Reduction of Qubits in Quantum Algorithm for Monte Carlo Simulation by Pseudo-random Number Generator

Koichi Miyamoto†
Mizuho-DL Financial Technology

Kenji Shiohara†
Division of Physics, Graduate School of Science, Hokkaido University
(Dated: December 2, 2019)

It is known that quantum computers can speed up Monte Carlo simulation compared to classical counterparts. There are already some proposals of application of the quantum algorithm to practical problems, including quantitative finance. In many problems in finance to which Monte Carlo simulation is applied, many random numbers are required to obtain one sample value of the integrand, since those problems are extremely high-dimensional integrations, for example, risk measurement of credit portfolio. This leads to the situation that the required qubit number is too large in the naive implementation where a quantum register is allocated per random number. In this paper, we point out that we can reduce qubits keeping quantum speed up if we perform calculation similar to classical one, that is, estimate the average of integrand values sampled by a pseudo-random number generator (PRNG) implemented on a quantum circuit. We present not only the overview of the idea but also concrete implementation of PRNG and application to credit risk measurement. Actually, reduction of qubits is a trade-off against increase of circuit depth. Therefore full reduction might be impractical, but such a trade-off between speed and memory space will be important in adjustment of calculation setting considering machine specs, if large-scale Monte Carlo simulation by quantum computer is in operation in the future.

I. INTRODUCTION

Among applications of quantum computers to numerical problems providing higher speed than classical computation is Monte Carlo simulation [1]. It has been shown that estimation error in the quantum-based Monte Carlo is proportional to $O(N^{-1})$, where $N$ is the number of computational steps, compared with $O(N^{1/2})$ in classical one. Quantitative finance is one of the fields where Monte Carlo simulation is heavily used and there are some proposal to apply the quantum algorithm to financial problem, for example, risk measurement of portfolio [2,3] and derivative pricing [4,5].

In application of the quantum algorithm for Monte Carlo to financial problems, required qubit number might be problematic. Many of such problems are extremely high-dimensional integration and require many independent random numbers to obtain one sample value of integrand. One of the most prominent example is risk measurement of credit portfolio [3]. Credit portfolio consists of many loans or debts and banks suffer losses when obligors default. Banks monitor such credit risks estimating some risk measures, for example, expected loss (EL), value-at-risk (VaR), which represents percentile point (say, 99%) of loss distribution, conditional VaR (CVaR), expectation value of loss conditioned it exceeds the VaR, and so on. One of popular mathematical models describing probability distribution of loss is Merton model [6] and risk measures under the model are usually estimated by Monte Carlo. We describe the model in the later section, but an important point is that the required number of random numbers to determine a default pattern of obligors is nearly equal to the number of obligors. In other words, it is necessary to generate random numbers as many as obligors to obtain a sample value of the integrand, that is, loss. The number of obligors can be $O(10^6)$ for large portfolios, so is the required random number. In the previous work [3], a quantum register is allocated to represent a random number, so the required qubit number is also $O(10^6)$. The qubit number of today’s largest quantum computer is $O(10)$, so it will be the far future when machines with such many qubits is realized. Therefore, it is meaningful to consider possibility to reduce qubits.

In this paper, we propose a way to reduce qubits. Although we will explain the detail in the next section, we here describe the outline. In short, it is classical Monte Carlo on a quantum computer. In classical Monte Carlo, we usually generate some sampled patterns of values of random numbers, not all patterns. More concretely, we generate sequences of pseudo-random number (PRN) using some pseudo-random number generator (PRNG) and use each sequence to obtain one sample value of the integrand. Finally, we calculate the average of the sample values and consider it as an approximation of the integral. We can do same thing on a quantum circuit. That is, we can sequentially generate pseudo-random bit strings on a quantum register and calculate the integrand into another register. On a quantum computer, we can parallelly perform such computation and finally obtain the superposition of states in which each of sampled integrand values is realized on a register. Then, we can estimate the average of sample values by methods commonly used in the quantum algorithm for Monte Carlo, for example [7,8]. This procedure leads to the estimation result same as classical Monte Carlo, but with smaller computational load.

We present not only the idea but also concrete implementation. We propose an example of PRNG which can be easily implemented on quantum circuit. It is permuted congruential generator (PCG) [10] and explained in detail in the later sec-
tion. This is the combination of linear congruential method and permutation of bit string and has advantages in the aspect of memory (that is, required qubit number) and computational load (that is, circuit depth) compared to other types of PRNG, for example Mersenne Twister [9]. It is possible to construct the quantum gate which progresses the PCG sequence as we present later.

We also consider application to concrete problems. The first one is credit risk measurement, which is mentioned above. We later present the quantum circuit which calculate sampled values of loss of a credit portfolio using PRNG. The second one is the integration of a simple multi-variable function, that is, a trigonometric function whose phase depends on two variables. We consider this for demonstrative purpose and present not only the circuit but also the numerical result calculated by a simulator.

The rest of this paper is organized as follows. Section II explains the overview of our idea. Section III presents concrete implementation of the gate which realizes PCG. In section IV and V we consider application to credit risk measurement and a simple integration, respectively. Section VI contains conclusion and discussion on some issues. Especially, we discuss the trade-off between qubit number and circuit depth and importance of such a memory-speed trade-off on adjustment of calculation configuration, which will be often necessary when large-scale Monte Carlo by quantum computer is in practical operation in the future.

II. OVERVIEW OF THE IDEA: QUANTUM ALGORITHM FOR MONTE CARLO USING PSEUDO-RANDOM NUMBER

A. Our Idea

Applications of the quantum algorithm for Monte Carlo to high-dimensional integration in financial problems can be found in previous works, especially credit risk measurement in [3]. In the paper, independent random numbers necessary to obtain a value of integrand are represented by different quantum registers, so the number of required qubits \( N_{\text{qubit}} \) is proportional to the number of random numbers \( N_{\text{ran}} \). If \( N_{\text{ran}} \) is large as in the aforementioned cases, this can lead to shortage of qubits.

Let us see the method in more detail. The way to represent a random number by quantum register is as follows. For example, a qubit with state \( \sqrt{1-p}|0\rangle + \sqrt{p}|1\rangle \) can be seen as a Bernoulli random number taking 1 with probability \( p \). We can also represent a discretized approximation of a continuous random number like a normal random number on a quantum register [11]. Then, referring to these registers, the value of the integrand is computed into another register and its expectation value is estimated by methods such as [12][8]. Note that this procedure intends to make a superposition of all possible patterns of random number and estimate the exact expectation value, which we hereafter write as \( E_{\text{true}} \).

In order to perform Monte Carlo enjoying quantum speed-up and reducing qubits, we first note that what we calculate in classical Monte Carlo is different from that in the quantum way. That is, we do not consider all patterns of random number values in the classical Monte Carlo. We sample only a part of patterns of the random numbers and the integrand and take a simple arithmetic average of the sampled integrand values as an approximation for \( E_{\text{true}} \). In other words, we calculate \( E_{\text{true}} \) the expectation value under the sample space which consists of a part of samples and the probability measure under which equal proportionality is allocated to each sample. Besides, in most cases, we use a sequence of PRN on behalf of random numbers to calculate the integrand, since strict randomness is difficult to realize on a classical computer. More concretely, we usually generate a PRN sequence with \( N_{\text{amp}}N_{\text{ran}} \) elements and divide them into \( N_{\text{amp}} \) subsequences with \( N_{\text{ran}} \) elements, then use each subsequence to calculate a sample value of the integrand.

Our idea is that we estimate not \( E_{\text{true}} \) but \( E_{\text{amp}} \) using a quantum computer in the way similar to classical Monte Carlo. The calculation flow is as follows. First, we choose some PRNG which consumes \( n_{\text{PRN}} \) bits, including random number itself and working space. We assume that we can construct two types of quantum gate: one is \( P_{\text{PRN}} \), which progresses a PRN sequence by one step, that is

\[
|x_{n+1}\rangle_{\text{PRN}} = |x_{n+1}\rangle_{\text{PRN}},
\]

where \( x_n \) is the \( n \)-th element of the PRN sequence, and another is \( J_{\text{PRN}} \), which gives \( x_{iN_{\text{ran}}+1} \) for given \( i \), that is,

\[
|i\rangle_{\text{amp}} |0\rangle_{\text{PRN}} \rightarrow |i\rangle_{\text{amp}} |x_{iN_{\text{ran}}+1}\rangle_{\text{PRN}}.
\]

Besides, we determine \( N_{\text{amp}} \), the number of samples. For simplicity, we assume \( N_{\text{amp}} = 2^n_{\text{amp}} \) for an integer \( n_{\text{amp}} \) which satisfies \( 0 < n_{\text{amp}} < n_{\text{PRN}} \). Then,

1. Prepare a register \( R_{\text{amp}} \) with \( n_{\text{amp}} \) qubits and generate a superposition of \( |0\rangle, |1\rangle, \ldots, |2^{n_{\text{amp}}-1}\rangle \) with equal amplitudes, that is,

\[
\frac{1}{\sqrt{2^{n_{\text{amp}}-1}}} |i\rangle_{\text{amp}}.
\]

This can be done by operating a Hadamard gate to each of \( n_{\text{amp}} \) qubit.

2. Operate \( J_{\text{PRN}} \), then the \( (iN_{\text{ran}}+1) \)-th element of the sequence is set to the register \( R_{\text{PRN}} \), where \( i \) is determined by the state of \( R_{\text{amp}} \). These are the starting points of subsequences.

3. Perform a calculation step of the integral referring to \( R_{\text{PRN}} \) and reflect the result into a register \( R_{\text{int}} \). Here,

---

1 If a continuous random number is approximated discretely, ‘all patterns’ means those of the discretized value.

2 Mathematically, using such subsequences might arise a statistical concern for large \( N_{\text{ran}} \) in terms of homogeneity of the distribution of tuples of consecutive PRNs in a high dimensional space [9]. However, such a way to use PRN are often adopted in practice in banks. We consider that using a tiny part in a large period PRN mitigates the concern [12].

3 Here and hereafter, a subscript of a ket basically denotes the qubit number of the register.
we assume that the integrand is calculated step-by-step using each random number.

4. Operate $P_{PRN}$ to the register $R_{PRN}$, then the PRN sequence progresses by one step.

5. Perform a calculation step of the integral referring to $R_{PRN}$ and reflect the result into a register $R_{int}$.

6. Iterate 4 and 5 until the calculation of the integrand ends. This corresponds to sequential generation of PRN and calculation using it. Finally, we obtain an equiprobable superposition of states, in each of which $R_{int}$ holds a sampled integrand value.

7. Estimate the expectation value of $R_{int}$ by methods like [7], [8]. This is an estimate for the arithmetic average of sampled integrand values, that is, $E_{samp}$.

The flow of state transformation is as follows:

\[
\begin{align*}
|0\rangle_{n_{samp}} |0\rangle_{n_{PRN}} |0\rangle_{n_{int}} &\xrightarrow{1} \frac{1}{\sqrt{n_{samp}}} \sum_{i=0}^{N_{samp}-1} |i\rangle_{n_{samp}} |0\rangle_{n_{PRN}} |0\rangle_{n_{int}} \\
\vdots &\xrightarrow{2} \frac{1}{\sqrt{n_{PRN}}} \sum_{j=0}^{N_{PRN}-1} |j\rangle_{n_{PRN}} |x_{1}^{(j)}\rangle_{n_{PRN}} |0\rangle_{n_{int}} \\
\vdots &\xrightarrow{3} \frac{1}{\sqrt{n_{int}}} \sum_{k=0}^{N_{int}-1} |k\rangle_{n_{int}} |x_{1}^{(k)}\rangle_{n_{PRN}} |f_{1}(x_{1}^{(k)})\rangle_{n_{int}} \\
\vdots &\xrightarrow{4} \frac{1}{\sqrt{n_{PRN}}} \sum_{j=0}^{N_{PRN}-1} |j\rangle_{n_{PRN}} |x_{1}^{(j)}\rangle_{n_{PRN}} |f_{1}(x_{1}^{(j)})\rangle_{n_{int}} \\
\vdots &\xrightarrow{5} \frac{1}{\sqrt{n_{int}}} \sum_{k=0}^{N_{int}-1} |k\rangle_{n_{int}} |x_{2}^{(k)}\rangle_{n_{PRN}} |f_{2}(x_{1}^{(k)}, x_{2}^{(k)})\rangle_{n_{int}} \\
\vdots &\xrightarrow{6} \frac{1}{\sqrt{n_{PRN}}} \sum_{j=0}^{N_{PRN}-1} |j\rangle_{n_{PRN}} |x_{N_{ran}}^{(j)}\rangle_{n_{PRN}} \\
&\quad \otimes |f_{N_{ran}}(x_{1}^{(j)}, \ldots, x_{N_{ran}}^{(j)})\rangle_{n_{int}}
\end{align*}
\]

Here, $x_{n}^{(i)} = x_{n_{ran}+n}$ and this is the $n$-th element of the $i$-th subsequence. $n_{int}$ is the qubit number of $R_{int}$. $f_{N_{ran}}$ is the integrand function, which takes $N_{ran}$ random numbers as arguments. We assume that it is sequentially computed in $N_{ran}$ steps and each step requires a random number and the output of the previous step as inputs. That is, the $n$-th intermediate value $f_{n}$ is in the following form:

\[
f_{1} = f_{1}(x), \\
f_{n} = f_{n}(y, x) \quad \text{for } n = 2, \ldots, N_{ran}
\]

and $f_{2}(x_{1}(1), x_{2})$ is simply written as $f_{2}(x_{1}, x_{2}), f_{3}(f_{2}(x_{1}, x_{2}), x_{3})$ as $f_{3}(x_{1}, x_{2}, x_{3})$, and so on. In the credit risk measurement, which we will consider later, the integrand satisfies this property.

We present an outline of the quantum circuit for the above method in Figure 1a. We also present that for the method in the previous papers for comparison. In our method, as shown in Figure 1a after the operation which create a superposition of $|x_{1}^{(0)}\rangle, |x_{1}^{(1)}\rangle, \ldots, |x_{1}^{(N_{ran}-1)}\rangle$ on $R_{PRN}$ and the gate $f_{1}$, the first step of calculation of the integrand, we sequentially operate $P_{PRN}$ and $f_{n}$, $n$-th calculation step. The register which represents (pseudo) random numbers is only $R_{PRN}$ and PRNs $x_{n}^{(i)}$ are sequentially generated on it. The intermediate value of the integrand $f_{n}(x_{1}, \ldots, x_{n})$ is calculated into $R_{int}$ using $x_{n}^{(i)}$ and $f_{n-1}(x_{1}, \ldots, x_{n-1})$ as inputs and finally $f_{N_{ran}}(x_{1}^{(0)}, \ldots, x_{N_{ran}}^{(0)})$ is reached. On the other hand, in the method in previous works, as shown in Figure 1b, quantum registers $R_{RN,1}, \ldots, R_{RN,N_{ran}}$ are prepared to represent all random numbers simultaneously and a superposition of numbers following the desired probability distribution (for example, normal) is generated on each register by the gate ‘dist’ in Figure 1b. Then, the integrand value is calculated using all of $R_{RN,1}, \ldots, R_{RN,N_{ran}}$ at the same time.

Here we make a comment on probability distribution of random number. In the previous method, a random number under the desired distribution is generated on a register using the gate ‘dist’. On the other hand, in the method of this paper,
sequentially generated PRNs basically obey uniform distribution, since most PRNGs are for that distribution. Therefore, we have to convert uniform random numbers to random numbers obey a desired distribution. Such a conversion is actually a common step in the classical Monte Carlo and there are many well-known methods, for example, the Box-Muller method for standard normal distribution. We assume such a conversion is implementable as a quantum gate and contained in $f_a$. Actually, implementation of trigonometric functions and logarithm, which are necessary to the Box-Muller method, has been investigated in previous papers [13–15].

Let us make another comment on how the distribution of the integrand value is taken into account in the method of this paper. In the previous method, the desired distribution of the integrand value is realized through the distribution of random numbers on registers. On the other hand, the distribution of the integrand value is generated through the PRNG in the method of this paper. Although the final state is a superposition of various integrand values with equal probability, the appearance pattern of the value reflects the distribution. For example, if the integrand value obeys distribution with a peak at some value $F$, the integrand values close to $F$ frequently appear on $R_{int}$ in the set of states compose the final superposition.

### B. Relationship between Computational Load and Error

Now, we roughly estimate the relationship between computational load and error in our method. There are two sources of error: one is the difference between $E_{\text{samp}}$ and $E_{\text{true}}$, which we write as $\Delta_{\text{TrSm}}$, and another is the error of estimation of $E_{\text{samp}}$ in the quantum algorithm, which we write as $\Delta_{\text{Est}}$. The former is proportional to $N_{\text{samp}}^{-1/2} = 2^{-n_{\text{amp}}/2}$ as the classical Monte Carlo. The latter depends on the quantum algorithm, but in many algorithms it is proportional to the inverse of $N_{\text{trac}}$, the number of performing the oracle circuit, which is that in Figure 14 in the current case. Since the computational load $N_{\text{comp}}$ is proportional to $N_{\text{trac}}$, the latter error is proportional to $N_{\text{comp}}^{-1}$ as a result. In total, the error in our method behaves as

$$\Delta_{\text{out}} \sim \Delta_{\text{TrSm}} + \Delta_{\text{Est}} \sim O(N_{\text{amp}}^{-1/2}) + O(N_{\text{comp}}^{-1}) \sim O(2^{-n_{\text{amp}}/2}) + O(N_{\text{comp}}^{-1}).$$

In the previous method of the quantum-based Monte Carlo, where $E_{\text{true}}$ itself is estimated, the first term does not exist, so

$$\Delta_{\text{prev}} \sim O(N_{\text{comp}}^{-1}).$$

On the other hand, the error is proportional to $N_{\text{samp}}^{-1/2}$ in the classical Monte Carlo and the classical computer takes time proportional to $N_{\text{amp}}$ to get $N_{\text{amp}}$ samples. So the error in the classical Monte Carlo behaves as

$$\Delta_{\text{class}} \sim O(N_{\text{amp}}^{-1/2}).$$

Figure 2 shows the rough image of the above relationships. $\Delta_{\text{out}}$ decreases faster than $\Delta_{\text{class}}$ and similar to $\Delta_{\text{prev}}$ as $N_{\text{comp}}$ increases, although it has a lower bound given by $\Delta_{\text{TrSm}}$. Therefore, in aspects of computational load, our method has an advantage compared to the classical way and is similar to the previous quantum way as long as $N_{\text{amp}}$ is large enough. We also note that increasing $n_{\text{amp}}$ by a few leads to an increase of $N_{\text{amp}}$ and decrease of $\Delta_{\text{TrSm}}$ by orders of magnitude. $N_{\text{ran}}N_{\text{amp}}$ cannot exceed $P$, the period of PRN, but we expect that it is unnecessary to concern such a upper bound, as long as we use a widely-used PRNG, which has a period, say $2^{64}$. This is much longer than $N_{\text{ran}}N_{\text{amp}}$ in practice, since each of these is at most $10^6$ and the product is at most $10^{12} \sim 2^{40}$.

### III. IMPLEMENTATION OF PSEUDO-RANDOM NUMBER GENERATOR ON QUANTUM CIRCUIT

#### A. PCG

We next consider how to implement a PRNG on a quantum circuit, that is, the gates $P_{\text{PRN}}$ and $J_{\text{PRN}}$. Remembering the motivation of this work, reduction of qubits, PRNGs which require small working space are desirable. Besides, in order to decrease circuit depth as much as possible, we desire a simpler and shorter calculation step to progress PRN sequence. Of course, the longer period and better statistical property is preferred. We propose PCG [10] as a PRNG which satisfies these properties.

PCG is combination of linear congruential generator (LCG), a popular and elementary PRNG, and permutation of bit string. The $n$-th element of a PCG sequence $x_n$ is recursively defined as follows:

$$\tilde{x}_{n+1} = f_{a,c,m}^{\text{prog}}(\tilde{x}_n) := (a\tilde{x}_n + c) \mod m \quad (8)$$

where $a, c$ and $m$ are integer parameters satisfying $a > 0, c \geq 0, m > 0$ and the seed $\tilde{x}_0$ is also given as an integer satisfying $0 \leq \tilde{x}_0 < m$. $\tilde{x}_n$ is the background sequence and defined by LCG recurrence formula as above. $g$ is the permutation of a bit string, which is explained in detail later. Therefore, the calculation steps to progress a PCG sequence is the sequence of modular multiplication, modular addition and permutation. Besides, for LCG we can easily jump ahead by $k$ steps using the following formula:

$$\tilde{x}_{n+k} = \left(d^k \tilde{x}_n + \frac{c(d^k - 1)}{a - 1}\right) \mod m. \quad (9)$$

Especially, we can obtain $\tilde{x}_{N_{\text{amp}}+1}$ from a seed $\tilde{x}_0$ as

$$\tilde{x}_{N_{\text{amp}}+1} = f_{a,c,m}^{\text{jump}}(\tilde{x}_0) := \left(d^{N_{\text{amp}}+1} \tilde{x}_0 + \frac{c(d^{N_{\text{amp}}+1} - 1)}{a - 1}\right) \mod m. \quad (10)$$

Given the above formulae, we can construct quantum gates $P_{\text{PRN}}$ and $J_{\text{PRN}}$ for PCG. The rough images of the circuit diagrams are shown in Figure 5. To construct $P_{\text{PRN}}$, we first get back PCG to LCG with the inverse of $g$, then progress LCG with the $f_{a,c,m}^{\text{prog}}$ gate and finally perform the permutation $g$. The $f_{a,c,m}^{\text{prog}}$ gate maps $|x\rangle$ to $f_{a,c,m}^{\text{prog}}(|x\rangle)$ and is constructed as
modular multiplication $|x⟩ → |ax \mod m⟩$ followed by modular addition $|x⟩ → |(x+c) \mod m⟩$. To construct $f_{PRN}$, we first operate $f_{jump}^{jump}$ gate, which refers to the first register as an input and transforms the second register from $|0⟩$ to $|\bar{x}_{N_{in}}⟩$ if the first register is $|0⟩$, then $g$ to the second register.

The $f_{jump}^{jump}$ gate is constructed as combination of modular addition, subtraction, multiplication, division and exponentiation $|k⟩|x⟩ → |k⟩|a^kx \mod m⟩$. Implementation of (modular) adder, multiplier and exponentiator has been investigated in many papers, for example, [16–27]. Modular subtraction is the inverse of addition. Division by $a - 1$ modulo $m$ is implemented as multiplication by an integer $b$ such that $(a - 1)b \equiv 1 \mod m$, which can be found by the extended Euclidean algorithm [28].

There is a comment on implementation of $f_{prog}^{prog}$. It should be implemented not in the form that it output the result in the register other than the input register, that is, $|x⟩|0⟩ → |x⟩|f_{prog}^{prog}(x)⟩$, but in the form that it updates the input register itself into the resulting state, that is, $|x⟩ → |f_{prog}^{prog}(x)⟩$. This is because this gate is repeatedly used in the method of this paper, so the qubit number required for the entire calculation explodes if it is necessary to add a register in each calculation step. Most of previous implementation of modular addition are the self-updating type, so can be used with no change. On the other hand, some implementation of modular multiplication output the result into ancilla, $|x⟩|0⟩ → |x⟩|ax \mod m⟩$, but the trick described in [29] solves the problem as follows. First, using an integer $a'$ such that $aa' \equiv 1 \mod m$, we construct a gate which performs $|x⟩|0⟩ → |x⟩|a'x \mod m⟩$ and its inverse. Then, we can implement the following sequence:

$$
|x⟩|0⟩ → |x⟩|ax \mod m⟩ → |ax \mod m⟩|x⟩ → |ax \mod m⟩|0⟩ .
$$

Here, the first, second and third steps are modular multiplication by $a$, swap and the inverse of modular multiplication by $a'$, respectively.

### B. Permutation

It is well-known that LCG suffers from some statistical flaws. [10] points out that performing permutation on LCG enhances its statistical properties. Here, permutation is transformation of binary representation of a PRN to another bit string. We here take some of permutations described in [10] as examples and show how to implement them in quantum circuit.

The first one is random rotation. We first divide a $n$-bit binary $x ∈ \mathbb{Z}_2^n$ into three parts: the top $t$ bits $x^t$, the middle $r$ bits $x^m$ and the bottom $n - t - r$ bits $x^{n-t-r}$, where $r$ is a power of 2 and $t = \log_2 r$. Then random rotation is a map from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2^n$ defined as

$$
x \mapsto \sigma_{rot}(x^t, x^m).
$$

Here,

$$
\sigma_{rot}(k, y) := \begin{cases} y & ; k = 0 \\ y_{r-k+1}y_1...y_{r-k} & ; 1 \leq k \leq r - 1 \end{cases}
$$

for an integer $k$ satisfying $0 \leq k \leq r - 1$, $y = y_1y_2...y_r \in \mathbb{Z}_2^r$ and $ab...$ represents a bit string whose first digit is $a \in \{0, 1\}$, second digit is $b \in \{0, 1\}$ and so on. In short, random rotation is clockwise rotation of middle digits of a binary where the

---

4 Such $b$ can be found if and only if $a - 1$ and $m$ are coprime. This condition is satisfied for many of widely used combination of $a$ and $m$. 
rotation width is determined by the value of top digits. Only the middle digits \(x^n_m\) are used to calculate the integrand as a \(r\) bit random number. Especially, the bottom digits \(x^n_{n-r}\) are discarded since their statistical properties are not good.

Random rotation is easily implemented in quantum circuit using controlled swap gate (Fredkin gate). The circuit diagram is shown in Figure [4]. The middle bits \(x^n_m\), is rotated by \(2^{m-1}\) bits by Rot_{\text{PRN}} under the control of the top \(i\)-th bit, for \(1 \leq i \leq t\). This leads to \(x^n_i\)-bit rotation of \(x^n\). We can construct the gate Rot_{\text{PRN}}, \(j = 0, 1, ..., t - 1\) connecting qubits with swap gates (actually Fredkin gates since these gates are controlled) as follows. Setting \(J = 2^t\):

- Connect \(|x_{t+1}\rangle_{\text{PRN}}\) and \(|x_{t+1}\rangle_{\text{PRN}}\) of Fredkin gates in Rot_{\text{PRN}} as explained above.

That is, there are \(J\) groups containing \(n/J\) qubits connected by \(n/J - 1\) swap gates. Note that \(r/J\) is an integer.

The second type of permutation is xorshift. This is a map from \(\mathbb{Z}_{2^n}\) to \(\mathbb{Z}_{2^n}\) defined as follows:

\[
x = x_1...x_n \rightarrow x_1...x_{n-s}y_1...y_s, \quad y_i := x_i \oplus x_{i-s+i}, \quad i = 1,...,s.
\]

Here, \(s\) is an integer satisfies \(1 \leq s \leq n - 1\) and typically comparable with \(n\), for example, \(n/2\) as proposed in [10]. Note that we do not need to take xorshift over the whole qubits in the PRN register. That is, we can take XOR between top qubits and middle qubits and discard bottom ones, as random rotation.

We can construct a gate which performs this permutation using CNOT gates. That is, put NOT on \(|x_{n-s+i}\rangle\) under control by \(|x_i\rangle\), for \(i = 1,...,s\), as shown in Figure [5]. Note the order to set CNOT gates, that is, from bottom to top. This is necessary in the case where \(s > n/2\) so that some middle qubits are used as both a target and a control. Such a qubit must work as a control before it becomes a target.

### C. Qubit Number and Circuit Depth

Here, we roughly estimate qubit number and depth of PCG circuits. We focus on \(P_{\text{PRN}}\), which is repeatedly used to progress PRN sequences. We consider PCG with \(r\)-bit output and \(n\)-bit background LCG. \(n\) should be so large that the period, \(2^n\) at most, is long enough and \(r\) is typically comparable with \(n\). For example, in many of settings considered in [10], \(n = 64\) and \(r = 32\).

The LCG part consists of modular addition and multiplication and dominant contribution comes from the latter. For \(n\)-bit operands, many of proposed modular adder require \(O(n)\) qubits including ancilla and \(O(n)\) depth. On the other hand, modular multipliers basically require \(O(n)\) qubits and \(O(n^2)\) depth, so this is dominant in the LCG part.[5]

The permutation part does not require any ancillas, at least two examples mentioned above. Circuit depth is found as follows. For random rotation on \(r\) bits with \(t = \log_2 r\) control bits, which we considered above, we first note that depth of swap gates in Rot_{\text{PRN}} is \(r/2 - 1\), since Rot_{\text{PRN}} consists of \(2^t\) groups of \(r/2\)qubits and \(r/2 - 1\) swap gates, as explained above. Summing up this for \(j = 0, 1, ..., t - 1\), it is found that the depth of Fredkin gates in the random rotation is \(2r - \log_2 r - 2\), that is, \(O(r)\). For xorshift with shift width \(s\), it is obvious that the depth of CNOT gates is \(s\) and if \(s\) is comparable with \(r\), say \(s = r/2\) as considered in [10], so is the depth.

In summary, in terms of both ancilla qubit number and circuit depth, dominan contribution comes from a multiplication and is \(O(n)\) and \(O(n^2)\) respectively. Therefore, if each calculation step for integrand \(f_i\) contains computations heavier than several multiplications, PRN generation makes subdominant contributions to qubit number and circuit depth.

### IV. APPLICATION TO CREDIT RISK MEASUREMENT

#### A. Merton Model

Now, let us consider the application of the aforementioned method to the actual problem in finance. We take credit risk measurement, which is mentioned in the introduction, as an example. First, we briefly explain Merton model[6], which is widely used in practice in many banks.

In this model, the stochastic loss amount \(L\) in a credit port-
The meaning of each symbol is as follows. $E_i$ is the exposure of the $i$th obligor, that is, the loss arising if he defaults. $1_C$ is the indicator function which is 1 if the condition $C$ is satisfied and 0 otherwise. The stochastic variable $Z_i$ is interpreted as "the value of the firm" for the $i$th obligor. We consider that he defaults if $Z_i$ becomes smaller than a threshold $z_i$. Usually, given a probability of default $p_i$ exogenously, $z_i$ is set as $z_i = \Phi_{SN}^{-1}(p_i)$, where $\Phi_{SN}$ is the distribution function for standard normal distribution and $\Phi_{SN}^{-1}$ is its inverse. $Z_i$ is given as a linear combination of two independent standard normal random variables $x$ and $\epsilon_i$. $x$ is common for all obligors and called a systematic risk factor, which is interpreted as a factor reflecting the situation of macro economy. $\epsilon_i$ is called an idiosyncratic risk factor and represents the effect of the matters unique to the $i$th obligor on his credit. We take the coefficient $\alpha_i$ such that $0 < \alpha_i < 1$, therefore $Z_i$ is also standard normal. $\alpha_i$ determines the correlation between $Z_i$ for different obligors: the larger $\alpha_i$ means stronger correlation and larger probability of simultaneous defaults of many obligors.

**B. Calculation of Loss Using PRNG on Quantum Circuit**

Then, we describe how to calculate credit risk measures using a PRNG on a quantum circuit. What we have to develop

---

6 Here, we assume that loss given default is 1.

7 Although we consider the case there is a single systematic risk factor, we can extend the model with multiple ones.
(a) Overview of the quantum circuit. Some ancilla qubits shown in (b) are omitted.

| $|x_i\rangle_{\text{nPRN}}$ | $|y_i\rangle_{L}$ | $|\sum_{j=1}^{N_{\text{obs}}} E_j |z_i, c_i\rangle_{R_L}\rangle$ |
|----------------|----------------|-------------------------------------------------|
| $|0\rangle$     | $|1\rangle$   | $|0\rangle$                                      |

(b) Detail of $f_i$. The first, second and fourth registers are $R_{\text{PRN}}, R_x$ and $R_L$ respectively. The third register, to which the result of comparison is output, is omitted in (a).

Figure 6: The Quantum circuit to calculate the loss amount in Merton model.

is the circuit which calculate stochastic loss amount $L$. Once we develop the circuit which create a superposition of states in which the value of the loss is encoded in some register, we can estimate VaR and CVaR as explained in [3]. The difference between the way in this paper and those in previous works is how to create such a superposition.

As mentioned in the introduction, calculation flow is similar to that in classical Monte Carlo. The steps to obtain one sample value of $L$ are as follows.

1. Generate a standard normal random variable and let it be $x$.
2. Set $i = 1$ and $L = 0$.
3. Set the first elements of the PRN sequence $x_1$.
4. Calculate $Y_i(x) = M_{\text{PRN}} P_i(x)$. Here,
   \[ P_i(x) = \Phi_{\text{SN}} \left( \frac{z_i - \alpha_i x}{\sqrt{1 - \alpha_i^2}} \right). \]
   \[ (16) \]
   is the conditional probability that the $i$ obligor defaults given $x$ and $M_{\text{PRN}}$ is the maximum number that the PRN can take.
5. Compare $x_i$ with $Y_i(x)$. If the former is smaller than the latter, update $L \leftarrow L + E_i$.
6. If $i = N_{\text{obs}}$, finish. Otherwise, progress the PRN sequence to get $x_{i+1}$, update $i \leftarrow i + 1$ and go to 4.

This flow is performed by the circuit in Figure 6. As explained in Section 11 we first create a superposition of $|x_1^{(1)}\rangle_{\text{nPRN}}, \ldots, |x_{(N_{\text{max}}-1)}^{(1)}\rangle_{\text{nPRN}}$ on $R_{\text{PRN}}$ using $H^{\text{PRN}}$ and $J_{\text{PRN}}$. These are starting elements of PRN sequences. Besides, we create a superposition of $x$, that is, numbers which obey the standard normal distribution in the register $R_x$. This is done by method described in [11] and depicted as the gate ‘SN’ in Figure 6. Then, progressing the PRN sequence by $P_{\text{PRN}}, L$ is sequentially calculated by $f_1, \ldots, f_{N_{\text{obs}}}$.

In $f_i$, at first, $x$ is converted to $Y_i(x)$ by the gate $Y_i$. Here simply assume that such a gate exists. In [2, 3], several ways to calculate such a function on a quantum computer are proposed, for example, linear approximation or piecewise polynomial approximation [13]. Then $x_i$ is compared with $Y_i(x)$ and a ancillary qubit is set to 1 if $x_i > Y_i(x)$. Such a comparator has been presented in [30]. With the control by the ancilla, $E_i$ is added to the loss register $R_L$. The controlled adder is also presented in previous papers, such as [10]. Finally, the inverses of $Y_i$ and comparison are performed to uncompute $R_x$ and the ancilla.

V. DEMONSTRATION: APPLICATION TO INTEGRATION OF SIMPLE MULTI-VARIABLE FUNCTION

Although the method proposed in this paper reduces required qubits, the circuit presented in the last section is still too large to perform in simulators or machines which can be publicly used today. We therefore consider a more small-scale problem performable in a simulator. It is an integral of a
We assume the PRNG outputs \( r \) and \( R \) for this calculation is shown in Figure 7. In addition to the aforementioned method, an alternative way is Monte Carlo integration numerically. Discretizing the integrand values on grid points set with equal interval in each axis, \( \int_0^\alpha x dx = \sum_{i=1}^{N-1} \sin^2 \left( \frac{\pi}{N} i + \frac{1}{2} \theta \right) \rho \), where \( N \) is the number of intervals in each axis. However, in this way, the computational load increases exponentially with the number of variables. So the alternative way is Monte Carlo integration, that is, taking the average of the integrand values on grid points which are randomly sampled using PRN:

\[
I \approx \frac{1}{N_{\text{samp}}} \sum_{i=1}^{N_{\text{samp}}} \sin^2 \left( \frac{\pi}{N} i + \frac{1}{2} \theta \right).
\]

Here, we take PRN sequences \( \{ x_j \}_{j=1}^{N_{\text{samp}}} \) for \( i = 1, \ldots, N_{\text{samp}} \). We assume the PRNG outputs \( r \)-bit numbers and they take values in \( \{0, 1, \ldots, 2^r - 1\} \). \[19\] is in the form to which the aforementioned method can be applied. The quantum circuit for this calculation is shown in Figure 7. In addition to \( R_{\text{samp}} \) and \( R_{\text{PRN}} \), the circuit has a qubit, which we hereafter write as \( R_{\text{rot}} \). The value of the integration \[19\] is encoded into its phase by the gate \( f \) in Figure 7. This gate is a sequence of rotations around \( y \)-axis \( R_y \), controlled by output qubits in \( R_{\text{PRN}} \). That is, if \( R_{\text{rot}} \) is in the state \( \cos \alpha |0\rangle + \sin \alpha |1\rangle \) for some real number \( \alpha \) and \( R_{\text{PRN}} \) is in the state corresponding to a random number \( x \) before \( f \), going through it transforms the state as follows:

\[
\cos \alpha |0\rangle + \sin \alpha |1\rangle \rightarrow \cos \left( \frac{x + 1/2}{2^r} - \theta \right) |0\rangle + \sin \left( \frac{x + 1/2}{2^r} - \theta \right) |1\rangle,
\]

that is, rotation by the angle \( \frac{x + 1/2}{2^r} \). Therefore, starting from the state in which all registers are 0, the entire circuit transforms the state as follows:

\[
|0\rangle \rightarrow |0\rangle_{\text{samp}} |0\rangle_{\text{PRN}} |0\rangle_{\text{rot}} \rightarrow \frac{1}{\sqrt{N_{\text{samp}}} \sum_{\alpha=1}^{N_{\text{samp}}}} |\alpha\rangle_{\text{samp}} |x_{\text{PRN}}^{(i)}\rangle_{\text{PRN}} \otimes \cos \left( \frac{\pi}{N} i + \frac{1}{2} \theta \right) |0\rangle + \sin \left( \frac{\pi}{N} i + \frac{1}{2} \theta \right) |1\rangle.
\]

So the probability to observe \( |1\rangle \) is equal to \[19\].

The probability to observe \( |1\rangle \) can be estimated, for example, in the way proposed in \[8\], which we here explain briefly.

---

Note that not all qubits in \( R_{\text{PRN}} \) represent output random numbers. For example, bottom bits in PCG are not used due to poor statistical property.
First we construct the operation

\[ Q = -AS_0A^{-1}S_1, \]

(22)

where \( A \) corresponds to the entire circuit in Figure\(^7\) \( S_0 \) multiplies \(-1\) to the state if all qubits are 0 or does nothing otherwise and \( S_1 \) multiplies \(-1\) to the state if \( R_{\text{rot}} \) is 1 or does nothing otherwise. If we write the probability to observe \( |1\rangle \) in \( R_{\text{rot}} \) in \( A|0\rangle \) all as \( \sin^2 \theta_k \), where \( \theta_k \in [0, \pi/2] \), that in \( |\Psi_{\text{m}}\rangle := Q^n A|0\rangle \) all is \( \sin^2((2m + 1)\theta_k) \). So, choosing a set of non-negative integers \( m_0, m_1, ..., m_N \) and making \( N_k \) observations of \( R_{\text{rot}} \) in \( |\Psi_{\text{m}}\rangle \) for each \( m_k \), we can estimate \( \theta_k \) as the maximum point of the following likelihood function:

\[
L_{\theta_k}(\theta_k) := \prod_{k=0}^{M} \sin^2((2m_k + 1)\theta_k) \cos^2((2m_k + 1)\theta_k) \]

(23)

where \( h_k \) is the number of observations where \( R_{\text{rot}} \) is \( |1\rangle \) in \( |\Psi_{\text{m}}\rangle \).

We have performed the actual calculation based on the above method using the quantum circuit simulator Qiskit on IBM\(^{[3]}\). The detailed setting is as follows. We estimate the integral (17) for \( \theta = \pi/6 \) and \( N_{\text{var}} = 2 \). Although such a two-dimensional integral can be done analytically, the problem must be small-scale enough to be performed in the simulator and we consider it is sufficient for proof-of-concept. For PRNG, we use LCG with parameters \( a = 11, c = 0, m = 31 \) and the seed 1. Then the PRN is 5-bit and the period is 30. We take \( N_{\text{rand}} = 8 \) sample points, so use 16 elements in the PRN sequence. Of course there are statistical concerns on the estimate based on such a small number of samples generated by such a small-scale PRNG, but it is inevitable in calculation on a simulator and sufficient for the current proof-of-concept purpose. If we can use a real quantum computer with sufficient qubits, say 100, we should use PCG under an appropriate setting: with sufficiently many qubits (say, 32-bit output and 64-bit background LCG), widely-used LCG parameters and permutation recommended in [10]. For the implementation of LCG, we use the adder presented in [27] and construct modular adder, multiplier and exponentiator based on the adder following the way in [16]. For \( \theta_k \) estimation, we take \( M = 8, N_k = 100 \) and \( m_k = 2^k \), as in [8].

We show the result in Table\(^{10}\). At the time when the integral (17) is approximated as (19), some error arises. This is the difference between (i) and (ii) in Table\(^{10}\) which will become smaller if we can take more sample points generated by a larger-scale PRNG. Estimation by quantum computer should converge to (ii), and the estimation obtained actually (iii) is close to (ii) as expected.

### VI. CONCLUSION AND DISCUSSION

In this paper, we considered reduction of qubit number in the quantum algorithm for Monte Carlo. Although its application to problems in finance are proposed in previous works, high-dimensionality of some of such problems requires many qubits if a quantum register is prepared for each of random numbers required to calculate one sample value of the integrand. Especially, for credit risk measurement, the required qubit number is proportional to the number of obligors, which can be \( \mathcal{O}(10^9) \). Then we proposed a novel way to reduce the qubit number. Considering the difference between what we calculate in the previous way of quantum-based Monte Carlo and that in classical Monte Carlo, we pointed out that estimating the average of sampled integrand values, which is calculated in classical Monte Carlo, by the quantum algorithm provides us with both quantum speed-up and qubit reduction. We saw that such a way is realized by the PRNG on quantum computer and presented a candidate for a PRNG implementable on a quantum computer, PCG, with concrete circuit diagrams. We also described how to implement credit risk measurement using PRNG on quantum computer and demonstrated a simple integral on a quantum circuit simulator as a proof-of-concept.

As a final note, let us consider the trade-off between qubit number and circuit depth. It is clear that qubit number reduction proposed in this paper increases circuit depth. It is change of the design of the circuit, from that parallely generate random numbers in different registers to that sequentially generate them in a register. Therefore, circuit depth is now proportional to the number of random numbers \( N_{\text{ran}} \). This might make full reduction of qubit number by this way impractical. Without quantum error correction\(^{[32,33]}\), which is expected not to be realized in near-term quantum computer, such a deep circuit will not be performable. Even if a machine with error correction is developed, deep circuits might suffer from long runtime of fault-tolerant gates\(^{[3,35]}\) and quantum computation with small computational load might not necessarily lead to short computational time.

However, we consider the above trade-off itself meaningful. Even if a quantum computer with large qubit number becomes in operation in the future, management of memory (that is, qubit) will be an important issue when it is applied to large-scale problems such as credit risk measurement. That is, when fully parallel computation is impossible due to shortage of memory, we have to perform some procedures in sequence. This is an issue which frequently arises also in today’s classical computation.

The method proposed in this paper provides a way to solve such a problem in large-scale Monte Carlo simulation by quantum computer. Consider the situation where \( N_{\text{ran}} \) random numbers are required to calculate the integrand but the available machine has so small number of qubits that only \( N_{\text{ran}}/n \) random numbers can be generated at the same time, where \( n \) is an integer satisfying \( n \geq 2 \). In such a case, we can parallely generate \( N_{\text{ran}}/n \) PRN sequences with \( n \) elements, calculate a

---

\(^{10}\) Here, “parallel” means not parallel computation in quantum superposition but that in separate memories, which is possible also in classical computers.

\(^{11}\) Note that, depending on problems, circuit depth can be proportional to \( N_{\text{ran}} \) even if random numbers are generated on different registers. That is, if there is no other way than calculating the integrand using random numbers in sequence, circuit depth is inevitably \( \mathcal{O}(N_{\text{ran}}) \), whether we generate random numbers sequentially or parallely. Calculation of loss in a credit portfolio can be parallelized, as explained in [33].
part of the integrand using the elements in each sequence one-by-one and finally merge partial results to obtain the entire integrand value. This leads to the circuit depth proportional to $n$. This is partial but maximum parallelism which can be done in the machine, although the depth is $n$ times larger than the full parallelism.

ACKNOWLEDGEMENTS

The authors thank Shumpei Uno of Mizuho Information & Research Institute and Kazuyoshi Yoshino of Mizuho-DL Financial Technology for helpful comments.

[1] A. Montanaro, Quantum speedup of Monte Carlo methods, Proc. Roy. Soc. Ser. A, 471, 2181 (2015)
[2] S. Woerner et. al., Quantum risk analysis, npj Quantum Information 5, 15 (2019)
[3] D. J. Egger et. al., "Credit Risk Analysis using Quantum Computers", arXiv:1907.03044
[4] P. Rebentrost et. al., "Quantum computational finance: Monte Carlo pricing of financial derivatives", Phys. Rev. A 98, 022321 (2018)
[5] N. Stamatopoulos et. al., "Option Pricing using Quantum Computers", arXiv:1905.02666
[6] R. C. Merton, "On the pricing of corporate debt: The risk structure of interest rates", J. Finance, 29, 449 (1974)
[7] G. Bassard et. al., Quantum amplitude amplification and estimation, Contemporary Mathematics, 305, 53 (2002)
[8] Y. Suzuki et. al., Amplitude Estimation without Phase Estimation, arXiv:1904.10246
[9] M. Matsumoto et. al., "Mersenne Twister: A 623-dimensionally Equidistributed Uniform Pseudo-random Number Generator", ACM Transactions on Modeling and Computer Simulation (TOMACS) 8, 1 , 3 (1998)
[10] M. E. ONeill, PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation, Harvey Mudd College Computer Science Department Technical Report (2014); http://www.pcg-random.org/
[11] L. Grover et. al., Creating superpositions that correspond to efficiently integrable probability distributions, arXiv:quant-ph/0208112
[12] P. Glasserman, "Monte Carlo Methods in Financial Engineering", Springer (2003)
[13] T. Haner et. al., "Optimizing Quantum Circuits for Arithmetic", arXiv:1805.12445
[14] Y. Cao et. al., "Quantum algorithm and circuit design solving the Poisson equation", New J. Phys. 15, 013021 (2013)
[15] M. K. Bhaskar et. al., "Quantum Algorithms and Circuits for Scientific Computing", Quantum Information and Computation, 16(3&4), 0197 (2016)
[16] V. Vedral et. al., Quantum Networks for Elementary Arithmetic Operations, Phys. Rev. A 54, 147 (1996)
[17] D. Beckman et. al., "Efficient networks for quantum factoring", Phys. Rev. A, 54, 1034 (1996)
[18] T. G. Draper, "Addition on a quantum computer", arXiv:quant-ph/0008033
[19] S. A. Cuccaro et. al., "A new quantum ripple-carry addition circuit", The Eighth Workshop on Quantum Information Processing (2004)
[20] Y. Takahashi et. al., "A linear-size quantum circuit for addition with no ancillary qubits", Quantum Information and Computation, 5(6), 440448 (2005)
[21] R. Van Meter et. al., "Fast quantum modular exponentiation", Phys. Rev. A 71(5), 052320 (2005)
[22] T. G. Draper et. al., "A logarithmic-depth quantum carry-lookahead adder", Quantum Information and Computation, 6(4), 351 (2006)
[23] Y. Takahashi et. al., "Quantum addition circuits and unbounded fan-out", Quantum Information and Computation, 10(9&10), 0872 (2010)
[24] R. Portugal et. al., "Reversible Karatsuba algorithm", Journal of Universal Computer Science, 12(5), 499 (2006)
[25] J. I. Alvarez-Sanchez et al., "A quantum architecture for multiplying signed integers", Journal of Physics: Conference Series, 128(1), 012013 (2008)
[26] Y. Takahashi et. al., "A fast quantum circuit for addition with few qubits", Quantum Information and Computation, 8(6), 636 (2008)
[27] H. Thapliyal et. al., Design of Efficient Reversible Logic Based Binary and BCD Adder Circuits, J. Emerg. Technol. Comput. Syst. 9, 17, 1 (2013)
[28] D. E. Knuth, "The Art of Computer Programming, Volume 2: Seminumerical Algorithms", Addison-Wesley (1981)
[29] I. L. Markov et. al., Constant-Optimized Quantum Circuits for Modular Multiplication and Exponentiation, Quantum Information and Computation, 12(5&6), 0361 (2012)
[30] D. S. Oliveira et. al., "Quantum bit string comparator: circuits and applications", Quantum Computers and Computing, 7, 17 (2007)
[31] G. Aleksandrowicz et al. Qiskit: An open-source framework for quantum computing (2019); https://qiskit.org/
[32] P. W. Shor, Fault-tolerant quantum computation, in Proceedings of the 37th Annual Symposium on Foundations of Computer Science, FOCS 96 (IEEE Computer Society, Washington, DC, USA, 1996) pp. 56
[33] A. Y. Kitaev, Fault-tolerant quantum computation by anyons, Annals of Physics 303, 2 (2003)
[34] A. G. Fowler et. al., Surface codes: Towards practical large-scale quantum computation", Phys. Rev. A 86, 032324 (2012)
[35] A. G. Fowler et. al., Low overhead quantum computation using lattice surgery, arXiv:1803.06709

\[12\] Again, this is possible only if the integrand allows such calculation.