Operating with Quantum Integers: An Efficient 'Multiples of' Oracle

Javier Sanchez-Rivero[1](✉), Daniel Talaván[1], Jose Garcia-Alonso[2], Antonio Ruiz-Cortés[3], and Juan Manuel Murillo[1,2]

[1] COMPUTAEX, Cáceres, Spain
jszrivero@gmail.com
[2] University of Extremadura, Cáceres, Spain
[3] Universidad de Sevilla, Sevilla, Spain

Abstract. Quantum algorithms are a very promising field. However, creating and manipulating these kind of algorithms is a very complex task, specially for software engineers used to work at higher abstraction levels. The work presented here is part of a broader research focused on providing operations of a higher abstraction level to manipulate integers codified as a superposition. These operations are designed to be composable and efficient, so quantum software developers can reuse them to create more complex solutions. Specifically, in this paper we present a 'multiples of' operation. To validate this operation, we show several examples of quantum circuits and their simulations, including its composition possibilities. A theoretical analysis proves that both the complexity of the required classical calculations and the depth of the circuit scale linearly with the number of qubits. Hence, the 'multiples of' oracle is efficient in terms of complexity and depth. Finally, an empirical study of the circuit depth is conducted to further reinforce the theoretical analysis.

Keywords: Quantum computing · Amplitude Amplification · Oracle · Multiples · Qiskit

1 Introduction

Quantum computing [17] uses quantum mechanics to perform computations in a different manner than classical computing [18]. Nowadays, quantum computers are in the NISQ (Noisy Intermediate-Scale Quantum) Era [19], which means their practical use is still limited by errors and the low number of qubits (quantum bits). However, the recent developments on quantum devices have allowed researchers to start testing on real quantum hardware the theoretical work on quantum algorithms, which has been a very active field for decades [12].

Quantum algorithms are useful when they can solve certain problems faster than any known classical algorithm [16]. This speedup is measured in terms of asymptotic scaling of complexity [4]. The work presented here is part of ongoing

M. Aiello et al. (Eds.): SummerSOC 2023, CCIS 1847, pp. 105–124, 2023.
https://doi.org/10.1007/978-3-031-45728-9_7

research aimed at providing programmers with operations on quantum states at a higher level of abstraction than the base quantum gates. The design of these operations aims at composability and efficiency, such that they can be reused to create larger solutions. More specifically, our research has begun with the goal of providing operations on a superposition quantum state that encodes integers with size determined by the number of qubits in the state [20]. These operations are not only useful for manipulating a quantum state encoding integers, they are also more efficient than the same operations in the classical domain. In addition, the quantum circuits that implement these operations are optimised in depth, with respect to Qiskit's automatic methods [23], as well as in the number of qubits (ancilla and non-ancilla) they use [21].

In particular, this paper presents an operation that computes multiples. Thus, given an integer and a quantum state that encodes integers, the operation phase-tags[1] the configurations of the quantum state that correspond to multiples of the given integer. While the complexity of finding the multiples in the classical domain is $\mathcal{O}(2^{n_N})$,[2] the complexity of the operation presented here is $\mathcal{O}(n_N)$, where n_N is the number of bits codifying the maximum size of the wanted multiples, N. This is a logarithmic scaling in the total number of states, which provides an exponential speedup with respect to classical calculations.

As mentioned before, in our research this 'multiples of' is part of a larger set of quantum operations on integers [20,21]. An important feature that we want the whole set of operations to preserve is composability. Thus, all the resources of this set are composable with each other and, for example, the 'multiples of' can be composed with a 'less than' operation to obtain the multiples of a given integer a less than another given integer b. In particular, the operation 'multiples of' can also be composed with itself to, for example, mark in a quantum state the multiples of an integer a and then, in the resulting state, mark the multiples of another integer b. Preserving composability offers the possibility to reuse such operations to build more complex operations with a higher level of abstraction. Achieving the best quality attributes, such as reusability or composability, in each operation is important because their compositions will inherit those attributes [14,24].

This paper is organized as follows. In Sect. 2 we provide the necessary background for this work. Next, the description of the 'multiples of' operation is presented in Sect. 4. The operation takes the shape of a quantum oracle and the section details both the idea inspiring the oracle and the quantum circuit exact implementation. In Sect. 5 some examples of circuits and simulations are shown to prove the functionality of the oracle. Then, in Sect. 6, the complexity of classical calculations as well as the quantum circuit is discussed. Section 7 shows composability and further uses of the oracle. Finally, in Sect. 8, the conclusions and future work are explored.

[1] Phase-tagging a state is giving that state a π-phase.

[2] Given $k \in \mathbb{N}$, there are $\lceil N/k \rceil$ multiples of k in $[0, N]$. As k is fixed, the number of multiples grows linearly in N, $\mathcal{O}(N)$. Namely, if $N = 2^{n_N}$, then $\mathcal{O}(2^{n_N})$.

2 Background

Oracles have been identified as a recurring pattern in quantum software [15,24]. Following this trend, the work presented here is built as an oracle for Grover's algorithm [9]. This algorithm searches for one quantum state in an unordered database faster than any known classical algorithm. Its generalization is called Amplitude Amplification [3,10] and allows to search for multiple values. This algorithm works by applying two quantum operations: an oracle, which marks with a π-phase the desired quantum states, and a diffuser, which tries to amplify the amplitude of those marked states. Often, to reach a satisfactory amplification, it is needed to repeat the pair oracle-diffuser several times. This pair oracle-diffuser is usually called the Grover iterator.

The 'multiples of' oracle is built reusing two pieces of existing quantum software, the linear multi-controlled gate [22] and the modulo addition [1,7].

In [22] the authors present an efficient implementation of a multi-controlled gate whose depth scales linearly with the number of qubits and thus avoids the polynomial growth of previous implementations. Furthermore, it does not require the use of ancilla qubits. The linear multi-control gate outperforms Qiskit [23] implementation from five qubits onwards, which supposes a clear improvement for any meaningful use. Furthermore, the authors also conduct an analysis on the utility of the linear multi-controlled gate on NISQ devices, showing that the depth reduction can help achieve more accurate results. Because of these quality attributes, we chose to reuse it in this work.

The modulo addition operation $a + b \bmod k$ is defined as the remainder of dividing $a + b$ by k. In this work, this operation will only be performed with values $a, b < k$. Hence, in our case, the modulo addition can be written as:

$$a + b \bmod k = \begin{cases} a + b & \text{if } \ a + b < k \\ a + b - k & \text{if } \ a + b \geq k \end{cases} \qquad a, b < k \tag{1}$$

For this modulo addition we use the implementation presented in [1]. It is heavily based on Draper's algorithm [7] for quantum addition. This method uses the quantum Fourier transform [6] and hence is done on the frequency domain[3]. It allows the addition of an integer to a quantum superposed state without the need to encode the integer in a quantum register. This reduces the number of necessary qubits. The depth of this operation is linear on the number of qubits, as it is a composition of linear-depth primitives.

Once the addition gate is built, the modulo addition conducts the following operations [1]: adds a classical value $a < k$ to a quantum state holding the classical value $b < k$, and then it subtracts k if $a + b \geq k$. This methods requires two ancilla qubits to perform the operation, one for the overflowing of the sum, and another one for checking whether it is needed to subtract k or not.

[3] In the frequency domain, integers are represented by superposition of states with the same different in phase between the state. That difference in phase is the unique identifier of the integer [18].

This paper showcases how by carefully composing existing pieces of quantum software a new non-trivial software can be obtained. The ideas hereby described are the ones which allow to build the oracle ‘multiples of’, presented in detail in Sect. 4.

3 Related Works

There are several approaches which seek to create higher-level quantum programming languages. Quipper [8] is a scalable quantum programming language which can be used to program several quantum algorithms, such as HHL [11]. Silq [2] is a high-level quantum language which focuses, among other objectives, in the automatic uncomputation of operations usually required in quantum programming. Another construction of higher-level quantum program is Classiq [5], which researches in building oracles for arithmetic expressions. Latter one is the closest to our work, previous ones focus on quantum primitives with a more general approach. Operations in Classiq, range from addition or subtraction to built-in functions such as Amplitude Estimation. These languages aim at creating a whole set of operations to be able to create quantum software without the need of deep knowledge on quantum circuits. Our work follows the same idea and attempts to create new more complex operations with efficient classical calculations.

4 Implementation of the ‘Multiples of’ Oracle

In this section, we provide a description of the oracle. It comprises two differentiated parts, the first one is the mathematical ideas inspiring the oracle, where basic modulo theory shows a condition for identifying multiples. The second part describes the quantum circuit of the oracle, how the multiples are given a π-phase, and how to adapt the oracle for a full Amplitude Amplification implementation.

4.1 Mathematical Properties Inspiring the Oracle

A number $M \in \mathbb{N}$ is multiple of another number k if the remainder of the division is 0, formally expressed as $M \equiv 0 \bmod k$. If M is not a multiple of k, then $M \not\equiv 0 \bmod k$.

The number M can be be expressed in binary form, also known as binary decomposition:

$$M = a_m \cdot 2^m + a_{m-1} \cdot 2^{m-1} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 = \sum_{i=0}^{m} a_i \cdot 2^i \qquad (2)$$

where $a_i \in \{0,1\}$ and $m = \lceil \log_2 M \rceil$.

Let r_i, with $0 \leq r_i < k$, be the remainder of 2^i when divided by k, formally:

$$2^i \equiv r_i \bmod k \tag{3}$$

Then by the properties of the ring of remainders $\mathbb{Z}_k$ it can be noticed that:

$$M \equiv \sum_{i=0}^{m} a_i \cdot 2^i \equiv \sum_{i=0}^{m} a_i \cdot r_i \bmod k \tag{4}$$

Hence, M is a multiple of k if the sum of the remainders of the powers of 2 modulo k of its binary decomposition is equivalent to 0, formally:

$$M \equiv 0 \bmod k \iff \sum_{i=0}^{m} a_i \cdot r_i \equiv 0 \bmod k \tag{5}$$

Therefore, the 'multiples of' oracle is built in two parts, first adding the remainders of the powers of two, and then giving a π-phase to those which are 0, thus the multiples.

4.2 Algorithm for the 'Multiples of' Oracle

This subsection provides a detailed explanation of the implementation of the 'multiples of' oracle.

Let $k \in \mathbb{N}$ be the number whose multiples want to be calculated. The quantum circuit for the 'multiples of k' oracle consists of three registers of qubits.

The first is the input register, which holds the quantum states in which the multiples will be searched. It is formed by n qubits and the i-th qubit of this register is denoted q_i. The number n is an input parameter and does not depend on any other value. Thus, the numbers in which the multiples will be searched range from 0 to $N-1$, where $N = 2^n$.

The second register holds the remainder of the numbers. At most, the remainder of dividing by k is $k-1$, hence the required number of qubits for this register is $n_k = \lceil \log_2(k-1) \rceil$. The i-th qubit of this register is denoted rq_i.

Finally, an ancilla register with two qubits is needed to perform the modulo addition introduced in the Sect. 2, as described in detail in [1]. These qubits are denoted ancilla$_0$ and ancilla$_1$. Both the registers and the ancilla registers are initialized to state $|0\rangle$.

Algorithm 1 builds the circuit. It follows an explanation which describes it thoroughly.

The remainders of each power of 2, $r_i \equiv 2^i \bmod k$, are added modulo k to the remainders register, where the addition is controlled by the input qubit q_i. As the remainders register is initialized as $|0\rangle$ and $r_i < k \ \forall i$, the result of each modulo addition will never be larger than k, as shown in the definition of this operation in Sect. 2. Figure 1 shows the general case of this implementation. This image and all the others showing circuits have been done with the *quantikz* package [13].

```
Data: Number of qubits $n$ and a natural number $k$
Result: Quantum Circuit which gives a $\pi$-phase to all states representing
        binary forms of natural numbers multiples of $k$
Calculate $r_i \equiv 2^i \bmod k$ for $i \in [0, n-1]$ ;            /* see Appendix A */
$n_k \leftarrow \lceil \log_2(k-1) \rceil$;
input_register $(q) \leftarrow QuantumRegister(n)$;
remainder_register $(rq) \leftarrow QuantumRegister(n_k)$;
ancilla_register $\leftarrow QuantumRegister(2)$;
$n_{total} \leftarrow n + n_k + 2$;
$circ \leftarrow QuantumCircuit(n_{total})$;
Initialize input_register to $|0\rangle$;
for $i = 0$ to $n-1$ do
    $circ^{+} = ModuloAddition(r_i, n_k + 2)$;
    append $circ^{+}$ to $circ$:
      - Target: remainder_register and ancilla_register;
      - Control: $q_i$;
end
for $j = 0$ to $n_k - 1$ do
    $X$ gate to $rq_j$;
end
$CZ^{n_k}$ gate to qubits $rq_0, \ldots, rq_{n_k-1}$;
;                      /* Target: $rq_{n_k-1}$, Control: $rq_0, \ldots, rq_{n_k-2}$ */
for $j = 0$ to $n_k - 1$ do
    $X$ gate to $rq_j$;
end
for $i = 0$ to $n-1$ do
    $circ^{-} = ModuloSubstraction(r_i, n_k + 2)$;
    append $circ^{-}$ to $circ$:
      - Target: remainder_register and ancilla_register;
      - Control: $q_i$;
end
```

Algorithm 1: Algorithm for building the 'multiples of' oracle

After applying the modulo additions, the ancilla register is always at state $|00\rangle$ [1]. The remainders register holds states from $|0\rangle$ up to $|k-1\rangle$. From Eq. 5, it can be seen that the multiples of k are those states of the form:

$$|rq_{n_k-1} \ldots rq_1\, rq_0\, q_{n-1} \ldots q_1\, q_0\rangle = |\underbrace{0 \ldots 0\, 0}_{rq \text{ register}}\ \underbrace{q_{n-1} \ldots q_1\, q_0}_{\text{input register}}\rangle \tag{6}$$

Hence, these states and only these ones will be given a π-phase by means of the gate 7:

$$X^{\otimes n_k} \cdot CZ^{n_k} \cdot X^{\otimes n_k} \tag{7}$$

where CZ^{n_k} is a multi-controlled Z-gate whose target is qubit rq_{n_k-1} and controlled by qubits $rq_0, \ldots, rq_{n_k-2}$. This gate is built using the linear multi-controlled Z-gate introduced in Sect. 2. Figure 2 shows how this part of the circuit is built.

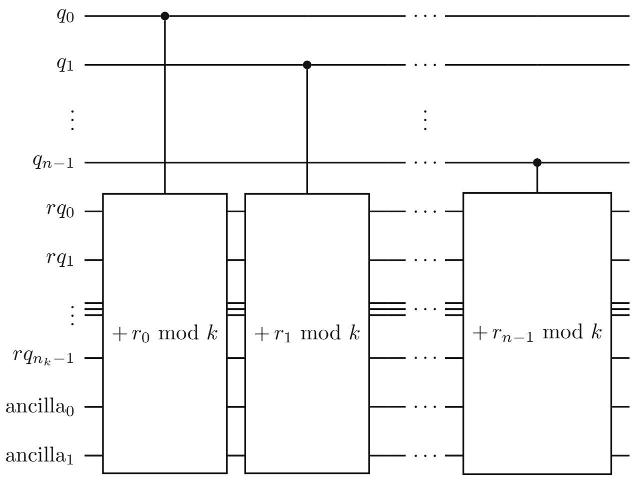

Fig. 1. Modulo addition of the remainders of the powers of 2.

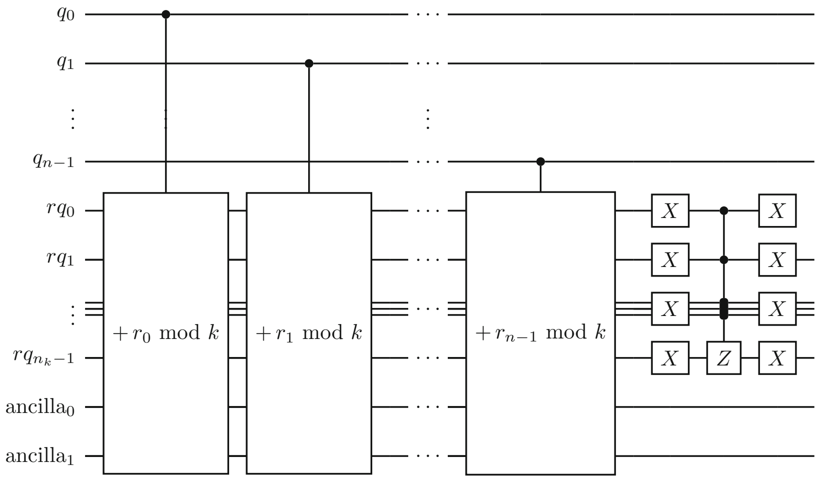

Fig. 2. Oracle that marks multiples of k.

Afterwards, in order to apply the diffuser, it is required to return auxiliary qubits to their initial states, that is, to perform an uncomputation on this register [18]. As the multiples are already given a π-phase, if the modulo additions of the remainders are uncomputed, the states would keep the phase. The circuit would be as in Fig. 3.

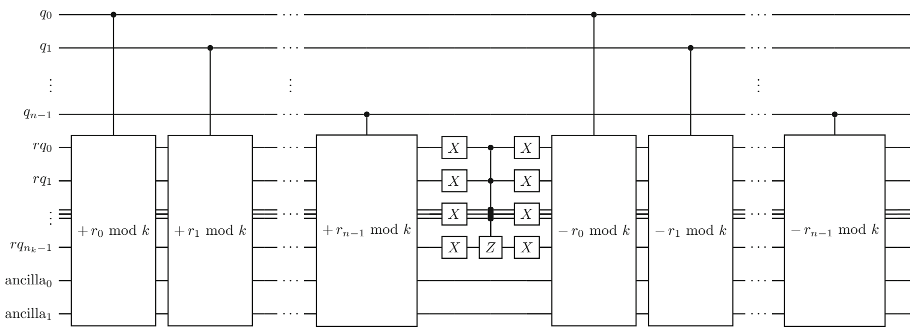

Fig. 3. Oracle that marks multiples of k and returns all auxiliary qubits to state $|0\rangle$.

Finally, the diffuser can be applied to the input register (the rest of the registers, remainders and ancilla, are in the state $|0\rangle$). The complete implementation is shown in Fig. 4.

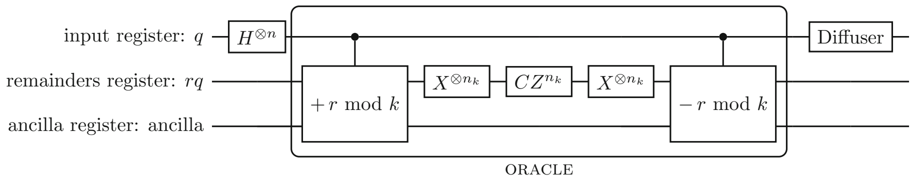

Fig. 4. Implementation of 'multiples of' oracle plus diffuser with full superposition as input.

A documentation for this 'multiples of' oracle following the guidelines proposed in [21] can be found in the following repository.

5 Simulations and Results

In this section, we show some examples of the oracle, both the implementations and the results, which are obtained by means of a simulator with no noise and no error model applied. Different values for k, the number whose multiples are calculated, and n, the number of input qubits[4], are chosen to showcase the functionality of the 'multiples of' oracle. We display the full circuit for the multiples of 3 oracle with 4 qubits input as well as its simulation. We also show the full circuit and the simulation of multiples of 5 with 6 qubits. We have chosen these values for k and number of input qubits to improve readability of the circuits.

[4] We codify all our integers as quantum states in these n qubits, hence the multiples are calculated up to $N = 2^n$ integers.

In addition, we also show a simulation of multiples of 14 with 5 qubits using one and two repetitions of the Grover iterator to showcase the difference in the amplified amplitude in both cases.

We have used Qiskit [23] to generate the circuits and simulate them. To be able to amplify the marked quantum states we have chosen a full superposition of 0*s* and 1*s* as our input state and have applied the Grover's algorithm diffuser [18] after the oracle. All the simulations are conducted with 20,000 shots[5] as it is the maximum allowed by Qiskit.

5.1 Multiples of 3

The 'multiples of 3' oracle with 4 qubits as input can be found in Fig. 5. The remainders of the powers of 2 when divided by 3 follow the cycle 1, 2, as:

$$\begin{aligned} 2^0 &\equiv 1 \bmod 3 \\ 2^1 &\equiv 2 \bmod 3 \\ 2^2 &\equiv 1 \bmod 3 \end{aligned} \tag{8}$$

As there are 4 input qubits, the remainders of the first 4 powers of 2 are added in the remainders register. These 4 remainders are 1, 2, 1, 2, repeating the whole cycle once.

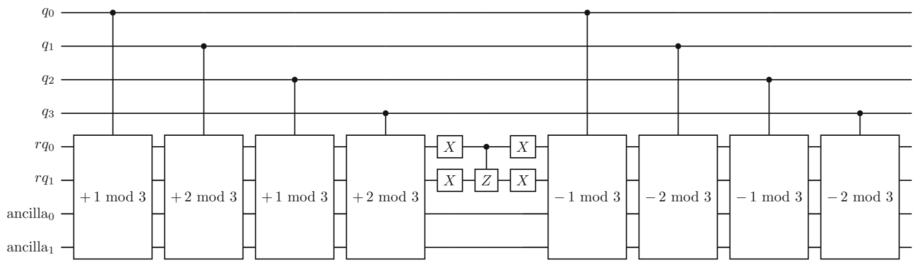

Fig. 5. Multiples of 3 oracle with a 4 qubits input register.

The result of simulating the entire circuit, including the initialisation of the state, the shown oracle, and the diffuser, are shown on Fig. 6. The x-axis represents the final quantum states and the y-axis represents the relative frequency. Desired states (multiples of 3) are in blue with a thick border, undesired states (not multiples of 3) are in red without border. It can be noticed that with just one repetition, the desired states are clearly amplified to differentiate the multiples of 3 from the rest of the numbers.

5 Each shot is one simulation of the circuit, the final result is the aggregation of all shots.

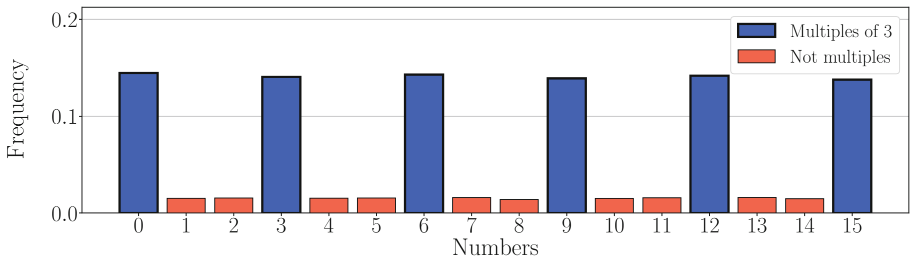

Fig. 6. Results of simulating the circuit of multiples of 3 with a 4 qubits input.

5.2 Multiples of 5

The ‘multiples of 5’ oracle with 6 qubits as input can be found in Fig. 7. The remainders of the powers of 2 when divided by 5 follow the cycle 1, 2, 4, 3, as:

$$\begin{aligned} 2^0 &\equiv 1 \bmod 5 \\ 2^1 &\equiv 2 \bmod 5 \\ 2^2 &\equiv 4 \bmod 5 \\ 2^3 &\equiv 3 \bmod 5 \\ 2^4 &\equiv 1 \bmod 5 \end{aligned} \tag{9}$$

As there are 6 input qubits, the remainders of the first 6 powers of 2 are added in the remainders register. These 6 remainders are 1, 2, 4, 3, 1, 2, repeating the first remainders of the cycle, 1 and 2.

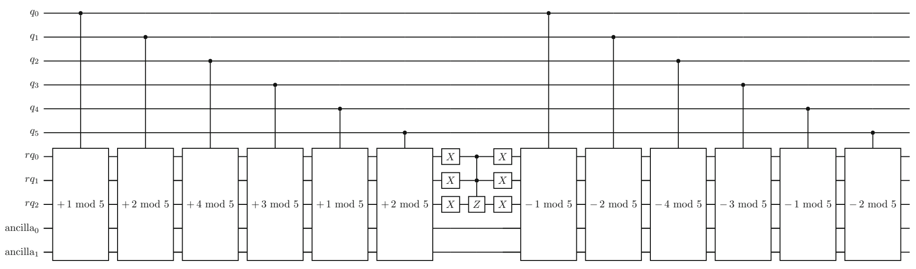

Fig. 7. Multiples of 5 oracle with a 6 qubits input register.

The result of simulating the full circuit, including the initialisation of the state, the shown oracle and the diffuser; are shown on Fig. 8. It can be seen that, in this case, the amplification with one iteration is almost perfect.

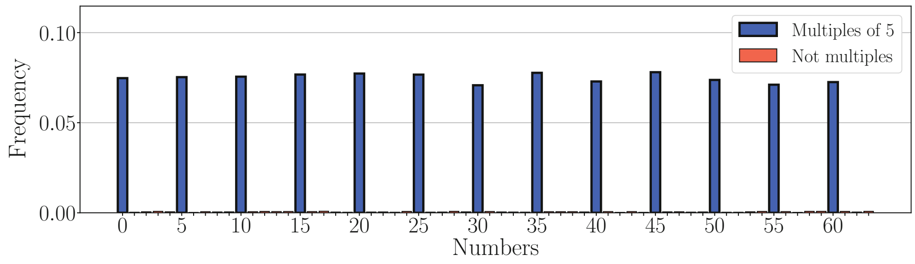

Fig. 8. Results of simulating the circuit of multiples of 5 with a 6 qubits input.

5.3 Multiples of 14

In this section, we show the results of simulating the ‘multiples of 14’ with 5 input qubits in full superposition with one and two repetitions of the Grover iterator. The remainders of the powers of 2 are 1, 2, 4, 8, and 2. We do not show the circuits for the sake of readability, however they can be found in the provided repository.

Figure 9 shows the results of the simulation using one repetition. The total amount of amplification of desired states is ≈ 64%. Although from an absolute perspective this may not seem a favourable result, the three desired states (0, 14 and 28, multiples of 14 up to 31) have their amplitude enlarged by a factor ≈ 6.82. This amplification allows the distinction of multiples of 14 among the input states.

On Fig. 10 are depicted the results of the simulation using two repetitions. In this case, the total amount of amplification is ≈ 100%. This is the best possible amplification and shows that this oracle may improve its applicability by repeating the pair oracle-diffuser. However, the increased depth of this operation has to be taken into account when implementing the operation in a real quantum device.

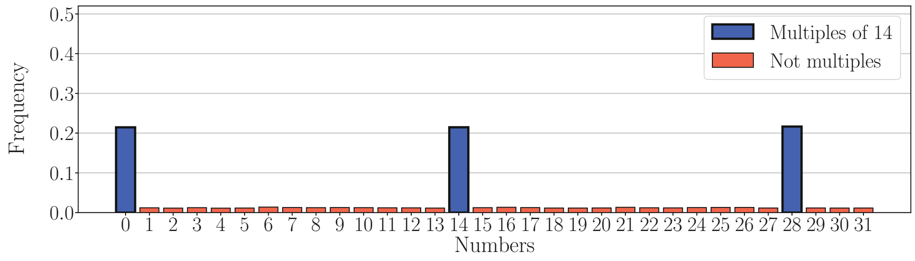

Fig. 9. Results of simulating the circuit of multiples of 14 with a 5 qubits input with one repetition of the Grover iterator.

Knowing the number of desired states, M, and the total number of states, N, the number of repetitions to reach maximum amplification can be calculated exactly [18]. Further analyses on the number of repetitions are conducted in [9] [10].

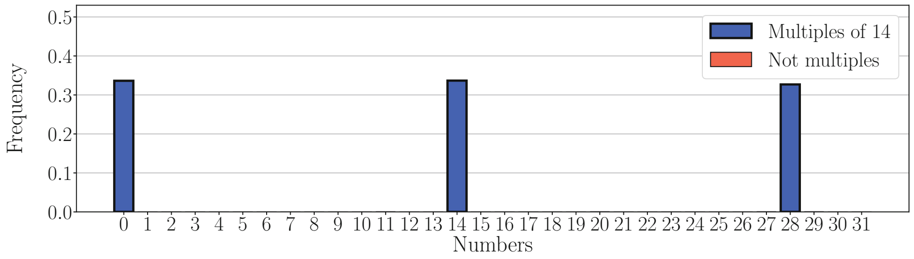

Fig. 10. Results of simulating the circuit of multiples of 14 with a 5 qubits input with two repetitions of the Grover iterator.

6 Complexity and Depth

In this section, we present both an analysis of the classical calculations required to build the quantum oracle and also a theoretical and empirical study of the depth of the resulting quantum oracle.

Let us bear in mind that $k \in \mathbb{N}$ is the number whose multiples want to be calculated; n is the number of input qubits, in which the multiples of k are going to be calculated; $N = 2^n$ is the total number of quantum states; and n_k is the required number of qubits to store the remainders of dividing by k (at most $k-1$).

6.1 Classical Calculations Complexity

In this subsection, we analyse the classical calculations needed to implement the 'multiples of' quantum oracle. This classical part is divided in two tasks. First one is computing the remainders of the powers of 2 divided by k. Second task is building the quantum circuit.

The first task consists on the calculations of the remainders $r_i \equiv 2^i \bmod k$, $0 \leq r_i < k$ and $i \in [0, n-1]$. At most, only n remainders need to be calculated, as only n modulo additions are conducted. Therefore, this operation is $\mathcal{O}(n) = \mathcal{O}(\log N)$. The algorithm to do these computations can be found in Appendix A.

The second task is building the quantum circuit. The construction of the controlled circuit '$+r \bmod k$' which performs the modulo addition is linear on the number of qubits [1]. In this case, the number of qubits on the remainders register, n_k. This means that the complexity of this operation is $\mathcal{O}(\log k) =$

$\mathcal{O}(n_k)$. This is smaller than $\mathcal{O}(n)$, otherwise, k would be greater than $N = 2^n$ and there would be only one multiple in those integers, the number 0.

Moreover, the complexity of appending the modulo addition circuits to the full quantum circuit is linear on the number of qubits on the input register, n, thus, $\mathcal{O}(n) = \mathcal{O}(\log N)$. The rest of needed appends (Hadamard, X and multi-controlled Z) are also linear with the number of qubits. Therefore, the complete procedure required in classical computations holds a complexity of $\mathcal{O}(n) = \mathcal{O}(\log N)$.

It can be noted that for obtaining the multiples of a given number k up to N classically, it is needed to calculate $\lceil N/k \rceil$ multiples. Hence, as k is already fixed, this calculation grows exponentially with the number of binary bits, n_N, needed to encode N in binary form, $\mathcal{O}(N) = \mathcal{O}(2^{n_N})$. To apply our method, we need to encode N in a quantum circuit and n_N qubits are needed. As showed above, the complexity of the classical computations of our method is $\mathcal{O}(n_N)$, hence, our method presents an exponential reduction of the complexity of the classical computations.

6.2 Theoretical Analysis of Quantum Circuit Depth

As stated in Sect. 4.2, the quantum circuit consists of three registers of qubits, the input qubits, which hold the information for all the possible numbers, formed by n qubits, which is input from the user. The register which holds the remainder of the numbers, which has $n_k = \lceil \log_2(k-1) \rceil$ qubits. At most, the remainder of dividing by k is $k-1$, hence not more qubits are required. Finally, an ancilla register with two qubits is needed to perform the modulo addition, as described in detail in [1]. The depth of this circuit is determined by the depth of its two reused oracles, the modulo addition and the phase-marking operation.

The modulo addition '$+r \bmod k$' has linear depth on the number of qubits, in this case $\mathcal{O}(n_k) = \mathcal{O}(\log k)$, as it is applied on the remainders register. Once k is chosen, the depth of this circuit is fixed. This operation needs to be applied $2n$ times, firstly to compute the remainders and afterwards to uncompute them. Therefore, the depth of this operation is $\mathcal{O}(n) = \mathcal{O}(\log N)$.

The phase-marking operation requires a multi-controlled Z-gate. This is implemented following [22], which provides a linear depth on the number of qubits, $\mathcal{O}(n_k) = \mathcal{O}(\log k)$. As stated in the previous Subsect. 6.1, this is upper-bounded by $\mathcal{O}(n)$.

Therefore, the depth of the full implementation of the 'multiples of' oracle is linear on the number of input qubits n, $\mathcal{O}(n) = \mathcal{O}(\log N)$.

6.3 Empirical Measurement of Circuit Depth

To further reinforce the depth complexity study, an empirical analysis is also presented. In order to do this, we have generated the oracles for different numbers of k, n_k, and n. To properly perform this analysis, before measuring depth, all the circuits have been transpiled using one of the IBM quantum computer backends. In particular, the one used has been *fake_washington_v2*, which has the same properties (gate set, coupling map, etc.) as the real quantum device Washington.[6].

Figure 11 shows the depth of the oracle with respect to the number of input qubits n, for different values of k and n_k. It can be noticed that the depth grows linearly as the number of input qubits increases. This is an expected behaviour as theoretically explained above.

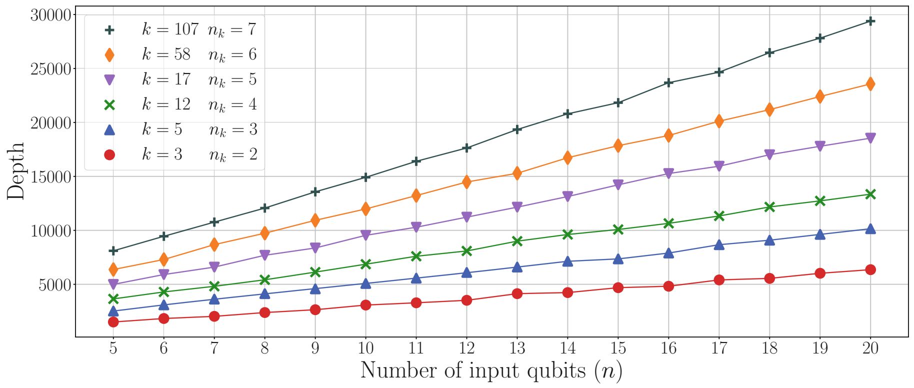

Fig. 11. Depth (y-axis) against number of input qubits n (x-axis) for different values of k.

It can also be noticed that the slope of the graphic grows as the number of qubits in the remainders register n_k increases. This is also an expected behaviour, as the depth of the modulo addition grows linearly with the number of qubits on which it is applied. This behaviour can be observed in Fig. 12. This figure shows the growth of the depth with respect to the number k whose multiples are to be computed. This analysis has been conducted by choosing several pseudo-random numbers in each interval $[2^{n_k-1}, 2^{n_k})$, with $n_k \in \{3, 4, 5, 6, 7\}$. These intervals are delimited by vertical dotted lines on the figure. It can be observed that the depth for each value holds mostly constant in these intervals. This means that the depth increments are mainly caused by the growing number of n_k qubits required to store the remainders of k (largest number stored is $k - 1$).

Lines in both figures are mere visual guides and do not represent any data.

7 Composability and Further Uses

In this section, we show how the proposed oracle can be further reused by providing some examples. First, we showcase how the 'multiples of' oracle can be composed with other oracles. Second, we explain how the oracle can be modified to obtain, instead of multiples of a number k, numbers with a determined

[6] https://qiskit.org/documentation/apidoc/providers_fake_provider.html.

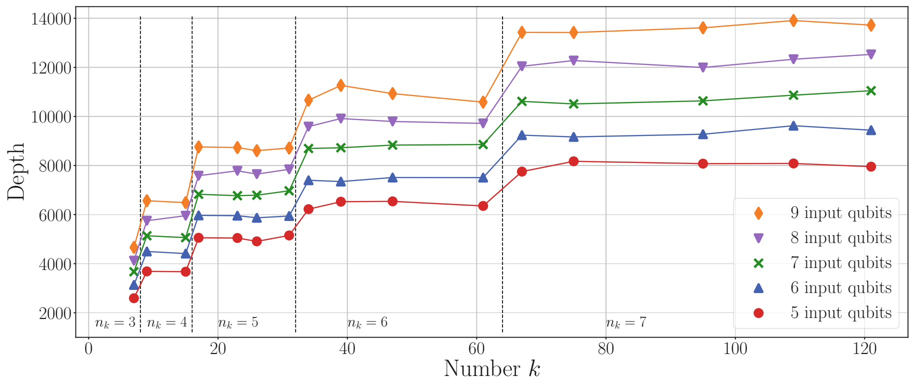

Fig. 12. Depth (y-axis) against k (x-axis) for different amounts of input qubits (n). The corresponding value of n_k for each k is displayed.

remainder when dividing by k. Last, both of these options are combined. Both circuit and results of simulations are displayed in each case. The conditions for the simulations are the same as previously described in Sect. 5.

7.1 Multiples and Less-Than Oracle

We show an example on how to obtain the multiples of a given number k smaller than m. In order to do so, the 'multiples of' oracle and the 'less-than' oracle [20] are composed[7]. However, this composition is not trivial since it must be applied in an specific way. The oracle to compose with ('less-than' in this example) must be applied controlled by the qubits in the remainders register $rq_0, \ldots, rq_{n_k-1}$ and targeted on the input register $q_0, \ldots, q_{n-1}$. This oracle substitutes the multi-controlled Z-gate which is used originally to mark all the multiples.

In this example, the choices are $k = 5$, $m = 14$, $n = 5$. Hence, the desired states are the multiples of 5 smaller than 14 from 0 to 31. The implementation of this oracle can be found in Fig. 13. The results of the simulation using only one repetition of the Grover iterator is in Fig. 14. The results are, as expected, the states amplified of the multiples of 5 less than 14.

7.2 Numbers with Any Remainder

This subsection shows how to change the multiples oracle in order to obtain numbers with any remainder r when dividing by a given integer k. The operation 'multiples of' explained so far is the particular case $r = 0$. In this example, we show the oracle taking $k = 6$, $r = 3$, $n = 5$, formally, $p \equiv 3 \bmod 6$. The oracle can be found in Fig. 15. Notice that, when giving a π-phase with gate CCZ in the remainders register, there are only X gates in the qubit rq_2, hence marking

[7] The 'multiples of' oracle can be combined with any other phase-marking oracle.

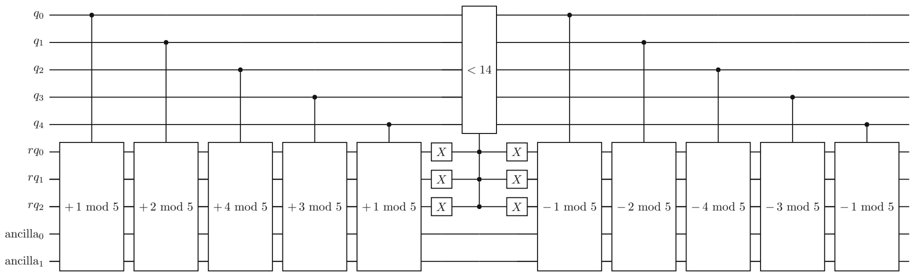

Fig. 13. Multiples of 5 oracle combined with less than 14 oracle with a 5 qubits input.

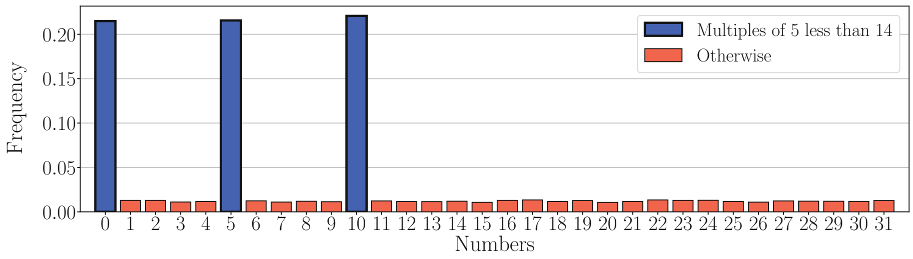

Fig. 14. Results of simulating the circuit of multiples of 5 less than 14 with a 5 qubits input.

those states where $|rq_2\, rq_1\, rq_0\rangle = |011\rangle = |3\rangle = |r\rangle$. The results of the simulation using only one repetition of the Grover iterator is shown in Fig. 16 and match the expected results for this operation.

7.3 Numbers with Any Remainder and Range of Integers

This subsection shows how to combine the oracle of numbers with a determined remainder when dividing by a number and the range of integers oracle presented in [21]. For instance, here we show the oracle for integers $p \equiv 5 \bmod 9$ and $p \in [12, 28]$. The oracle can be found in Fig. 17. Notice that, as in Subsect. 7.1, there is an oracle controlled by the qubits in the remainders register. However, in this case, the X gates are arranged such that the oracle is activated when the qubits in the remainders register are in the state $|rq_3\, rq_2\, rq_1\, rq_0\rangle = |0101\rangle = |5\rangle$. The results of the simulation using only one repetition of the Grover iterator is shown in Fig. 18.

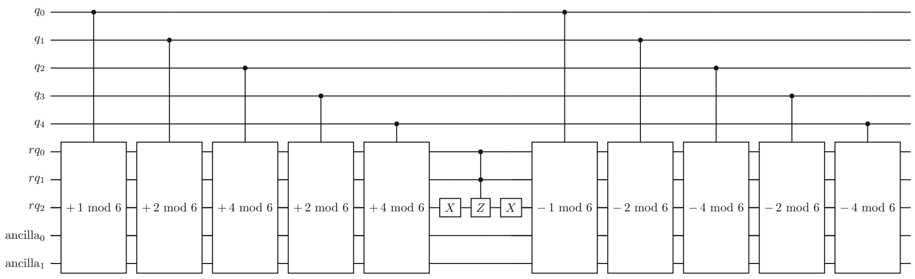

Fig. 15. Numbers $p \equiv 3 \bmod 6$ oracle with a 5 qubits input.

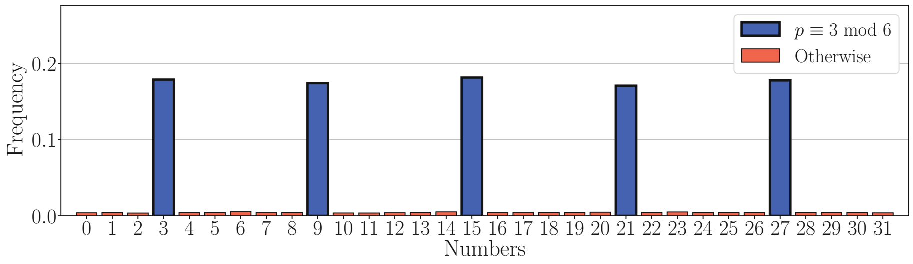

Fig. 16. Results of simulating the circuit of numbers $p \equiv 3 \bmod 6$ with a 5 qubits input.

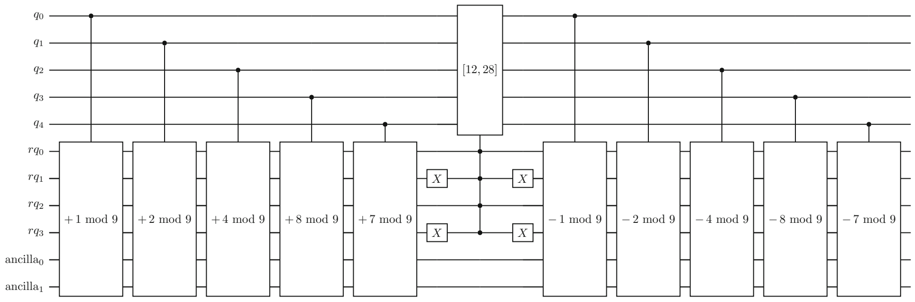

Fig. 17. Numbers $p \equiv 5 \bmod 9$ with $p \in [12, 28]$ oracle with a 5 qubits input.

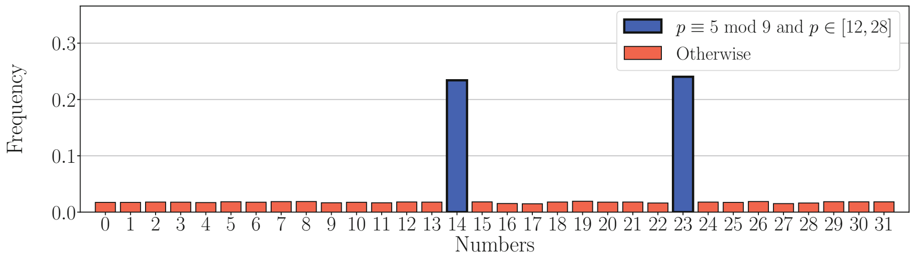

Fig. 18. Results of simulating the circuit of numbers $p \equiv 5 \bmod 9$ with $p \in [12, 28]$ with a 5 qubits input.

8 Conclusions

In this work, we have presented a method to build an efficient oracle for phase-marking multiples of a given number. We have shown the theoretical ideas behind this construction and how to build the quantum circuit. Moreover, we have conducted a theoretical analysis of the complexity of both the classical calculations needed to build the oracle and the oracle itself. The result of this analysis is that our method leads to an exponential speedup over the classical one in terms of the required classical computations. Finally, further functionalities are explored. Through examples and simulations we show how to compose the 'multiples of' oracle with other oracles and also how numbers with other properties can be obtained.

This work is one of the steps taken to create an efficient set of tools of quantum software for working with integers. We hope these tools can be reused by quantum software developers to create new quantum algorithms.

Acknowledgements. This work has been financially supported by the Ministry of Economic Affairs and Digital Transformation of the Spanish Government through the QUANTUM ENIA project call - Quantum Spain project, by the Spanish Ministry of Science and Innovation under project PID2021-124054OB-C31, by the Regional Ministry of Economy, Science and Digital Agenda, and the Department of Economy and Infrastructure of the Government of Extremadura under project GR21133, and by the European Union through the Recovery, Transformation and Resilience Plan - NextGenerationEU within the framework of the Digital Spain 2026 Agenda.

We are grateful to COMPUTAEX Foundation for allowing us to use the supercomputing facilities (LUSITANIA II) for calculations.

Repository. The code used for this paper can be found in the following repository: https://github.com/JSRivero/oracle-multiples

Appendix A

```
Data: Number of powers $n$ and a natural number $k$
Result: List of remainders $r_i$ of $2^i$ when divided by $k$
        $r_i \equiv 2^i \bmod k$ for $i \in [0, n-1]$
list_remainders ← $list(n)$;
$r \leftarrow 1$ ;                          /* as $2^0 \equiv 1$ mod $k$ for any $k \in \mathbb{N}$ */
for $i = 1$ to $n - 1$ do
    $r' \leftarrow 2 \cdot r$;
    if $r' < k$ then
        $r \leftarrow r'$
    else
        $r \leftarrow r' - k$
    end
    list_remainders[$i$] ← $r$
end
```

Algorithm 2: Algorithm for computing the remainders of the first n powers of 2 when divided by k

It can be noticed that this algorithm performs at most 3 operations each iteration, and has n iterations, hence its complexity is $\mathcal{O}(n)$.

References

1. Beauregard, S.: Circuit for shor's algorithm using 2n+3 qubits (2002). https://doi.org/10.48550/ARXIV.QUANT-PH/0205095, https://arxiv.org/abs/quant-ph/0205095
2. Bichsel, B., Baader, M., Gehr, T., Vechev, M.: SILQ: a high-level quantum language with safe uncomputation and intuitive semantics. In: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 286–300. PLDI 2020, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3385412.3386007
3. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemp. Math. **305**, 53–74 (2002)
4. Chivers, I., Sleightholme, J., Chivers, I., Sleightholme, J.: An introduction to algorithms and the big o notation. Introduction to Programming with Fortran: With Coverage of Fortran 90, 95, 2003, 2008 and 77, pp. 359–364 (2015)
5. Classiq: Classiq arithmetic oracle. https://docs.classiq.io/0-13/user-guide/builtin-functions/arithmetic/arithmetic-expression.html
6. Coppersmith, D.: An approximate Fourier transform useful in quantum factoring. arXiv preprint quant-ph/0201067 (2002)
7. Draper, T.G.: Addition on a quantum computer (2000). https://doi.org/10.48550/ARXIV.QUANT-PH/0008033, https://arxiv.org/abs/quant-ph/0008033
8. Green, A.S., Lumsdaine, P.L., Ross, N.J., Selinger, P., Valiron, B.: Quipper: A scalable quantum programming language. SIGPLAN Not. **48**(6), 333–342 (2013). https://doi.org/10.1145/2499370.2462177

9. Grover, L.K.: A fast quantum mechanical algorithm for database search (1996). https://doi.org/10.48550/ARXIV.QUANT-PH/9605043, https://arxiv.org/abs/quant-ph/9605043
10. Grover, L.K.: Quantum computers can search rapidly by using almost any transformation. Phys. Rev. Lett. **80**(19), 4329–4332 (1998). https://doi.org/10.1103/physrevlett.80.4329
11. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. **103**, 150502 (2009). https://doi.org/10.1103/PhysRevLett.103.150502, https://link.aps.org/doi/10.1103/PhysRevLett.103.150502
12. Hidary, J.D., Hidary, J.D.: A brief history of quantum computing. Quant. Comput. Appl. Approach. 15–21 (2021)
13. Kay, A.: Tutorial on the quantikz package. arXiv preprint arXiv:1809.03842 (2018)
14. Klappenecker, A., Roetteler, M.: Quantum software reusability. Int. J. Found. Comput. Sci. **14**(05), 777–796 (2003)
15. Leymann, F.: Towards a pattern language for quantum algorithms. In: Feld, S., Linnhoff-Popien, C. (eds.) QTOP 2019. LNCS, vol. 11413, pp. 218–230. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-14082-3_19
16. Montanaro, A.: Quantum algorithms: an overview. npj Quant. Inf. **2**(1), 1–8 (2016)
17. National Academies of Sciences, Engineering, and Medicine and others: Quantum computing: progress and prospects (2019)
18. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information. Phys. Today. **54**, 60 (2002)
19. Preskill, J.: Quantum computing in the NISQ era and beyond. Quantum **2**, 79 (2018)
20. Sanchez-Rivero, J., Talaván, D., Garcia-Alonso, J., Ruiz-Cortés, A., Murillo, J.M.: Automatic generation of an efficient less-than oracle for quantum amplitude amplification (2023). https://doi.org/10.48550/ARXIV.2303.07120, https://arxiv.org/abs/2303.07120
21. Sanchez-Rivero, J., Talaván, D., Garcia-Alonso, J., Ruiz-Cortés, A., Murillo, J.M.: Some initial guidelines for building reusable quantum oracles (2023). https://doi.org/10.48550/arXiv.2303.14959
22. da Silva, A.J., Park, D.K.: Linear-depth quantum circuits for multiqubit controlled gates. Phys. Rev. A. **106**, 042602 (2022). https://doi.org/10.1103/PhysRevA.106.042602, https://link.aps.org/doi/10.1103/PhysRevA.106.042602
23. Qiskit, A., et al.: An open-source framework for quantum computing (2021). https://doi.org/10.5281/zenodo.2573505
24. Zhao, J.: Quantum software engineering: Landscapes and horizons (2021). https://doi.org/10.48550/ARXIV.2007.07047, https://arxiv.org/abs/2007.07047