Exploiting Quantum Assertions for Error Mitigation and Quantum Program Debugging

Peiyi Li
Dept. of ECE
NC State University
Raleigh, NC, USA
pli11@ncsu.edu

Ji Liu
Dept. of ECE
NC State University
Raleigh, NC, USA
jliu45@ncsu.edu

Yangjia Li
State Key Lab. of CS
Institute of Software
Beijing, China
yangjia@ios.ac.cn

Huiyang Zhou
Dept. of ECE
NC State University
Raleigh, NC, USA
hzhou@ncsu.edu

***Abstract*—An assertion is a predicate that should be evaluated true during program execution. In this paper, we present the development of quantum assertion schemes and show how they are used for hardware error mitigation and software debugging. Compared to assertions in classical programs, quantum assertions are challenging due to the no-cloning theorem and potentially destructive measurement. We discuss how these challenges can be circumvented such that certain properties of quantum states can be verified non-destructively during program execution. Furthermore, we show that besides detecting program bugs, dynamic assertion circuits can mitigate noise effects via post-selection of the assertion results. Our case studies demonstrate the use of quantum assertions in various quantum algorithms.**

***Index Terms*—quantum computing, error mitigation, debugging, assertion**

I. Introduction

Quantum computing offers high speedup potentials over classical computing in multiple important domains, including quantum simulation, optimization, etc. To realize such potentials, there are barriers in both quantum hardware and software to be overcome. On the software side, developing quantum programs is difficult. Based on the lessons learnt from classical computing, program bugs can be common in quantum programs given their conceptual complexity. On the hardware side, quantum devices are sensitive to environment and are error prone. Therefore, error mitigation and ultimately error correction are required for quantum computing systems. In this paper, we discuss quantum assertions as a way to help debug quantum programs as well as to perform lightweight error detection and correction.

An assertion is a program predicate that should always be evaluated true during program execution. In classical computing, assertions are commonly used to monitor some intermediate program states so as to verify certain properties of a program or to detect runtime anomalies. Assertion errors, if any, would help to pinpoint the location of program bugs.

In quantum program execution, intermediate states are much more difficult to monitor than in classical computation. There are two reasons. First, direct measurement of some qubits may collapse their superposition states and may affect other qubits if they are entangled with the qubits under measurement. For example, in an entangled 2-qubit Bell state, $\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, the measurement of the first qubit would not only collapse the first qubit to a classical state, $|0\rangle$ or $|1\rangle$, but also affects the second qubit by forcing it to be in the same classical state as a result of entanglement. In other words, direct measurement of quantum states can be destructive and affect subsequent program execution. Second, indirect measurement is also limited by the non-cloning theorem, which states that it is impossible to copy an arbitrary qubit state such that the copied qubits and the original ones are not correlated.

To tackle these challenges, Li and Ying [1] proposed a protocol to debug quantum processes, in which an error detector, that is a projection operator orthogonal to the anticipated quantum system state, is used to check the quantum system at a sequence of time points. The similar principle of projection based monitoring is leveraged in later works on quantum assertions with different assertion circuit implementations [3]–[5]. On the other hand, statistical assertions, which rely on multiple measurements of the quantum states to infer their statistical properties, have been proposed [6]. Such statistical approaches can be viewed as a limited form of quantum tomography to characterize quantum system states.

In this paper, we present a detailed overview of these quantum assertion techniques and discuss their trade offs. Besides checking that intermediate quantum states are the same as expected ones, assertions can be used to check certain properties of the quantum states. State symmetry or bit-flip invariance is one such example [7].

In classical computation, assertions can be used for detecting both software bugs and/or hardware errors as they both lead to unexpected program states. Similarly, quantum assertions can be leveraged for detecting either software bugs or hardware errors. Furthermore, as a consequence of measurements, successful assertion checks may actually collapse erroneous quantum states into the correct ones. In other words, besides debugging, quantum assertions can achieve error correction to a certain degree. In our case studies, we present our experimental results of using quantum assertion techniques for debugging as well as error mitigation.

The remainder of this paper is organized as follows. Section II provides a brief background on quantum computation and

The NC State team is funded in part by NSF grants 1818914 (with subcontract to NC State University from Duke University) and OMA-2120757 (with subcontract to NC State University from University of Maryland).

measurement. Section III discusses the different quantum assertion techniques. Section IV and Section V present case studies of using quantum assertions for software debugging and hardware error mitigation, respectively. Section VI concludes the paper and discusses the future directions.

II. Background

In a digital quantum computing system, information is encoded in qubit (quantum bits) and the system state is represented with its qubit states. A quantum program is essentially a sequence quantum gates applied upon qubits. In a way, a quantum program controls how the quantum system state evolves over time by applying the instructions, i.e., quantum gates, at different times. During program execution, the quantum system state can be in a pure or mixed state.

A pure quantum state can be represented with a vector of complex numbers with norm one. For example, a single-qubit pure state can be represented as $|\psi\rangle = a|0\rangle + b|1\rangle$ where $|a|^2 + |b|^2 = 1$. An n-qubit quantum gate operation is represented by a $2^n \times 2^n$ unitary matrix U. A gate U operating on a pure state $|\psi\rangle$ results in the state $|\psi'\rangle = U|\psi\rangle$.

A mixed quantum state means a mixture of more than one pure state. A mixed state is described with a density matrix $\rho = \sum_i P_i |\psi_i\rangle\langle\psi_i|$, where P_i are the probabilities of each pure state $|\psi_i\rangle$ and $\sum_i P_i = 1$. Mixed states are more generic than pure ones as a pure state $|\psi\rangle$ can also be represented with a density matrix: $\rho = |\psi\rangle\langle\psi|$. When a quantum gate U operates on a mixed state with density matrix ρ, the resulting state's density matrix is $\rho' = U\rho U^\dagger$.

For a quantum state, $|\psi\rangle$, a measurement will collapse it into one of the basis states, $|m_i\rangle$, associated with the measurement. And the probability of the measurement outcome that the basis state $|m_i\rangle$ is observed is $p_i = \langle\psi| M_i^\dagger M_i |\psi\rangle = |\langle\psi|m_i\rangle|^2$, where the measurement operator $M_i = |m_i\rangle\langle m_i|$. The state after the measurement is $\frac{M_i|\psi\rangle}{\sqrt{p_i}} = \frac{\langle m_i|\psi\rangle}{\sqrt{p_i}}|m_i\rangle$ [11], which is the same as $|m_i\rangle$ except a non-distinguishable global phase. For example, for the computational basis, $M_0 = |0\rangle\langle 0|$ and $M_1 = |1\rangle\langle 1|$, the state $|\psi\rangle = a|0\rangle + b|1\rangle$ has a probability of $|a|^2$ (or $|b|^2$) being measured as $|0\rangle$ (or $|1\rangle$) and the state after the measurement becomes $|0\rangle$ (or $|1\rangle$). In the same way, when the state $|\psi\rangle = a|0\rangle + b|1\rangle$ is measured using the $|+\rangle, |-\rangle$ basis, the state after measurement becomes the $|+\rangle$ or $|-\rangle$ state with the probability of $\frac{1}{2}|a+b|^2$ and $\frac{1}{2}|a-b|^2$, respectively. The implication is that when being measured at a proper basis, the state after measurement can be corrected to the expected state. This property can be leveraged for hardware error mitigation when assertion checks pass without any error.

The same property also holds for a mixed state with a density matrix ρ. Given a set of orthogonal measurement operators M_n, the probability of observing $|m_i\rangle$ is $Tr(M_i\rho M_i^\dagger)$ and the density matrix after measurement collapses to $\rho' = \frac{M_i\rho M_i^\dagger}{Tr(M_i\rho M_i^\dagger)}$ if the state $|m_i\rangle$ is observed [11]. As we can see, the density matrix after measurement is a normalized form of $M_i\rho M_i^\dagger$. With $M_i = |m_i\rangle\langle m_i|$ and $\rho = \sum_k P_k |\psi_k\rangle\langle\psi_k|$, we can derive that $M_i\rho M_i^\dagger = \sum_k P_k |m_i\rangle\langle m_i|\psi_k\rangle\langle\psi_k|m_i\rangle\langle m_i| = CM_i$ where C is a scaled factor. This means that the state after the measurement is the same as the measured state $|m_i\rangle$.

III. Quantum Assertion Schemes

In this section, we summarize the recently proposed quantum assertion schemes and discuss their trade offs. We start with assertions whose measurements may destruct the quantum state under test and then present the ones that preserve or correct to the expected states.

A. *Statistical Assertions*

Huang et al. [6] proposed to use statistical tests to implement quantum assertions. In their approach, the assertion points in a quantum program become breakpoints and the measurements are directly performed upon the qubits under test. Three types assertions are proposed.

- Classical assertions: asserting that a quantum variable should take an expected integer value upon measurement.
- Superposition assertions: asserting that a quantum variable should be in the uniform superposition state.
- Entanglement assertions: asserting that the control and target quantum variables are entangled, which means that they have correlated measurement outcomes.

Ensembles of measurements are needed to determine whether an assertion passes or fails. Each measurement requires the program to run from the beginning and stop at the assertion point. After multiple measurements, the chi-square test is used to check for classical and superposition quantum states. Contingency table analysis coupled with the chi-square test is used to check for entangled states.

Direct measurement of the quantum variables (or qubits) using the computational basis limits the information to be collected. For example, the phase information (e.g., $|+\rangle$ vs. $|-\rangle$) cannot be determined from the ensemble of measurements. This limitation can be overcome adding a rotation gate before measurement such that the measurement is done at a different basis. This process would make statistical assertions similar to quantum tomography, whose drawback is the number of measurements can be exceedingly high when considering the different bases.

B. *Assertions using Swap Tests*

Swap tests have been suggested as a way for quantum assertions [8]. The swap test circuit, as shown in Fig. 1, can be used to test two states, pure or mixed, are equal. Therefore, it makes intuitive sense to use this circuit implement the assertion check $AssertEqual(|\psi\rangle, |\mu\rangle)$, which passes when the two states are the same and fails otherwise.

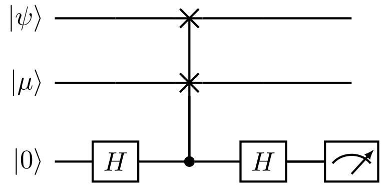


Fig. 1: Swap-test circuit.

With $|\psi\rangle$ and $|\mu\rangle$ being pure states, the probability of the measurement outcome being $|0\rangle$ is shown in Equation 1.

$$P(measured\_outcome = 0) = \frac{1}{2} + \frac{1}{2}|\langle\psi|\mu\rangle|^2 \qquad (1)$$

As shown in Equation 1, if the two states are identical, the probability of measuring $|0\rangle$ is 1. In other words, if the two states are equal, the measured state is always $|0\rangle$ and the assertion check always passes. If the two states $|\psi\rangle$ and $|\mu\rangle$ are different, the probability would be lower than 1. The problem of using swap tests for assertions is that it is possible for the measured state to be $|0\rangle$ when $|\psi\rangle$ and $|\mu\rangle$ are different. This means that a single measured state being $|0\rangle$ does not mean that the assertion passes since there is 50% chance that the measured state is $|0\rangle$ even when $|\psi\rangle$ and $|\mu\rangle$ are orthogonal. Therefore, swap tests can only be used as a statistical approach to check the difference between the two input states. Furthermore, the swap test may entangle the two output states even when the measured state is $|0\rangle$.

One way to utilize the SWAP test for dynamic assertion is to assert whether a multi-qubit superposition state satisfies certain position equivalence. For example, for a 2-qubit state $|\psi\rangle = a_0|00\rangle + a_1|01\rangle + a_2|10\rangle + a_3|11\rangle$, the swap test will not raise an assertion error if a_1 is the same as a_2 and may raise an error if $a_1! = a_2$. In other words, the assertion passes when switching the positions of the two qubits has no impact on the quantum state.

C. Assertions in debugging quantum processes

Li and Ying [1] firstly studied the protocols for debugging quantum processes in which assertions are defined as projective measurements that are successively taken to monitor a quantum system until an error is detected. Consider a quantum system that is initially in state $|\psi_0\rangle$ and then evolves according to a computing process. Assume that the process is designed to transform the quantum system into state $|\psi_t\rangle$ at any time $t > 0$ during the evolution. A projection operator $P \neq 0$ of the system can be introduced as an assertion for debugging the process if the following condition holds:

- For a sequence of time points $t_1, t_2, \cdots$ of the process, $P|\psi_{t_k}\rangle = 0$ for all k.

The debugging protocol is as follows. The assertion P is implemented by a projective measurement $\{M_0 = I - P, M_1 = P\}$ which is successively inserted into the process at time $t_1, t_2, \cdots$ to check if the system state at time t_k is as anticipated $|\psi_{t_k}\rangle$. There are two cases of the measurement outcomes at time t_k:

1) The outcome is 0, then the system state is regarded as correct and the process goes on after the measurement. Note that if the system at time t_k is correctly in state $|\psi_{t_k}\rangle$, then the outcome is always 0, and the system state keeps unchanged after the measurement.
2) The outcome is 1, which means that an error of the system is detected at the time. In this case, one needs to stop the process and then carefully checks the process implementation to find the bugs. Note that if the system at time t_k is in an incorrect state ρ_k, then with probability $Tr(P\rho_k)$ the error would be detected at the time.

Suppose a bug of the system will be involved in the process at time t', then for time $t < t'$ the system is correctly in state $|\psi_t\rangle$ and for time $t \geq t'$ the system might be in some incorrect state. Using the debugging protocol for this process, an error can only be detected after time t'. So an error detected at time t_k means that $t' \in [0, t_k]$. In practice, we can run the debugging protocol many times, and get the smallest time t_k of detecting an error, then the bug is more likely located in $[t_{k-1}, t_k]$ and the relevant component (e.g, a quantum circuit fragment or a quantum subprogram) of the process should be carefully checked to find the bug.

A major advantage of this debugging approach is that the external measurement will not disturb the original evolution of the quantum system. So, the protocol is efficient, as the system can be checked many times in a single run of the process; moreover, the protocol is conclusive, i.e., no false positive results would be reported when the process runs correctly. The main problem of applying this approach in practice is how to find a satisfying assertion P. In [1] this problem is solved only for the simple case of time-independent evolution, i.e., the discrete-time transformation of the quantum system is achieved by a fixed unitary operation U; in this case, the measurement should be periodically taken in the process with some period p, and thus the assertion can be found according to the invariant subspace of U^p. The construction of assertions for debugging general quantum computing processes is further studied in following works, such as [2]–[5].

D. Dynamic Runtime Assertions

Zhou et al. [2] and Liu et al. [4] proposed to achieve dynamic runtime assertions by collecting the quantum state information through ancilla qubits. They define dynamic assertion as assertion checks that are performed during program execution and the program continues execution if there is no assertion error. They also showcased that dynamic assertions can be used for hardware error correction/mitigation. The proposed assertion circuits implement the types of assertions proposed by Huang et al [6]. And these circuits can be viewed as assertion primitives as they test specific quantum states. The later works [3], [5] generalize the design methodology for assertion circuits.

E. Projection Based Assertions

Li et al. [3] proposed projection based quantum assertion $assert(\bar{q}; P)$ for general quantum programs, where $\bar{q} = q_1, ..., q_n$ is a collection of quantum variables of the quantum program and P is a projection in the state space. The semantics of the assertion is that based on the projection operator P in the assertion, a projective measurement is constructed such that when applied upon $\bar{q}$, the assertion check passes if the measurement outcome corresponds to P and fails if it corresponds to $I - P$. With the projection operator, projection-

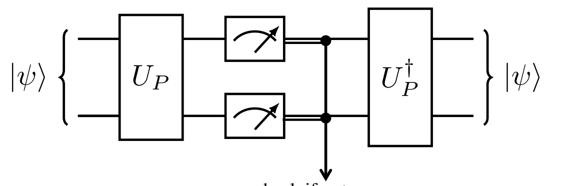


Fig. 2: Projection-based assertion circuit.

based assertion is a generic form and can be used to implement the assertions proposed in [6] as well as others [5], [7].

Since only projective measurement that lie in the computational basis is supported on the quantum computers available so far, additional unitary transformation is needed, as shown in Fig. 2. The state under test, $|\psi\rangle$, first goes through a unitary transformation U_P, which is designed based on the projection operator P. The key is the expected state will be mapped to $|0\rangle$. This way, if $|\psi\rangle$ matches the expected state, the measured state will be $|0\rangle$. An assertion failure is signaled if the measured state is not $|0\rangle$ and the program aborts execution. With the measurement property discussed in Section II, the measured state being $|0\rangle$ would force the state after measurement to be $|0\rangle$. Then, the inverse transformation $U_P^\dagger$ converts the $|0\rangle$ state back to the expected state, achieving error correction/mitigation.

F. Swap Based Assertions

Liu et al. [5] proposed a way to accomplish the assertion $assertEqual(|\psi\rangle, |\mu\rangle)$ using swap gates, where the state $|\psi\rangle$ is an n-qubit state under test and $|\mu\rangle$ is the expected state. Both states can be pure or mixed ones. Therefore, this assertion is more expressive than the three assertions proposed in statistical assertions [6]. The swap-based design can also be viewed as a way to build the U_P and $U_P^\dagger$ gates in Fig. 2 from a different perspective.

The circuit of swap-based assertions is shown in Fig. 3. The key idea is that the unitary gate $U^\dagger$ needs to transform the n-qubit "correct" state $|\mu\rangle$ to the zero state $|0\rangle^{\otimes n}$, and other "incorrect" states to the states in the computational basis which contain at least one $|1\rangle$. This way, when we measure the qubits after $U^\dagger$, if all the qubits are in the $|0\rangle$ state, the circuit won't raise an assertion error. If any of the qubit is in $|1\rangle$ state, it means that the state $|\psi\rangle$ was not the same as the $|\mu\rangle$ state, and the circuit would raise an assertion error.

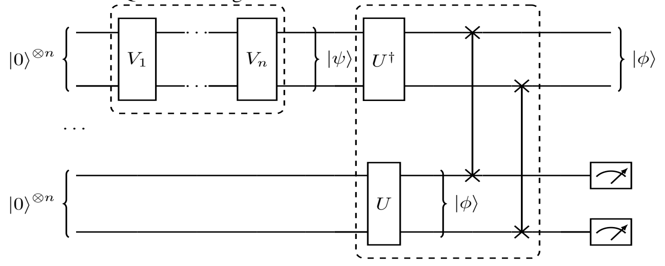


Fig. 3: The idea of SWAP-based assertion circuits.

The U or $U^\dagger$ gate can be generated as follows. First, from the expected state $|\mu\rangle$, we find an orthonormal basis $\{|\psi_i\rangle\}_{i\in[0,2^n-1]}$ that includes state $|\mu\rangle$ as $|\psi_0\rangle$ using the Gram-Schmidt process [10]. Second, we generate an unitary matrix $U^\dagger$ to transform the states in this orthonormal basis to the states in the computational basis, and $U^\dagger = |0\rangle^{\otimes n}\langle\psi_0| + |0\rangle^{\otimes n-1}|1\rangle\langle\psi_1| + ... + |1\rangle^{\otimes n}\langle\psi_{2^n-1}|$. This way, when applying $U^\dagger$ to $|\psi\rangle$, if $|\psi\rangle == |\mu\rangle$, the output would be $|0\rangle$. Otherwise, one or more of the measurement outcomes would be 1.

Depending on where we place the U gate and $U^\dagger$ gate w.r.t. the SWAP gate in Fig. 3, there would be different circuit designs. The design shown in Fig. 4 is typically preferred as it enables more opportunities for transpiler optimization to reduce the circuit complexity. From Fig. 2 and Fig. 4, we can see similarity between the two approaches.

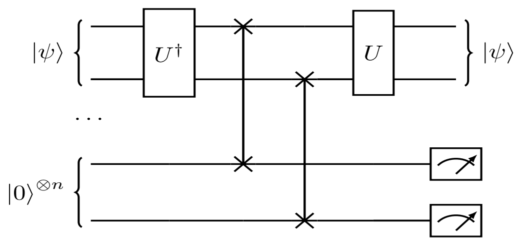


Fig. 4: General scheme of SWAP based assertion circuit for pure state assertions.

Another optimization proposed by Liu et al. [5] is based on the observation that an assertion failure happens if any of the state after $U^\dagger$ is $|1\rangle$. Therefore, we can use a logical OR operation to reduce the number of ancilla qubits, as shown in Fig. 5.

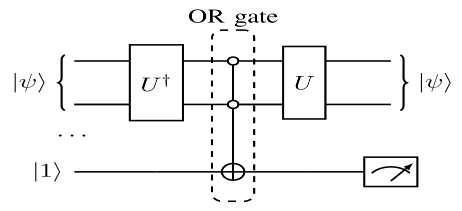


Fig. 5: Logical OR based assertion circuits.

When asserting a mixed state, the density matrix of the expected state needs to be diagonalized. Then, depending on the rank of the density matrix, different numbers of qubits need to be measured for assertion checks [5]. A similar process is also used in [3].

G. NDD/Stabilizer Based Assertions

Non Destructive Discrimination (NDD) is a protocol leveraged by Liu et al. [4], [5] for quantum assertions. As shown in Fig. 6, the NDD-based design consists of a controlled-N gate and two Hadamard (H) gates. When the input quantum state $|\psi\rangle$ is the eigenstate of the unitary matrix N with the eigenvalue of 1, i.e., $|\psi\rangle = N|\psi\rangle$, the ancilla qubit's output state is $|0\rangle$. When the quantum state $|\psi\rangle$ is the eigenstate of the unitary matrix U with the eigenvalue of -1, the ancilla qubit's output state is $|1\rangle$. When designing for the assertion $assertEqual(|\psi\rangle, |\mu\rangle)$, where $|\psi\rangle$ is the state under test and

$|\mu\rangle$ is the expected state, the key is to design the unitary N such that $N|\mu\rangle = |\mu\rangle$. From the description, we can see that the NDD based assertion check is essentially the stabilizer designed for the expected state. Compared to the stabilizers that are widely used in quantum error correction (QEC), two distinctions exist. (1) In QEC, the quantum states are encoded such that the codespace satisfies the corresponding stabilizers. For example, with 3-qubit repetition code, the qubit states are encoded such that they pass the stabilizers, ZZI and IZZ, if there is no single bit-flip error. In comparison, in assertions, the qubits are not encoded on purpose. (2) As it is sufficient for QEC to detect and correct X and Z Pauli errors, the stabilizers are mainly Pauli Z and X stabilizers. In contrast, for assertions, the stabilizers are specific for expected states.

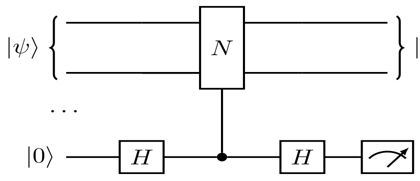


Fig. 6: General scheme of NDD/Stabilizer-based assertion circuits.

The unitary gate or the stabilizer N can be designed for assertion $assertEqual(|\psi\rangle, |\mu\rangle)$ as follows. First, based on the expected state $|\mu\rangle$, we find an orthonormal basis $\{|\psi_i\rangle\}_{i\in[0,2^n-1]}$ that includes state $|\mu\rangle$ as $|\psi_0\rangle$. Second, we can construct the unitary matrix N as $N = |\psi_0\rangle\langle\psi_0| - \sum_{i=1}^{2^n-1} |\psi_i\rangle\langle\psi_i|$.

Similar to the projection-based assertions or swap-based assertions, a passing assertion forces/stabilizes the state under test to the same as the expected state. Therefore, we can abort program execution if an assertion fails or use post-selection to only consider the runs that pass the assertion checks to mitigate hardware errors.

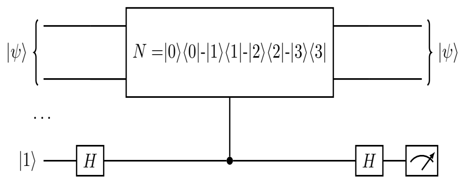
H. *Assertions for Symmetry States*

Besides checking whether a quantum state is the same as an expected one using $assertEqual(|\psi\rangle, |\mu\rangle)$, other properties can be checked during program execution. One such example is symmetry assertion [7], [9]. In some application domains, such as finding max cuts in graphs, the information encoding is inherently symmetric. For example, a bi-partition of nodes in a graph encoded with '110001' is the same as '001110' as one can label the partition 0 and partition 1 interchangeably. This symmetry is referred to bit-flip symmetry [9]. Similarly, the location of a node in the graph does not alter the partition, which is referred to as permutation symmetry [9].

The assertion circuits to verify the symmetric property are very similar to the NDD/stabilizer assertion circuit. A simple example is the circuit asserting the $|+\rangle$ state, for which the N gate in Fig. 6 becomes an X gate as shown in [4]. The $|+\rangle$ state satisfies bit-flip symmetry as flipping the bit has no effect. The X gate is the stabilizer of the $|+\rangle$ state, i.e., passing the assertion means that the output state is the $|+\rangle$ state.

I. *Assertions for Memberships / Approximate Assertions*

Liu et al. [5] proposed assertions to check whether a quantum state under test belongs to a given set of states or a superposition of the states in the set. Such assertions are referred to as approximate assertions in [5]. Approximate assertions can be implemented using the swap-based or NDD/stabilizer-based approach. It is similar to asserting for mixed states. Compared to pure state precise assertions, the difference is that there would be multiple correct states rather than a single correct state. For example, to check a 2-qubit state $|\psi\rangle \in \{|01\rangle, |10\rangle, |11\rangle\}$, the unitary matrix of the N gate in Fig. 6 can be constructed as $N = |01\rangle\langle 01| + |10\rangle\langle 10| + |11\rangle\langle 11| - |00\rangle\langle 00|$. Or it can be implemented using $N = |00\rangle\langle 00| - |01\rangle\langle 01| - |10\rangle\langle 10| - |11\rangle\langle 11|$ with the ancilla qubit input as $|1\rangle$, as shown in Fig. 7.



$|\psi\rangle$
$N = |0\rangle\langle 0| - |1\rangle\langle 1| - |2\rangle\langle 2| - |3\rangle\langle 3|$
$|\psi\rangle$
$\dots$
$|1\rangle$
H
H

Fig. 7: Approximate Assertion for $|\psi\rangle \in \{|01\rangle, |10\rangle, |11\rangle\}$ or $|\psi\rangle! = |00\rangle$.

Note that for a 2-qubit state, the assertion $|\psi\rangle \in \{|01\rangle, |10\rangle, |11\rangle\}$ is essentially checking $|\psi\rangle \neq |00\rangle$. Such assertions are common for applications where some bit encoding are known to be impossible/forbidden. For example, in a graph 3-coloring problem, if three colors are encoded with $|01\rangle, |10\rangle, |11\rangle$, for any node, the color encoding $|00\rangle$ would be impossible, i.e., we can assert that $|q_1q_2\rangle \neq |00\rangle$, where q_1 and q_2 are the qubits encoding the color of a node.

In general, assertion $assertNotEqual(|\psi\rangle, |\mu\rangle)$, i.e., $|\psi\rangle \neq |\mu\rangle$, where $|\psi\rangle$ is the n-qubit state under test and $|\mu\rangle$ is a known impossible state, can be constructed as approximate assertion $|\psi\rangle \in$\{space orthogonal to $|\mu\rangle$\}. When $|\mu\rangle$ is a pure state, the assertion circuit can be designed as follows. First, based on the known state $|\mu\rangle$, we find an orthonormal basis $\{|\psi_i\rangle\}_{i\in[0,2^n-1]}$ that includes state $|\mu\rangle$ as $|\psi_0\rangle$. Second, we can construct the unitary matrix N as $N = |\psi_0\rangle\langle\psi_0| - \sum_{i=1}^{2^n-1} |\psi_i\rangle\langle\psi_i|$. Then, we can use this stabilizer with the ancilla qubit input as $|1\rangle$, similar to Fig. 7, for assertion checks. When $|\mu\rangle$ is specified as a density matrix or a mixed state, the design process is similar except that N would have more components with eigenvalues of 1.

For assertion $assertNotEqual(|\psi\rangle, |\mu\rangle)$, where both $|\psi\rangle$ and $|\mu\rangle$ are states under test and cannot be known statically, we need to construct the combined qubit-state set by listing all the states that do not violate the condition and then use approximate assertion to ensure that the combined state $|\psi \otimes \mu\rangle$ satisfies the membership check of the set. For example, in the graph 3-coloring problem, one can assert that for each pair of nodes that are connected with an edge, their color encoding should be different, e.g., $|q_1q_2\rangle \neq |q_3q_4\rangle$, which can be implemented with $|q_1q_2q_3q_4\rangle \in$

$\{|0110\rangle, |0111\rangle, |1001\rangle, |1011\rangle, |1101\rangle, |1110\rangle\}$ when the color is encoded with $|01\rangle$,$|10\rangle$, and $|11\rangle$.

Approximate assertion can also be used for precisely checking properties of quantum states, such as parity and qubit position equivalence. For example, for a 3-qubit state, one can include all the even parity states in the membership check, i.e., $|\psi\rangle \in \{|000\rangle, |011\rangle, |101\rangle, |110\rangle\}$. As an another example, the membership check $|\psi\rangle \in \{|00\rangle, |11\rangle, |01\rangle + |10\rangle\}$ would result in the swap test circuit.

J. A Comparison Among the Assertions

Among the assertion schemes discussed above, the statistical assertions [6] and the ones using swap-tests [8] need numerous measurements to infer the statistical significance. Between the two, the statistical assertions require measurements of multiple qubits whereas swap-tests need to measure a single ancilla qubit to compare the difference between the state under test and the expected state.

Assertion primitives [2], [4], Projection-based assertions [3], Swap-based assertions [5], NDD/stabilizer-based assertions [5], symmetry assertions [9], and approximate assertions [5] won't affect subsequent program execution when there is no assertion error. They can also be used for error correction/mitigation. The projection-based assertion does not involve ancilla qubits but require mid-circuit measurement. The complexity of these assertion circuits varies for different scenarios and one may explore different assertion approaches to select the one with lowest implementation complexity, e.g., in terms of circuit depth or the number of gates.

IV. Case Study: Assertions for Software Debugging

Assertions are useful for software debugging. To facilitate programmers, the tket framework [13] has implemented two types of assertions: the projection-based assertion [3], and the stabilizer-based assertion. Moreover, Liu et al. [5] have augmented the Qiskit [12] to incorporate the SWAP-based assertion and NDD/stabilizer-based assertion so that programmers can use these two types of assertions conveniently. In this section, we study how to leverage these assertion methods and compare these methods available in tket and Qiskit.

The circuit of this case study is a 3-qubit cluster state circuit. The cluster state can be prepared by firstly applying the Hadamard gate to each qubit, then applying the controlled-Z gate between all the adjacent qubits. Listing 1 and Listing 2 show the Qiskit code and the tket code for generating the cluster state with the assertions, respectively.

Listing 1 illustrates how to insert NDD/stabilizer-based assertion in Qiskit code. Two assertions are inserted in line 21 and line 30. The assertion in line 21 is at the location after the circuit applies the H gate to all the qubits, so the correct state at this location should be $|+++\rangle$. We save this state vector into the stateList as state0 in line 13, and use it for assertion in line 21. The second assertion located in line 30 asserts the final state of the circuit, which should be the cluster state $\frac{1}{\sqrt{2}}(|+0+\rangle + |-1-\rangle)$. We save this cluster state as state1 in line 13 and use it for asserting the final output state in line 30.

Listing 2 illustrates how to insert stabilizer-based assertion in tket code. Although the stabilizer-based assertion in tket generates the assertion circuit shown in Fig. 6 just as the NDD/stabilizer-based assertion implemented in Qiksit, these two assertion schemes are different in creating the unitary N in the assertion circuit. For the stabilizer-based assertion in tket, the unitary gate in the assertion circuit must be composed of Pauli gates, while the NDD/stabilizer-based assertion in Qiskit can use any kind of gates to create the unitary N in Fig. 6, which means the NDD/stabilizer-based assertion has more flexibility. Moreover, for the stabilizer-based assertion in tket, users need to provide a set of Pauli operators which will be used to create the unitary gate in the assertion circuit. And if users make some mistakes and provide non-ideal Pauli operators, the assertion can be inefficient/ineffective in detecting errors.

We didn't list the code for using SWAP-based assertion and Projection-based assertion. The reason is that the usage of these two types of assertions is very similar to Listing 1 and Listing 2. For SWAP-based assertion, users can also specify the design type to choose between the SWAP-based assertion and the logical-OR-based assertion. For the Projection-based assertion, users need to provide the matrix of the projector.

The circuits with assertions generated from the Qiskit code and the tket code are all sent to Aer Simulator from Qiskit to run without noise. The number of shots is set to be 1000. If all the 1000 shots do not raise an error for an assertion, then this assertion succeeds. Otherwise, it fails. Table I shows the assertion results for the circuits which contain different bugs. The assertion results for the bug-free circuit are not included in the table since there is no assertion failure for bug-free circuits. Two kinds of bugs are considered. Bug 1 misses the H gate on the third qubit of the circuit, and bug 2 mistakenly replaces all the controlled-Z gates of the circuit with controlled-X gates. Using the Qiskit code in Listing 1 as an example, bug 1 can happen when we incorrectly set the iteration number of the first For loop to be $nqubits - 1$ in line 17, and bug 2 can happen when we wrongly apply CX gate instead of CZ gate in line 27.

In Table I, the first two lines of results correspond to stabilizer-based assertion 1 and 2, the difference between these two assertions is the user-defined Pauli operator sets. For stabilizer-based assertion 1, we set the two Pauli operator sets just as line 15 and line 26 of Listing 2. For stabilizer-based assertion 2, we set the first Pauli operator sets to be $\{XXI\}$ and the second to be $\{XIX\}$. From Table I, we can see for bug 1, stabilizer-based assertion 1 on both of the two locations fails, because bug 1 happens before these two locations; for bug 2, stabilizer-based assertion 1 on location 2 fails because bug 2 happens after location 1 and before location 2. However, for the stabilizer-based assertion 2, when the circuit has bug 1, assertion succeeds in location 1 as it doesn't identify the incorrect state of $|++0\rangle$. The reason is that the Pauli operator XXI does not only stabilize the correct state $|+++\rangle$ but also stabilizes the incorrect state of $|++0\rangle$ as both of these two states have +1 eigenvalue corresponding to this Pauli operator XXI. The same happens for the circuit with bug 2, the stabilizer-based assertion 2 at location 2 succeeds because it doesn't distinguish between the incorrect state and the correct state. Therefore, it is important that the user defines the proper Pauli operators to make an effective stabilizer-based assertion in tket. Compared to the stabilizer-based assertion in tket, NDD/stabilizer-based assertion in Qiskit is more user-friendly since the unitary gate of the stabilizer circuit is generated by Qiskit's function `UnitaryGate` and users don't need to define the gate using Pauli operators.

TABLE I: Assertion results for different types of assertions

Assertion type	Bug 1		Bug 2	
	Location1	Location2	Location1	Location2
Stabilizer-based assertion 1	Fail	Fail	Succeed	Fail
Stabilizer-based assertion 2	Succeed	Fail	Succeed	Succeed
NDD/stabilizer-based assertion	Fail	Fail	Succeed	Fail
Projector-based assertion	Fail	Fail	Succeed	Fail
SWAP-based assertion	Fail	Fail	Succeed	Fail
Logical-OR-based assertion	Fail	Fail	Succeed	Fail

```
#Create a QuantumRegister with 3 qubits
nqubits=3
qr = QuantumRegister(nqubits, 'q')

#ancilla qubit for assertions
nqubits_ancilla=2
ar = QuantumRegister(nqubits_ancilla, 'a')

#classical bit to save the measurement result
cr = ClassicalRegister(nqubits+nqubits_ancilla, 'c')

#a list of state vectors that we are asserting for
stateList=[state0, state1]

circuit = QuantumCircuit(qr,ar,cr)
#apply H gate to all the qubits
for q in range(nqubits):
    circuit.h(q)

#insert an NDD/stabilizer assertion
assertion_NDD(circuit, qr[0:3], [ar[0]], [stateList
    [0]])
#measure the ancilla qubit ar[0]
circuit.measure(ar[0],cr[nqubits:nqubits+1])

#apply CZ gate to adjacent qubit
for q in range(nqubits-1):
    circuit.cz(q,q+1)

#insert an NDD/stabilizer assertion
assertion_NDD(circuit, qr[0:3], [ar[1]], [stateList
    [1]])
#measure the ancilla qubit ar[1]
circuit.measure(ar[1],cr[nqubits+1:nqubits+2])

#measure the circuit output
circuit.measure(qr,cr[0:nqubits])
```

Listing 1: The Qiskit code of generating the cluster state with assertions.

```
#create a Quantum circuit with 3 qubits
nqubits=3
circuit = Circuit(nqubits)

#specify a qubit list which will be used for
    assertion
qubit_list=[]
for i in range(nqubits):
    qubit_list.append(i)

#apply H gate to all the qubits
for i in range(nqubits):
    circuit.H(i)

#define a set of Pauli operators for generating the
    stabilizer
stabilizer1 = ['XXX']
# add an ancilla qubit for this stabilizer-based
    assertion
circuit.add_qubit(Qubit(nqubits+1))
#insert the stabilizer-based assertion
circuit.add_assertion(stabilizerAssertionBox(
    stabilizer1), qubit_list, ancilla=nqubits+1,
    name="|assertion1>")

#apply CZ gate to all the adjacent qubits
for i in range(nqubits-1):
    circuit.CZ(i,i+1)

#define a set of Pauli operators for generating the
    stabilizer
stabilizer2 = ['XZI','ZXZ','IZX']
# add an ancilla qubit for this stabilizer-based
    assertion
circuit.add_qubit(Qubit(nqubits+2))
#insert the stabilizer-based assertion
circuit.add_assertion(stabilizerAssertionBox(
    stabilizer2), qubit_list, ancilla=nqubits+2,
    name="|assertion2>")
```

Listing 2: The tket code of generating the cluster state with assertions.

V. Case Study: Assertions for Hardware Error Mitigation

In this section, we study the error mitigation effects when adding the NDD/stabilizer-based assertion into a quantum circuit. The circuit under study is an Iterative Phase Estimation (IPE) circuit. The IPE circuit is used to estimate the phase of a unitary operator and realizes the same function as the Quantum Phase Estimation (QPE) circuit. Compared to the QPE circuit which may suffer severely from circuit noise when the circuit size grows, IPE circuits utilize reset gates and conditioned gates to reduce the circuit noise effects. Fig. 8 shows the circuit for a 2-qubit IPE circuit. This circuit estimates the phase of a single-qubit Z-rotation gate, and the rotation angle is set to $\frac{6\pi}{4}$. Therefore, we can see the phase φ of this gate is $\frac{3}{4}$ because $2\pi\varphi = \frac{6\pi}{4}$. If we use a binary number to represent the decimal part of this phase, it is $\varphi = 0.\varphi_1\varphi_2 = 0.11$, so the precision of this gate phase is 2 and we can use the circuit containing 2 iterations to estimate this phase. The first iteration estimates the value of φ_2 by performing mid-circuit measurement of the ancilla qubit $q0$, then in the second iteration, the ancilla qubit $q0$ will be reset and will be used to estimate the value of φ_1.

From the IPE circuits in Fig. 8, we can see the second qubit $q1$ first goes through an X gate, and after the X gate or after the first grey dotted line in Fig. 8, the state of $q1$ should remain $|1\rangle$ no matter how many controlled-phase gates are applied. This creates an opportunity for inserting assertions on $q1$ because we can actually insert the assertions at any location we want, and the state to be asserted is $|1\rangle$. The assertion type we used in the IPE circuit is the NDD/stabilizer-based assertion. By using post-selection to filter out the results with assertion errors, we can realize error mitigation effects. Since we want to precisely assert qubit $q1$, and the state we want to assert is $|1\rangle$, we can easily know that the Pauli operator $-Z$ can be used to stabilize the state $|1\rangle$, and the circuit cost of the CNOT gate number for inserting such one NDD/stabilizer-based assertion is just 1, which means our assertion will not adding too much gate noise to the IPE circuit.

IPE circuit is a dynamic circuit that contains conditioned gates which perform operations conditioned by the classical information generated from previous measurements. In order to run a dynamic circuit with conditioned gates on a real IBM machine, the circuit is converted into the OpenQASM3 program and then sent to the machine which supports OpenQASM3 to run. The IBM machine `ibm_perth` which supports OpenQASM3 is selected to run the experiments and the result is shown in Table II. All of these three IPE circuits in the table estimate the phase for a 1-qubit unitary gate with different numbers of iterations 2, 3, and 4, which compute the phase with different precision, 2, 3, and 4, respectively. In this case study, We run the experiments with one assertion inserted in one of the iterations or with multiple assertions inserted in each of the iterations. We choose the assertion location to be right before the first controlled-phase gate of each of the circuit iterations, and the location is marked with the grey dotted line in Fig. 8.

We run the circuit and calculate the success rate based on the output distributions, and the success rate is calculated as the ratio of the number of shots of getting the correct output over the total number of shots, and the total number of shots is set to be 1000 in our experiments. For the circuit with assertions, we first drop all the output results which have assertion errors, then calculate the success rate using the output results which don't have assertion errors. From the results shown in Table II, we can see that without the NDD/stabilizer-based assertions, the successful rate decreases quickly as the number of iterations increases. After we add the assertions, almost all the circuits have a great improvement in success rate and the largest success rate improved can be 3.19 times compared with the original circuit without assertion, except for the IPE with 2 iterations and with assertions inserted into both iterations. It indicates that when the circuit size is small, it is better to not include too many assertions in the circuit at the same time, which incurs too much noise.

VI. Conclusion

In this paper, we present a detailed overview of different quantum assertion techniques proposed recently and discuss

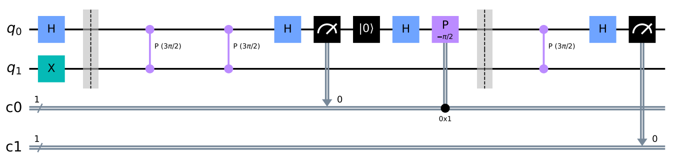


Fig. 8: 2-qubit IPE circuit with two iterations

TABLE II: Success rates of the IPE circuit with/without assertions

	no assertion	assertion in iteration1	assertion in iteration2	assertion in iteration3	assertion in iteration4	assertions in all the iterations
IPE with 2 iteration	0.731	0.731	0.918			0.301
IPE with 3 iteration	0.371	0.604	0.752	0.828		0.777
IPE with 4 iteration	0.192	0.495	0.663	0.805	0.661	0.533

different types of assertion checks that can be supported with them. We highlight that with non-destructive measurement, i.e., the stabilizer designed specific to the expected states, assertions can be effective in either software debugging or hardware error correction/mitigation. Case studies are presented to illustrate the usage of assertions in different quantum algorithms.

Our future research directions on quantum assertion include (a) expanding the scope of assertions such that they can check more invariants in quantum programs, and (b) reducing the cost of assertion circuits such that they can be more effective in error mitigation. One possible way is to break down a mixed state assertion into a combination of a few approximate assertions.

References

[1] Y. Li and M. Ying. Debugging quantum processes using monitoring measurements. Phys. Rev. A, 89 (2014), Apr, 042338. https://doi.org/10.1103/PhysRevA.89.042338.

[2] H. Zhou and G. T. Byrd. Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation. Comput. Archit. Lett. 2019.

[3] G. S. Li, L. Zhou, N. K. Yu, Y. F. Ding, M. S. Ying and Y. Xie, Projection-Based Runtime Assertions for Testing and Debugging Quantum Programs, Proceedings of the ACM on Programming Languages, 4.OOPSLA(2020): 1-29.

[4] J. Liu, G. Byrd and H. Zhou, Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation, In:Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2020), 1017-1030.

[5] J. Liu and H. Zhou, Systematic Approaches for Precise and Approximate Quantum State Runtime Assertion, In:Proceedings of the 27th IEEE International Symposium on High-Performance Computer Architecture (HPCA 2021), 179-193..

[6] Y. P. Huang and M. Martonosi, Statistical assertions for validating patterns and finding bugs in quantum programs, In: Proceedings of the 46th International Symposium on Computer Architecture (ISCA 2019), ACM, 541-553.

[7] X. Bonet-Monroig, R. Sagastizabal, M. Singh, and T. E. O'Brien. Low-cost error mitigation by symmetry verification. Phys. Rev. A, 98:062339, Dec 2018.

[8] Yongshan Ding and Frederic T. Chong. Quantum Computer Systems: Research for Noisy Intermediate-Scale Quantum Computers, Synthesis Lectures on Computer Architecture, Morgan Claypool Publishers, 2020.

[9] Ruslan Shaydulin and Alexey Galda. Error mitigation for deep quantum optimization circuits by leveraging problem symmetries. arXiv preprint arXiv:2106.04410, 2021

[10] G. H. Golub and C. F. Van Loan, *Matrix computations*. JHU press, 2012, vol. 3.

[11] M. A. Nielsen and I. Chuang, "Quantum computation and quantum information," 2002.

[12] ANIS, M., et al., Qiskit: An Open-source Framework for Quantum Computing. (2021)

[13] Sivarajah, S., Dilkes, S., Cowtan, A., Simmons, W., Edgington, A. & Duncan, R. $t|ket\rangle$: a retargetable compiler for NISQ devices. *Quantum Science And Technology*. **6**, 014003 (2020,11), https://doi.org/10.1088/2058-9565/ab8e92