QDiff: Differential Testing of Quantum Software Stacks

Jiyuan Wang, Qian Zhang, Harry Xu, and Miryung Kim

University of California, Los Angeles
TEAM MEMBERS

Jiyuan Wang  Qian Zhang  Harry Xu  Miryung Kim
People are using hardware accelerators

FPGA

GPU

Quantum Computer
Quantum Computers (QCs) are promising hardware accelerators.

**Grover algorithm** can speed up unstructured search with a quantum computer.

**Shor algorithm** can speed up integer factorization with a quantum computer.
Quantum platforms are becoming publicly available
A study of real world errors in Qiskit, Cirq, and Pyquil

- We studied 76 bugs posted on StackOverflow and Github regarding Qiskit, Cirq, and Pyquil.

- Errors happen in three parts: language constructs, compilers, and simulators, which we call quantum software stacks (QSS).

<table>
<thead>
<tr>
<th>Survey Statistics</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Keywords Searched</strong></td>
<td>Quantum framework bugs, error, unexpected behavior</td>
</tr>
<tr>
<td><strong>Posts Studied in total</strong></td>
<td>76 posts</td>
</tr>
<tr>
<td><strong>Common Fault Types</strong></td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Error Types</th>
<th>Percentage</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bugs in <strong>Language Constructs</strong></td>
<td>11.8%</td>
<td>Qiskit crashes on cnot() gate</td>
</tr>
<tr>
<td>Bugs in <strong>Compiler Setting</strong></td>
<td>53.9%</td>
<td>Qiskit has bugs at the compiler optimization level</td>
</tr>
<tr>
<td>Bugs in Different <strong>Simulators</strong></td>
<td>19.7%</td>
<td>Pyquil crashes on PyQvm with controlled gates</td>
</tr>
<tr>
<td>Bugs in Installation / Interaction with Other Tools</td>
<td>14.6%</td>
<td>Qiskit installation fails with Python 3.9</td>
</tr>
</tbody>
</table>
Errors in QSS are hard to identify

Challenge 1: Inherent probabilistic nature of quantum computing makes the error detection hard.

Challenge 2: There are very few quantum programs out here.

Challenge 3: Wait time is long due to limited public access to expensive quantum hardware.
QDiff’s solutions

Challenge 1: Inherent probabilistic nature of quantum computing makes the error detection hard.

Challenge 2: There are very few quantum programs out here.

Challenge 3: Wait time is long due to limited public access to expensive quantum hardware.


Generate more quantum programs with equivalent transformations.

Avoid unnecessary invocations of a quantum simulator and hardware with static characteristics of quantum circuits.

We found 4 software crashes in the Cirq and Pyquil simulations, and 2 possible root causes of 29 divergence cases on IBM quantum hardware.

QDiff approach overview

- QSS
- Equivalent Programs
- Differential Execution
- Testing loop

- Seed Program

- Exploration of program variants, backends, and compiler settings
- Filtering mechanism to avoid unnecessary hardware and simulator invocations
- K-S statistics based comparison [1]
Approach 1: Equivalent program transformation

- QDiff transforms gates across their equivalent forms
  - We pick 7 commonly used rules in quantum compilation [2] and users can easily extend their own rules.
- QDiff also varies backend settings and compiler optimizations.

<table>
<thead>
<tr>
<th>Rule ID</th>
<th>Original Gate</th>
<th>Equivalent Gate</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>SWAP(p,q)</td>
<td>CNOT(p,q)CNOT(q,p)CNOT(p,q)</td>
</tr>
<tr>
<td>G2</td>
<td>S(p)</td>
<td>T(p)T(p)</td>
</tr>
<tr>
<td>G3</td>
<td>X(p)</td>
<td>H(p)S(p)S(p)H(p)</td>
</tr>
<tr>
<td>G4</td>
<td>Z(p)</td>
<td>S(p)S(p)</td>
</tr>
<tr>
<td>G5</td>
<td>CZ(p,q)</td>
<td>H(q)CX(p,q)H(q)</td>
</tr>
<tr>
<td>G6</td>
<td>CZ(p,q)</td>
<td>merged to Identity Matrix</td>
</tr>
<tr>
<td>G7</td>
<td>CCNOT(p,q)</td>
<td>6 CNOT gates with 9 one-qubit gates</td>
</tr>
</tbody>
</table>

QDiff uses K-S test [1] to compare the distributions. We report the divergence if the K-S statistic is larger than a threshold.
Approach 2: KS based comparison

- **Cumulative Probability** with respect to the total trial number $N$.
- **K-S Statistic** is the max distance $D$ between two measurement results.
- QDiff checks if $D > t$ where $t$ is a user-defined threshold.
- The total trial number $N$ can be determined by $t$ and the confidence level $p$, which are given by users [3].

<table>
<thead>
<tr>
<th>Measurement Distribution</th>
<th>‘00’</th>
<th>‘01’</th>
<th>‘10’</th>
<th>‘11’</th>
</tr>
</thead>
<tbody>
<tr>
<td>N1</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>N2</td>
<td>247</td>
<td>247</td>
<td>253</td>
<td>253</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cumulative Probability</th>
<th>‘00’</th>
<th>‘01’</th>
<th>‘10’</th>
<th>‘11’</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>0.250</td>
<td>0.500</td>
<td>0.750</td>
<td>1.00</td>
</tr>
<tr>
<td>S2</td>
<td>0.247</td>
<td>0.494</td>
<td>0.747</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Approach 3: Filtering mechanism

• In the quantum theory, a qubit can retain a correct state for only a limited amount of time, called T1 time.
• 2-qubit gate dominates the error rate of the circuit.

Filtering mechanism: $\text{execution\_time} < T1 \& \#(2qgate) < th$

[Diagram showing quantum circuits, analyzing the number of 2-qubit gates, and executing on quantum hardware]
We generated 730 different quantum programs with semantic modifying mutations. The programs in turn produced 14799 circuits with equivalent transformations.

We reduced 66% unnecessary quantum hardware and simulator invocations.

We found 4 crashes in the quantum simulators and 2 root causes of 29 divergences with the quantum hardware.
Differential testing on quantum simulators

4 crashes we found in quantum simulators and they have been confirmed with QSS developers.

```python
qvm = get_qc('3q-qvm')
try:
    qvm.run(Program())
except:
    print("empty")
.....
qvm.run(Program(H(1)))
```

Example: Pyquil’s simulator is stuck after executing an empty circuit
Differential testing on IBM quantum hardware

● We found 29 divergence cases in hardware executions:
  ○ These divergence cases might due to quantum circuit synthesis or compilation.
  ○ Equivalent programs produce measurements that are flagged as divergence in terms of K-S statistics.

● We speculated 2 possible root causes for 25 of them:
  ○ qubit dephasing [4]: 9 cases.
  ○ 2-qubit gate mapping [5]: 16 cases.

Differential testing on IBM quantum hardware

Qubit dephasing: no operation on #qc4 for too long time [4].

QDiff: Differential Testing of Quantum Software Stacks

● We are the first one to adapt **differential testing** to quantum software stacks.

● QDiff is effective in
  ○ generating quantum program variants;
  ○ avoiding unnecessary invocations of the quantum simulator and hardware;

● We need deeper hardware knowledge to understand the root causes of our findings.

● We need further effort to isolate software stack errors from hardware noises.

● QDiff on Github: [https://github.com/UCLA-SEAL/QDiff](https://github.com/UCLA-SEAL/QDiff)
Thanks for listening!
Q&A