Breakpoint Debugging on Real Machine

Palvit Garg, Shawn Salekin, Harsh Joshi

Problem Statement

- There is no known way to add a breakpoint in a quantum circuit and examine the properties of a quantum circuit that runs on the hardware
- Debugging is possible in simulators (a group in previous year have done this)

Challenges

- Adding a breakpoint entails measuring the circuit
- Measurement forces the circuit to collapse into one of the possible states
- We have probability distribution upon measurement
- But not enough to “recreate” the program state as it was before the breakpoint
- Amplitude can be recovered (with error) from the measured states, but phase is lost.

Solution Approach: Hybrid (Simulator + Hardware)

1. Run until breakpoint on both hardware and simulator.
2. Show measurements from hardware to developer.
3. Compute unitary matrix from the simulator run.
4. Use this unitary matrix in place of actual circuit for next run on hardware until next breakpoint.
5. Pro: Not introducing any noise in old circuit

Solution Approach (Only hardware)

1. Run circuit until breakpoint on hardware and get probability distribution of output state.

2. Using this, we computed the amplitude of the state vector. {00: 1/2, 01: 1/2, 10: 1/2, 11: 1/2}

```
{'00': 0.249999970197678, '01': 0.249999970197678, '10': 0.249999949296019, '11': 0.249999949296019}
```

3. Further, to compute the final state vector, we need to compute exact relative phase for each output. QPE can be helpful at, this step. Phase will look something like this:
{00: 0 deg, 01: 180 deg, 10: 45 deg, 11: -135 deg (225 deg)}

4. This relative phase must be used to compute state vector as below:
{00: 1/2(cos(0 deg) + sin (0 deg) * i) → 0.5 + 0i
01: 1/2(cos(180 deg) + sin (180 deg) * i) → -0.5 +0i
10: 1/2(cos(45 deg) + sin (45 deg) * i) → 0.35355 + 0.35355i
11: 1/2(cos(-135 deg) + sin (-135 deg) * i) → -0.35355 -0.35355i}

5. The final state vector looks something like this:

```
st0 = Statevector.from_instruction(qc)
st0
```

```
Statevector([ 0.5       +0.j        , -0.5       +0.j        ,
              0.35355339+0.35355339j, -0.35355339-0.35355339j],
            dims=(2, 2))
```

6. Now this state vector can be used to prepare state for this first part run until breakpoint. Thus, we don't need to rerun this part of the circuit, because the previous state was collapsed.
7. This approach will not give exact results, cause there will be noise at each part, which will flow with each state vector.

Caveats?

Hardware Solution

- It is not possible to extract local phase of entangled qubits (you get a random value).
- To project the phase of one qubit with QPE, we need multiple ancilla qubits (3-4). This limits the size of the circuit, as wider circuit decohere faster
- 4 qubits still doesn’t give you lots of precision to estimate the local phase.
- This approach will introduce a lot of noise at each step. There is a good chance that the final result deviates far from actual measurements.

Hybrid Solution

- It is limited to small circuits only, because the unitary matrix grows exponentially.

Progress So Far

- Completed the literature review to understand and brainstorm all possible approaches.
- Scrapped a few ideas after experimentation, like preparing unitary matrix from hardware run.
- Worked on few proofs of concepts to understand if they actually work as we expect (code available in repo).
- Almost all other parts of both approaches can be integrated into code, except phase estimation from hardware.

Open Items

- QPE seems to be working fine for single qubit, without any entanglement. But we are still working on figuring out how to make it work with 2 qubit system and get phase for each possible outcome in probability distribution.
- Once that part is done, we need to figure out ways to reduce time and space complexity on ancilla bits. Because phase estimation adds significant overhead, instead rerunning the same circuit seems more optimized as of now.