# Eliminating Redundant Computation in Noisy Quantum Computing Simulation

Gushu Li
*ECE Department*
*University of California*
Santa Barbara, USA
gushuli@ece.ucsb.edu

Yufei Ding
*CS Department*
*University of California*
Santa Barbara, USA
yufeiding@cs.ucsb.edu

Yuan Xie
*ECE Department*
*University of California*
Santa Barbara, USA
yuanxie@ece.ucsb.edu

***Abstract*—Noisy Quantum Computing (QC) simulation on a classical machine is very time consuming since it requires Monte Carlo simulation with a large number of error-injection trials to model the effect of random noises. Orthogonal to existing QC simulation optimizations, we aim to accelerate the simulation by eliminating the redundant computation among those Monte Carlo simulation trials. We observe that the intermediate states of many trials can often be the same. Once these states are computed in one trial, they can be temporarily stored and reused in other trials. However, storing such states will consume significant memory space. To leverage the shared intermediate states without introducing too much storage overhead, we propose to statically generate and analyze the Monte Carlo simulation simulation trials before the actual simulation. Those trials are reordered to maximize the overlapped computation between two consecutive trials. The states that cannot be reused in follow-up simulation are dropped, so that we only need to store a few states. Experiment results show that the proposed optimization scheme can save on average 80% computation with only a small number of state vectors stored. In addition, the proposed simulation scheme demonstrates great scalability as more computation can be saved with more simulation trials or on future QC devices with reduced error rates.**

***Index Terms*—quantum computing, simulation, noise**

## I. Introduction

Quantum Computing (QC) has attracted great interest from both academia and industry in the last decades due to its great potential in accelerating various important applications, such as integer factorization [1], database search [2], and molecule simulation [3]. Recently, several Noisy Intermediate-Scale Quantum (NISQ) devices have been released [4]–[6] and *Quantum Supremacy* has been experimentally demonstrated [7], indicating that the advantages of quantum computing against classical computing is achievable.

Ideally, quantum algorithms should be executed on realistic NISQ hardware for evaluation. However, NISQ devices require an extreme execution environment and most of them remain in physics laboratories. Existing QC cloud services, e.g., IBM Quantum Experience [8], Rigetti's QPU [9], only provide limited access which cannot satisfy the ever-increasing demand for experiments to evaluate new NISQ algorithm/hardware designs. Therefore, noisy QC simulation that could take various noise effects [10] into consideration is still a practical way for algorithm development and evaluation in the NISQ era.

Monte Carlo simulation is widely adopted in noisy QC simulation [9], [11], [12] but it is very time-consuming. In such simulation, noise effects can be treated as errors that are randomly injected during the computation. To model such random effects, the same input quantum program needs to be simulated for a large number of times, and in each simulation trial, errors are randomly injected based on an error model of the target NISQ device. Previous QC simulation optimizations, no matter from the algorithm level [13]–[19] or the system level [12], [20]–[24], focus on single trial simulation optimization while little consideration has been given to the inter-trial optimization.

Orthogonal to these prior QC simulation optimizations, we observe that there exists significant redundant computation which is never leveraged in existing Monte Carlo noisy QC simulation [9], [11], [12]. For multiple error injected Monte Carlo simulation trials, it is possible that they share the same intermediate states. Such shared intermediate states can be temporarily stored and reused among different trials to save computation. However, these reusable intermediate states are often hidden in the huge numbers of trials. It is thus critical to have an efficient and effective heuristics for locating these shared states and maximizing the reused computation. Meanwhile, saving a state takes significant memory space, which may limit the size of the program that could be simulated. Therefore, it would be desirable to remove redundant computation with the stored intermediate state as few as possible.

To this end, we propose a Monte Carlo simulation trial reorder scheme to 1) efficiently identify and remove the computation redundancy in the Monte Carlo noisy QC simulation, 2) minimize the number of stored intermediate states. Our optimization scheme will not affect the final simulation result since it is mathematically equivalent to the original simulation. Specifically, instead of direct running the Monte Carlo simulation, we first generate all the simulation trials without actually running the simulation. We statically analyze the generated trials and reorder them based on the locations of the injected errors. The overlapped computation between two consecutive trials is maximized so that more computation results can be shared and reused. Moreover, we dynamically

This work was supported in part by NSF 1730309 and 1925717.

978-1-7281-1085-1/20/$31.00 ©2020 IEEE

drop the intermediate states that cannot be reused in the follow-up computation to reduce the memory requirement for temporary intermediate state storage.

We evaluate the noisy simulation optimization scheme on both realistic NISQ devices models and artificial models of larger sizes expected in the future. Experiment results show that we can save around 80% computation on average with only a small number of state vectors stored at most on a realistic NISQ device model. The test on larger NISQ device models demonstrates that our noisy QC simulation optimization has great scalability as it could save even more computation when simulating future NISQ devices with lower error rates and more simulation trials.

## II. Related Work

In this section, we summarize related work about noisy QC simulation and QC simulator optimizations.

**Noisy QC Simulator** Several existing QC simulators have supported error modeling and noisy simulation, such as IBM QISKit [11], QX [12], and Rigetti QVM [9]. These simulators model a realistic quantum processor with straight forward Monte Carlo simulation while none of them leverages the shared computation among multiple simulation trials.

**Density Matrix Simulation** Another approach in noisy QC simulation is to manipulate the density matrix of a quantum system [23], [25]. The density matrix approach could model the noise effect in one simulation trial but the size of the density matrix of a $N$-qubit system will be $2^{2N}$, which is much larger than that of a state vector. This paper focuses on state vector simulation which is capable of simulating a system of more qubits with the same hardware resources.

**QC Simulator Optimization** Previous optimizations for QC simulators can be summarized into two categories. Some simulators increase the simulation capability from the algorithm level [13]–[19]. These works exploited sparsity or redundancy inside a single QC simulation trial while the proposed optimization leverages the redundancy among multiple simulation trials. The other type of optimizations is from the computer system level, including vector instructions [12], [20], specialized linear algebra library [22], multi-thread [12], [20], [21], distributed system [20]–[22], GPU [23], [24]. Our acceleration is from algorithm-level and is compatible with these system-level approaches.

## III. Background

In this section, we present a brief review of relevant background to help understand the noisy QC simulation.

### A. QC Basics

**Qubit** Classical computing uses bits as the basic information unit with two deterministic states, '0' and '1', while QC employs qubits with basis states denoted as $|0\rangle$ and $|1\rangle$. The state of one qubit can be the linear combination of the two basis states, represented by $|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$, where $\alpha, \beta \in \mathbb{C}$ and $|\alpha|^2 + |\beta|^2 = 1$. Two or more qubits can be in a superposition of more basis states. For example, a two-qubit system can be in the state $|\Psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle$ and represented by a four-dimensional complex vector $(\alpha_{00}, \alpha_{01}, \alpha_{10}, \alpha_{11})$. In general, a $2^N$-dimensional vector is required to describe the state of a system with $N$ qubits.

**Quantum Operation** The state of a QC system can be manipulated by two main types of quantum operations. The first type is quantum gates, which are unitary operators applied on one or more qubits to change the state vector. The second type of operation is the measurement, which will collapse the superposition state to the basis states with different probabilities based on the amplitudes of the state vector.

**Quantum Circuit and Computation** Quantum circuit is a diagram to represent a quantum program in the well-adopted quantum circuit model [26]. Figure 1 shows an example of a quantum circuit and its computation. On the left is the quantum circuit which contains two qubits and two $H$ gates (in the two squares). The initial state is $S_0 = |00\rangle$ and its state vector $(1, 0, 0, 0)$ is shown on the right. To compute the state $S_1$, the two $H$ gates are applied on $S_0$ and the result is $S_1 = \frac{1}{2}|00\rangle + \frac{1}{2}|01\rangle + \frac{1}{2}|10\rangle + \frac{1}{2}|11\rangle$. This process can be considered as a matrix-vector multiplication since the quantum system is linear and quantum gates are linear transformations. The applied matrix is determined by the Kronecker product of the applied quantum gates. To simulate a quantum circuit, we need to apply the gates to the state vector sequentially and the measurement output will be determined by the amplitudes in the state vector.

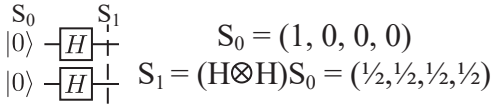



Fig. 1. Example of Quantum Circuit and Computation

### B. Noisy QC Simulation

Noisy QC simulation needs to simulate the input quantum circuit a large number of times with error randomly injected during the simulation. We first introduce error modeling in QC simulation and then introduce the Monte Carlo noisy QC simulation.

*1) Error Modeling:* An error model indicates how error happens during the computation process. It consists of three parts, error operator, error position, and error probability.

**Error Operator** Error operators are some special operators that will be randomly injected in the quantum circuit in order to model the noise effect in the QC program execution on noisy quantum hardware. For example, the three Pauli matrices, $X$, $Y$, and $Z$ (given in Equation 1), are commonly used error operators. When an error happens, an error operator will be applied to the target qubit(s).

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix}, Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \quad (1)$$

**Error Position** Error positions are the places where an error could possibly be injected in the simulated quantum circuit.

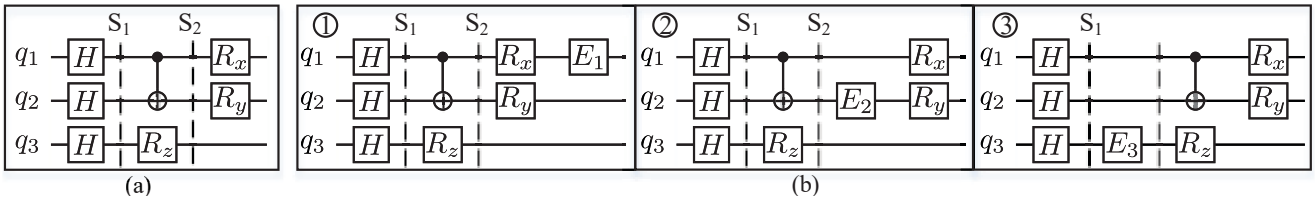


Fig. 2. Example for Computation Redundancy and Execution Reordering (Details in Section IV.A).

For gate errors, error operators can be injected after a gate. Some other errors like decaying from high-energy state $|1\rangle$ to low-energy state $|0\rangle$ or interacting with the environment can happen without an operation. Such an error could appear at any place across the quantum circuit.

**Error Probability** After the error operators and positions are determined, we still need to know the probability for each error position with each error operator. Each time when we meet an error position during the simulation, we will randomly inject one error operator based on the error probability for each operator at this position.

**Measurement Error** Errors can also happen after a measurement. An error operator can only be applied to quantum states while the result after the measurement is a classical bit. To model a measurement error, we directly flip the measurement result bit with a specified probability right after the measurement operation.

*2) Monte Carlo Noisy QC Simulation:* The error operator, position, and probability can construct an error model which can be used in the noisy QC simulation. The error injection simulation trials will then be generated under the given error model. We use the symmetric depolarization error channel, a widely used standard error model [11], [12], as an example to illustrate this procedure. Under this error model, the three error operators are $X$, $Y$, $Z$. There error probability for these three errors are equal, $p = P(X) = P(Y) = P(Z)$. The error probability and the simulated circuit are shown in Figure 3. Since the error is triggered by operations, we inject an error operator $E$ after each gate. On the right of Figure 3 is the final error injected circuit. We will simulate this error-injected circuit many times. In each simulation trial, every error operator $E$ is replaced by $X$, $Y$, and $Z$ with the same probability $p$, or by the identity operator $I$ with the probability $1 - 3p$. These operators will be applied to the state vector to model the noise effect. After a measurement, the classical bit may also be flipped to model the measurement error. Finally, the output result is recorded. Such a simulation procedure will be repeated for all simulation trials and the final results are averaged to show a distribution of the output on the modeled device.

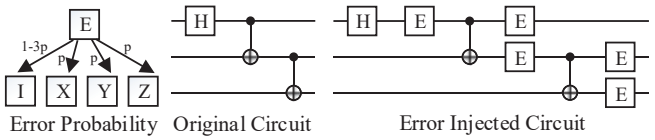


Fig. 3. Depolarization Error Channel and Injection [11], [12]

## IV. Noisy Simulation Optimization

The redundancy among the error-injected simulations can be leveraged to reduce the amount of computation. If two error-injection simulation trials share the same state in the middle, we can save this intermediate state in one simulation trial and then reuse it in the other simulation trial to eliminate the computation before this state. However, the size of a state grows exponentially as the number of qubits increases and it takes significant memory space to store a state vector. Thus, how to identify and store these states efficiently must be addressed to enable this inter-trial QC simulation optimization. In this section, we will first start from an example to illustrate the computation redundancy and then discuss how to efficiently run all the simulation trials.

### A. Computation Redundancy

Figure 2 shows an example to demonstrate the computation redundancy. There are totally four error injection executions in this example, represented by four quantum circuits. The first one in (a) is the original error-free execution. $S_1$ and $S_2$ are two intermediate states during the error-free execution. The other three in (b) (labeled with ①, ②, and ③) are error injected executions. Each of them has one error operator occurred, represented by the gates $E_{\{1,2,3\}}$. To run the noisy QC simulation, all these four quantum circuits will be simulated and then averaged to obtain a distribution of the final output. We can find that all the four quantum circuits are exactly the same before reaching $S_1$ state. The state vector of $S_1$ is the same for all four execution since no errors are injected before $S_1$. As a result, the computation from the initial state to $S_1$ can be shared by all four executions. The state vector at $S_1$ only needs to be calculated and stored in one execution. The rest three executions can start from the stored $S_1$ state instead of starting from the beginning. Such redundancy exists at multiple locations across the error injection Monte Carlo executions. For example, the state vector at $S_2$ can be also be shared by the error-free execution and the first two error injected executions ①②.

The motivating example above has shown the computation redundancy among Monte Carlo executions. We can store some state vectors when we first reach such states and the results will be reused in the following executions. However, the maximal number of state vectors we can store is limited since one state vector has $2^n$ amplitudes ($n$ is the number of qubits). Although several techniques have been proposed to store the state vector in a compressed form [18], [19], the memory requirement will still grow exponentially as the number of qubits increases. To allow circuits with more

**Algorithm 1:** $Trial\_Reorder(S, n)$

**Input:** Trials $S$, error index $n$
**Output:** Ordered Trials $S'$

```
if S has only one trial then
    return S;
end
Order the trials in S based on the location of n^th injected error;
Divide the trials into Groups based on the n^th error;
for Trial Group i do
    S_i = all the trials in Group i;
    Trial_Reorder(S_i, n + 1);
end
```

intermediate states to be simulated efficiently, we introduce an execution reorder technique to reduce the maximal number of concurrently maintained state vectors without loss of the benefit from the computation redundancy elimination.

### B. Trial Reorder

Different execution order can significantly affect the number of states that need to be stored. For the example in Figure 2 (b), ①②③ is an inefficient Monte Carlo execution order. When running ①, both the states $S_1$ and $S_2$ need to be stored so that ② can start from $S_2$ and ③ can start from $S_1$. An optimized execution order for this example can be ③②①. When executing ③, we only need to store state $S_1$. The execution of ② can directly start from the stored $S_1$ and then $S_1$ can be dropped since it is no longer used in the follow-up executions. During the execution of ②, $S_2$ will be stored and finally used when executing ①. Consequently, only one state vector needs to be stored during the entire simulation process. An optimized execution order reduced 50% of memory requirement (from two state vectors to one state vector) compared with a straight-forward order in this example.

We propose to find the optimized execution order with a trial reorder algorithm (shown in Algorithm 1), which is explained as follows. We first generate the Monte Carlo execution trials without actually running the simulation. The simulated quantum circuit is divided into layers, in which any two quantum operations are not applied to the same qubit. Error operators will only be injected at the end of each layer (shown in Figure 3). One execution trial will record the location and operator of each injected error. These trials will be ordered by the location of the first injected error. The trials with the first error injected in the first layer (e.g., ③ in Figure 2) will appear at the beginning of the execution order, followed by those trials with the first error injected in the second layer (e.g., ② in Figure 2), and so on.

After ordering the trials based on the location of the first error, we can further improve the ordering based on the location of the next error. If two or more error trials share the same first error (injected on the same qubit with the same error operator), these trials will be grouped. The simulation for these trials can be further optimized if we recurrently reorder the trials in the same group based on the location of the second injected error. Similarly, we reorder trials which share the first two injected errors based on the third one, and so on. This recurrent order will stop when there is only one trial left.

After the ordering procedure above, we begin our simulation by executing the first layer of the circuit with no error injected and store the state as $S_1$. This part of computation can be shared by all Monte Carlo trials. After finishing the trials with the first error in the first layer, we can execute one more layer without error and store the new state as $S_2$. Now $S_1$ can be dropped as no executions remaining will rely on it. Additional memory space is only required when recurrent reordering happens because these trials sharing the first error operator need to store the state vector after the shared error to help eliminate the computation redundancy among them. The maximal number of state vectors we need to store is the recursion depth during the reordering, which is small because the probability for two independently and randomly generated trials to have $m$ shared error operators decreases exponentially as $m$ increases.

This execution reorder technique leverages the inter-trial computation redundancy and can cooperate with existing QC simulation optimizations which focus on the execution of one simulation trial. The final simulation result will not be changed since the output of all trials are calculated and averaged, which makes our optimized simulation mathematically equivalent to the original one.

## V. Evaluation

In this section, we evaluate the computation saving and memory consumption of the optimized noisy simulation. We conducted two groups of experiments to give a full test of our accelerated noisy QC simulation: 1) small-scale circuits with realistic device error model, 2) large-scale circuits with artificial error models. In the first group of experiments, we will show that our noisy QC simulation scheme can accelerate state-of-the-art NISQ device modeling. In the later one, we focus on testing the scalability of the proposed noisy simulation scheme by varying the error rate of the modeled NISQ device and the input circuit size.

**Baseline** The baseline noisy QC simulation strategy is to execute the randomly generated error injection trials directly without ordering them. During the execution of each trial, errors are injected based on the error model and only the final result is stored. All the trials are treated individually, and the shared intermediate states are not considered. Such a strategy is widely adopted in full-state QC simulators, including Rigetti's QVM [27], QX [12], etc.

**Metrics** In order to perform a fair evaluation of our noisy simulator optimization, the metrics in this section are chosen to be independent of implementation and platform. For the computation time, we use the number of basic operations (matrix-vector multiplication) in the full-state QC simulation to indicate the computation amount normalized to the baseline.

TABLE I
BENCHMARK CHARACTERISTICS

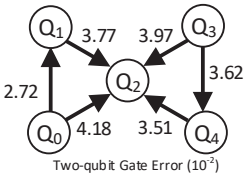
| Name | Qubit # | Single # | CNOT # | Measure # |
|---|---|---|---|---|
| rb | 2 | 9 | 2 | 2 |
| grover | 3 | 87 | 25 | 3 |
| wstate | 3 | 21 | 9 | 3 |
| 7x1mod15 | 4 | 17 | 9 | 4 |
| bv4 | 4 | 8 | 3 | 3 |
| bv5 | 5 | 10 | 4 | 4 |
| qft4 | 4 | 42 | 15 | 4 |
| qft5 | 5 | 83 | 26 | 5 |
| qv_n5d2 | 5 | 44 | 12 | 5 |
| qv_n5d3 | 5 | 74 | 21 | 5 |
| qv_n5d4 | 5 | 100 | 30 | 5 |
| qv_n5d5 | 5 | 130 | 36 | 5 |

For the memory consumption, we use the number of **Maintained State Vectors** (MSVs) during the noisy simulation since the memory space for the state vectors, which will grow exponentially as the number qubits increases, dominates the memory consumption. Note that MSV is an overhead purely for our approach since the baseline does not maintain any intermediate states.

### A. Realistic NISQ Device Error Modeling

The first group of experiments is performed on the error model of a realistic NISQ device, IBM's 5-qubit superconducting quantum processor. We generate various numbers of trials (from 1024 to 8192) to test the computation saving under different simulation configurations.

**Benchmarks** Table I shows the 12 quantum programs used in this experiment. They are collected from IBM OpenQASM benchmarks [28] and prior work [29]. These benchmarks include Bernstein-Vazirani algorithm (bv) [30], Quantum Fourier Transform (qft) [26], Quantum Volume (qv) [31], Grover algorithm [2], Randomized Benchmarking (rb) [32], Modular Multiplication (7x1mod15) [11], and W-state [33]. All the benchmarks are compiled and mapped to this IBM's 5-qubit device with the Enfield compiler [29] to determine the actual physical qubits. The four columns on the right in Table I show the numbers of qubits and quantum operations in the post-compilation programs for each benchmark. "Single" stands for single-qubit gate and "CNOT" stands for CNOT gate, the only supported two-qubit gate on this device.




| | Single-qubit Gate Error ($10^{-3}$) | Measurement Error ($10^{-2}$) |
|---|---|---|
| $Q_0$ | 1.37 | 2.40 |
| $Q_1$ | 1.37 | 2.60 |
| $Q_2$ | 2.23 | 3.00 |
| $Q_3$ | 1.72 | 2.20 |
| $Q_4$ | 0.94 | 4.50 |

Fig. 4. Error Rates on IBM Yorktown Chip [8]

**Error Model** Figure 4 shows the error probability of IBM's 5-qubit Yorktown quantum processor. For the error operator and position, we use the symmetric depolarization model (shown in Figure 3), in which the error probability is distributed to three Pauli operators equally and errors are injected after each gate. This a standard model employed in most noisy simulators [11], [12].

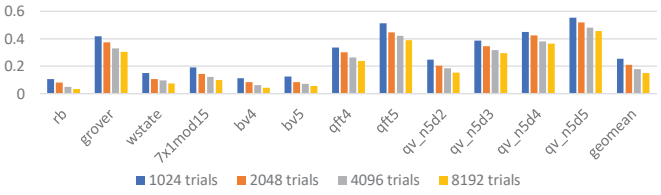



Fig. 5. Normalized Computation in Realistic Error Model Experiments

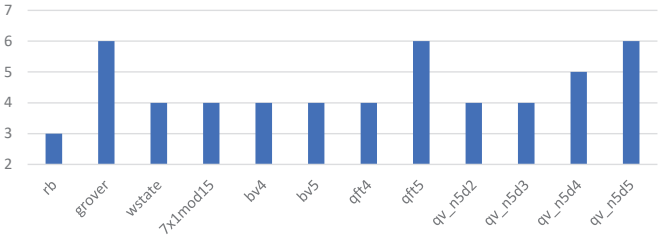



Fig. 6. Memory Consumption (MSVs) in Realistic Error Model Experiments

**Results** Figure 5 shows the computation saving for all benchmarks with different numbers of trials. The proposed optimization can save about $75\% \sim 85\%$ of computation on average with the number of trials increases from 1024 to 8192. In the worst case when the benchmark is large ('qv_n5d5'), the computation amount saving still achieves 57% with 8192 trials. We can also see that the more trials we execute, the more computation we will save because more overlapped computation can be identified. Figure 6 shows the number of MSVs in experiments with 1024 trials and this result does not significantly change when the number of trials increases from 1024 to 8192. The number of MSVs is 3 for the smallest benchmark 'rb' and only 6 in the largest benchmarks 'qft5' and 'qv_n5d5'. As discussed in Section IV, the number of MSVs will grow slowly since the probability for two trials to share the same $m$ injected errors decays exponentially as $m$ increases.

### B. Artificial Error Model for Scalability Test

In this scalability test, we choose input circuits of larger sizes and increase the number of simulation trials to $10^6$.

**Benchmarks** We use Quantum Volume (qv) benchmark, one type of random circuit proposed by IBM [31], to test the scalability of the proposed noisy simulation scheme since random circuit is widely-used in benchmarking QC simulators [15], [20]A group of qv programs is generated with various numbers of qubits (from 10 to 40) and circuit depth (from 5 to 20) to test the computation saving and memory consumption as the input circuit scales. For example, "n10,d10" means 10 qubits with circuit depth 10. The largest circuit used in this experiment with 40 qubits and depth of 20 is already close to the limit of existing full state QC simulators [15].

**Error Model** We construct error models for larger artificial NISQ devices expected in the future. We still use the symmetric depolarizing gate error model for the error operator and position. The error probabilities of single-qubit gates ranges from $10^{-3}$ to $10^{-4}$. $10^{-3}$ represents state-of-the-art superconducting quantum circuit technology and $10^{-4}$ reflects extrapolations of progress in hardware. The error rates of two-qubit gates and measurement operations are set

to be $10\times$ of single-qubit gates. We assume that all the qubits and qubit pairs share the same error probabilities since small error probability variance will not significantly affect the computation saving in the proposed noisy QC simulation scheme.

**Results** Figure 7 shows the computation amount for all benchmarks with different error probabilities. On average, we can save about 79% computation. In the worst case, for a quantum volume circuit of the largest size and highest error rate, we can still save about 31% computation. The computation amount drops dramatically with lower error rates which can be expected in future devices. Figure 8 shows the number of MSVs, which grows slowly as the circuit depth increases. On average we need to store about 6 intermediate state vectors. When the number of qubits increases, the number of MSVs decreases because there are more potential error positions which reduce the probability for two trials to share the same injected error.

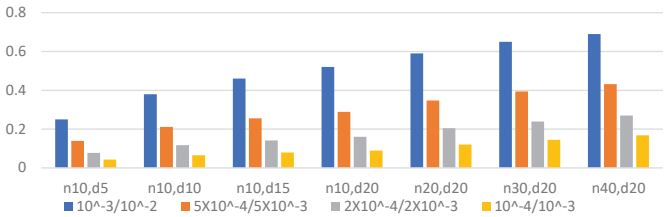



Fig. 7. Normalized Computation in Scalability Experiments

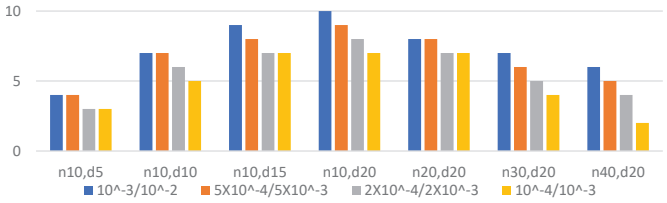



Fig. 8. Memory Consumption (MSVs) in Scalability Experiments

## VI. Conclusion

Although simulating quantum computing on a classical machine is ultimately not scalable, it is still of great interest due to its practical usage. In this paper, we propose to accelerate the time-consuming noisy QC simulation by eliminating the redundancy among the Monte Carlo simulation trials. By analyzing the Monte Carlo error injection simulation trials before actually running the simulation, we identify shared intermediate states among these trials and then reorder them to maximize the overlapped computation between two consecutive simulation trials. The number of saved intermediate states is also reduced since states that will no-longer be used are dropped immediately. Experiment results show that we can achieve around 80% computation saving on average with only a small number of state vectors maintained at the same time. The proposed simulation scheme also demonstrates great scalability when modeling larger size future QC devices as more computation can be saved with more simulation trials or on future device models with reduced error probabilities.

## References


[1] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[2] L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[3] A. Peruzzo et al. A variational eigenvalue solver on a photonic quantum processor. *Nature communications*, 5:4213, 2014.

[4] J. Kelly. A Preview of Bristlecone, Google's New Quantum Processor. https://ai.googleblog.com/2018/03/a-preview-of-bristlecone-googles-new.html, 2017.

[5] W. Knight. IBM Raises the Bar with a 50-Qubit Quantum Computer. https://www.technologyreview.com/s/609451/ibm-raises-the-bar-with-a-50-qubit-quantum-computer/, 2017.

[6] N. M. Linke et al. Experimental comparison of two quantum computing architectures. *Proceedings of the National Academy of Sciences*, 114(13):3305–3310, 2017.

[7] Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[8] IBM. https://quantumexperience.ng.bluemix.net/qx/devices, 2018.

[9] Rigetti. https://www.rigetti.com/qpu, 2018.

[10] J. Preskill. Quantum computing in the nisq era and beyond. *arXiv preprint arXiv:1801.00862*, 2018.

[11] G. Aleksandrowicz et al. Qiskit: An open-source framework for quantum computing, 2019.

[12] N. Khammassi et al. Qx: A high-performance quantum computer simulation platform. In *2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 464–469. IEEE, 2017.

[13] G. F. Viamontes et al. High-performance quidd-based simulation of quantum circuits. In *Proceedings of the conference on Design, automation and test in Europe-Volume 2*, page 21354. IEEE, 2004.

[14] G. F. Viamontes et al. *Quantum circuit simulation*. Springer Science & Business Media, 2009.

[15] J. Chen et al. Classical simulation of intermediate-size quantum circuits. *arXiv preprint arXiv:1805.01450*, 2018.

[16] I. L. Markov and Y. Shi. Simulating quantum computation by contracting tensor networks. *SIAM Journal on Computing*, 38(3):963–981, 2008.

[17] S. Aaronson and D. Gottesman. Improved simulation of stabilizer circuits. *Physical Review A*, 70(5):052328, 2004.

[18] S. Anders and H. J. Briegel. Fast simulation of stabilizer circuits using a graph-state representation. *Physical Review A*, 73(2):022334, 2006.

[19] A. Zulehner and R. Wille. Advanced simulation of quantum computations. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018.

[20] M. Smelyanskiy et al. qhipster: the quantum high performance software testing environment. *arXiv preprint arXiv:1601.07195*, 2016.

[21] D. S. Steiger et al. Projectq: an open source software framework for quantum computing. *Quantum*, 2:49, 2018.

[22] D. Wecker and K. M. Svore. Liqui$|>$ : A software design architecture and domain-specific language for quantum computing. *arXiv preprint arXiv:1402.4467*, 2014.

[23] B. Tarasinski. https://gitlab.com/quantumsim/quantumsim, 2018.

[24] T. Jones et al. Quest and high performance simulation of quantum computers. *arXiv preprint arXiv:1802.08032*, 2018.

[25] TE O'brien et al. Density-matrix simulation of small surface codes under current and projected experimental noise. *npj Quantum Information*, 3(1):39, 2017.

[26] Michael A Nielsen and Isaac L Chuang. Quantum computation and quantum information. *UK: Cambridge University Press*, 2010.

[27] Rigetti. https://pyquil.readthedocs.io/en/stable/noise.html, 2019.

[28] A. W. Cross et al. Open quantum assembly language. *arXiv preprint arXiv:1707.03429*, 2017.

[29] UFMG Compilers Laboratory. http://cuda.dcc.ufmg.br/enfield/, 2018.

[30] E. Bernstein and U. Vazirani. Quantum complexity theory. *SIAM Journal on computing*, 26(5):1411–1473, 1997.

[31] N. Moll et al. Quantum optimization using variational algorithms on near-term quantum devices. *Quantum Science and Technology*, 3(3):030503, 2018.

[32] Emanuel Knill et al. Randomized benchmarking of quantum gates. *Physical Review A*, 77(1):012307, 2008.

[33] J. Joo et al. Quantum teleportation via a w state. *New Journal of Physics*, 5(1):136, 2003.