SOFTWARE AND ARCHITECTURE
FOR RELIABLE QUANTUM COMPUTING

Poulami Das

ETH Zurich
Sep 27, 2022
Why Quantum Computing?

Quantum computers promise computational advantages over conventional machines for many important applications.

- Material Science
- High-energy Physics
- Optimization
- Integer Factorization
- Machine Learning
Quantum Computing 101

Fundamental unit of information- Qubit

Quantum algorithms use quantum gates to manipulate qubits

α|0⟩ + β|1⟩

Superposition

Entanglement

Exponential State Space

Encode Problem

Apply Gates

Obtain Outcome

Initialize Qubits

Manipulate Qubits

Measure Qubits

Q₀ |0⟩

H

Q₁ |0⟩

Obtain Outcome

Quantum computers get computational advantage by using properties of qubits
Quantum Computers are Here!!!

- Quantum computers with 100+ qubits are already available!
- High error-rates
- Noisy Intermediate Scale Quantum (NISQ)

Use Quantum Error Correction (QEC)

Errors will happen, live with them

Hardware errors limit us from running most practical quantum applications today
Reliability Challenges in the Quantum Roadmap

Can software and architecture solutions bridge the gap between applications and noisy devices?
Outline

- Background and Motivation
- Measurement Error Mitigation for the NISQ Era
- Understanding Quantum Error Correction in the Near-Team
- Enabling Accurate, Fast, and Scalable Decoding in Fault-Tolerant Systems
Quantum errors and NISQ computing model

Quantum programs are vulnerable to different sources of hardware errors:

- Gate Error
- Measurement Error
- Idle Error
- Correlated Error
The Problem: Measurement Errors

Measurement errors are dominant sources of errors in large programs.

Each trial measures all program qubits.

All measurements must be error-free.
Even Bigger Problem: Measurements at Scale

Measurement crosstalk can increase the effective error-rate

MEASUREMENT ERROR RATES ON GOOGLE SYCAMORE

<table>
<thead>
<tr>
<th></th>
<th>Average</th>
<th>Worst-Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISOLATED</td>
<td>6.14</td>
<td>11.7</td>
</tr>
<tr>
<td>MULTIPLE</td>
<td>7.73</td>
<td>20.9</td>
</tr>
</tbody>
</table>

Measurement crosstalk increases with program size
Prior Works on Measurement Error Mitigation

Matrix-based Approach from IBM

- $2^n \times 2^n$ matrix
- Probability of Success increases from 50 to 70%

State-transformation Approach

Flip-And-Measure (MICRO-2019): Measurement error depends on state

- Calibration
- Post-processing

Existing measurement error mitigation schemes rely on measuring all qubits.
Goal: Reduce Measurement Errors

Insight: Measure fewer qubits and reconstruct distribution

Are high fidelity circuits with partial measurements alone sufficient?
Need for Correlation

Ideally, we want full correlation and high fidelity

![Diagram showing correlation and fidelity]

<table>
<thead>
<tr>
<th>Outcomes (q1q0)</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0.25</td>
</tr>
<tr>
<td>01</td>
<td>0.25</td>
</tr>
<tr>
<td>10</td>
<td>0.25</td>
</tr>
<tr>
<td>11</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outcomes (q1)</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Outcomes (q0)</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Correlation? No
Program gives full correlation; CPM gives high fidelity. JigSaw combines both.
JigSaw-M: Multi-Layer JigSaw

Effectiveness of JigSaw can be improved further by using heterogeneous CPM

Default subset size: 2

Other subset size: 3
Impact of JigSaw

- Outperforms IBM’s measurement error mitigation and Flip-and-Measure
- Greater effectiveness when combined with other optimizations
Outline

- Background and Motivation
- Error Mitigation Techniques for the NISQ Era
- Understanding Quantum Error Correction in the Near-Team
- Enabling Accurate, Fast, and Scalable Decoding in Fault-Tolerant Systems
Quantum Error Correction (QEC) is more challenging than classical techniques.

- **No-Cloning**
- **Collapse on Measurement**

Did an error occur?  
Where did the error occur?  
What was the type of error?

QEC can protect quantum information by tracking errors periodically.

Diagram:
- Data Qubits
- Parity Qubits
- Logical Qubit
- Syndrome Extraction
- Decode Errors
- Initialize
- Measure Data Qubits
- Repeat

Example sequence: 0101
Demonstration of QEC codes

Data Qubits

Bit Flip (X) Errors -> Z stabilizers (A)
Phase Flip (Z) Errors -> X stabilizers (B)

X stabilizers

Z stabilizers

Code Distance (d=3)

Demonstration of small surface codes represent a significant milestone for QEC

Widely regarded as the most promising QEC candidate
Real-Time Accurate Decoding

Decoding or identification of errors must be in real-time in cryogenic environment.

Real-Time decoding is essential to prevent accumulation of errors.
Goal: Real-Time Decoding in Near-Term QEC

Insight: Don’t use software decoder, Lookup from Tables -> LILLIPUT

LILLIPUT: A lightweight low latency lookup table decoder for near-term quantum error correction, ASPLOS 2022

P. Das, A. Locharla, C. Jones, LILLIPUT: A Lightweight Low Latency Lookup Table Decoder for Near-Term Quantum Error Correction, ASPLOS 2022
Step 1: Detection of Errors

Error Detection Event: XOR between syndromes from consecutive cycles

LILLIPUT uses FIFOs and XOR operations to detect error events.
Step 2: Handling Different Types of Errors

- Errors on Data Qubits
- Gate Errors in Syndrome Extraction
- Readout Errors on Parity Qubits
- Readout Errors on Data Qubits
Step 2: Handling Different Types of Errors

- Errors on Data Qubits
- Gate Errors in Syndrome Extraction
- Readout Errors on Parity Qubits
- Readout Errors on Data Qubits

LILLIPUT can handle errors in any operation in the quantum hardware.
Step 3: Error Assignments

Program LUT using software Minimum Weight Perfect Matching Decoder

LILLIPUT programs the LUTs offline and performs decoding online.
LILLIPUT Operations Overview

1. Streaming Mode Operation
2. Assign Errors to Oldest Round
3. Track internal state (in LUT entry)

Account for window boundary crossings

Avoid premature matching

Avoid inaccurate matching

LILLIPUT maintains error logs and internal state to accurately track errors.
LILLIPUT: Design Overview

- Z Syndrome
  - FIFO
  - Event Detection Logic
  - LUT for Decoding
  - Error Log
  - Internal State
  - Compute Logical Error
  - Logical Error

- X Syndrome
  - FIFO
  - Event Detection Logic
  - LUT for Decoding
  - Error Log
  - Internal State

Interface to Readout Logic
Challenge: Memory Complexity of LILLIPUT

The size of the LUTs scale rapidly

Syndrome = 4 bits
Entry = 9 bits error assignment + 4 bits state

Cycle-1
Cycle-2

Syndrome = 4 bits
Entry = 9 bits error assignment + 4 bits state

[d=3, m=1] 13
2^4 832 B
[d=3, m=2] 13
2^8 [d=4, m=2] 24 238 KB
[d=5, m=2] 37 2^24 148 MB

The size of the LUTs scale rapidly
Tackling the Memory Complexity of LILLIPUT

LUT sizes scale exponentially with the distance of the QEC Code

Distance = 3
$p_{err}=0.1\%$

Uncorrectable

Not All Error Events are Equally likely

Store selective entries in LUTs

LUT entries themselves store sparse data -> compress

We propose Compressed LUTs (CLUTs) to reduce the memory complexity
Performance of LILLIPUT

CLUTs have negligible impact on accuracy

42 ns latency, < 7% FPGA logic, Sufficient up-to d=5

Logical Error Rate < Physical Error Rate

Up-to 107x memory reduction with CLUTs

LILLIPUT: A low-cost, accurate, and real-time decoder for practical adoption
Outline

- Background and Motivation
- Error Mitigation Techniques for the NISQ Era
- Understanding Quantum Error Correction in the Near-Team
- Enabling Accurate, Fast, and Scalable Decoding in Fault-Tolerant Systems
Goal: Real-Time Accurate Decoding at Scale

Decoders must be accurate, fast, and scalable for practical implementation.

AFS decoder translates the three steps of UFD into three distinct pipeline stages:

1. **Cluster Generation**
2. **Cluster Traversal**
3. **Peeling**

Leverage micro-architectural optimizations to improve decoding latency.

**Union-Find Decoding (UFD) Algorithm -> AFS Decoder**

- **Graph Generator**
- **DFS Engine**
- **Corr Engine**
AFS Decoder: Design Overview

Graph Generator (Gr-Gen)

Control Logic

Read/Write Interface

Zero Data Register (ZDR)

Parity/Traversal Registers

Root Table

Size Table

Runtime Stack

Depth First Search (DFS) Engine

State Machine

Edge Stack (S0)

Pending Edge Stack

Edge Stack (S1)

Correction (CORR) Engine

Select Logic

Apply Correction

Syndrome Hold Registers

Spanning Tree Memory (STM)
Challenge: Increasing Memory Cost

- AFS Decoder is a memory-intensive design
  - Registers
  - Tables
  - Stacks

- Technology for Implementation: Superconducting (tight memory budget) or CMOS (tight power budget) is an open problem

Memory requirement of AFS decoder scales very rapidly with system size
Conjoined Decoder Architecture (CDA)

- **Dedicated Decoders**
  - Linear increase in hardware cost
  - Least hardware cost
  - Poor decoding capability

- **Single Decoder**
  - Timeout?
  - Least hardware cost
  - Poor decoding capability

- **CDA**
  - Reduced hardware cost
  - Negligible impact

Insight: Allow sharing if $P_{\text{Timeout}} < P_{\text{logical error}}$

CDA allows restricted sharing of hardware with negligible impact on accuracy.
Performance of AFS Decoder

AFS Decoder is accurate, fast, and scalable

Accuracy: 9.96 MB

Speed: 2.81 MB

Scalability: Yes
Conclusion

• Hardware errors limit us from running most practical quantum applications

• Software techniques can improve the fidelity of applications in the NISQ Era

• Fault-tolerant quantum computers can power a wider range of applications

• Architecture and system-level solutions can help us to build accurate, fast, and scalable decoders for fault-tolerant quantum computing
Thank You!

A Quantum COMPUTER
BACK-UP Slides (JigSaw)
Bayesian Reconstruction

<table>
<thead>
<tr>
<th>Global Mode</th>
<th>Subset Mode</th>
<th>Update Coefficients</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Q_2Q_1Q_0)</td>
<td>(Q_1Q_0)</td>
<td>(Q_1Q_0)</td>
<td>(Q_2Q_1Q_0)</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 1 0</td>
<td>1 0</td>
<td>1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 0</td>
<td>1 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>1 1 1</td>
<td>1 1</td>
<td>1 1</td>
<td>0 1 1</td>
</tr>
</tbody>
</table>

\[
\frac{0.5 \times 0.1}{1 - 0.1} = 0.06
\]

Cannot infer solution

Correct!
Measurement Errors: Sources

0 1
Measure

1
Discriminator

ADC
300 K
Amplifier
3-4 K
SNR+
Resonator
20 mK

Software
Hardware

Frequency shift is sensitive to noise
Thermal noise
Long latency
Inaccurate classification
How many CPM do we need?

N-qubit program has $\binom{N}{2}$ possible CPM of subset size 2
Device Variability

**Measurement Error Rates**

- Mean: 4.70%
- Median: 2.76%
- Minimum: 0.85%
- Maximum: 22.2%

**Measurement Error Range (in percentiles)**

- <25
- 25-50
- 50-75
- >75
Impact of CPM

CPM for 6-qubit Bernstein Vazirani on IBMQ-Toronto
Impact of Number of Trials

Graph showing the impact of the number of trials on the Application PST. The graph compares different algorithms: GHZ-12, GHZ-16, GHZ-14, QAOA-10 (p1), QAOA-10 (p2), and QAOA-10 (p4). The x-axis represents the number of trials executed on IBMQ-Paris, ranging from 8K to 4 million. The y-axis represents the Application PST, ranging from 0.1 to 0.5.
Impact of Recompilation

Probability of Correctly Measuring the Qubit

Program Qubit in BV-6

- Baseline
- CPM of subset size 2
Impact of Recompilation

Average: No Recompilation -> 1.9x, With Recompilation -> 2.9x
Scalability Analysis

- Complexity is determined by number of unique outcomes
- JigSaw does updates only for non-zero outcomes (limited by trials)

<table>
<thead>
<tr>
<th>Program Size (Num. of Qubits)</th>
<th>JigSaw</th>
<th>JigSaw-M</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Memory (GB)</td>
<td>Operations (Billion)</td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td>0.4</td>
</tr>
<tr>
<td>500</td>
<td>5</td>
<td>2.1</td>
</tr>
</tbody>
</table>

*Assuming 1 Million trials, and pessimistically each trial yields a unique outcome
BACK-UP Slides (LILLIPUT)
<table>
<thead>
<tr>
<th>Decoder Configuration</th>
<th>Frequency (MHz)</th>
<th>Latency (ns)</th>
<th>Total LEs/ALMs</th>
<th>Total Registers</th>
<th>Memory</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>[d=3, m=2]</td>
<td>250</td>
<td>28</td>
<td>353</td>
<td>209</td>
<td>832 B</td>
<td>6%</td>
</tr>
<tr>
<td>[d=3, m=3]</td>
<td>240.7</td>
<td>29.1</td>
<td>418</td>
<td>239</td>
<td>13 KB</td>
<td>7%</td>
</tr>
<tr>
<td>[d=4, m=2]</td>
<td>209.8</td>
<td>33.4</td>
<td>557</td>
<td>340</td>
<td>238 KB</td>
<td>&lt;1%</td>
</tr>
<tr>
<td>[d=4, m=3]</td>
<td>244.4</td>
<td>40.8</td>
<td>217</td>
<td>409</td>
<td>53.8 MB</td>
<td>&lt;1%</td>
</tr>
<tr>
<td>[d=5, m=2]</td>
<td>232.9</td>
<td>42</td>
<td>246</td>
<td>486</td>
<td>148 MB</td>
<td>&lt;1%</td>
</tr>
</tbody>
</table>

- **Low-Latency**
- **Low-Cost**
- Up-to 107x reduction from CLUTs
How to use Compressed LUTs?

<table>
<thead>
<tr>
<th>d=3, m=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0x0F</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0xAA</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0xFF</td>
</tr>
</tbody>
</table>

Hamming Weight cut-off = 3

Memory Fragmentation

416 B

[d=3, m=2]

<table>
<thead>
<tr>
<th>16-entry Data Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>0x0F</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

10-entry Data Frame

| 0xA0 |
| ...  |
| 0xAF |
| ...  |

Discard

| 0xFF |
| ...  |

Segment-A

| 0xA0 |
| ...  |
| 0xAA |
| ...  |

Compress Entries

<table>
<thead>
<tr>
<th>36b -&gt; 16b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xA0</td>
</tr>
<tr>
<td>0xAA</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

140 B
BACK-UP Slides (AFS)
Error Decoding as a Matching Problem

Error Assignments

Measurement Errors
## Cost Analysis of Memory Reduction

<table>
<thead>
<tr>
<th>Design Component</th>
<th>AFS without CDA (in MB)</th>
<th>AFS with CDA (in MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM (Gr-Gen)</td>
<td>1.97</td>
<td>0.99 (2x)</td>
</tr>
<tr>
<td>Root Table (Gr-Gen)</td>
<td>3.17</td>
<td>0.79 (4x)</td>
</tr>
<tr>
<td>Size Table (Gr-Gen)</td>
<td>3.46</td>
<td>0.87 (4x)</td>
</tr>
<tr>
<td>Stacks (DFS Engine)</td>
<td>1.35</td>
<td>0.34 (4x)</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>9.96</strong></td>
<td><strong>2.81 (3.5x)</strong></td>
</tr>
</tbody>
</table>
Bandwidth Challenges: Scalability

![Graph showing the relationship between Code distance (d) and Bandwidth (in Gbps). The graph indicates an exponential increase in bandwidth with increasing code distance.]
Syndrome Compression for Bandwidth Reduction

- Each error flips two parity qubits
- Long error-chains have fewer parity flips
- $3d^3$ fault locations
  - $6d^3p$ possible non-zero syndrome bits
- $d = 11, p = 10^{-3}$
  - 8 non-zero bits in 1000-bit syndrome

Insight: Syndrome data is sparse, can be compressed
Bandwidth Reduction

- 30x reduction on average
- Effectiveness increases with code distance