



HPCA 2025 Tutorial

Topic 4. QuFEM: Fast and Accurate Quantum Readout Calibration Using the Finite Element Method



JanusQ
Cloud

Speaker: Kaiwen Zhou

College of Computer Science and Technology
Zhejiang University (ZJU)

https://janusq.github.io/HPCA_2025_Tutorial/



Kaiwen Zhou

kaiwenzhou@zju.edu.cn

Kaiwen Zhou is a first-year PhD student at the College of Computer Science, Zhejiang University. He is interested in quantum computer architecture and high-performance computing. He is currently working on designing a QLDPC decoding accelerator.

Outline of Presentation

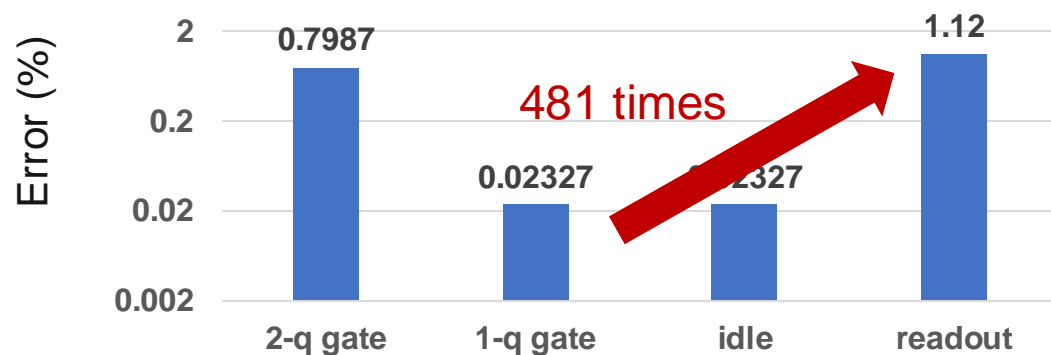


- **Background and challenges**
- Overview of QuFEM
- QuFEM characterization and calibration
- Experiment
- API of QuFEM

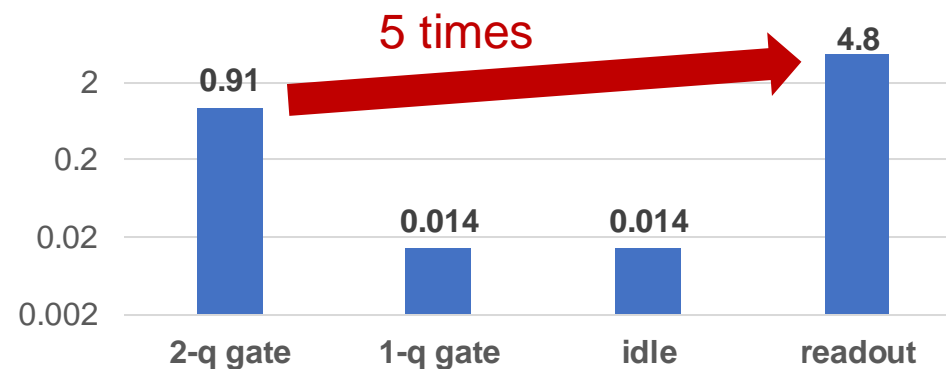
Quantum readout is an operation to **read the information from quantum bits to classical bits**.



Readout error is significant on current quantum hardware.



Noise on 127-qubit IBM Sherbrooke quantum device



Noise on 10-qubit Tianmu quantum device

Source of readout error

from 1 to 0



from 0 to 1

Excitation error

from 1/0 to 0/1

Crosstalk

e.g. Das, et al. JigSaw: Boosting Fidelity of NISQ Programs via Measurement Subsetting. MICRO 2021

Readout errors vary in different combinations of measured qubits due to crosstalk.

Crosstalk has different frequencies when Q2 is measured 0, 1 or not measured

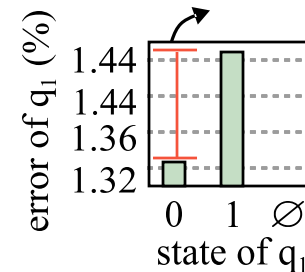
Example of state-dependent and readoutdependent noises on the IBMQ Perth quantum device.



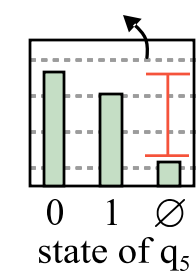
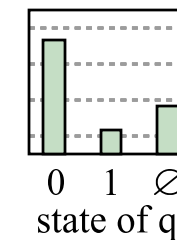
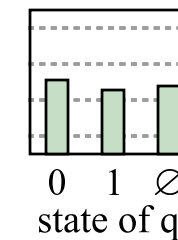
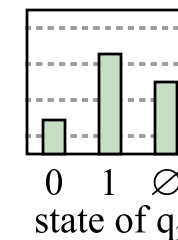
Similar to entanglement



the error of q_1 depends on its own operation.



the error also depends on the readout of q_5 .

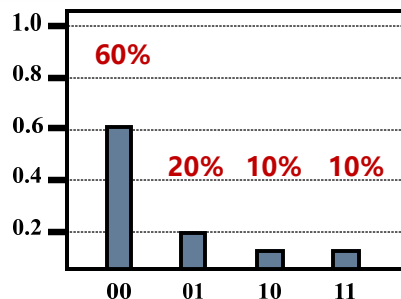


The readout output of qubits has correlations similar to the entanglement, making the calibration difficult.

Basic Matrix-based readout calibration



Ideal readout

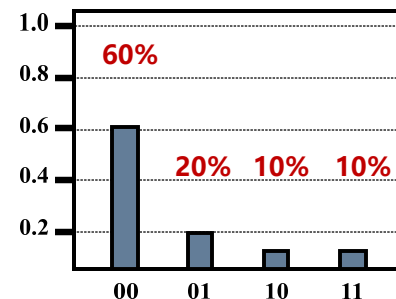


State vector

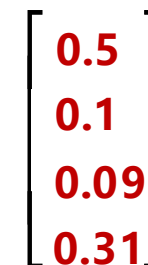


Ideal distribution
(ideal program output)

Readout with noise



State vector



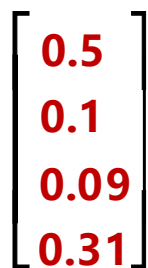
Noisy distribution
(noisy program output)

Matrix-based readout error calibration



Noise
matrix

-1

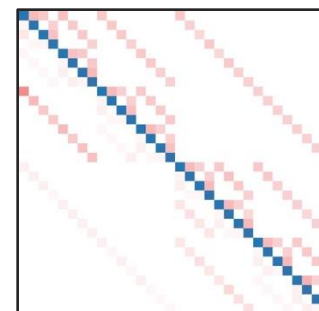


Noisy
distribution

$=$



Calibrated
distribution



Calibration matrix of
a 5-qubit readout

The size exponentially
increases!

$2^5 \times 2^5$

Step 1. Matrix characterization

Prepares qubits to different basis states and apply measurement.



Fill in a noise matrix.

$$M = \begin{bmatrix} 0.6 & 0.1 & 0.2 & 0.1 \\ 0 & 0.7 & 0.2 & 0.1 \\ 0.2 & 0.2 & 0.6 & 0 \\ 0 & 0.1 & 0.1 & 0.8 \end{bmatrix}$$

Inverse the noise matrix

$$M^{-1} = \begin{bmatrix} \text{blue} & \text{red} & \text{red} & \text{red} \\ \text{red} & \text{blue} & \text{red} & \text{red} \\ \text{red} & \text{red} & \text{blue} & \text{red} \\ \text{red} & \text{red} & \text{red} & \text{blue} \end{bmatrix}$$

Calibration matrix

Basic Matrix-based Readout Calibration



Step 1. Matrix characterization

Prepares qubits to different basis states and apply measurement.



Fill in a noise matrix.

$$M = \begin{bmatrix} 0.6 & 0.1 & 0.2 & 0.1 \\ 0 & 0.7 & 0.2 & 0.1 \\ 0.2 & 0.2 & 0.6 & 0 \\ 0 & 0.1 & 0.1 & 0.8 \end{bmatrix}$$

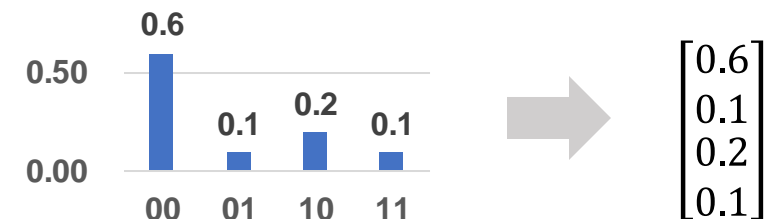
Inverse the noise matrix

$$M^{-1} =$$

Calibration matrix

Step 2. Calibration for any input

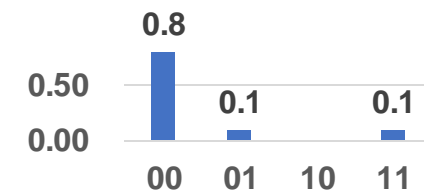
Represent the measured distribution as a vector.



Apply matrix-vector multiplication.

$$\begin{bmatrix} 0.6 & 0.1 & 0.2 & 0.1 \\ 0 & 0.7 & 0.2 & 0.1 \\ 0.2 & 0.2 & 0.6 & 0 \\ 0 & 0.1 & 0.1 & 0.8 \end{bmatrix}^{-1} \cdot \begin{bmatrix} 0.6 \\ 0.1 \\ 0.2 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.1 \\ 0 \\ 0.1 \end{bmatrix}$$

$$M^{-1} \cdot P_{\text{read}} = P_{\text{calibrated}}$$



Step 1. Matrix characterization

Prepares qubits to different basis states and apply measurement.

2^N circuits are executed to measure qubits on all basis states.

Fill in a noise matrix.

The size of the noise matrix is $2^N \times 2^N$.

Inverse the noise matrix

Calculating the inverse has $\mathcal{O}(4^N)$ complexity.

Step 2. Calibration for any input

Represent the measured distribution as a vector.

The transformation has linear complexity.

Apply matrix-vector multiplication.

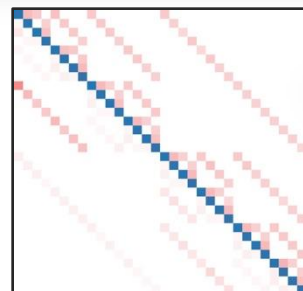
The multiplication has $\mathcal{O}(4^N)$ complexity.

8.8 TB and 10 hours for a 32-qubit calibration on a server with AMD EPYC 2.25GHz 64-core CPUs

Limitations of Current Methods

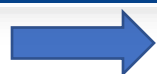


IBU (Google Science 2021) Realizing topologically ordered states on a quantum processor.



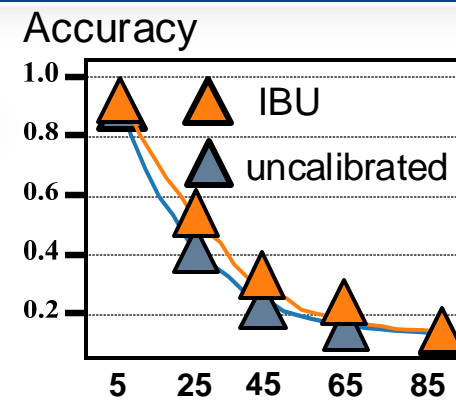
Real calibration matrix

Crosstalk makes the matrix not simple tensor-product result.



Single-qubit matrix

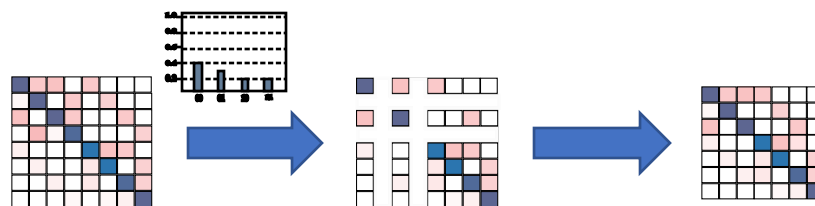
Use tensor product of a series of single-qubit meta-matrices



Fail to calibrate on 80-qubit readout output

Fast but not accurate: ignore the qubit interactions.

M3 (IBM PRA 2021): Scalable mitigation of measurement errors on quantum computers

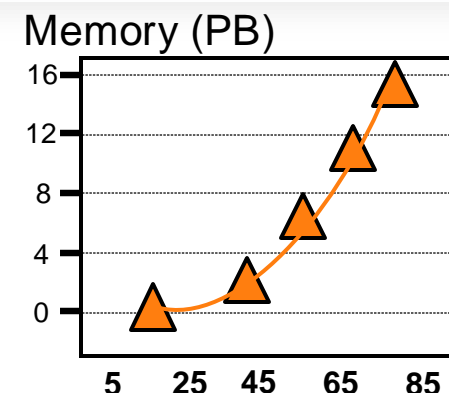


Before pruning

Pruning based on
program output

After pruning

Use a sparsity-aware method to prune on the matrix under a threshold of Hamming distance



Require 16PB to calibrate a 85-qubit result.
(4 times the Fugaku supercomputer)

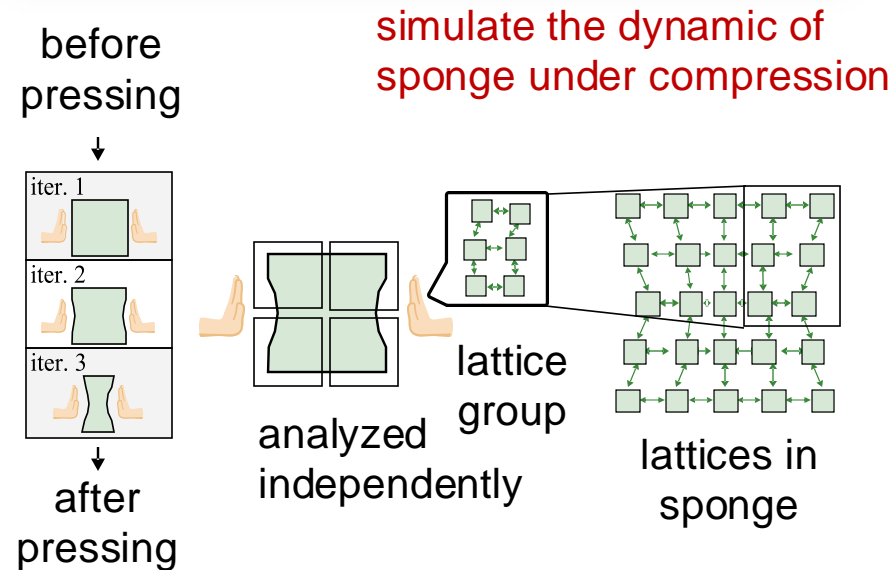
Accurate but not fast: many matrix elements cannot be ignored

Outline of Presentation



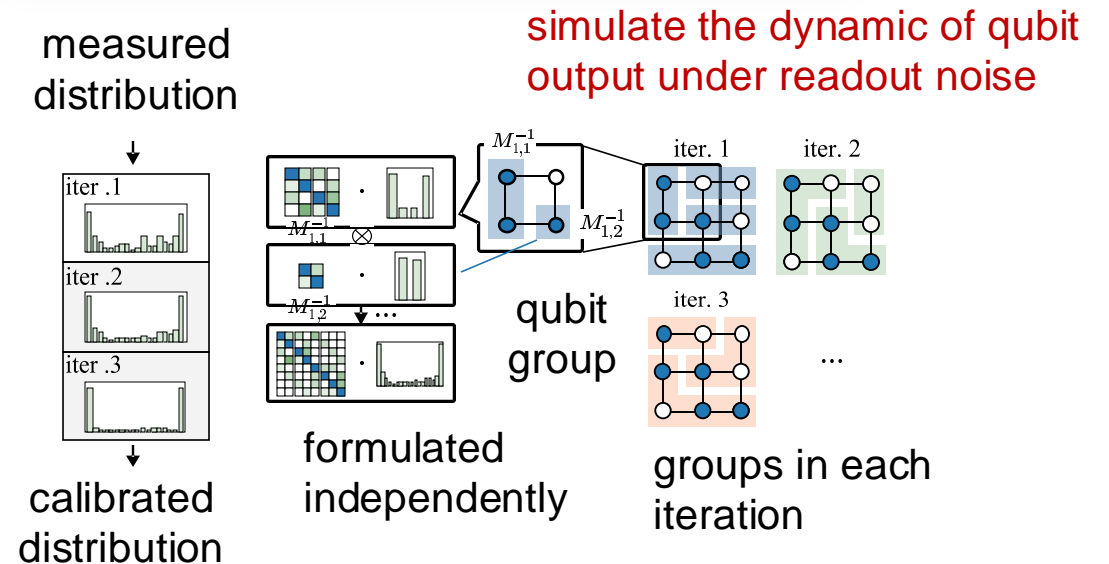
- Background and challenges
- **Overview of QuFEM**
- QuFEM characterization and calibration
- Experiment
- API of QuFEM

Classical Finite Element Method



- ① partitions the sponge into **lattices**
- ② analyzes the state of each lattice **independently**
- ③ simulate the **interaction**
- ④ update the state of **sponge**

Quantum Finite Element Method



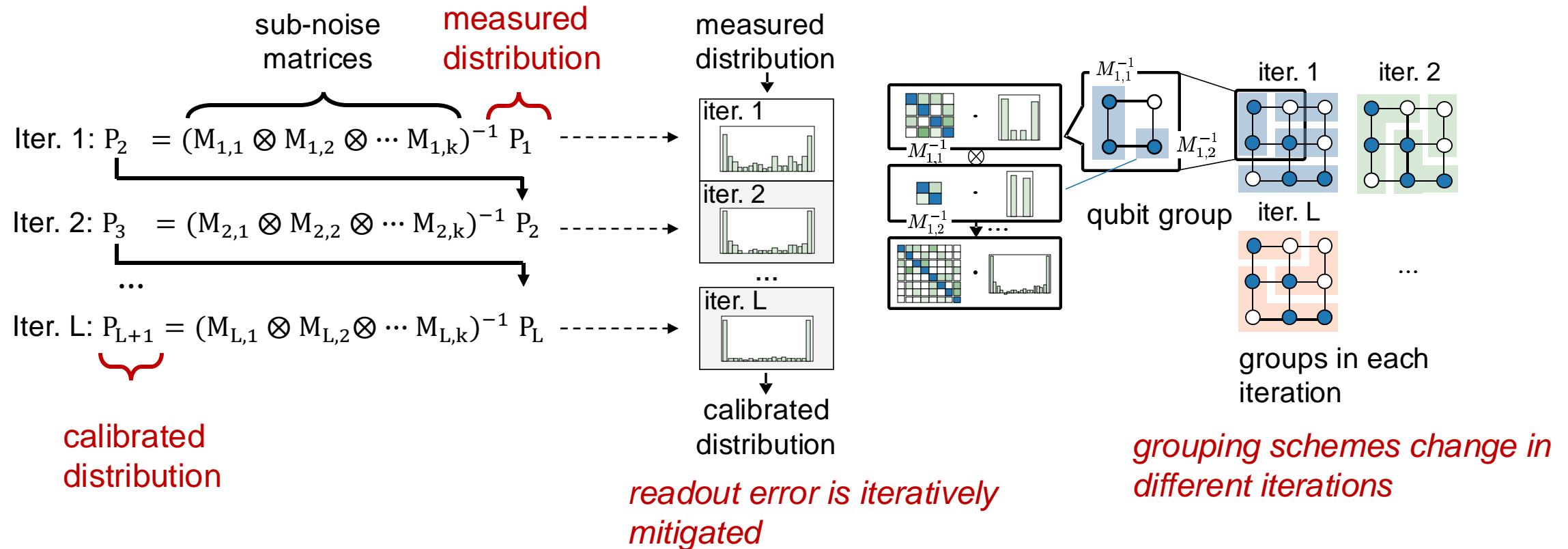
- ① partitions qubits into **groups**
- ② analyze the noise in each group **independently**
- ③ simulate the **interaction**
- ④ update the calibration result of **qubits**

A **divide-and-conquer strategy** to calibrate measured distribution

Calibration formulation



QuFEM reformulates the calibration as an iterative process with a series of sub-noise matrices.



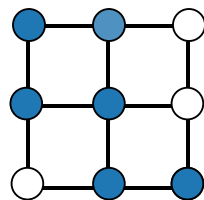
- Reason for fast: adopt finite element method
- Reason for accurate: dynamically generate noise matrices for different measured qubits

An example

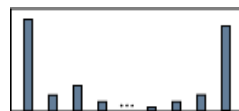


Input :

measured qubits

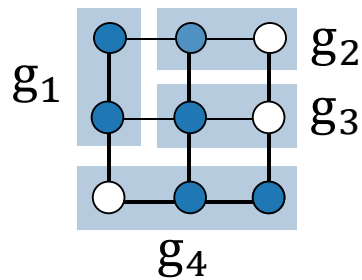


measured distribution



Iteration 1 :

grouping scheme

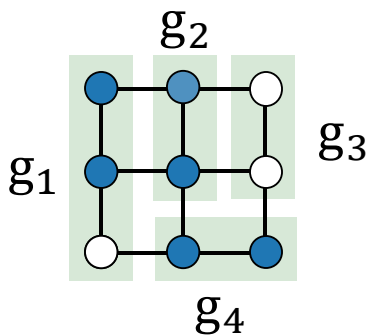


formulation

$$P_2 = (M_{1,1} \otimes M_{1,2} \otimes M_{1,3} \otimes M_{1,3})^{-1} \cdot P_1$$

Iteration 2 :

grouping scheme



formulation

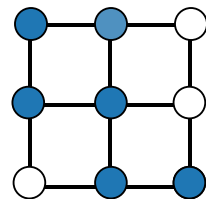
$$P_3 = (M_{2,1} \otimes M_{2,3} \otimes M_{2,4})^{-1} \cdot P_2$$

An example

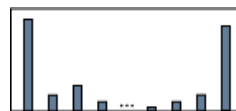


Input :

measured qubits



measured distribution

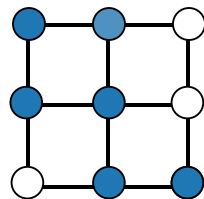


An example

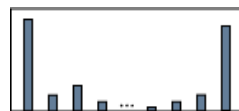


Input :

measured qubits

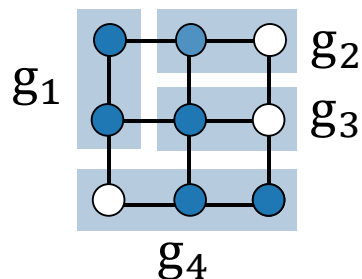


measured distribution



Iteration 1 :

grouping scheme



formulation

$$P_2 = (M_{1,1} \otimes M_{1,2} \otimes M_{1,3} \otimes M_{1,3})^{-1} \cdot P_1$$

Matrices are generated according to the measured qubits.

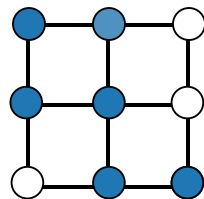
Since crosstalk varies in different combinations of measured qubits

An example

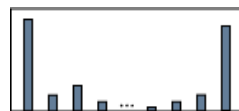


Input :

measured qubits

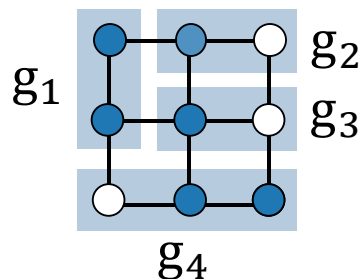


measured distribution



Iteration 1 :

grouping scheme

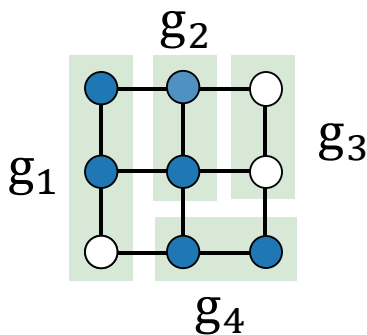


formulation

$$P_2 = \left(M_{1,1} \otimes M_{1,2} \otimes M_{1,3} \otimes M_{1,3} \right)^{-1} \cdot P_1$$

Iteration 2 :

grouping scheme



formulation

$$P_3 = \left(M_{2,1} \otimes M_{2,3} \otimes M_{2,4} \right)^{-1} \cdot P_2$$

Outline of Presentation

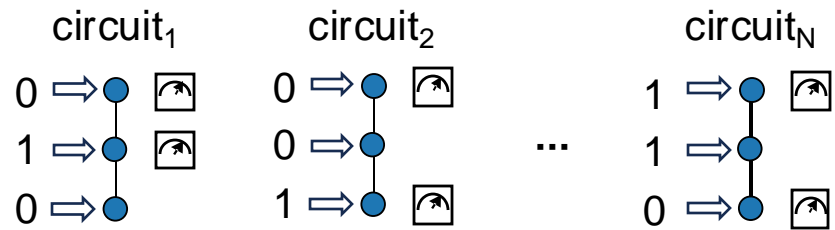


- Background and challenges
- Overview of QuFEM
- **QuFEM characterization and calibration**
- Experiment
- API of QuFEM

Technique 1: determine the grouping scheme

Data collection

Run benchmarking circuits.



Possible states of a qubit in a benchmarking circuit:

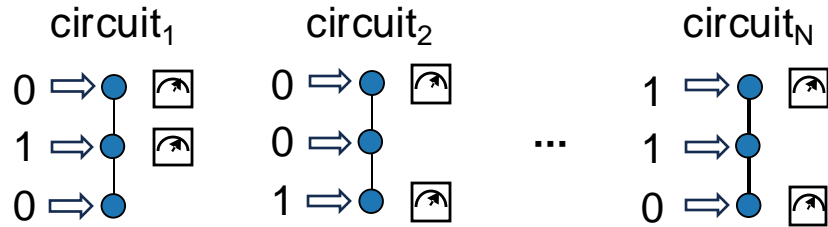
- 0: qubit is set 0 and measured
- 1: qubit is set 1 and measured
- 2: qubit is set 0 or 1 and not measured

Not all qubits are measured to maximize the variety.

Technique 1: determine the grouping scheme

Data collection

Run benchmarking circuits.



Possible states of a qubit in a benchmarking circuit:

- 1: qubit is set 0 and measured
- 2: qubit is set 1 and measured
- 3: qubit is set 0 or 1 and not measured

Not all qubits are measured to maximize the variety.

Qubit partition

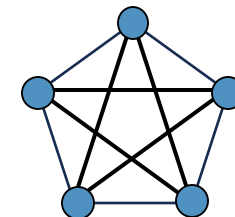
Characterize the **interaction** from one qubit to another qubit under different states:

$$\text{interact}(q_i. \text{state} = x \rightarrow q_j. \text{state} = x) \\ = \underbrace{P(q_j. \text{error} = 1 \mid C1, C2)}_{\text{error rate of } q_j \text{ under } C1, C2} - \underbrace{P(q_j. \text{error} = 1 \mid C2)}_{\text{average error rate of } q_j}$$

C1: $q_i. \text{state} = x$,

C2: $q_j. \text{state} = y$

Construct weighted graph



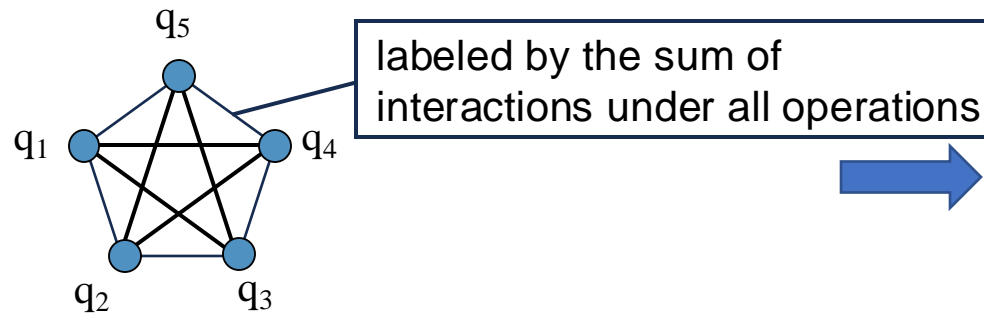
Deployment on Specific Quantum Device



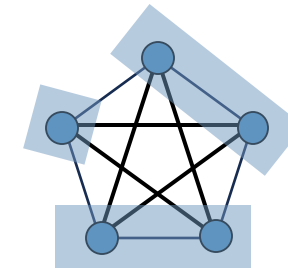
Technique 1: determine the grouping scheme

Qubit partition

Construct a **weighted qubit graph**:



Partitions with a **MAX-CUT solver**:



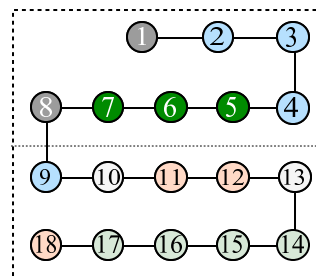
try to comprehensively capture the interactions between qubits

An Example

Prior knowledge of hardware helps grouping

Readout resonator 1

Readout resonator 2



18-qubit topology

group 1: 14 15 16 17
same readout resonator
group 2: 2 3 4 9
similar readout frequency
group 3: 1 8
overlapping frequency shift region

- demonstrated in the results from other quantum devices
- can be used as prior knowledge to facilitate the partition.

Technique 2: sub-noise matrix generation

Perform matrix-vector multiplication

$$\text{Iter. } i: P_{i+1} = (M_{i,1} \otimes M_{i,2} \otimes \cdots M_{i,k})^{-1} P_i$$

Matrix generation

Noise matrix formulates the transformation probability from the ideal state to measured state.

$$\begin{bmatrix} 0.6 & 0 & 0.1 & 0 \\ 0.1 & 0.7 & 0.2 & 0.1 \\ 0.2 & 0.2 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.8 \end{bmatrix} = \text{read state} \left\{ \begin{array}{c|cccc} & \text{set state} & & & \\ \hline & 00 & 01 & 10 & 11 \\ \hline 00 & 0.6 & 0 & 0.1 & 0 \\ 01 & 0.1 & 0.7 & 0.2 & 0.1 \\ 01 & 0.2 & 0.2 & 0.6 & 0.1 \\ 11 & 0.1 & 0.1 & 0.1 & 0.8 \end{array} \right.$$

Technique 2: sub-noise matrix generation

Perform matrix-vector multiplication

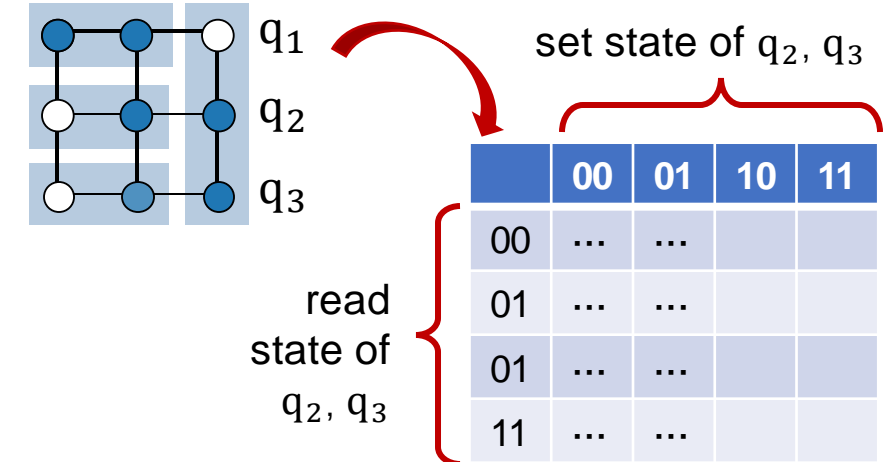
$$\text{Iter. } i: P_{i+1} = (M_{i,1} \otimes M_{i,2} \otimes \dots M_{i,k})^{-1} P_i$$

Matrix generation

Noise matrix formulates the transformation probability from the ideal state to measured state.

$$\begin{bmatrix} 0.6 & 0 & 0.1 & 0 \\ 0.1 & 0.7 & 0.2 & 0.1 \\ 0.2 & 0.2 & 0.6 & 0.1 \\ 0.1 & 0.1 & 0.1 & 0.8 \end{bmatrix} = \text{read state} \left\{ \begin{array}{c|cccc} & \text{set state} & & & \\ \hline & 00 & 01 & 10 & 11 \\ \hline 00 & 0.6 & 0 & 0.1 & 0 \\ 01 & 0.1 & 0.7 & 0.2 & 0.1 \\ 01 & 0.2 & 0.2 & 0.6 & 0.1 \\ 11 & 0.1 & 0.1 & 0.1 & 0.8 \end{array} \right.$$

Sub-noise matrices of QuFEM formulates the transformation probability of states inside the qubit groups.



$$M[x][y] = P(\{q_2, q_3\}. \text{read} = x | \{q_2, q_3\}. \text{set} = y, q_1 = 2)$$

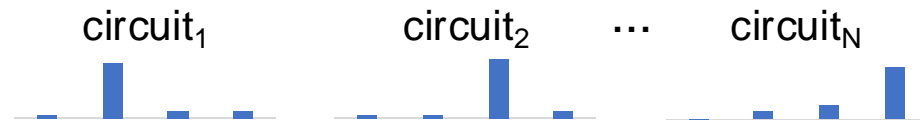
Transformation probability when q_1 is not measured

Put all together

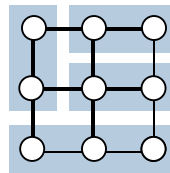


Characterization

Iteration 1. Run benchmarking circuits.



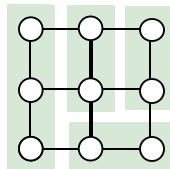
Iteration 1. Partition qubits.



Iteration 1. Calibrate.



Iteration 2. Partition qubits.



Iteration 2. Calibrate.

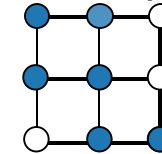


distributions, grouping
scheme of iteration 1.

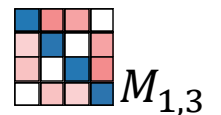
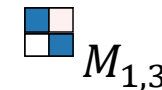
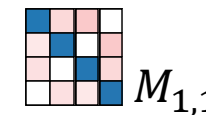
distributions, grouping
scheme of iteration 2.

Calibration

Input. *measured qubits* *measured distribution*



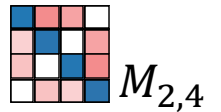
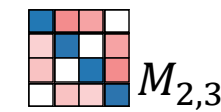
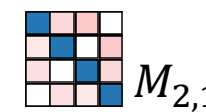
Iteration 1. Generate sub-noise matrices.



Iteration 1. Calibrate.



Iteration 2. Generate sub-noise matrices.

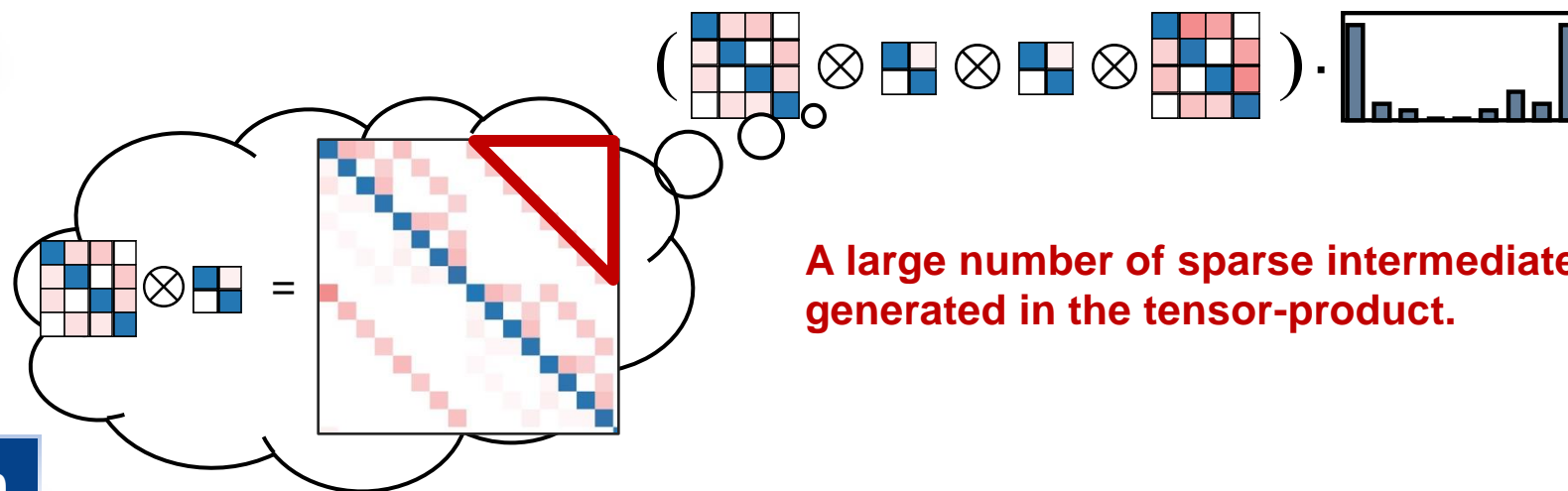


Iteration 2. Calibrate.



Output

Observation



Implementation

Use a key-value table to store sparse vector

$$(M_{2,1}^{-1} \otimes M_{2,2}^{-1}) \cdot \begin{matrix} P_1 \\ 0.47 \\ 0.00 \\ \dots \\ 0.53 \\ 0.00 \end{matrix}$$

target equation

x	prob.
$P_1(000)$	0.47
$P_1(011)$	0.53

	value
00	0.50
01	-0.02
10	0.01
11	0.01

Calculate the tensor product

	value
0	0.99
1	0.01

③ $|value| < \beta$

Aggregate the tensor-product result

	value
000	0.49
001	0.01
010	-0.01
100	0.01
101	10^{-4}
110	10^{-4}

x	prob.
$P_2(000)$	0.48
$P_2(001)$	6×10^{-3}
$P_2(010)$	6×10^{-3}
$P_2(011)$	0.50
$P_2(111)$	6×10^{-3}

Prune values $<$ threshold (e.g., 10^{-5})

For each basis states

- ① calculate the matrix-vector multiplication
- ② calculate the tensor-product
- ③ prune intermediate values
- ④ sum intermediate values to obtain output.

Compute the tensor-product of other basis states

Outline of Presentation



- Background and challenges
- Overview of QuFEM
- QuFEM characterization and calibration
- **Experiment**
- API of QuFEM

Setup

Platform	#Qubits	1-q fidelity	2-q fidelity	Instructions
Quafu	136	94.6±3.1%	94.6±3.0%	ID,RX,RY,RZ,H,CX
	18	95.9±1.3%	95.9±1.3%	ID,RX,RY,RZ,H,CX
Rigetti	79	99.5±1.1%	90.0±6.4%	CPHASE,XY
Self-developed	36	99.9±0.1%	98.7±0.8%	U3,CZ
IBMQ	7	99.9±0.1%	99.2±0.1%	CX,ID,RZ,SX,X

Evaluated hardware

IBU: KJ Satzinger, et al. Realizing topologically ordered states on a quantum processor. Science 2021

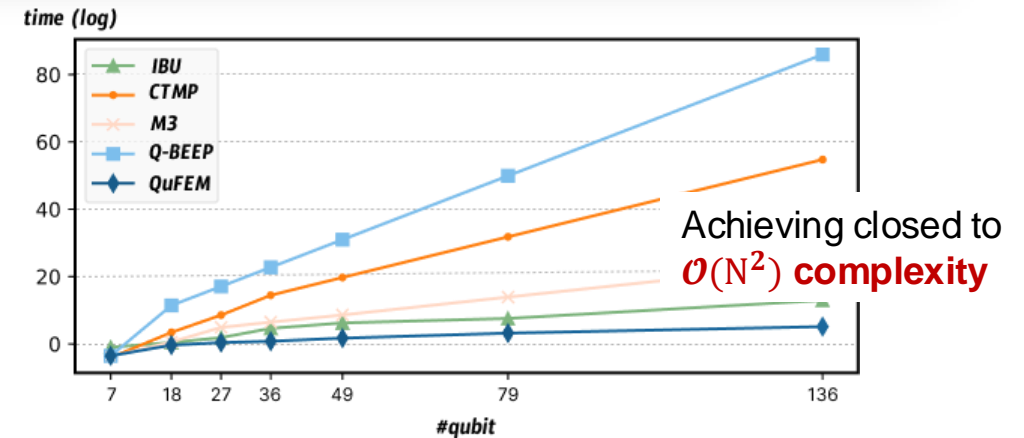
CTMP: Sergey, et al. Mitigating measurement errors in multiqubit experiments. PRA 2021.

M3: Paul D Nation , et al. Scalable mitigation of measurement errors on quantum computers. PRX Quantum 2021.

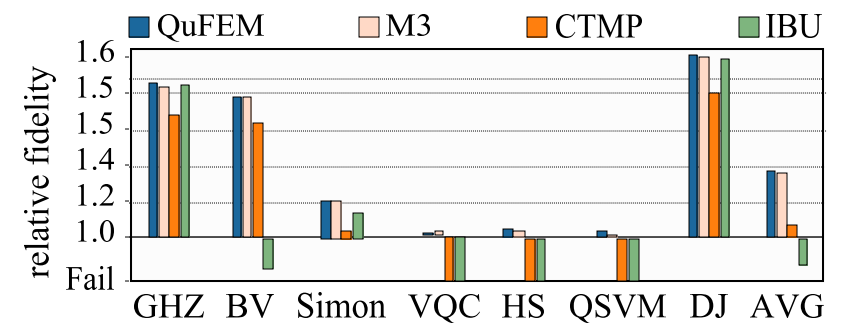
Q-BEEP: Nathan Wiebe, et al. QBEEP: Quantum Bayesian error mitigation employing Poisson modeling over the hamming spectrum. ISCA 2023.

Baselines

Result



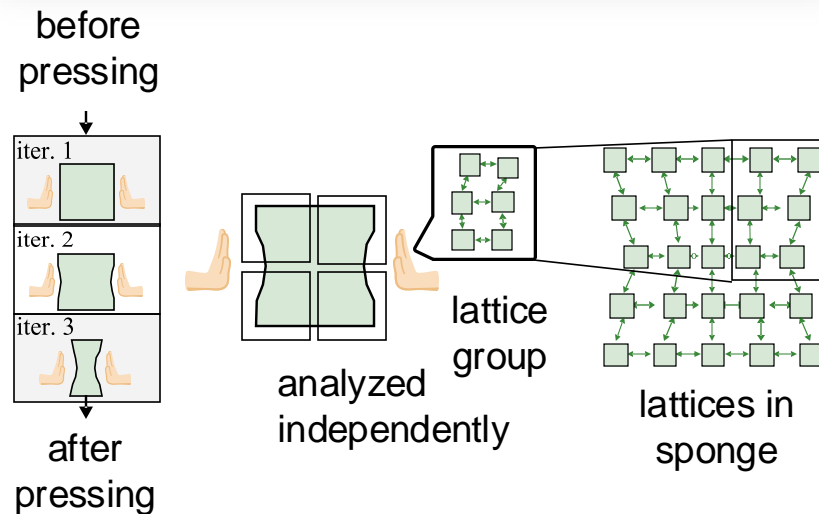
QuFEM reduces the calibration time of the 136-qubit program output from **119.44 hours** (IBU) to **169.65 seconds** (**119.44 × reduction**).



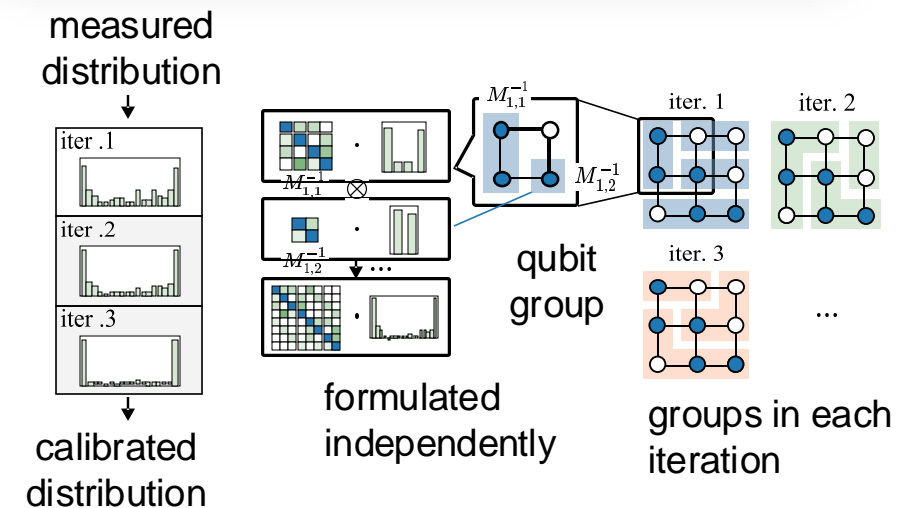
QuFEM shows an average improvement in relative fidelity of **1.003×**, **1.2×**, and **1.4×** compared to M3, CTMP, and IBU, respectively.

1. Limitations of prior matrix-based calibration methods: slow and inaccurate
2. Finite element method: a divide and conquer strategy
3. Detailed techniques to partition qubits and generate noise matrix
4. Sparse tensor product engine to speed up the computation

Classical Finite Element Method



Quantum Finite Element Method



Outline of Presentation



- Background and challenges
- Overview of QuFEM
- QuFEM characterization and calibration
- Experiment
- **API of QuFEM**

File:

- JanusQ/examples/ipynb/4_1_readout_calibration_simulator.ipynb
- JanusQ/examples/ipynb/4_2_readout_calibration_realqc.ipynb
- https://janusq.github.io/tutorials/demo/4_1_readout_calibration_simulator
- https://janusq.github.io/tutorials/demo/4_2_readout_calibration_realqc

construct mitigator {

calibrate output of
GHZ circuit {

```
from janusq.dataset import ghz_error, benchmark_circuits_and_results

from janusq.calibration.readout_mitigation.qufem import Mitigator
from janusq.calibration.readout_mitigation.qufem.tools import npformat_to_statuscnt

mitigator = Mitigator(n_qubits=8, n_iters=2)
scores = mitigator.init(benchmark_circuits_and_results, group_size=2,
multi_process=False, draw_grouping=True)

n_qubits = 4
outout_ideal = { '1'*n_qubits: 0.5, '0'*n_qubits: 0.5 }
output_fem = mitigator.mitigate(ghz_error[0], [ _ for _ in range(n_qubits) ], cho = 1)
output_fem = npformat_to_statuscnt(output_fem)
```



Thanks for listening

QuFEM: Fast and Accurate Quantum Readout Calibration Using the Finite Element Method

Siwei Tan, Liqiang Lu*, Hanyu Zhang, Jia Yu, Congliang Lang, Yongheng Shang, Xinkui Zhao, Mingshuai Chen, Yun Liang, and Jianwei Yin*