1 Execution Example

Execution result of the example

2 Comparison of Result to Traditional Quantum Simulators

In this section, we align the results of our novel implementation with those obtained from Qiskit's entire suite of simulators. Our aim is to verify that our results are consistent with those of well-established, widely-used simulators.

2.1 Chi-Square Goodness-of-Fit Test

To compare our results with those of the extant quantum simulators, we deployed the Chi-Square Goodness-of-Fit Test, a variant of Pearson's chi-square test. This test is employed to determine if the observed distribution of a particular execution result, which is treated as a discrete random variable, deviates from the expected distribution. This expectation is derived from the result of the result of our implementation.
For the reliability of a Chi-Square goodness-of-fit test, a widely accepted rule of thumb stipulates that the expected frequencies should be no less than 5 for all potential outputs. Consequently, we adhered to this guideline in conducting our experiment.
The p-value of the Chi-Square goodness-of-fit test is the probability of obtaining statistical results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct. A smaller p-value indicates that the observed data is less likely under the null hypothesis, and at a certain threshold (often 0.05), we might reject the null hypothesis in favor of the alternative hypothesis.
In our comparative analysis between the results of our implementation and those of Qiskit's simulators, we formed the null hypothesis stating, "the outcomes of the Qiskit simulator conform to the probability distribution derived from our implementation". The Chi-Square goodness-of-fit test was consequently conducted under this presupposition.

2.2 Compared Simulators

We conducted a comparison of the results from twelve quantum simulators provided by Qiskit to determine whether they align with the probability distribution derived from our implementation.

Quantum Simulator	**Explanation**	**Dynamic Circuit Support**
`aer_simulator`	Explanation for Simulator 1	?
`aer_simulator_statevector`	Explanation for Simulator 2	?
`aer_simulator_density_matrix`	Explanation for Simulator 3	?
`aer_simulator_stabilizer`	Explanation for Simulator 4	?
`aer_simulator_matrix_product_state`	Explanation for Simulator 5	?
`aer_simulator_extended_stabilizer`	Explanation for Simulator 6	?
`aer_simulator_unitary`	Explanation for Simulator 7	?
`aer_simulator_superop`	Explanation for Simulator 8	?
`qasm_simulator`	Explanation for Simulator 9	?
`statevector_simulator`	Explanation for Simulator 10	?
`unitary_simulator`	Explanation for Simulator 11	?
`pulse_simulator`	Explanation for Simulator 12	?

Table 1: Qiskit Quantum Simulators

2.3 Benchmarks

To compare the behaviors of both static and dynamic quantum circuits between our implementation and Qiskit Aer simulators, we executed a total of 34 benchmark programs from QASMBench[ ] and all examples found in the OpenQASM2.0 specification.

2.3.1 QASMBench

We executed all the small-scale benchmarks specified in the QASMBench paper along with some of the medium-scale benchmarks that involve 8 qubits or less, using our implementation (refer to Table 2). This enabled us to derive the expected probability distribution for each benchmark program.

Index	Benchmark	Scale	Qubits	Gates	CX
1	adder	small	4	23	10
2	basis_change	small	3	53	10
3	basis_trotter	small	4	1626	582
4	bell_state	small	2	3	1
5	cat_state	small	4	4	3
6	deutsch	small	2	5	1
7	dnn	small	2	268	84
8	fredkin_n3	small	3	19	9
9	qec_dist3	small	5	114	49
10	grover	small	2	16	2
11	hs4	small	4	28	4
12	inverseqft	small	4	8	0
13	iSWAP	small	2	9	2
14	linearsolver	small	3	19	4
15	lpn	small	5	11	2
16	pea	small	5	98	42
17	qaoa	small	3	15	6
18	qec_sm	small	5	5	4
19	qec_en	small	5	25	10
20	qft	small	4	36	12
21	qrng	small	4	4	0
22	quantumwalks	small	2	11	3
23	shor	small	5	64	30
24	toffoli	small	3	18	6
25	teleportation	small	3	8	2
26	jellium	small	4	54	16
27	vqe	small	4	89	9
27	vqe_uccsd	small	4	220	88
28	wstate	small	3	30	9
29	dnn	medium	8	1200	384
30	bb84	medium	8	27	0
31	qaoa	medium	6	270	54
32	simons	medium	6	44	14
33	vqe_uccsd	medium	8	10808	5488
34	hhl	medium	7	689	196

Table 2: QASMBench

2.3.2 OpenQASM2.0 Specification

All the benchmark programs listed in Table 2 are static circuits. To facilitate a comparison of dynamic circuit behaviors, we incorporated dynamic circuit benchmarks as introduced in the OpenQASM2.0 specification.

2.4 Experiment

3 Comparisons to Previous Works

In this section, we highlight the potential for using new ways to test quantum simulators that haven't been tried before.

Index	Benchmark	S/D	Qubits	Gates	CX
1	inverse QFT followed by measurement	dynamic	?	?	?
2	quantum error correction	dynamic	?	?	?
3	quantum fourier transform	static	?	?	?
4	quantum process tomography	static	?	?	?
5	quantum teleportation	dynamic	?	?	?
6	randomized benchmarking	static	?	?	?
7	ripple carry adder	static	?	?	?

Table 3: OpenQASM2.0 Specification Examples

3.1 Problems

Past techniques for testing quantum simulators, like QDiff and MorphQ, have made use of two-sample tests, specifically the Kolmogorov-Smirnov test (K-S test). There are two main issues with this method.

1. Two-sample testing is carried out without the expected probability distribution that the simulation results should match, so the test's accuracy is naturally lower.

2. The K-S test is basically a statistical test for a continuous probability distribution, which isn't a good fit for our situation where we have a discrete probability distribution. QDiff uses a modified version of the K-S test for discrete cases, but calculating the p-value in this situation is not easy. In MorphQ, the p-value is found using the standard K-S test for a continuous probability distribution without any changes, but it's unclear whether the p-value obtained in this way is meaningful.

3.2 One-sample testing

3.3 Experiment