DESIGN AUTOMATION CONFERENCE

59

JULY 10 - 14, 2022

MOSCONE WEST CENTER

SAN FRANCISCO, CA, USA

DESIGN AUTOMATION CONFERENCE 59

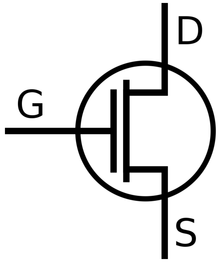
Enabling Deeper Quantum Compiler Optimization at High Level

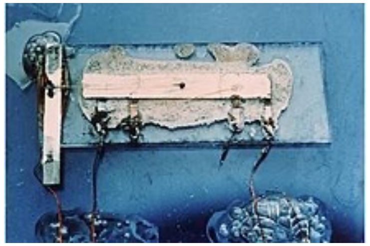
Yufei Ding, Gushu Li, Anbang Wu, Yuan Xie

07/11/2022

IBM amazon intel Pacific Northwest NATIONAL LABORATORY CISCO

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

Practical Quantum Computing

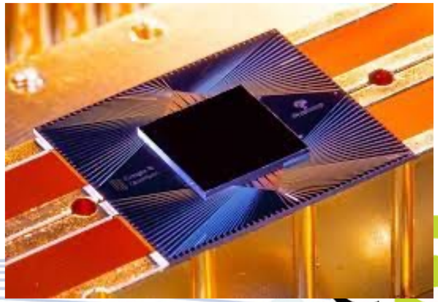
2nd Quantum Revolution

1980 Quantum Computer

1994 Cryptography

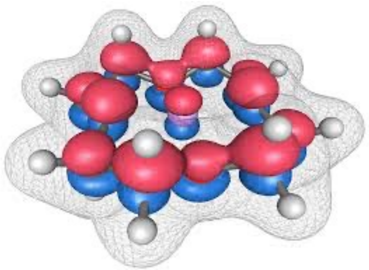
2019 Supremacy

Now

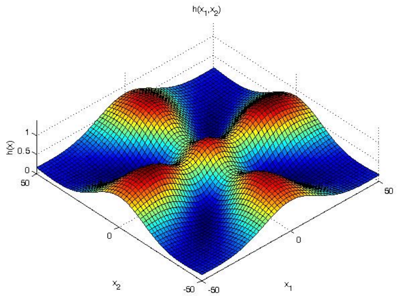


Quantum Computing System Stacks

Application →



Simulation



Optimization

Machine Learning

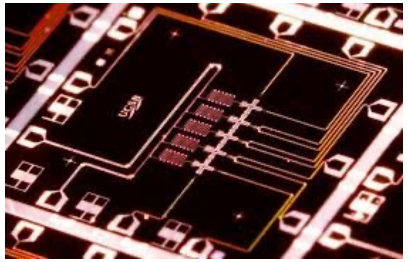
Cryptography

Technology Stacks

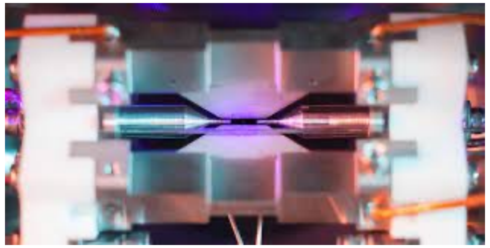
Quantum Computing System Stacks

Application

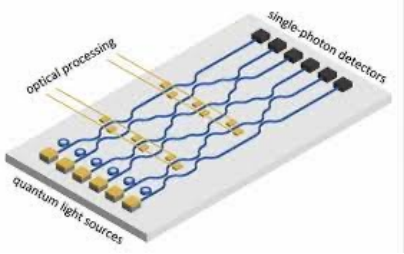
Device →



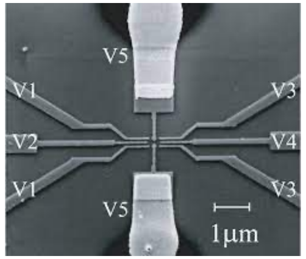
Superconducting



Ion Trap



Photonics



Quantum Dot

Technology Stacks

Quantum Computing System Stacks

Application

Language

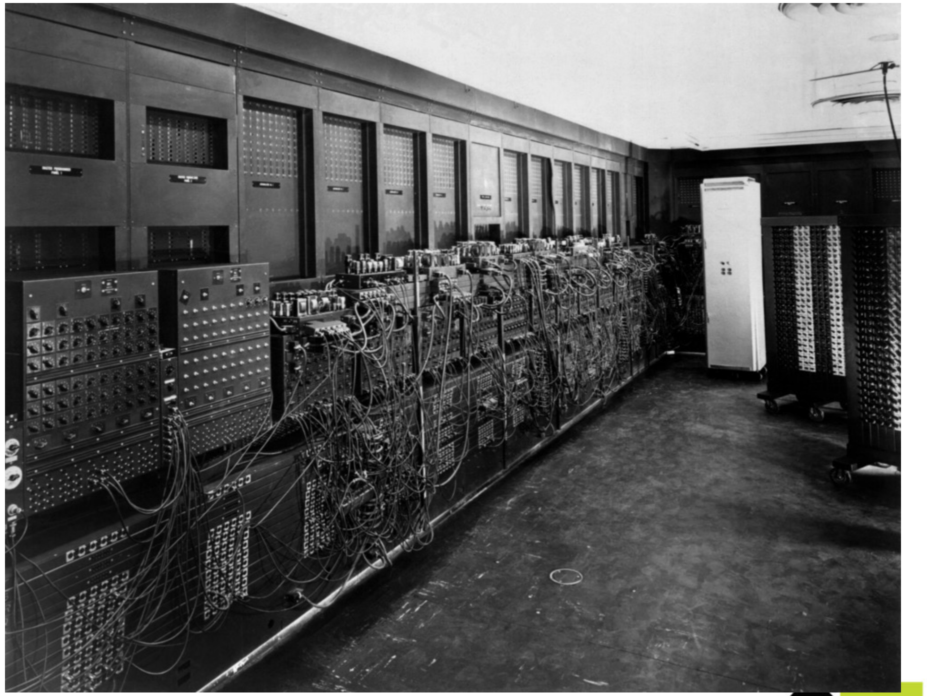
Compiler

Architecture

Device

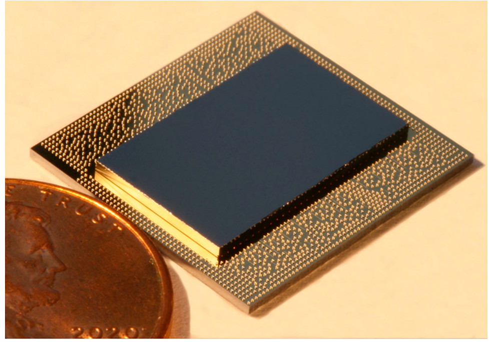
Technology Stacks

ENIAC, the first electronic general purpose digital computer, 1945



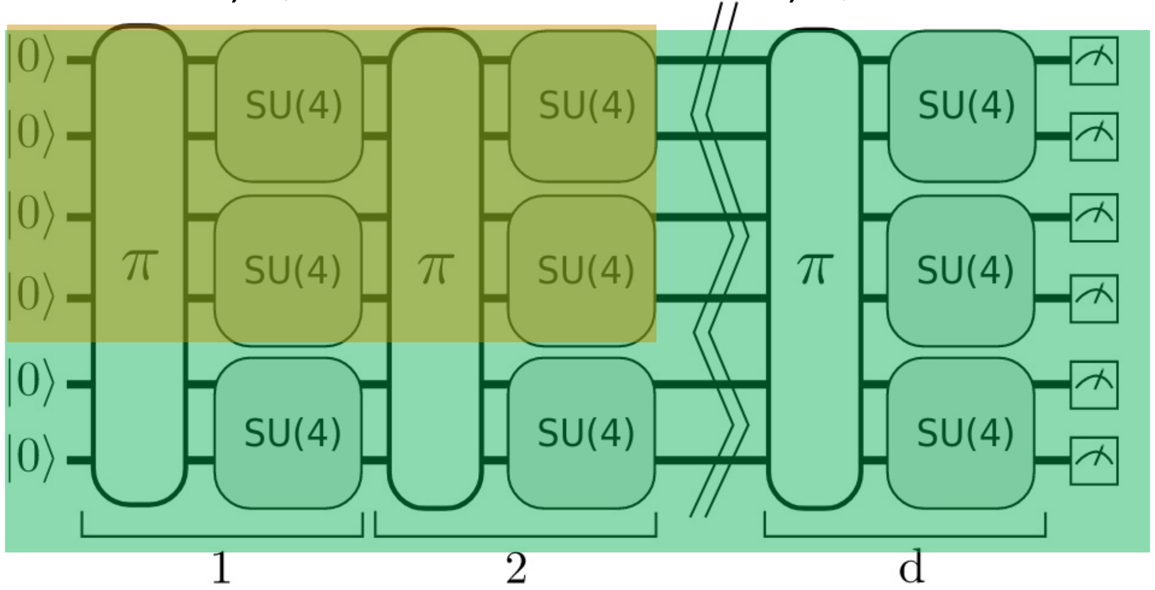
Hardware vs Software

- IBM benchmarking results



IBM Q Montreal

Hardware, QV=16 Software, QV=128



Quantum system research extends the computation capability!

And both software and hardware are important

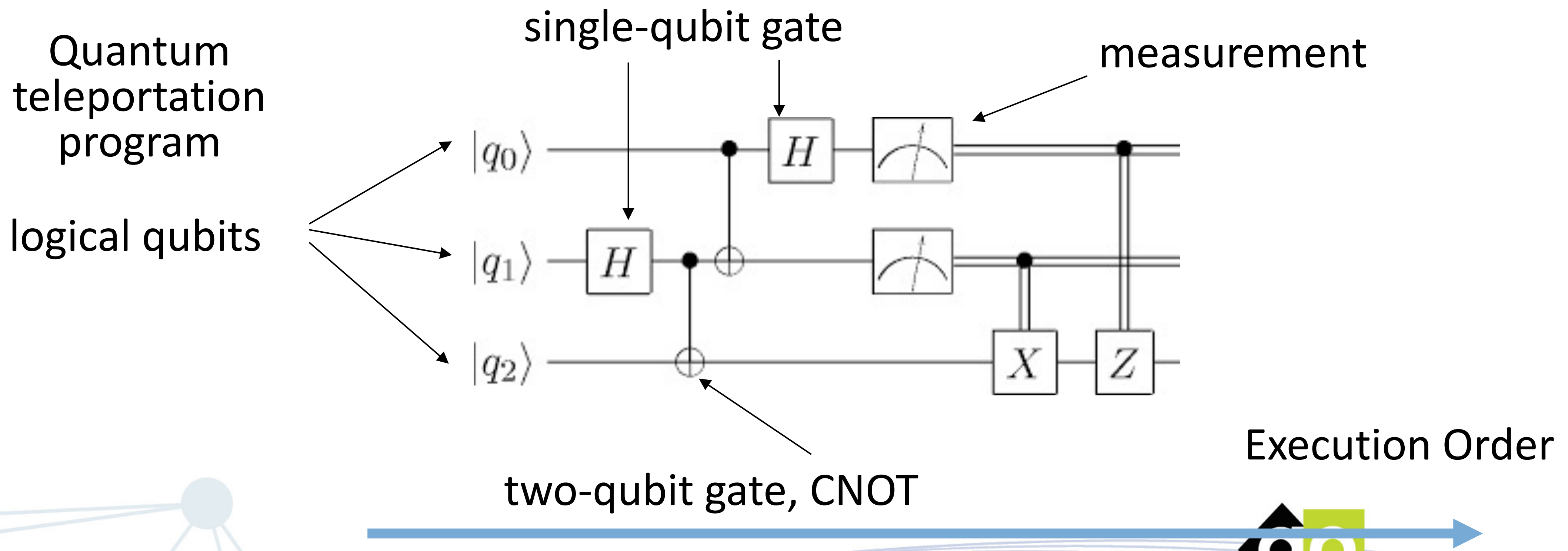
Contribution Breakdown **4 : 3**

[IBM Quantum]

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

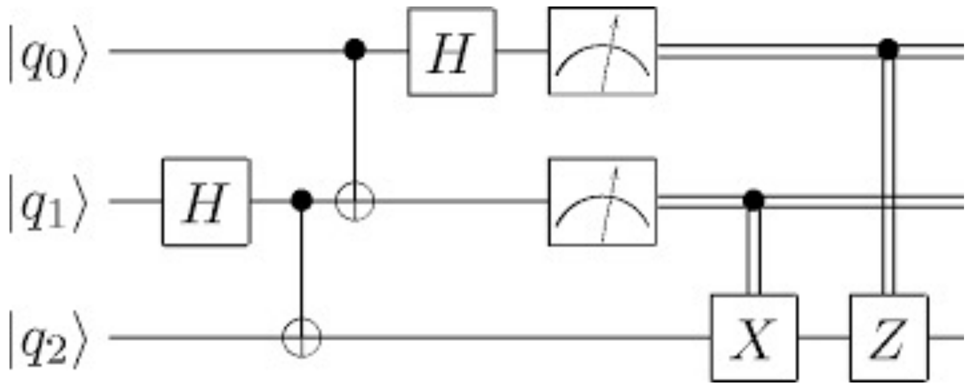
Background

- Quantum software – quantum circuit



Background

- Quantum software – program state



State vector:

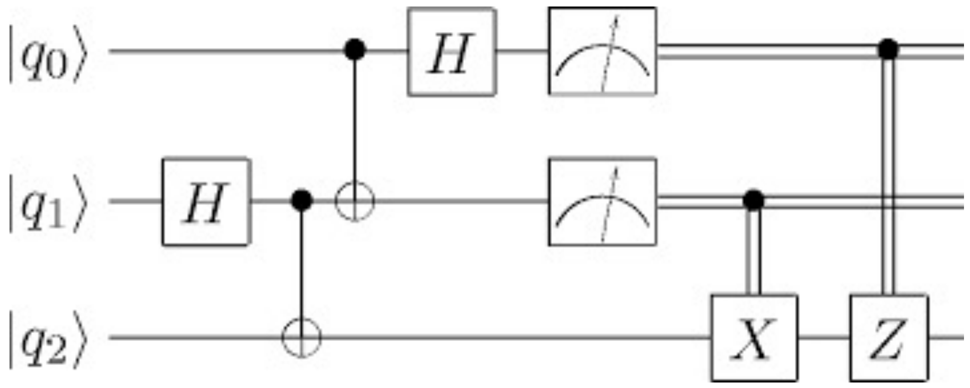
1-qubit, $|\psi\rangle = a_0|0\rangle + a_1|1\rangle$, $[a_0, a_1]$

3-qubit, $|\psi\rangle = a_{000}|000\rangle + a_{001}|001\rangle + \cdots + a_{111}|111\rangle$
$[a_{000}, \ldots, a_{111}]$

n-qubit, state vector $[a_0, \ldots, a_{2^n-1}]$ of size 2^n

Background

- Quantum software – program semantics



Gate matrices:

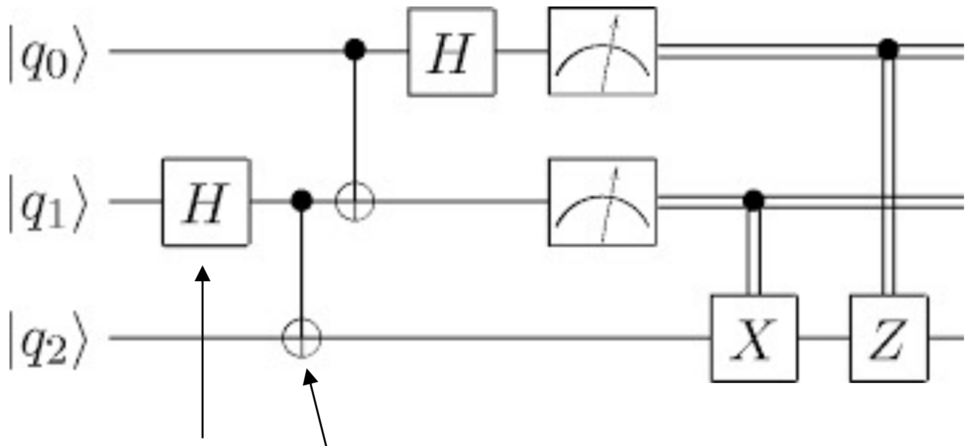
1-qubit, $\mathrm{H} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

2-qubit, $\mathrm{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

n-qubit, matrices of size 2^n by 2^n

Background

- Quantum software – program semantics



$I_0 \otimes H_1 \otimes I_2$

$I_0 \otimes CNOT_{2,3}$

Gate matrices:

1-qubit, $H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$

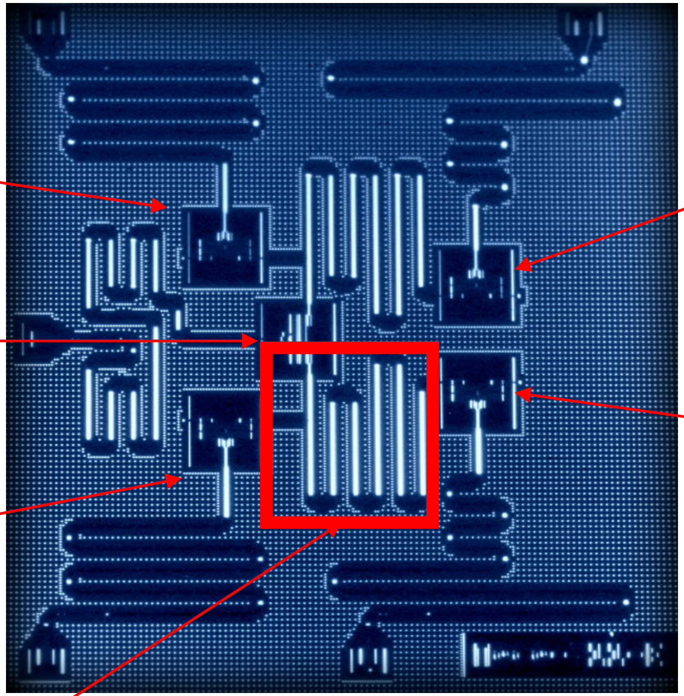
2-qubit, $CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

n-qubit, matrices of size 2^n by 2^n

Background

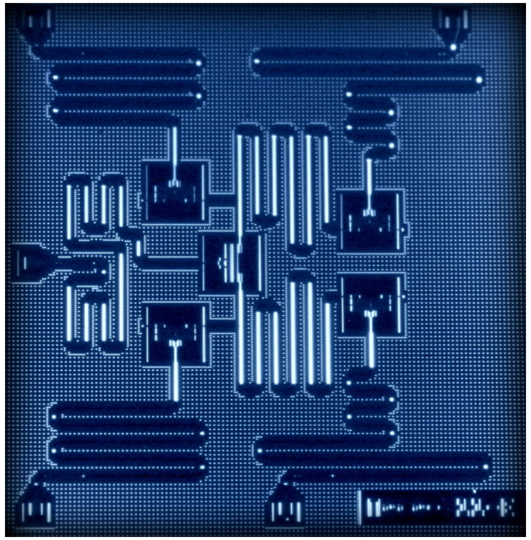
- Quantum hardware – the superconducting architecture

IBM’s 5-qubit superconducting quantum chip

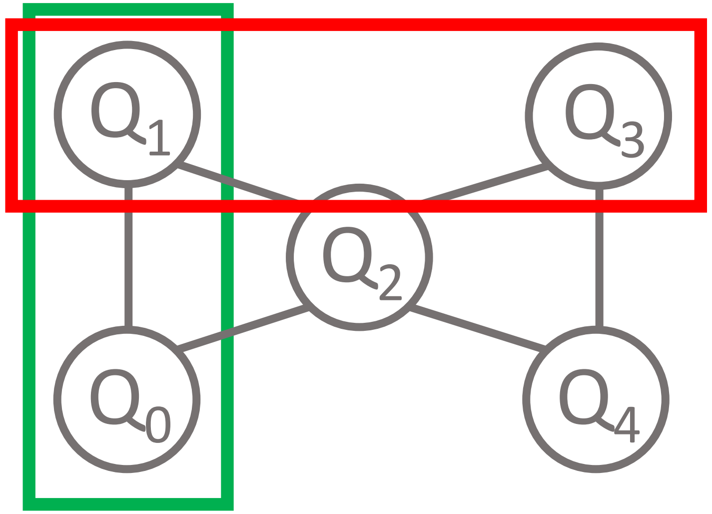


Background

- Quantum hardware – the superconducting architecture



IBM's 5-qubit superconducting quantum chip



Coupling graph – limited qubit connection

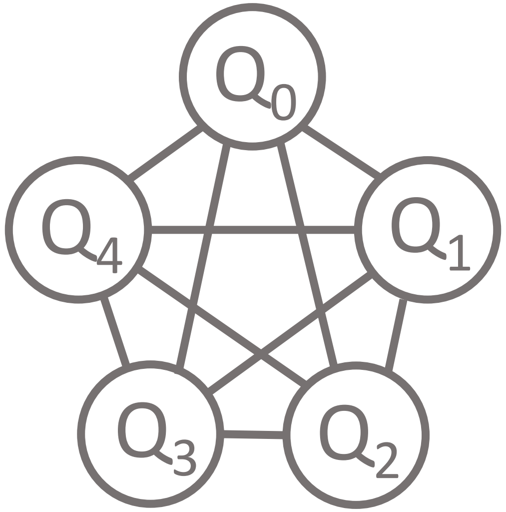
CNOT gates only allowed on the connected edges

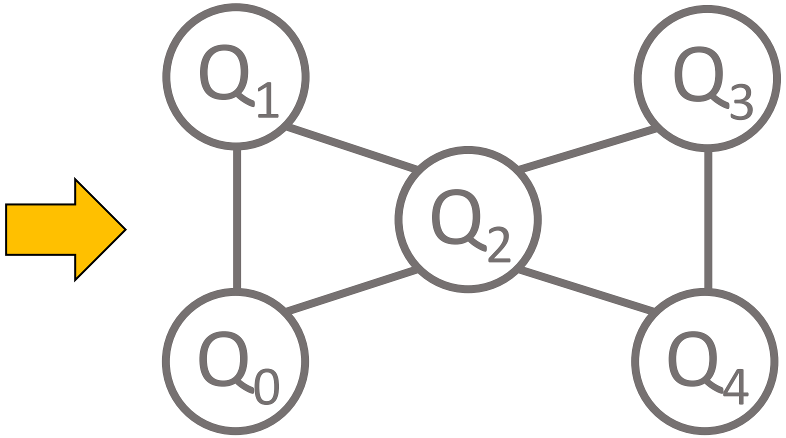
CNOT Q0, Q1

CNOT Q1, Q3

Mismatch

- When we write a quantum program, we may not know the underlying architecture





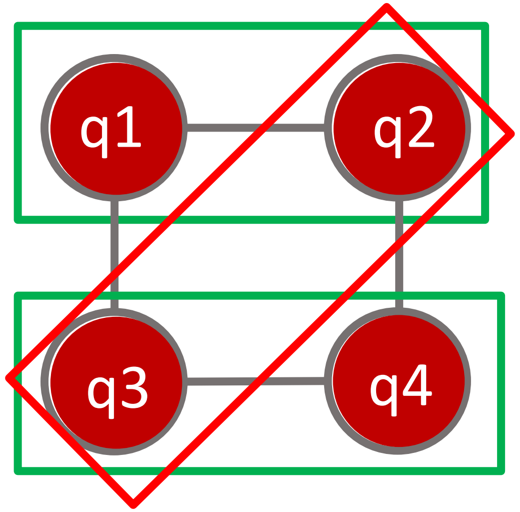
Some gates may not be executable

Ideal device – complete graph

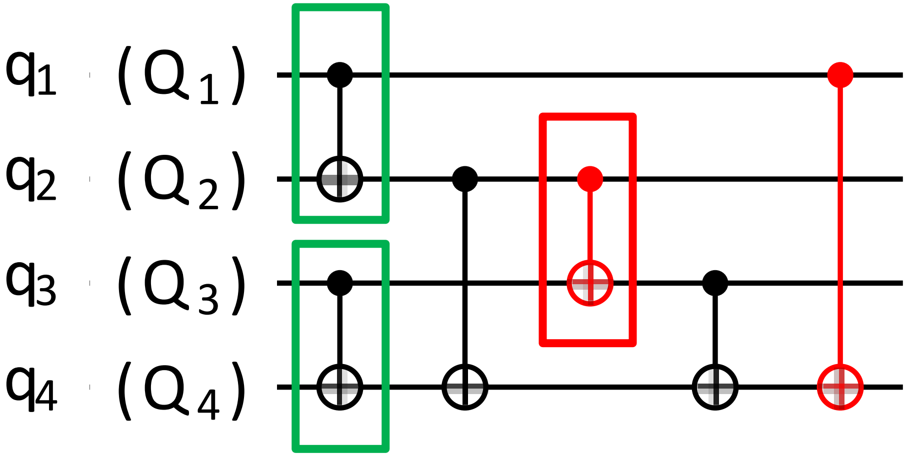
Coupling graph – limited qubit connection

Qubit Mapping

- An Example



4-qubit chip



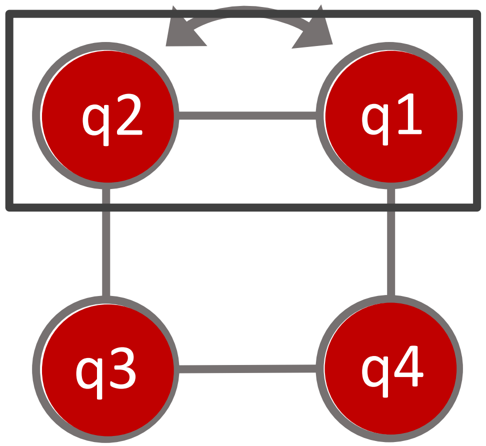
4-qubit circuit

Some CNOTs are executable

But some CNOTs are not executable

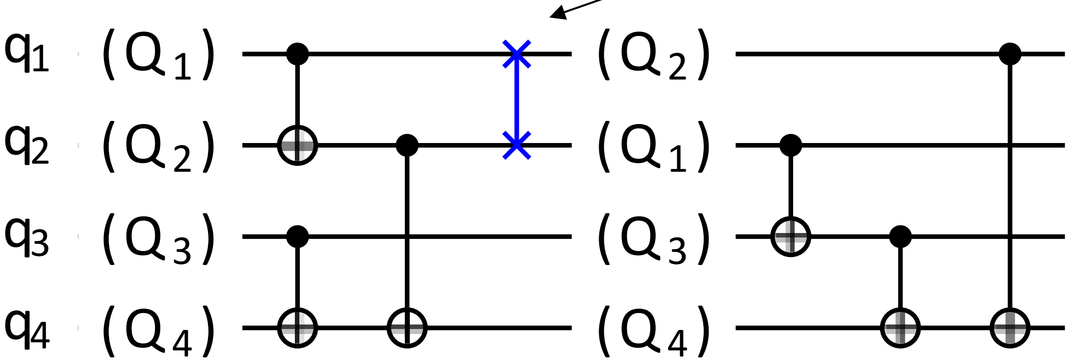
Qubit Mapping

- An Example

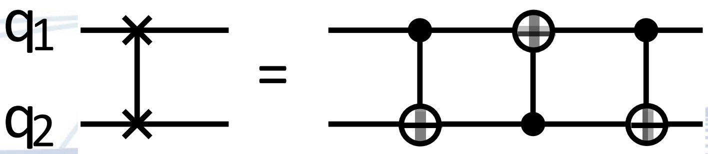


4-qubit chip

Insert additional gate SWAP: exchange the mapping



4-qubit circuit

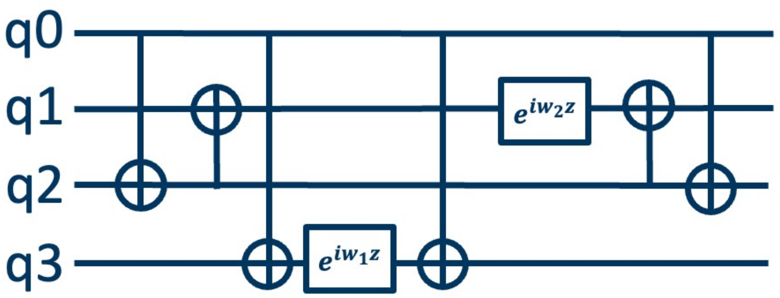


1 SWAP = 3 CNOT

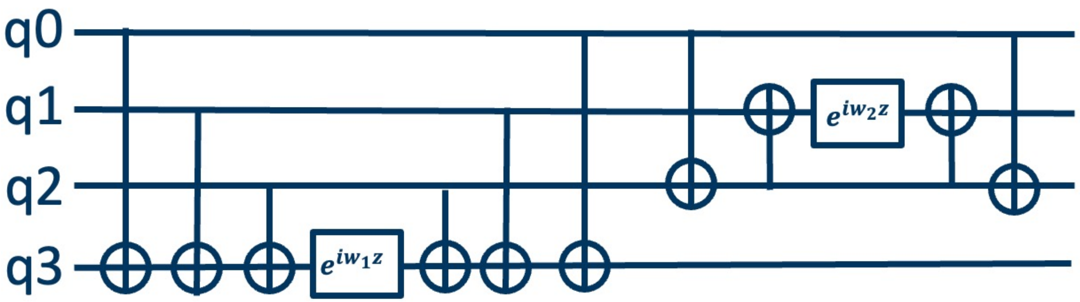
Each additional SWAP leads to more noise

Quantum Compiler Optimization

- Find some circuit identities
- Select the best one according to some metrics (e.g., # gate, # depth) and constraints (e.g., sparse connection on hardware)



Gate: 6 + 2; Depth: 6

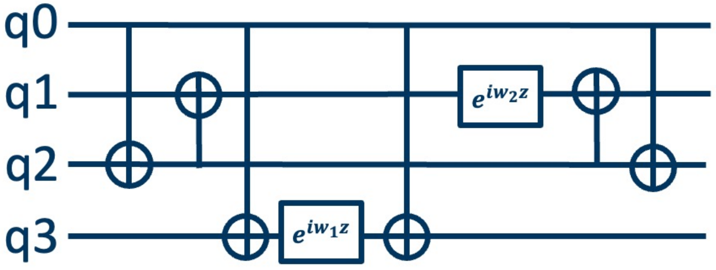


Gate: 10 + 2; Depth: 12

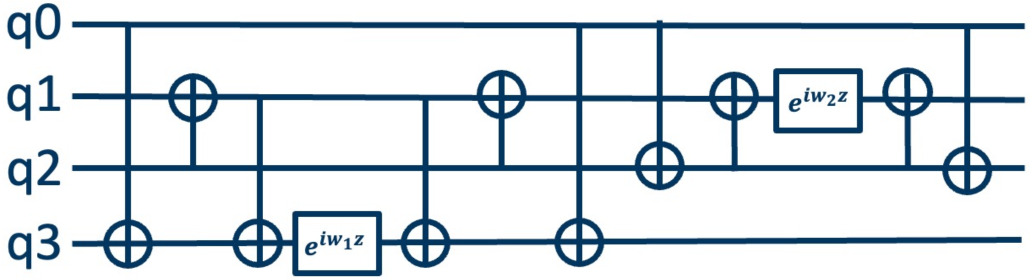
DESIGN AUTOMATION CONFERENCE

Challenge

- How can we find large-scale quantum circuit identities efficiently?



?=



❖ Calculate their matrix representations and check the equivalence?
 * Scalability issue: Matrix size of 2^n x 2^n and/or huge combinatorial search space.

❖ Limit our compiler optimization scope: peephole optimization, local swap insertions in qubit mapping, etc.
 * Missing large-scope optimizations.

Paulihedral: A Generalized Block-Wise Compiler Optimization Framework for Quantum Simulation Kernels

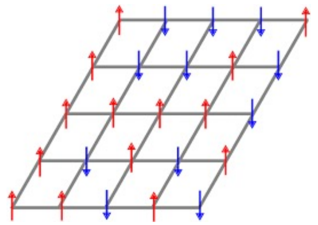
Gushu Li, Anbang Wu, Yunong Shi, Ali Javadi-Abhari, Yufei Ding, Yuan Xie

ASPLOS 2022

Opportunities at High-Level

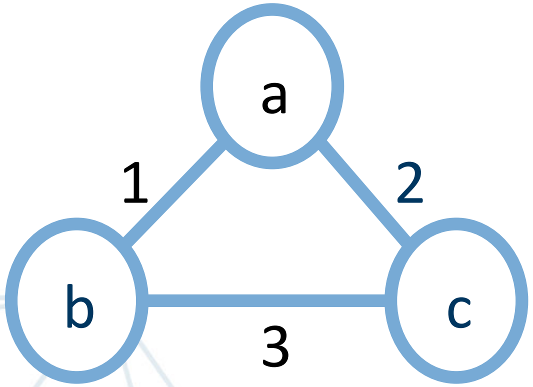
- More abstract compact form? Yes
- Simulation is widely used in quantum algorithm design

Simulation



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

Graph Cut



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

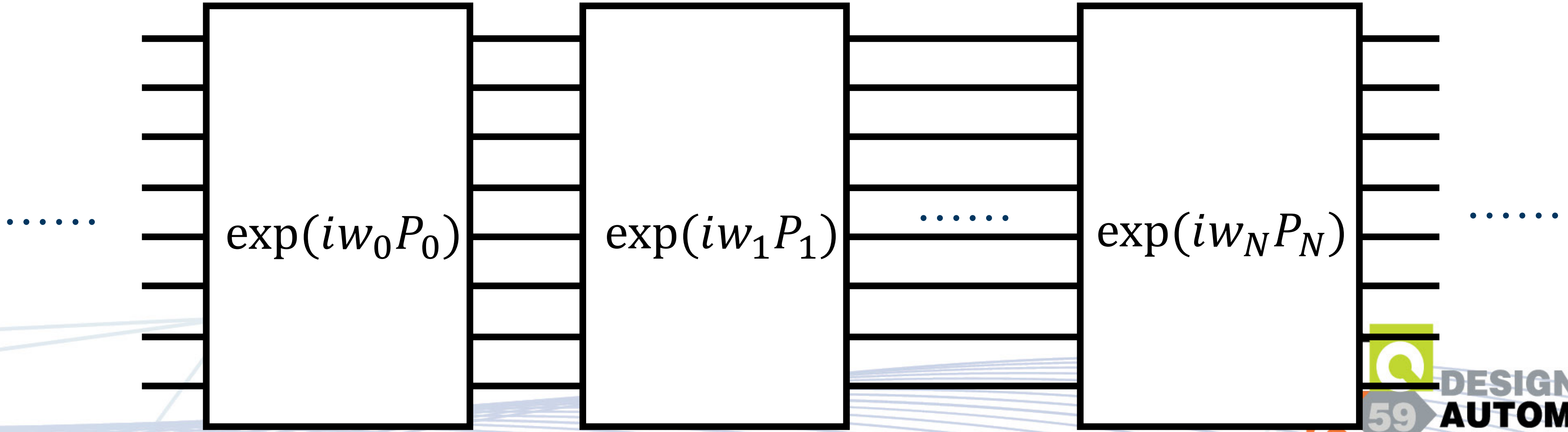
And many more

Quantum Simulation Kernel

- A widely-used subroutine

$$\exp(iHt)$$

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



High-Level IR: Pauli IR

(P, w) Basic unit of our IR: a pair of Pauli string **P** and a real number **w.**

- Pauli string **P** is just **Kronecker product** of 1-qubit Pauli matrices (I, X, Y, Z).

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \quad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \quad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

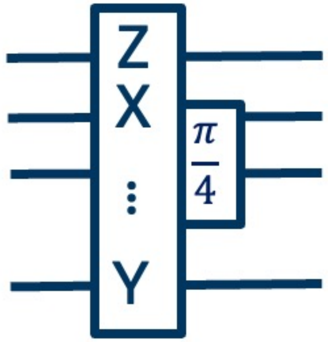
A size-**n** P can concisely express a $\mathbf{2^n \times 2^n}$ matrix

- Examples of 4-qubit Pauli string: $X_3Y_2Z_1I_0$, $Z_3Z_2Z_1Z_0$, $X_3Y_2Y_1X_0$
- **Active qubits** for a Pauli string: qubits with a non-I Pauli matrix.

High-Level IR: Pauli IR

(P, w) denotes a **2^n x 2^n unitary gate** $\exp(iwP)$.

Example: $(ZX\ldots Y, \frac{\pi}{4})$ → $\exp(i\frac{\pi}{4} ZX\ldots Y)$ →



Pauli IR (kernel)

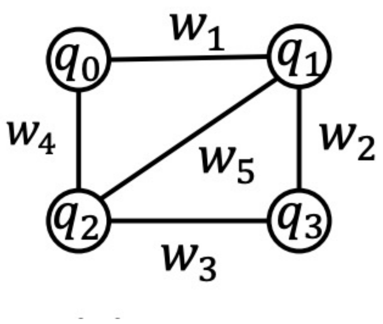
Exponential form

Circuit form

Universal in terms of unitary gates

Compile to Pauli IR

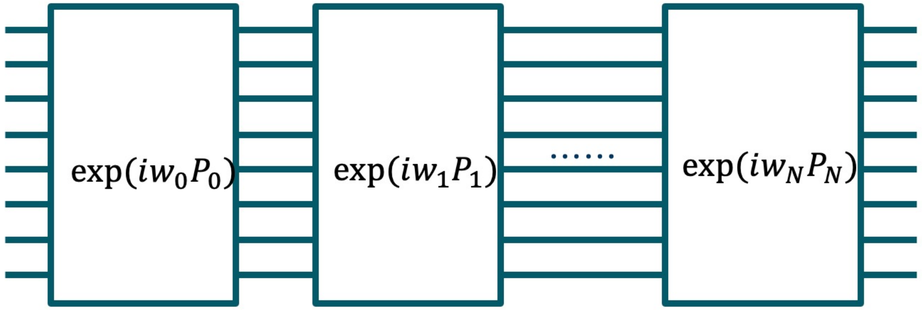
- **Great news:** It is already there, at least for many quantum algorithm design.
- (e.g., QAOA, VQE, and many other quantum simulation algorithms like UCCSD).



$$\exp(iw_1 I_3 I_2 Z_1 Z_0 \gamma)$$
$$\exp(iw_2 Z_3 I_2 Z_1 I_0 \gamma)$$
$$\exp(iw_3 Z_3 Z_2 I_1 I_0 \gamma)$$
$$\exp(iw_4 I_3 Z_2 I_1 Z_0 \gamma)$$
$$\exp(iw_5 I_3 Z_2 Z_1 I_0 \gamma)$$

QAQA Ansatz on graph Max-Cut

$$H = \sum_i w_i P_i$$

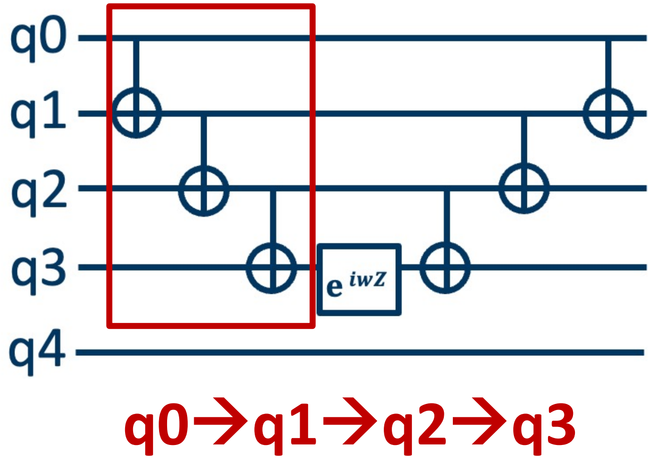


From Pauli IR to Gates

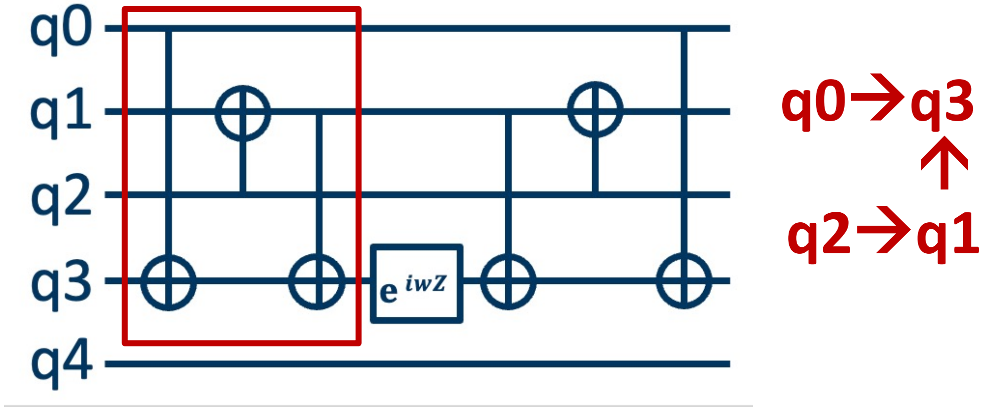
- **Very flexible compilation/synthesis**

$\exp(iwI_4Z_3Z_2Z_1Z_0)$

❖ if **z-basis parity on q3, q2, q1, q0** = 0, apply a global phase $\exp(iw)$; otherwise, apply $\exp(-iw)$.



CNOT tree for parity check



It thus could generate many great **circuit identities** with the same (P, w).

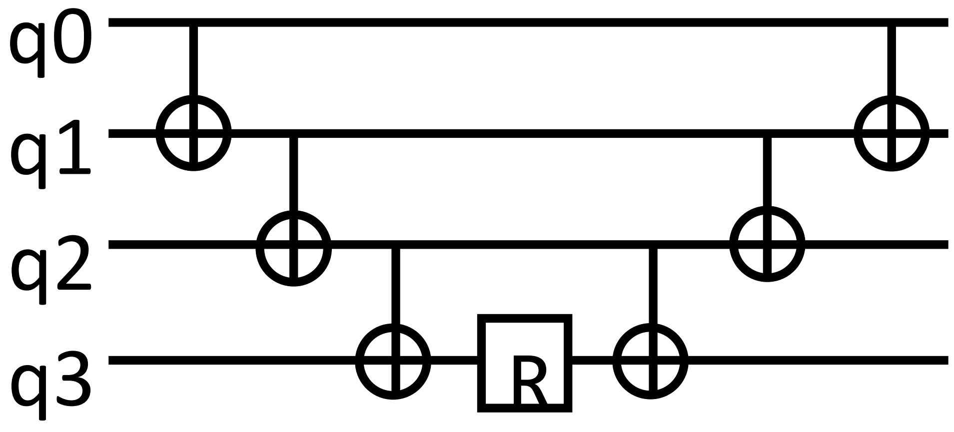
Example: Qubit Mapping

- How can Paulihedral change the mapping/SWAP insertion?

Conventional Compilation

$$\exp(iwZ_3Z_2Z_1Z_0)$$

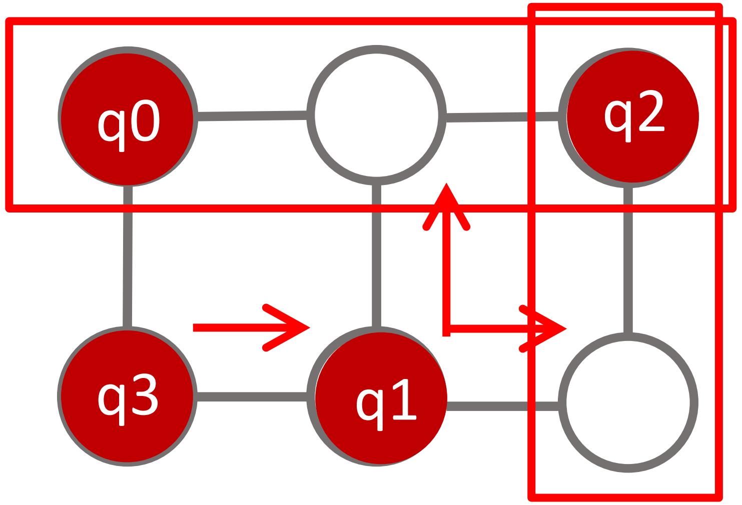
- Find SWAP in gate sequence



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

→

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



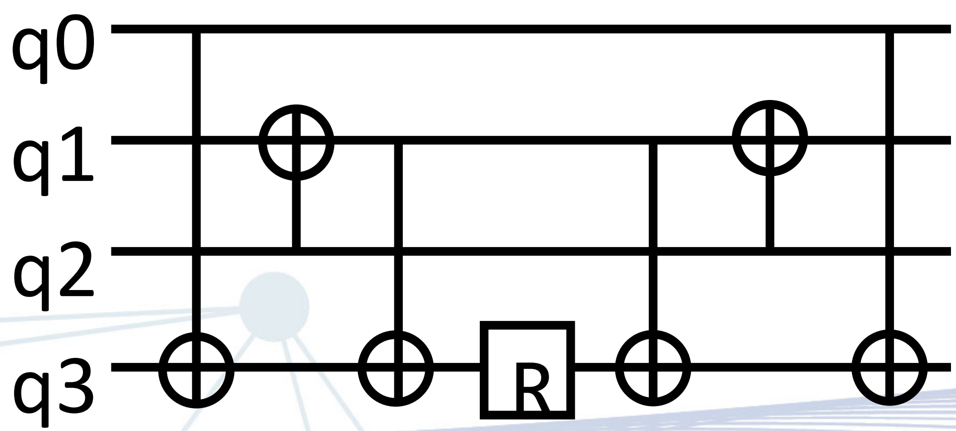
Paulihedral Compilation

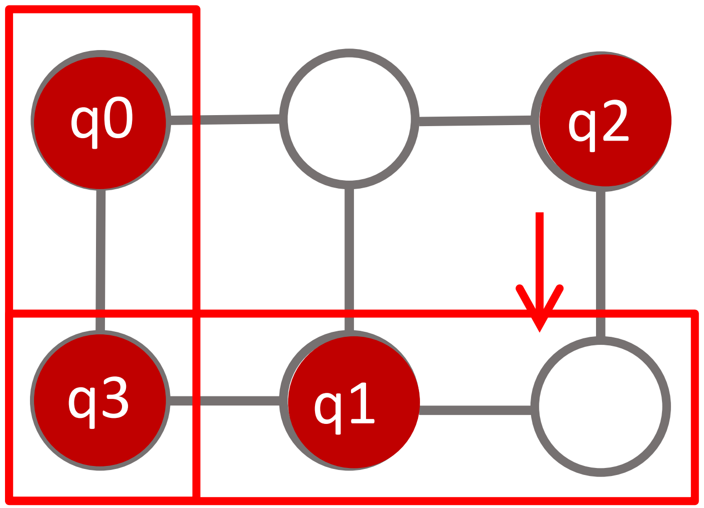
- Leverage high-level information

$$\exp(iwZ_3Z_2Z_1Z_0)$$

$$\exp(iwZ_3Z_2Z_1Z_0) \longrightarrow$$

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

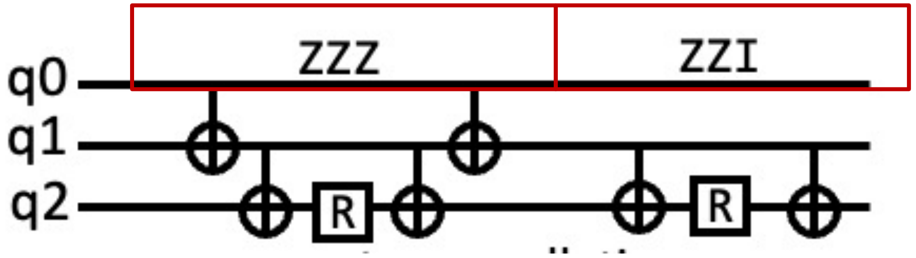




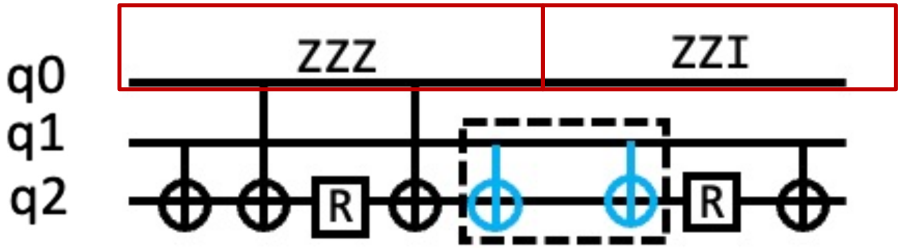
Find a tree embedding, then generate the CNOT tree

More Gate Cancellation

- **Find global gate cancellation among Pauli strings**
  - Circuit synthesis for $\exp(iZ_2Z_1I_0) \cdot \exp(iZ_2Z_1Z_0)$



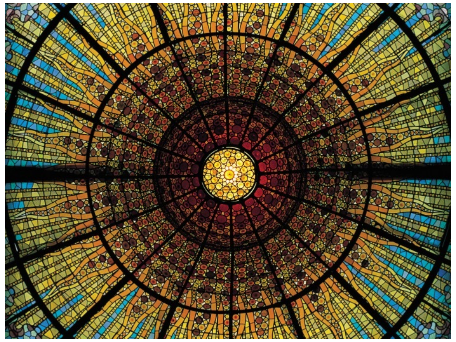
- ❖ Naïve synthesis for each separate kernel
- ❖ No gate cancellation.



- ❖ **Common subtree-centric gate synthesis** for large-scope gate cancellation.
- ❖ CNOTs cancel out with each other.

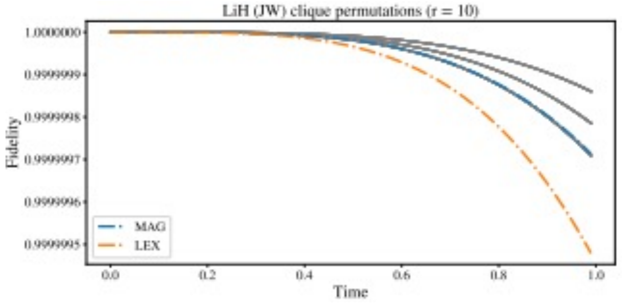
More active qubits overlapping between nearby Pauli IR kernels → more gate cancellation.

Moreover



[Livio, 2012]

symmetry preserving



[Gui et al. 2019]

error mitigation

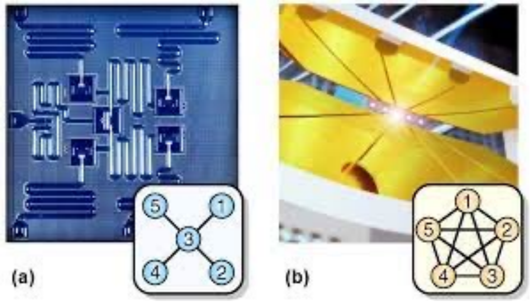
$$(a_2^\dagger a_0 - a_0^\dagger a_2) = \frac{i}{2}(X_2 Z_1 Y_0 - Y_2 Z_1 X_0)$$

$$(a_3^\dagger a_1 - a_1^\dagger a_3) = \frac{i}{2}(X_3 Z_2 Y_1 - Y_3 Z_2 X_1)$$

$$(a_3^\dagger a_2^\dagger a_1 a_0 - a_0^\dagger a_1^\dagger a_2 a_3) = \frac{i}{8}(X_3 Y_2 X_1 X_0 + Y_3 X_2 X_1 X_0 + Y_3 Y_2 Y_1 X_0 + Y_3 Y_2 X_1 Y_0 - X_3 X_2 Y_1 X_0 - X_3 X_2 X_1 Y_0 - Y_3 X_2 Y_1 Y_0 - X_3 Y_2 Y_1 Y_0).$$

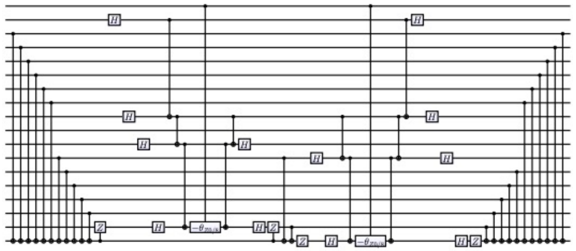
[McArdle et al. 2020]

parameter sharing



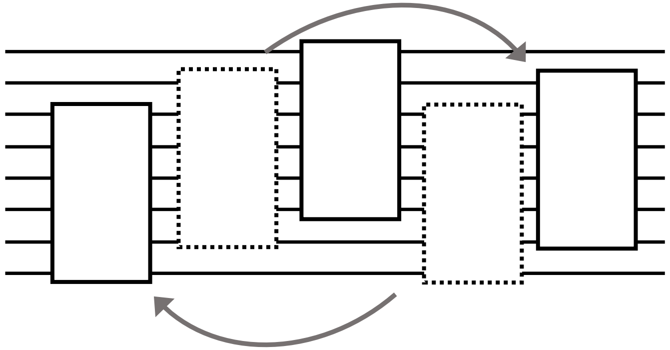
[Linke et al. 2017]

different backends



[Hastings et al. 2015]

more gate cancellation



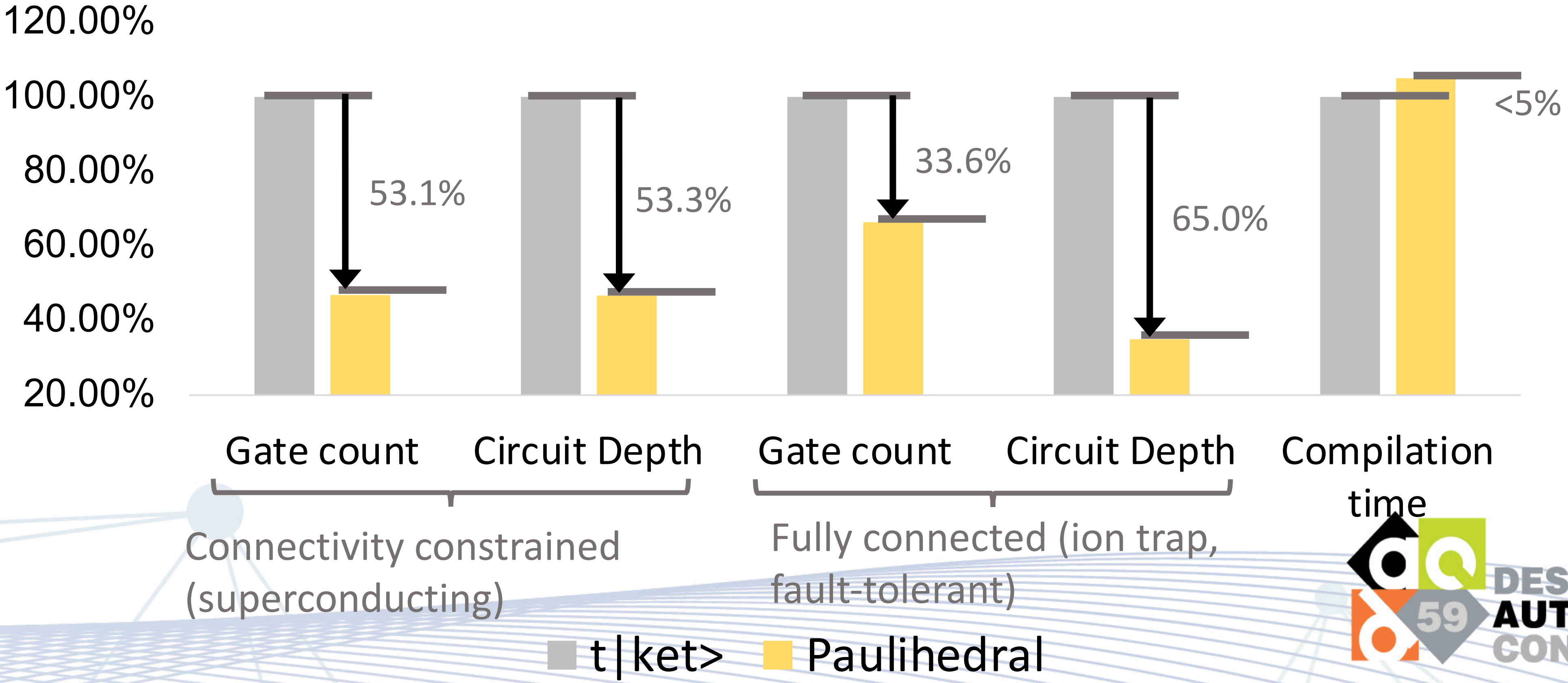
large-scope scheduling

Please see paper for details

DESIGN AUTOMATION CONFERENCE 59

Evaluation

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



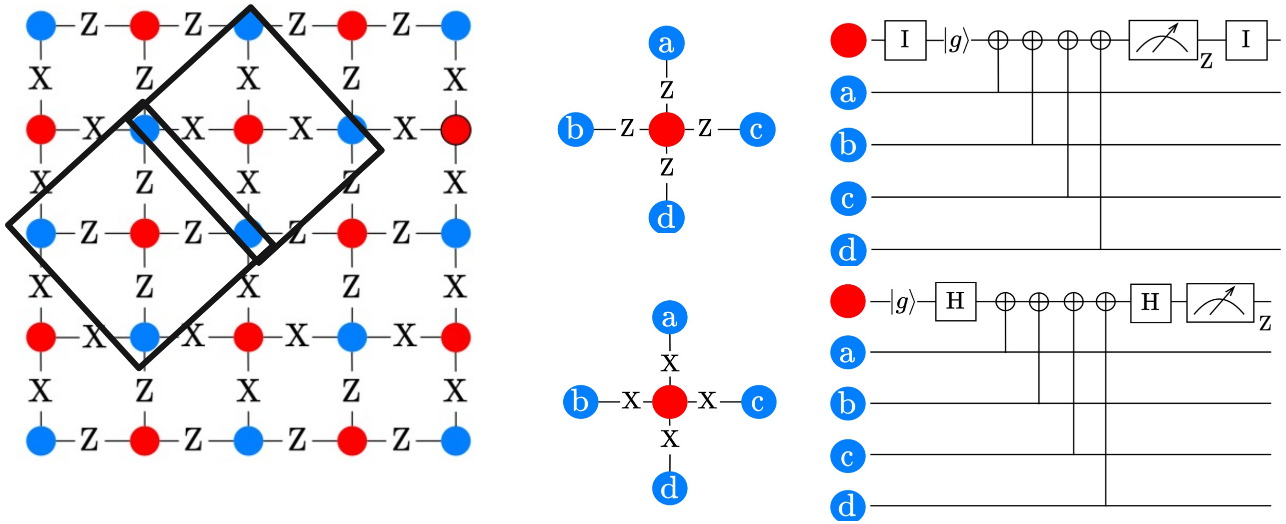
A Synthesis Framework for Stitching Surface Code with Superconducting Quantum Devices

Anbang Wu, Gushu Li, Hezi Zhang, Gian Giacomo Guerreschi, Yufei Ding, Yuan Xie

ISCA 2022

QEC Program: Surface Code

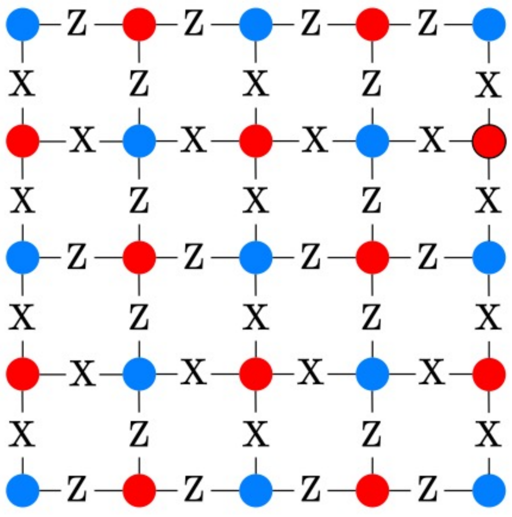
- Surface code: one of the best QEC in terms of error correction capabilities (up to about 1% error).



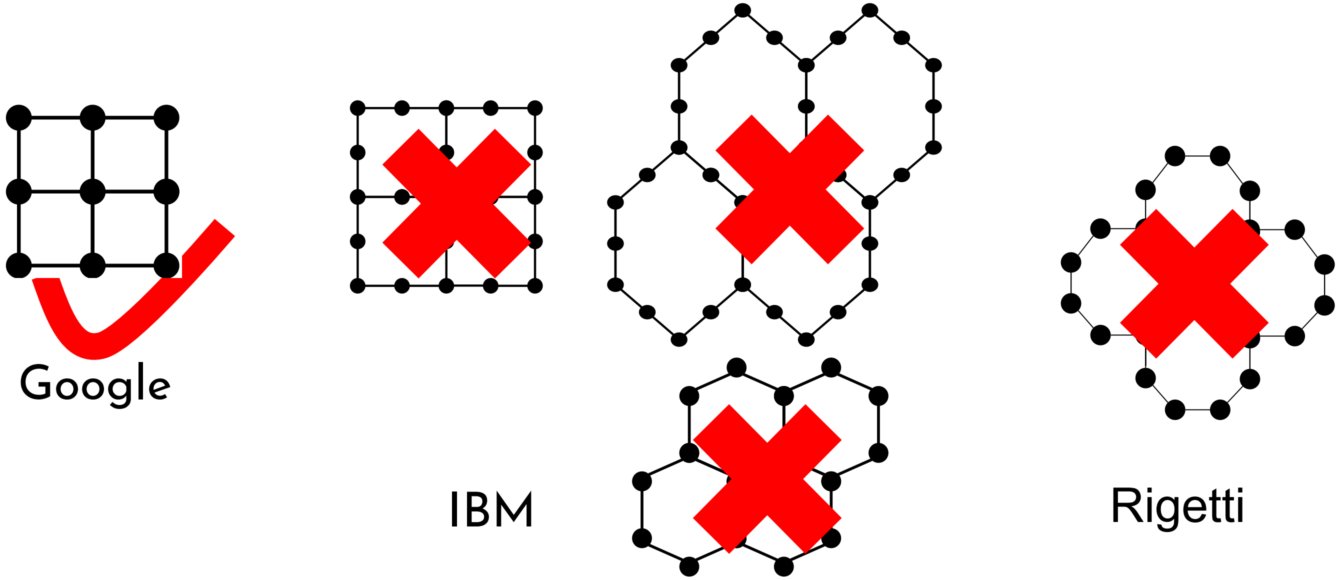
Circuits can be perfectly mapped to the hardware (coupling graph) on the left side.

- 2-D lattice qubit structure: long-range entanglement to protect the logic qubit from local noises.
  - Data qubits (blue): encode the correct subspace for logical operations.
  - Syndrome qubit (red): ensure data qubits are working collaboratively by checking their X, Z parity.

Mismatch between Surface Code and Sparse Architectures



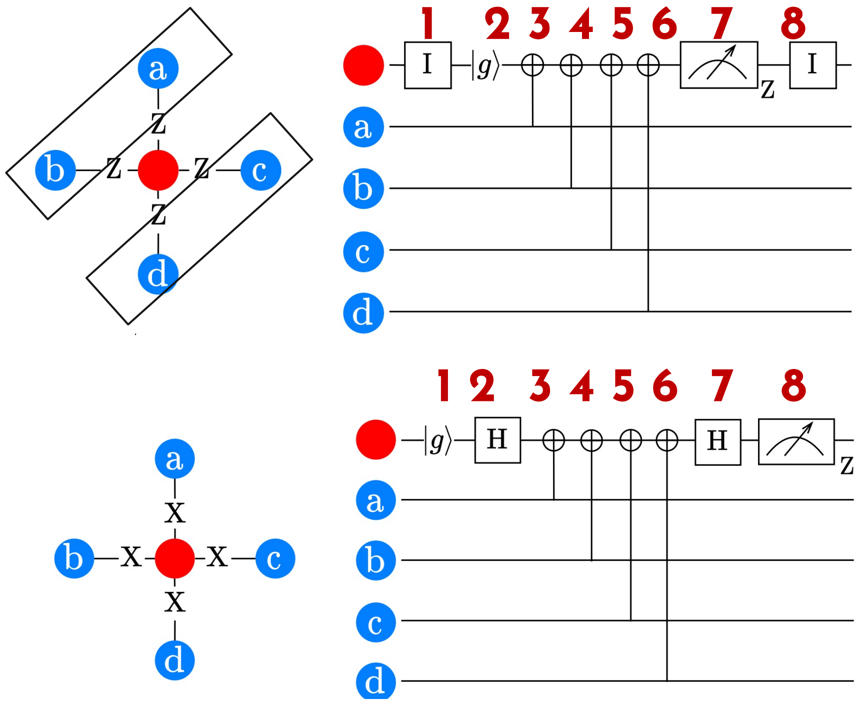
Surface Code

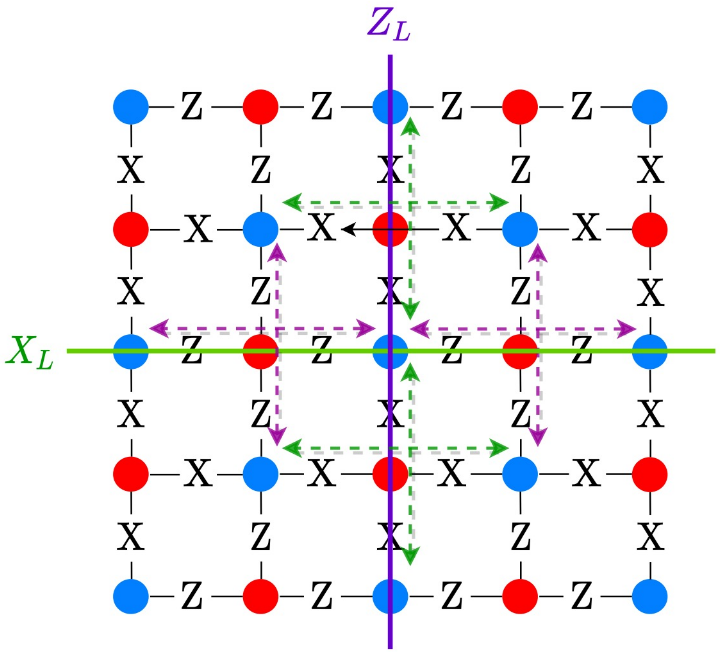


Some recent study designed new QEC codes tailored for these **sparse** architecture.

- ❖ Can the “mismatch” be mitigated by compiler optimization?

What is Special about QEC program?





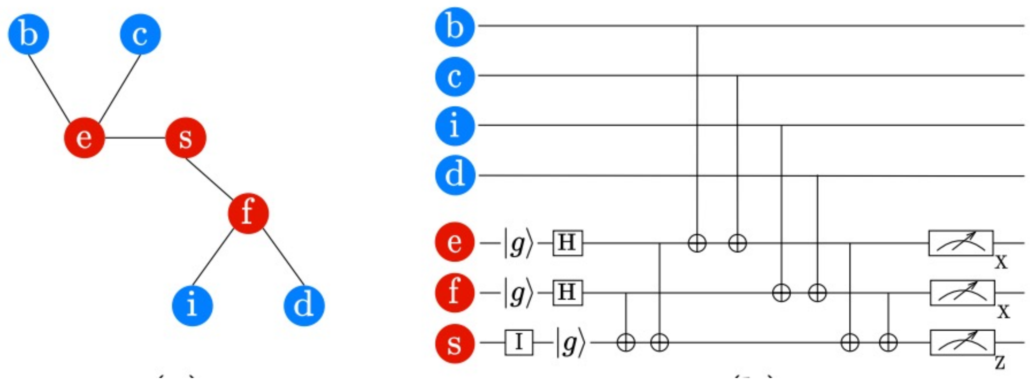
1) **4-degree qubit**: each syndrome qubit (red) measures the **parity** of **4** data qubits (blue).

2) **Fixed data qubit Layout:** moving data qubits would invalidate those high-level logical operations.

3) **Stabilizer measurement scheduling:** zigzag (instead of clockwise) gives maximum parallelism.

Our QEC Compiler

- How to resolve the mismatch problem from a compiler's perspective? If so, to what extent?



Bridge tree to encode a **"logic" syndrome qubit,** and use it to replace a 4-dgree node. [L. Lao et. al. PRA2020]

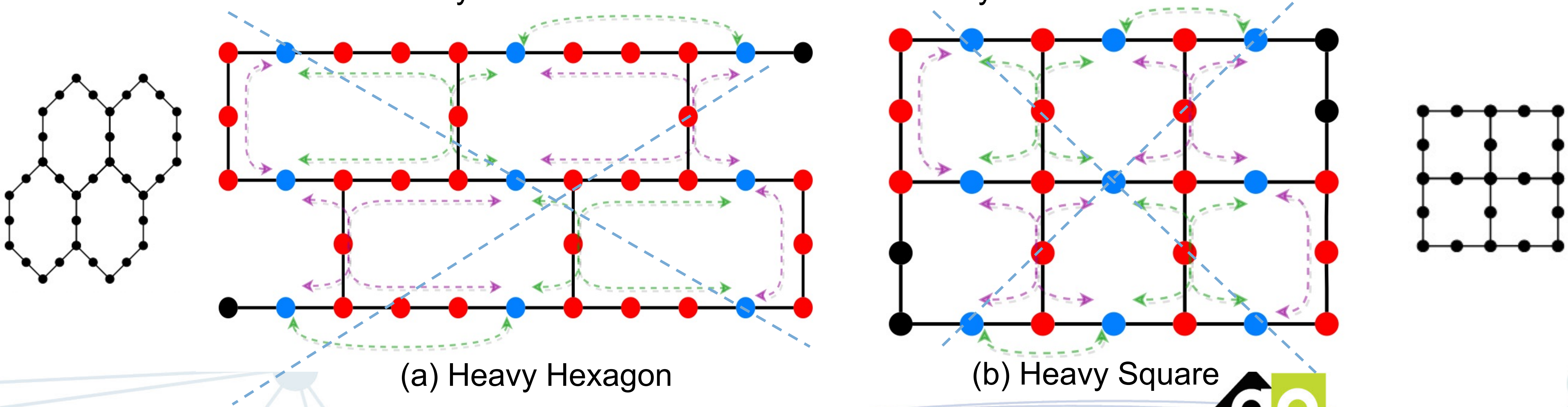
❖ **Three key submodules** to resolve the surface code mismatch problem:

1. How are data qubits allocated?
2. How to find "small and local " bridge trees?
3. How to schedule stabilizer measurement circuits?

Our QEC Compiler

- We build the first automated framework for compiling surface code to sparse quantum devices with a modularized design.

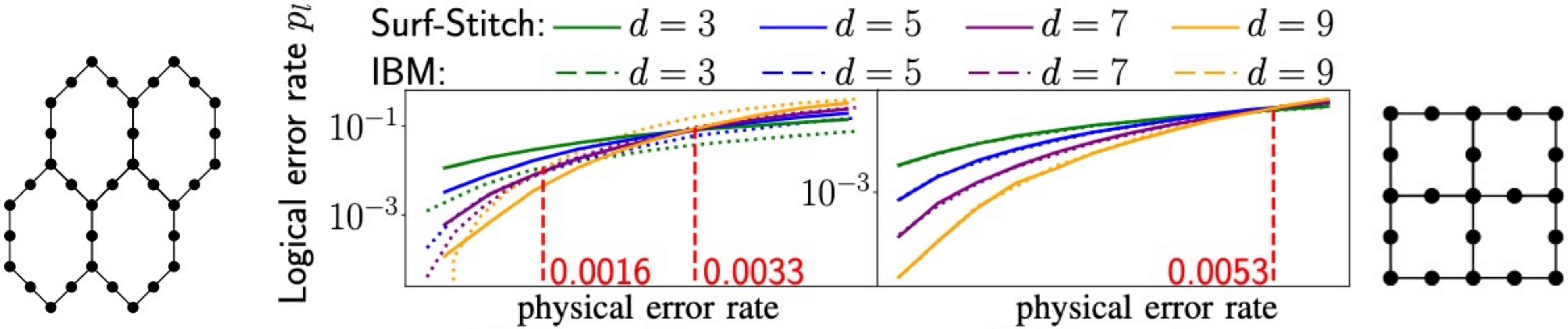
➢ The synthesized distance-3 surface code by our framework



We could work with different architectures + missing links/qubits.

Performance of Our QEC Compiler

- The *error threshold* of the compiled surface code is comparable or even better than IBM's manually designed QEC codes tailored for the sparse architectures.



(a) Heavy Hexagon
(b) Heavy Square

More results in our paper.

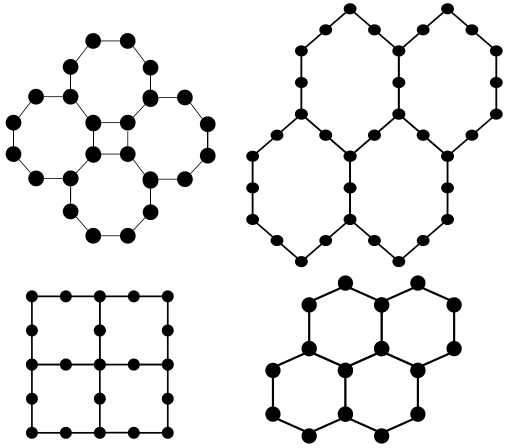
Our Vision for Future QEC Development

- General QEC design and mapping could be formulated as a compiler problem.

Different stabilizer codes,
subsystem codes,
and their hybrid concatenation,
etc.
…

Automatic Mapping and Optimization Framework

Metrics:
Error Threshold?
Qubit Utilization?
Fabrication Yield?
….



Hardware

A compiler framework to automate the designs of hardware-aware QEC codes + its associated error decoder!!!

Q & A

- Thank you!