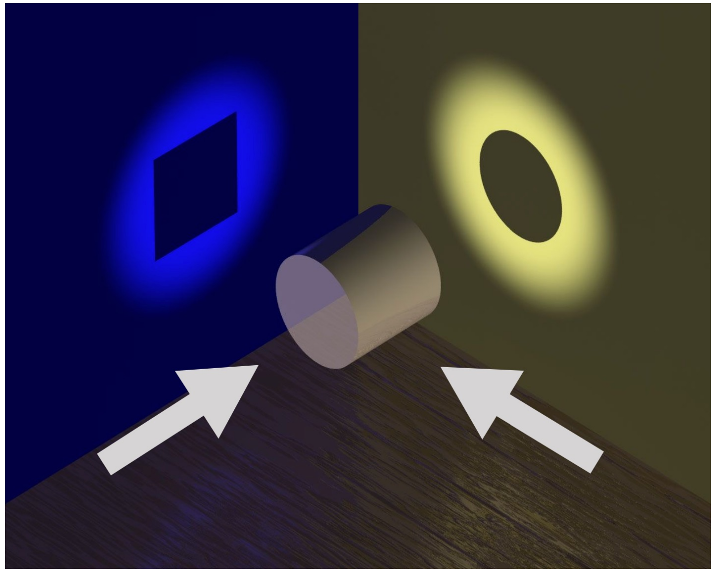
X and Z corrections should be interchanged, otherwise we get a MINUS (-) global phase when the final measurement result is [1, 1]. Proposed fix in #48.

Quantum teleportation - quantum ‘Hello World’

Bug in the quantum ‘Hello World’

Test a Quantum State

- Existing approaches

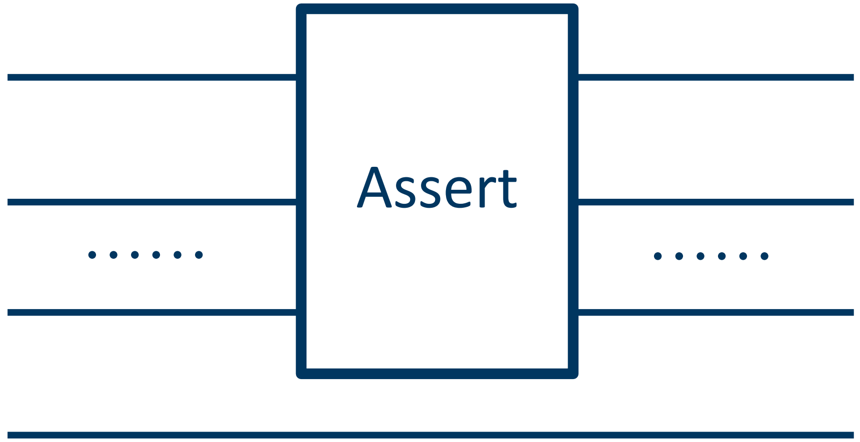


Quantum State Tomography

- a protocol to fully characterize any state
- very **expensive** – in a nutshell, observe a quantum state from many dimensions, repeated state preparation and measurement many times

Test a Quantum State

- Existing approaches



Quantum Program Assertion

- describe a quantum state property using classical languages

$$|1010 \ldots 0101\rangle$$
$$|++ + \cdots + +\rangle$$
$$|000 \ldots 00\rangle + |111 \ldots 11\rangle$$

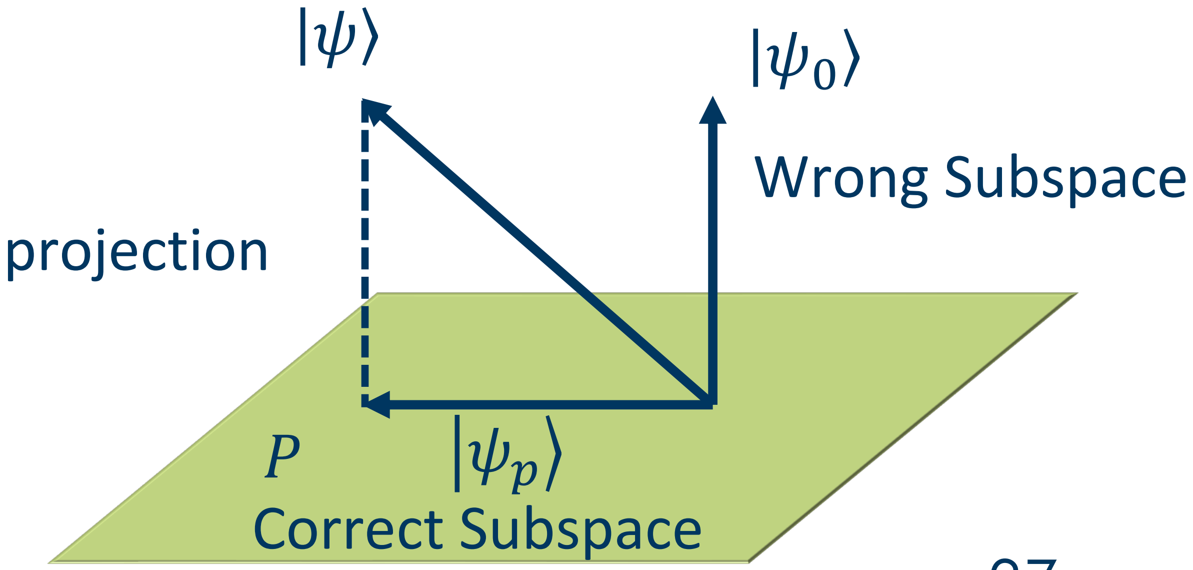
- **poor expressive power**
- direct measurement will usually destroy the tested state

Our Objective

- We hope to develop quantum **assertions** for testing and debugging:
  - **Strong expressive power**: specify complex quantum state properties
  - **Efficient checking**: efficiently quantum computers

- We select **Projection** for both needs.

- Subspaces as predicates

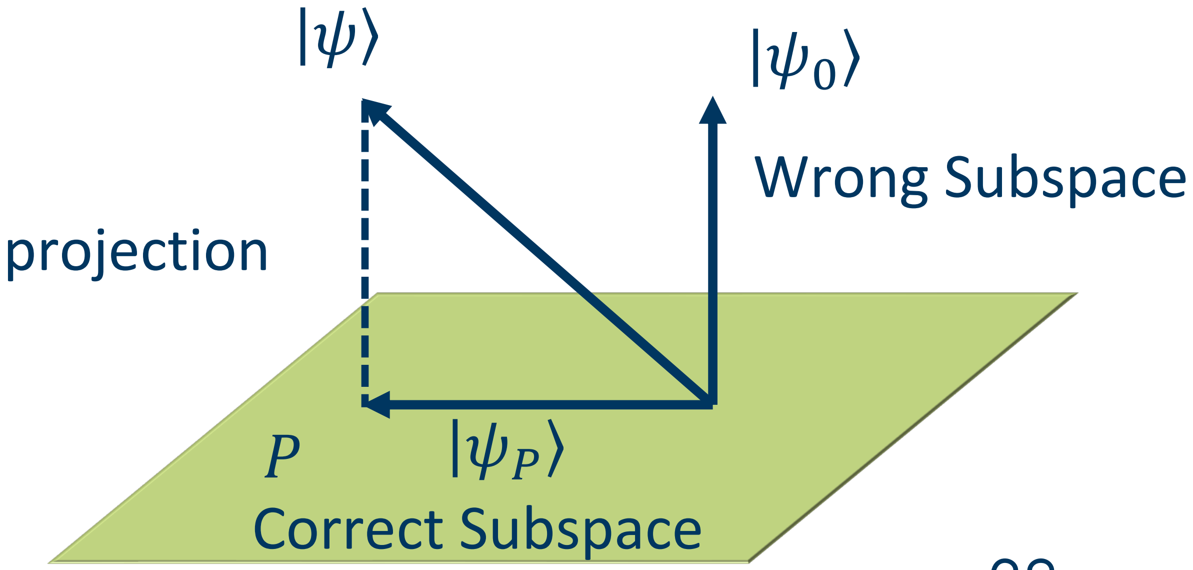


Expression Power

- Projections allows to specify subspaces of any dimensions (rank of projection)

  Example 1: $P = |00\rangle\langle 00|$, Subspace: $\{|00\rangle\}$ (rank 1)

  Example 2: $P = |00\rangle\langle 00| + |11\rangle\langle 11|$, Subspace: $\{|00\rangle, |11\rangle\}$ (rank 2)



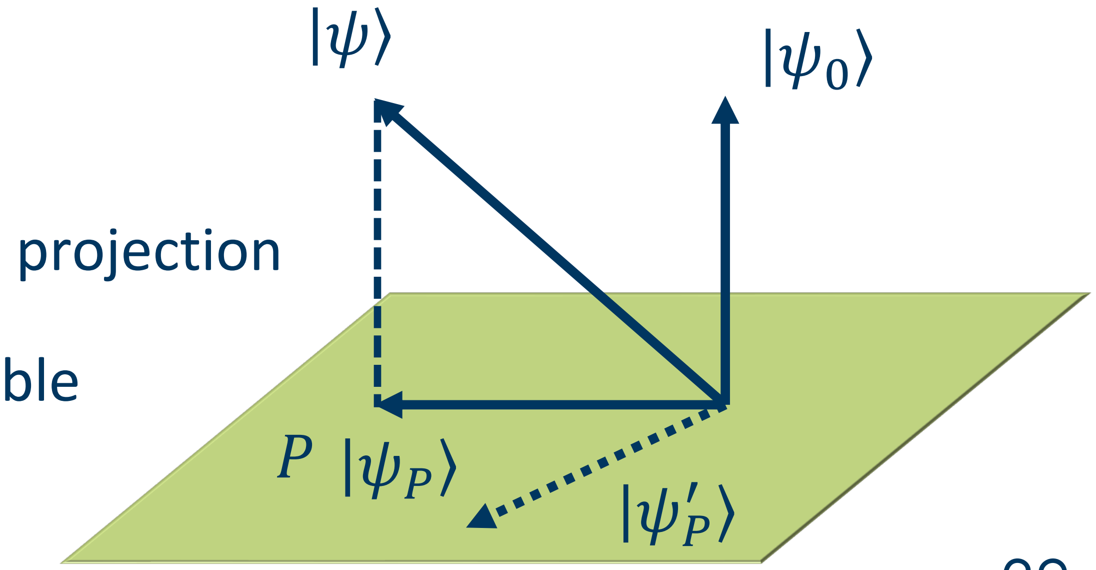
Expression Power

- Projections allows to specify subspaces of any dimensions (rank of projection)

  Example 1: $P = |00\rangle\langle 00|$, Subspace: $\{|00\rangle\}$ (rank 1)

  Example 2: $P = |00\rangle\langle 00| + |11\rangle\langle 11|$, Subspace: $\{|00\rangle, |11\rangle\}$ (rank 2)

- **Caveat:** indistinguishable states



$|\psi_P\rangle$ and $|\psi'_P\rangle$ are indistinguishable

Because we are not doing tomography

---> A sweet point, which will allow efficient checking

Efficient Checking

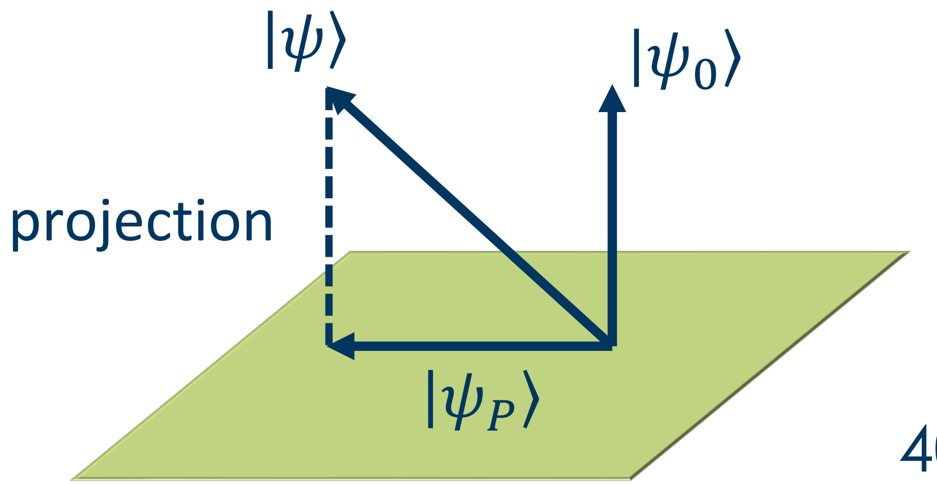
- Turn characterization into determination (sound approximation)

  Whether in a subspace is much easier to check

- Projective measurement may preserve the measured state

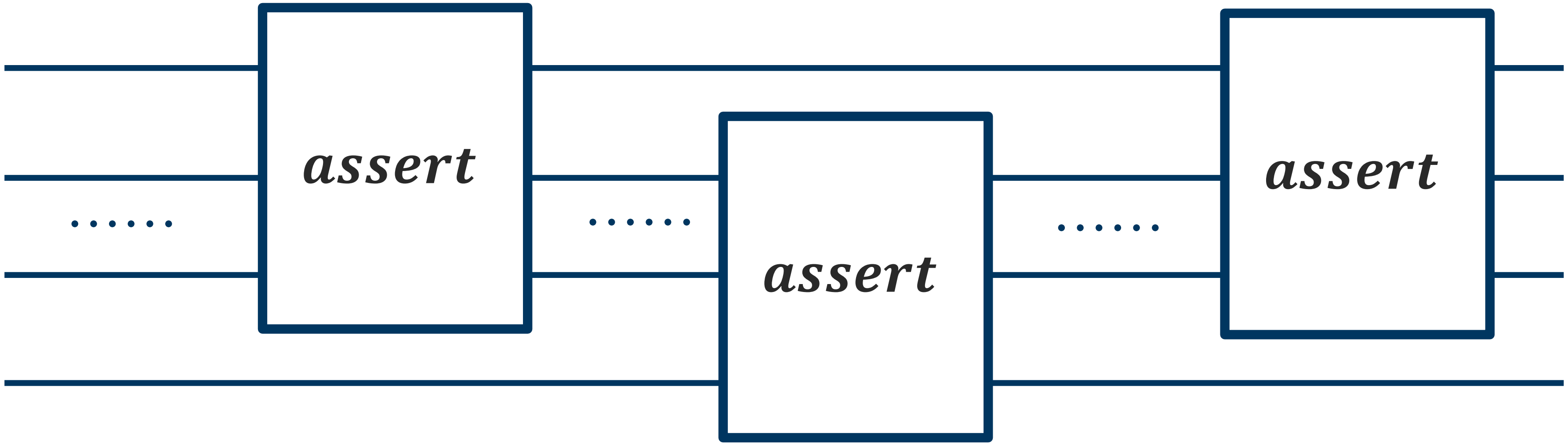
$$P|\psi_P\rangle = |\psi_P\rangle$$

Allowing follow-up execution



Projection-Based Assertion

- Language primitive $\boldsymbol{assert}(\overline{\boldsymbol{q}}; \boldsymbol{P})$, where P is a projection and $\bar{q}$ is a set of qubits



- $M_P = \{M_{true} = P,\ M_{false} = I - P\}$

  If true, continue

  If false, abort and report

Practical Implementation Issues

- $M_P = \{M_{true} = P, M_{false} = I - P\}$, this constructed measurement may not be directly executable on a quantum computer.

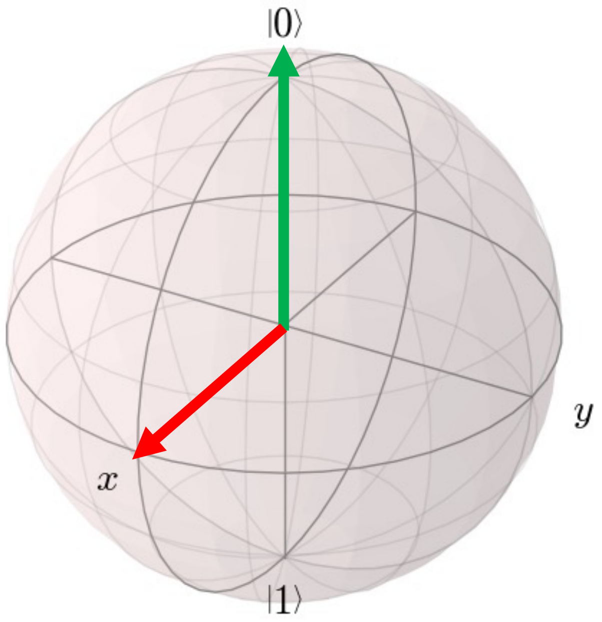
- Two key constraints:

  Limited measurement basis

  Dimension (rank) mismatch

Limited Measurement Basis

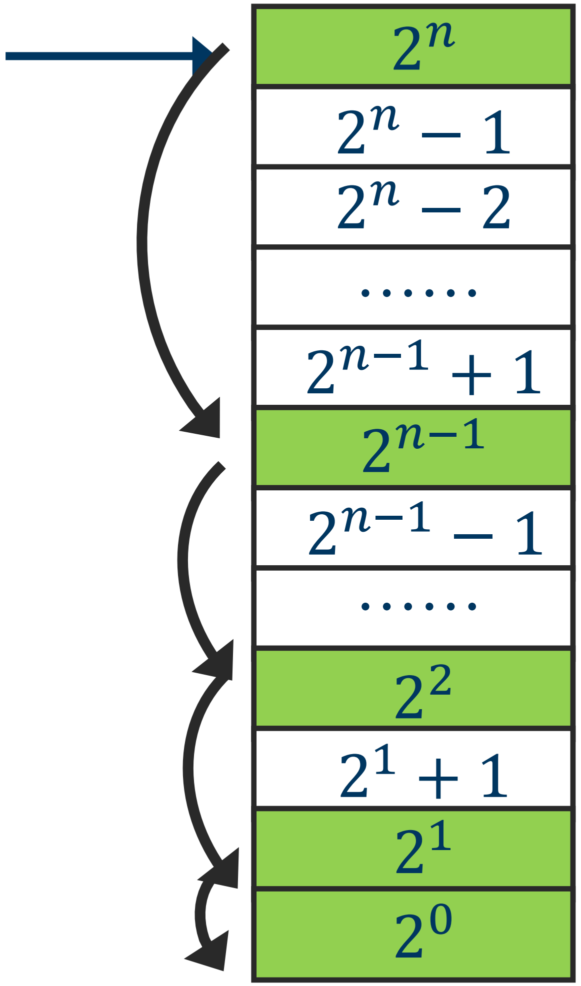
- Most physical quantum computers only support direct projective measurement along the computational basis $\{|0\rangle, |1\rangle\}$
- $M_P = \{M_{true} = |0\rangle\langle 0|, M_{false} = |1\rangle\langle 1|\}$ 😇
- $M_P = \{M_{true} = |+\rangle\langle +|, M_{false} = |-\rangle\langle -|\}$ 😡



$$|+\rangle = \frac{|0\rangle+|1\rangle}{\sqrt{2}}, |-\rangle = \frac{|0\rangle-|1\rangle}{\sqrt{2}}$$

Dimension Mismatch

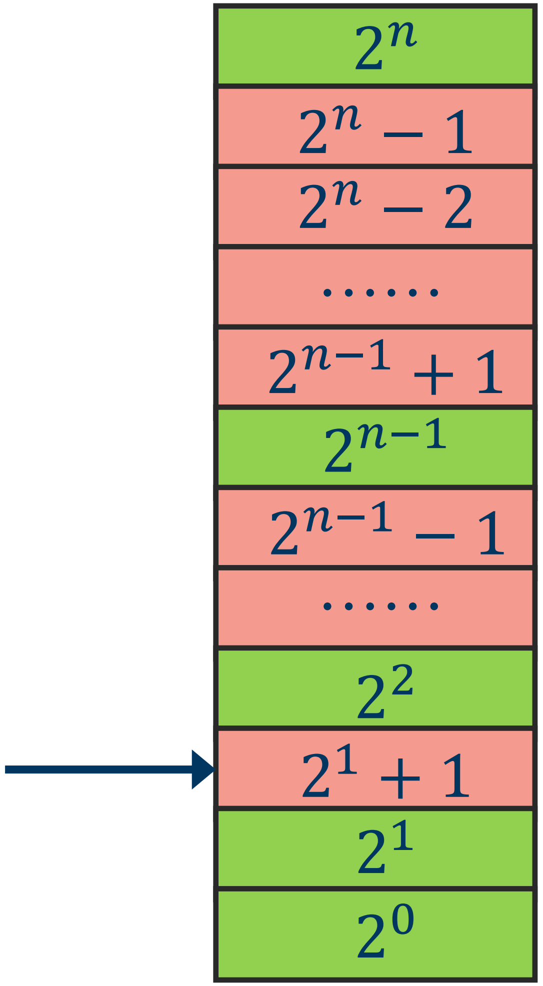
- 2^n-dimensional space for n qubits
- Measure one qubit, the space is reduced by half, 2^(n−1)-dimensional space
- We can only measure an integer number of qubits
- Only support rank $P \in \{2^{n-1}, 2^{n-2}, \dots, 2^0\}$



All possible ranks

Dimension Mismatch

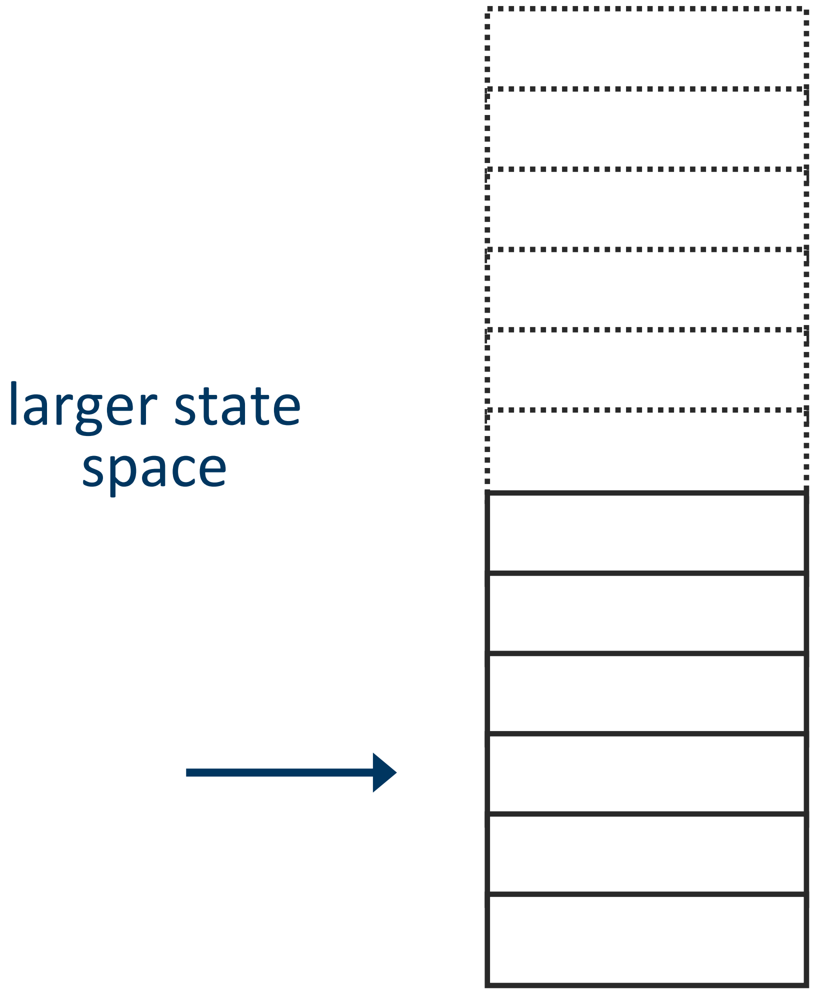
- 2^n-dimensional space for n qubits
- Measure one qubit, the space is reduced by half, 2^(n−1)-dimensional space
- We can only measure an integer number of qubits
- Only support rank $P \in \{2^{n-1}, 2^{n-2}, \ldots, 2^0\}$
- $P = |00\rangle\langle 00| + |01\rangle\langle 01| + |11\rangle\langle 11|$



All possible ranks

Assertion Compilation Flow

- We propose a compilation flow
    1. Control the dimension using **ancilla qubit**



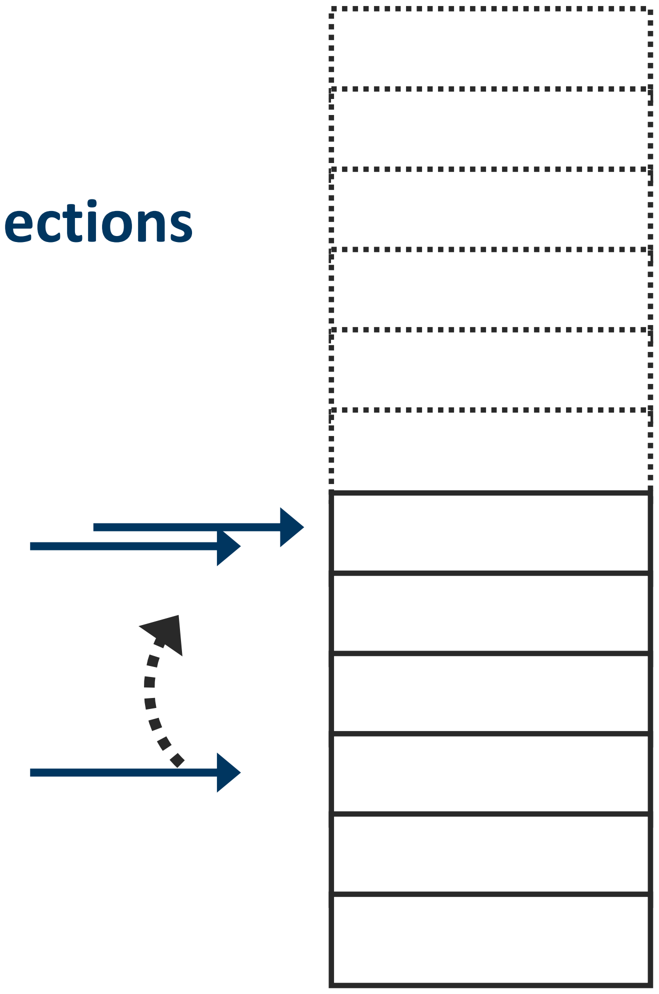
Assertion Compilation Flow

- We propose a compilation flow

    1. Control the dimension using

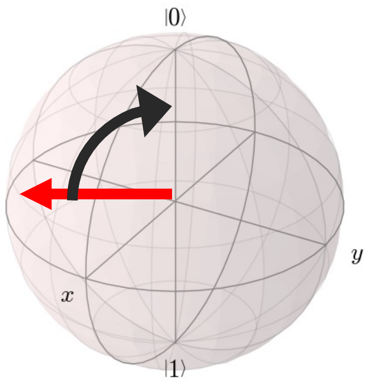
    ancilla qubit and **intersection of larger projections**

Change the relative position

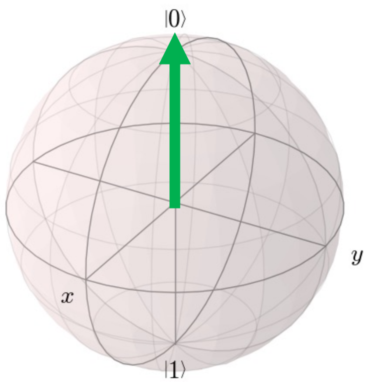


Assertion Compilation Flow

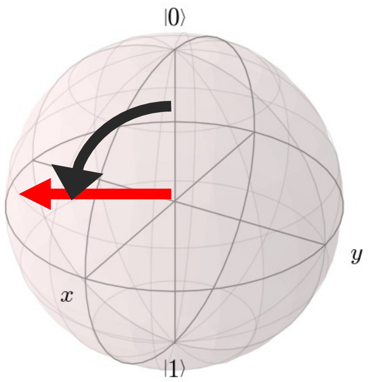
- We propose a compilation flow
    1. Control the dimension using **ancilla qubit** and **intersection of larger projections**
    2. Tune the measurement basis with **unitary transformations**



rotate



measure



rotate back

Assertion Compilation Flow

- We propose a compilation flow

  1. Control the dimension using

     ancilla qubit and **intersection of larger projections**

  2. Tune the measurement basis with

     unitary transformations

- Making all projection-based assertion physically executable
- More details (statistical efficiency proof, example, etc.) can be found in our paper

Error Mitigation

Assertions as Error Mitigation

- One point calculation of energy, no variational loop
- Experiment A : no verification
- Experiment B : runtime verification of 3 symmetries
- Despite the increased circuit depth the error is halved!

	Circuit size (CX gates)	Energy (Ha)	Error
No verification	15	-1.1032	0.1321
Mid-circuit verification	**25**	**-1.1738**	**0.0615**
Exact	-	-1.2353	-

Cambridge Quantum Computing

Projection-based deployed in molecule simulation experiments
Reduce simulation error by 50%

Take Home Message

- High-level information is very useful
- Reconstructing high-level semantics from quantum assembly is hard
- Enhance your software with build-in high-level operators
- Design corresponding compilation/transformation

Q & A

Application

Language

Compiler

Architecture

Device

Technology Stacks

Projection-Based Runtime Assertion [OOPSLA'20]

Qubit Mapping onto Superconducting Device [ASPLOS'19]

Efficient Superconducting Architecture Design [ASPLOS'20]

Paulihedral Compiler for Quantum Simulation [ASPLOS'22]

Surface-Code Compiler [ISCA'22]

Cross-Layer Co-Design for Quantum Chemistry [ISCA'21]

Cambridge Quantum Computing

Amazon Braket

QCOR QUANTUM COMPILER

Thank You!