Quantum Information Processing (2020) 19:170
https://doi.org/10.1007/s11128-020-02669-7

Quantum fast Poisson solver: the algorithm and complete and modular circuit design

Shengbin Wang[1] · Zhimin Wang[1] · Wendong Li[1] · Lixin Fan[1] · Zhiqiang Wei[2] · Yongjian Gu[1]

Received: 29 November 2019 / Accepted: 2 April 2020 / Published online: 15 April 2020

Abstract
The Poisson equation has applications across many areas of physics and engineering, such as the dynamic process simulation of ocean current. Here we present a quantum algorithm for solving Poisson equation, as well as a complete and modular circuit design. The algorithm takes the HHL algorithm as the framework (where HHL is for solving linear equations). A more efficient way of implementing the controlled rotation, one of the crucial steps in HHL, is developed based on the arc cotangent function. The key point is that the inverse trigonometric function can be evaluated in a very simple recursive way by a binary expansion method. Quantum algorithms for solving square root and reciprocal functions are proposed based on the classical non-restoring method. These advances not only reduce the algorithm's complexity, but more importantly make the circuit more complete and practical. We demonstrate our circuits on a quantum virtual computing system installed on the Sunway TaihuLight supercomputer. This is an important step toward practical applications of the present circuits as a fast Poisson solver in the near-term hybrid classical/quantum devices.

Keywords Quantum circuit · Poisson equation · HHL algorithm · Trigonometric function · Function-value binary expansion

Shengbin Wang and Zhimin Wang have contributed equally to this work.

Electronic supplementary material The online version of this article (https://doi.org/10.1007/s11128-020-02669-7) contains supplementary material, which is available to authorized users.

✉ Zhimin Wang
wangzhimin@ouc.edu.cn

✉ Yongjian Gu
guyj@ouc.edu.cn

[1] Department of Physics, College of Information Science and Engineering, Ocean University of China, Qingdao 266100, China

[2] Department of Computer Science and Technology, College of Information Science and Engineering, Ocean University of China, Qingdao 266100, China

1 Introduction

Quantum computing exploits quantum mechanical features, especially the quantum superposition and entanglement, to solve efficiently the problems intractable for classical computing [1, 2]. Applications of quantum computing are based on quantum algorithms which include Shor's factoring algorithm [3], Grover's searching algorithm [4], the HHL algorithm [5], etc. HHL algorithm has been at the center of quantum machine learning research [6] and has inspired the development of quantum algorithms for solving differential equations [7–11].

The Poisson equation is a widely used linear differential equation which is usually expressed as $\Delta\varphi = f$. It plays a key role across many areas of physics and engineering. For instance, when simulating the dynamic process of ocean current, we could, in theory, use the Navier–Stokes equations to calculate the velocity field of the current. The *N–S* equations are a set of coupled partial differential equations of fluid velocity, density, pressure and force, and these equations are notoriously difficult to solve with classical computation methods [12, 13]. Using the well-studied vortex-in-cell method, the velocity field can be computed easily from a vector potential instead which satisfies the Poisson equation [14, 15]. Therefore, solving the Poisson equation constitutes the most computationally intensive part of ocean current simulation. Several analytical approaches have been developed for solving the Poisson equation faster [16–18]. Any direct or iterative classical algorithms for solving the d-dimensional Poisson equation with error ε have a cost of at least $\varepsilon^{-\alpha d}$, where $\alpha > 0$ is a smoothness constant [19]. Therefore, the cost of the problem grows exponentially in dimensions in classical computing, and it suffers from the curse of dimensionality.

In the field of quantum computing, several algorithms have been proposed for solving the Poisson equation which achieve an exponential speedup against classical methods [9–11, 20, 21]. From a practicality point of view, the work of Cao et al. among them is more intriguing, because they tried to show the circuit model of the algorithm [11]. In general, Cao's method is to discretize the Poisson equation into linear equations and use the HHL algorithm to solve them. The bottleneck of Cao's algorithm is the way of implementing the controlled rotation of HHL, or in other words doing the linear map from state $|\lambda_j\rangle$ to $1/\lambda_j|\lambda_j\rangle$ [5, 11]. The key point is a rather mundane task of evaluating the reciprocal and inverse trigonometric functions. More specifically, after having the eigenvalue state $|\lambda_j\rangle$ by phase estimation, firstly its reciprocal state $|1/\lambda_j\rangle$ is obtained through the Newton iteration method. The procedure is to solve the equation $f(x) = 1/x - \lambda_j$ using the iterative formula $x_{i+1} = -\lambda_j x_i^2 + 2x_i$ with an appropriate initial approximation value x_0. And then the binary string of $|1/\lambda_j\rangle$ is converted to the probability amplitude $1/\lambda_j$ through the controlled R_y rotations with the angle of $\theta = \arcsin(1/\lambda_j)$. They evaluated the arc sine function by a cut-and-try method. In each trial, they selected the trial angles by the method of bisection, then calculate the sine function and compare it with $1/\lambda_j$. Suppose the cost of evaluating the sine function is $O(m^3)$ where m is the number of qubits of input register, then the cost of evaluating the arc sine function is $O(m^4)$. In fact, the bottleneck part constitutes the most complex part of Cao's algorithm. However, there is nearly no circuit design for this part in Cao's work, because their algorithm for solving the sine and arc sine functions is not friendly for quantum circuit design.

In the present work, we try to advance the quantum algorithm for solving Poisson equation and present a complete and implementable quantum circuit. We intend to establish a quantum fast Poisson solver that is implementable on a near-term middle-scale quantum device. To this end, firstly we propose a new way of implementing the controlled rotation of HHL, which take the state $|\lambda_j\rangle$ to $1/\lambda_j|\lambda_j\rangle$ directly without through $|1/\lambda_j\rangle$ state. The key point is to evaluate the arc cotangent function using a novel method named qFBE (quantum function-value binary expansion) [22]. We can reduce the cost of the problem from $O(m^4)$ to $O(m^3)$ which is one order lower than that of Cao's algorithm. Secondly, we develop the inverse qFBE method to compute the cosine function to simplify the Hamiltonian simulation process of HHL. This method makes the circuit design easier and more modular. Thirdly, we develop quantum algorithms for solving the reciprocal and square root operations based on the classical non-restoring method. The algorithm for reciprocal and square root operations is the fundamental components of quantum arithmetic for trigonometric and inverse trigonometric functions. The non-restoring method is a digit-recurrence arithmetic method in classical computer science [23, 24]. It is derived from the binary arithmetic rules [25], which is a more efficient way for implementing reciprocal and square root operations than the Newton iteration method in quantum computing. The above three aspects of work not only reduce the algorithm's complexity, but also make the circuit complete and implementable. Finally, we demonstrate our circuits on a quantum virtual computing system installed on the Sunway TaihuLight supercomputer.

The paper is organized as follows. In Sect. 2, the overview of the problem is described. In Sect. 3, we describe the quantum circuit in detail, including the implementations of each module, algorithm complexity and error analysis. Section 4 shows the demonstration results of the present circuits on the quantum virtual system. Finally, conclusions and outlook of the present work are discussed in Sect. 5.

2 Overview of the problem

The problem is to solve the Poisson equation over a unit rectangular or cube domain with Dirichlet boundary conditions. Let us first focus on the one-dimensional Poisson equation which could be described by the following equations,

$$\begin{aligned} -\frac{\mathrm{d}^2 v(x)}{\mathrm{d}x^2} &= b(x), x \in (0,1), \\ v(0) &= v(1) = 0, \end{aligned} \tag{1}$$

where $b(x)$ is a given smooth function representing the charge or velocity distribution in different questions and $v(x)$ the solution to compute. Using the central-difference approximation to discretize the second derivative, Eq. (1) could be written in finite difference form as

$$h^{-2}(-v_{i-1} + 2v_i - v_{i+1}) = b_i, i = 1, 2 \ldots, N-1,$$

$$v_0 = v_N = 0, \tag{2}$$

where the number of discrete points is $N+1$ and the mesh size h equals to $1\big/N$. Then we have $N-1$ linear equations which can be expressed as follows:

$$A \cdot \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix} = h^{-2} \begin{pmatrix} 2 & -1 & & 0 \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{pmatrix} \cdot \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_{N-1} \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_{N-1} \end{pmatrix}. \tag{3}$$

Now the problem of solving Poisson equation transfers into solving the linear system of equations, i.e., $A\vec{v} = \vec{b}$, which can be done by the HHL algorithm.

One of the crucial step in HHL is to simulate the unitary operator e^{iAt}, which could be done efficiently utilizing the properties of the discretized matrix [11]. Matrix A is a well-studied Hermitian matrix, of which the eigenvectors are $u_j(k) = \sqrt{2\big/N}\sin(j\pi k\big/N)$ and the eigenvalues are $\lambda_j = 4N^2\sin^2(j\pi\big/2N)$ [26]. The eigenvalue decomposition of matrix A can be written as $A = S\Lambda S^T$, where Λ is a diagonal matrix with elements of the eigenvalues, i.e., $\Lambda = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_{N-1})$ and S is an orthogonal matrix composed of the eigenvectors, i.e., $S = [u_1, u_2, \ldots, u_{N-1}]$. Matrix S is actually the sine transform with the elements of $S_{j,k} = \sqrt{2\big/N}\sin(\pi jk\big/N)$. Obviously, S is Hermitian and $S^2 = 1$. Utilizing these properties of matrix A, the exponential A can be decomposed as follows:

$$\mathrm{e}^{iAt} = S \cdot \mathrm{e}^{iAt} \cdot S. \tag{4}$$

This equation provides us an efficient way to simulate the unitary operator e^{iAt} as will be discussed in the subsequent section.

The above results for the one-dimensional Poisson equation could be extended immediately to the multidimensional cases. As shown in Ref. [11, 26], the discretized matrix A_d for d-dimensional Poisson equation can be expressed using the Kronecker products as follows:

$$A_d = \underbrace{A \otimes I \otimes \cdots \otimes I}_{d} + \underbrace{I \otimes A \otimes I \otimes \cdots \otimes I}_{d} + \cdots + \underbrace{I \otimes \cdots \otimes I \otimes A}_{d}. \tag{5}$$

Then the exponential A_d can be written as

$$\mathrm{e}^{iA_d t} = \underbrace{\mathrm{e}^{iAt} \otimes \mathrm{e}^{iAt} \otimes \cdots \otimes \mathrm{e}^{iAt}}_{d}. \tag{6}$$

Therefore, the quantum circuit simulating $\mathrm{e}^{iA_d t}$ is just the parallel repetitions of the circuit simulating e^{iAt} with the number of d. In the following sections, we will mainly focus on the one-dimensional Poisson equation.

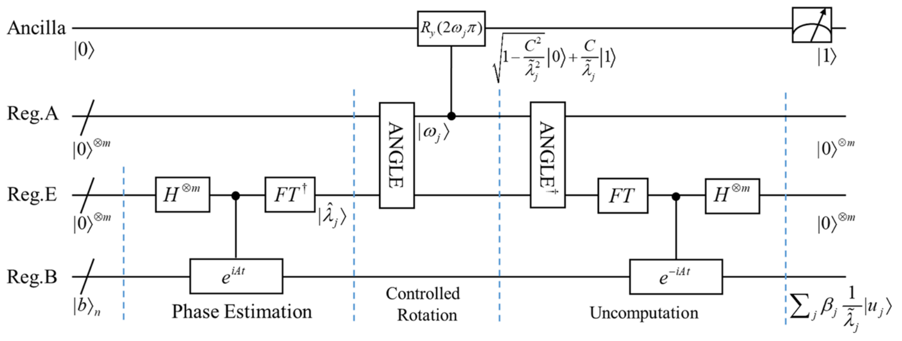


Fig. 1 Overall circuit of the present algorithm for solving one-dimensional Poisson equation. The number of qubits of registers B, E and A is n, m and m, respectively. Note the difference among λ_j, $\hat{\lambda}_j$ and $\tilde{\lambda}_j$. λ_j represents the eigenvalue of matrix A. $\hat{\lambda}_j$ denotes the approximation of λ_j with precision of m qubits. ω_j is the angular coefficient evolved from the $\hat{\lambda}_j$. $\tilde{\lambda}_j$ denotes the approximation of $\hat{\lambda}_j$ after the controlled rotation operation

3 Details of the quantum circuit

The present algorithm for solving one-dimensional Poisson equation consists of three main stages as shown in Fig. 1, which are the phase estimation, the controlled rotation and the uncomputation.

The circuit has three main registers, i.e., registers B, E and A. Register B is used to encode the coefficients of the right-hand side of Eq. (3). Its number of qubits is $n = \lceil \log(N) \rceil$, where N is the number of discrete points in Eq. (2). Register E is used to store the approximated eigenvalues. Its number of qubits is $m = 2n + 2 + f$, where the most significant $2n+2$ bits hold the integer part and the remaining f bits the fractional part. Register A is used to store calculated angular coefficients for the controlled rotation operation. Its number of qubits is chosen to be equal to that of register E.

Several important caveats to the present algorithm, essentially to the HHL algorithm, should be pointed out here [27]. First, we assume that the input state $|b\rangle$ of register B is already available to us. It is prepared as $\sum_i b_i |i\rangle$, where b_i is the value in Eq. (3) and $|i\rangle$ is the computational basis. Several techniques could be used to prepare the state [28–31]. Second, the output of the algorithm is a quantum state which encodes the solutions of Poisson equation as the probability amplitudes. Put another way, the implementation of the circuit could be considered as a process of quantum state preparation. The output state can be written as

$$|v\rangle = A^{-1}|b\rangle = \sum_i \alpha_i |i\rangle, \tag{7}$$

where α_i is the value of the solutions of Poisson equation after normalization.

3.1 Phase estimation

Phase estimation is used to approximate the eigenvalues of the discretized matrix A and entangle the states encoding the eigenvalues with the corresponding eigenstates [32].

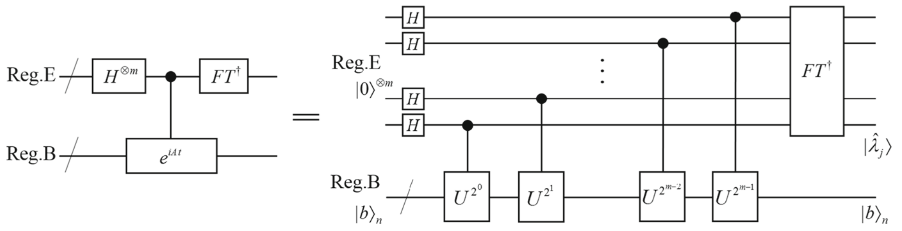


Fig. 2 Circuit for phase estimation. The U^{2^l} represents the unitary operator of $\exp(i2\pi A / 2^{m-l})$. $FT^{\dagger}$ represents the inverse quantum Fourier transform

The overall circuit for phase estimation is shown in Fig. 2. Hamiltonian simulation of e^{iAt} is the crucial part in phase estimation.

Before proceeding to the details of the circuit design, let us first sketch how the quantum states evolve through the circuit. The initial state of registers E and B is

$$|0\rangle^{\otimes m}|b\rangle = \sum_{i=1}^{2^n-1} b_i|0\rangle^{\otimes m}|i\rangle = \sum_{j=1}^{2^n-1} \beta_j|0\rangle^{\otimes m}|u_j\rangle, \tag{8}$$

where $|i\rangle$ is the computational basis and $|u_j\rangle$ is the jth eigenvector of matrix A. Then the Hadamard gates at the start of register E produce a uniformly superposition state. The following sequence of controlled U^{2^l} operation evolves the state as follows:

$$\sum_{k'=0}^{2^m-1}(|k'\rangle\langle k'|\otimes U^{k'}) \cdot \frac{1}{\sqrt{2^m}}\sum_{k=0}^{2^m-1}|k\rangle\otimes\sum_{j=1}^{2^n-1}\beta_j|u_j\rangle = \sum_{j=1}^{2^n-1}\beta_j\left[\frac{1}{\sqrt{2^m}}\sum_{k=0}^{2^m-1}\exp\left(2\pi i\frac{\hat{\lambda}_j}{2^m}k\right)|k\rangle\right]|u_j\rangle. \tag{9}$$

The state in the square bracket of Eq. (9) is just the output of the quantum Fourier transform acting on the state $|\hat{\lambda}_j\rangle$ [2, 33]. So after application of the inverse Fourier transform, the states evolve into

$$\sum_{j=1}^{2^n-1}\beta_j|\hat{\lambda}_j\rangle|u_j\rangle. \tag{10}$$

The states encoding the approximated eigenvalues, namely $|\hat{\lambda}_j\rangle$, are prepared and entangled with the eigenstates $|u_j\rangle$.

We now show the details of the circuit for the time evolution of e^{iAt}. There are general methods to simulate e^{iAt} [34, 35], but here we utilize the specific properties of matrix A to deal with it to reduce the complexity. According to Eq. (4), e^{iAt} can be diagonalized via the sine transform. And the sine transform S can be performed based on the T_N and Fourier transform through the way of $T_N^{\dagger}FT_{2N}T_N = C_{N+1}\oplus(-iS_{N-1})$,

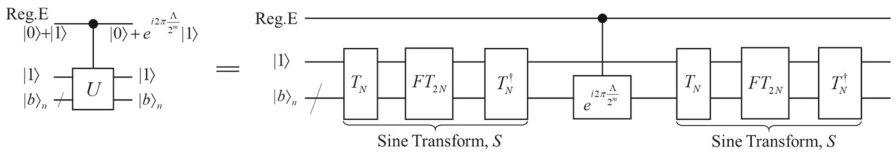


Fig. 3 Circuit for the controlled U operation. One ancillary qubit prepared in state $|1\rangle$ is used to obtain sine transform from the transformation of $T_N^{\dagger} F T_{2N} T_N$

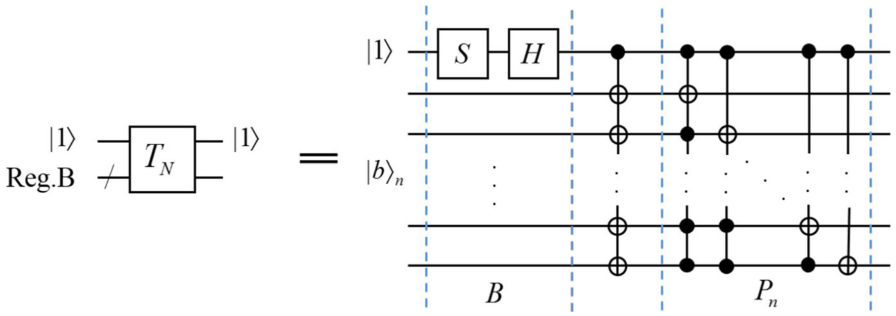


Fig. 4 Circuit for the T_N transform in $T_N^{\dagger} F T_{2N} T_N$. The present circuit is a simplified one of that in Ref. [37]. The controlled Hadamard and phase gates are ignored because $|b_n\rangle$ has no $|0\rangle^{\otimes n}$ state. The multiple controlled NOT gates could be decomposed into TOFFOLI gates using the method in Ref. [38]

where C_{N+1} represents the cosine transform and the subscript denotes the dimensions of the matrix. More details about the sine transform and its quantum implementation can be found in Refs. [36, 37]. Figures 3 and 4 show the circuits of the controlled U^{2^0} operation and the T_N transform, respectively.

For the operator $\exp(i2\pi \Lambda / 2^m)$ in Fig. 3, its eigenvectors are the computational basis, and its eigenvalues are the exponential of matrix A's eigenvalue, namely λ_j. The action of the operator $\exp(i2\pi \Lambda / 2^m)$ on a state, the $|b\rangle$ state after sine transform, could be expressed as follows:

$$\exp\left(i\Lambda \frac{2\pi}{2^m}\right) \cdot \sum_{j=1}^{2^n-1} b'_j |j\rangle = \sum_{j=1}^{2^n-1} b'_j \exp\left(i\hat{\lambda}_j \frac{2\pi}{2^m}\right)|j\rangle. \tag{11}$$

This action can be implemented using the trick of phase kickback [1]. The general idea of phase kickback is that a computational basis adds a constant integer $\lambda \bmod 2^m$ will kickback a phase change of $\exp(i2\pi\lambda / 2^m)$. Figure 5 shows an example circuit for the controlled $\exp(i2\pi \Lambda / 2^m)$ operator with $m = 3$. Phase kickback is accomplished by the (inverse) Fourier transformation and the controlled modular-addition operation. A ripple-carry adder (see Ref. [39]) is used to do the modular addition operation by omitting the highest carry bit. The ripple-carry adder consists of two basic units, which are SUM and CARRY units. The adder could become a controlled one by changing the CNOT gates of SUM unit into TOFFOLI gates and leaving CARRY unit unchanged.

The module of *EVC* in Fig. 5 is used to approximate the eigenvalues of matrix A and obtain the state $|\hat{\lambda}_j\rangle$. It is already known that the eigenvalues are $\lambda_j =$

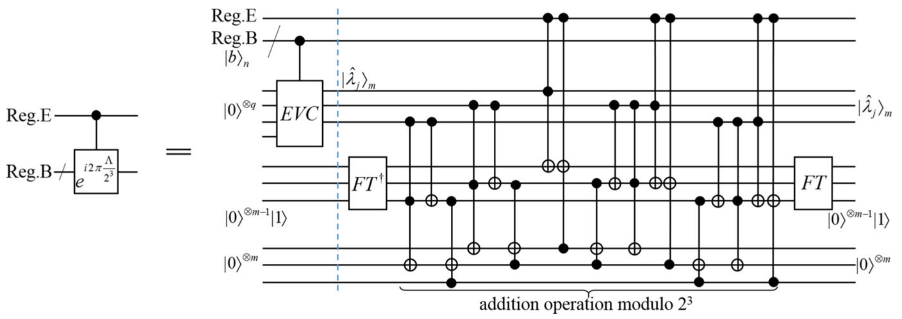

Reg.E
Reg.B
$e^{i2\pi\frac{A}{2^3}}$
=
Reg.E
Reg.B
$|b\rangle_n$
$|\hat{\lambda}_j\rangle_m$
$|0\rangle^{\otimes q}$
EVC
$FT^\dagger$
$|0\rangle^{\otimes m-1}|1\rangle$
$|0\rangle^{\otimes m}$
$|\hat{\lambda}_j\rangle_m$
FT
$|0\rangle^{\otimes m-1}|1\rangle$
$|0\rangle^{\otimes m}$
addition operation modulo 2^3

Fig. 5 Circuit for the controlled $\exp(i2\pi A/2^m)$ transformation with $m = 3$. The *EVC* module, which is short for eigenvalue calculation, is used to approximate the eigenvalues. Note that the initial state of *EVC* module is $|0\rangle^{\otimes q}$, and the output state is truncated to m qubits to store the approximated eigenvalues. The total number of qubits and operations is $(n + 4)q + m$ and $m^3/3 + 11m^2/2$, respectively

$4N^2 \sin^2(j\pi/2N)$, i.e., $\lambda_j = 2N^2(1-\cos j\pi/N)$, so the state $|\hat{\lambda}_j\rangle$ could be prepared by computing the cosine function.

The cosine function can be calculated based on the Taylor expansion and repeated squaring method [11], but this method requires a large number of reversible multiplication and square operations, which require a large number of auxiliary qubits. Here we propose a more simple and programmable method to evaluate the cosine function. This method is actually the inverse version of the qFBE method which is discussed in the next Sect. 3.2. For the present specific case, the cosine function $\cos(j\pi/N)$ is computed as follows:

Firstly, binary expansion of the coefficient $j/N = (0.v_{n-1}\cdots v_2v_1v_0)_2$, $v_i \in \{0, 1\}$ has already been prepared in register B before the controlled *EVC* module as shown in Fig. 5. Then the absolute value of cosine function, i.e., $|\cos(j\pi/N)|$, can be approximated step by step using the following iterative formula,

$$a_0 = 1; \quad a_{i+1} = \begin{cases} \sqrt{(1+a_i)/2}, if\ v_i = v_{i-1} \\ \sqrt{(1-a_i)/2}, if\ v_i \neq v_{i-1} \end{cases}, 0 \le i \le n-1,\ v_{-1} = 0. \quad (12)$$

As can be seen from the equation, the solution of $a_n = |\cos(j\pi/N)|$ is obtained after n iterations conditioned on each digit of the binary string of the input.

Figure 6 shows the circuit of *EVC* module, which is designed to perform the calculation of $2N^2(1 - \cos(j\pi/N))$. The module is initialized with q qubits which are used to implement the intermediate iteration operations, and at the end the outputs are truncated to m qubits to obtain the eigenvalues. Appropriate q is selected to guarantee that the m-bits of output are totally exact (see "Error accumulation in the iteration process of EVC module" in Appendix 3 for details). The COS modules with the number of n are used to implement the iterative operations of Eq. (12) to obtain $|\cos(j\pi/N)|$. Calculations are performed in fixed precision arithmetic, so multiplications of 2 and N^2 can be performed easily by just keeping in track the position of the binary point.

The circuit for each COS module is almost the same except with different control qubits. Figure 7 shows the circuit of the ith COS module. The dominant cost of the

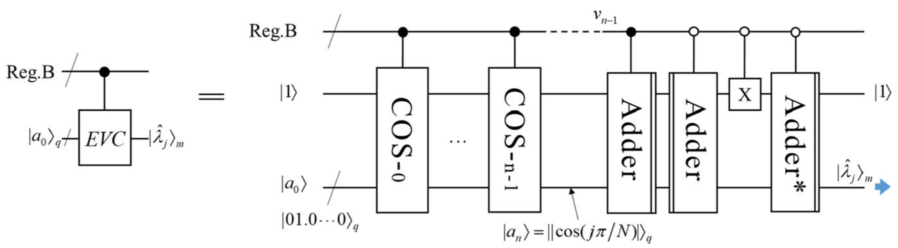


Fig. 6 Overall circuit for eigenvalues computation. Adder denotes a ripple-carry adder omitting the highest carry bit, which is the same as that in Fig. 5. The ancillary qubit $|1\rangle$ is for addition operation, where for Adder module the augend is designed to be $|1\rangle|0\rangle^{\otimes q}$ and for Adder* be $|0\rangle^{\otimes q}|1\rangle$. Note the position of a bar on the right- or left-hand side of Adder network. The bar on the right-hand side denotes that the module is a normal ripple-carry adder, while the left-hand side denotes that the Adder's elementary gates are in a reversed sequence and it is actually a subtracter. The total number of qubits and operations is $(n+4)q$ and $n(33q^2/2+32q)$, respectively

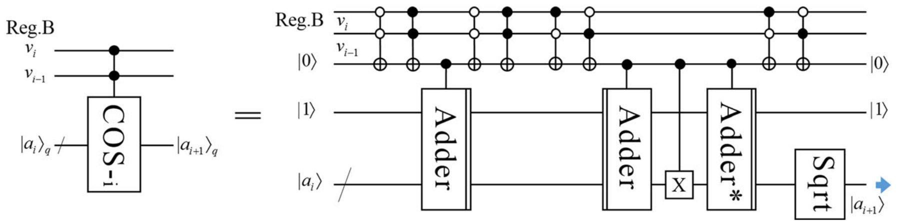


Fig. 7 Circuit of the ith COS module, which is designed to accomplish one iteration in Eq. (12). The Adder and Adder* modules are the same as those in Fig. 6. The total number of qubits and operations is $5q$ and $33q^2/2+32q$, respectively

COS module results from the square root operation, and the error during the calculation process is induced by the truncation of square root.

Bhaskar et al. developed the quantum algorithm for square root operation based on the Newton iteration method [40]. The procedure is to solve the equation $f(y) = 1/y^2 - 1/\omega$ using the iterative formula $y_{i+1} = (3y_i - y_i^3/\omega)/2$ with an appropriate initial approximation value y_0, where ω is the input and $1/\omega$ is calculated using the Newton iteration method as mentioned in the Introduction section. This method requires $O(\log(n))$ iteration steps, and in each iteration step there are square and cube multiplication operations. Here we develop a new kind of quantum algorithm for the square root operation based on the classical non-restoring method [24]. Like the qFBE method, the non-restoring method outputs the solution in binary form digit by digit, of which each bit is the exact one. There are only addition and subtraction operations in each recursion [23–25], so it can greatly reduce the usage of qubits and quantum gates. In "Non-restoring method for solving the Square root function" in Appendix 2, we describe the computing procedure of this method in detail and give a simple example illuminating the procedure further.

Figure 8 shows the circuit design for the square root operation based on the non-restoring method. The circuit consists of $2q$ recursive steps, where q is the number of qubits of the input state of EVC. As can be seen from the figure, each recursive step of Part I includes only one subtraction and one controlled addition operation, and Part

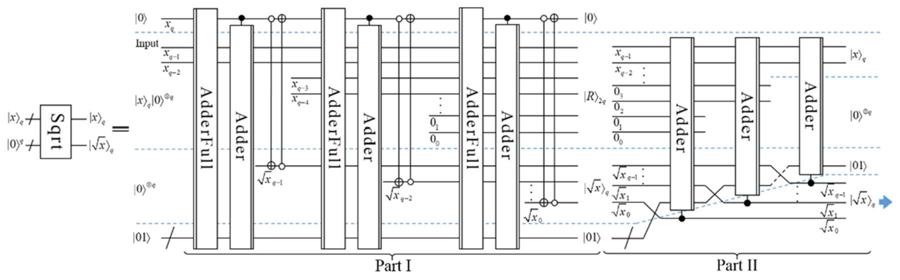


Fig. 8 Circuit for the square root operation. It consists of two parts, where part I is used to calculate the square root and part II to uncompute the remainder register $|R\rangle_{2q}$. AdderFull is an unabridged ripple-carry adder comparing with Adder, in other words it has the highest carry bit. The total number of qubits and operations is $5q$ and $33q^2/2 + 22q$, respectively

II only one controlled addition. So the complexity of the present method is $O(q)$ in qubits and $O(q^2)$ in operations, which are the same as that of multiplication operation.

To sum up, the dominant cost of the phase estimation result from the *EVC* module, in fact from computing the cosine function. The complexity of the present method for calculating cosine function is $O(q^2)$ in qubits and $O(q^3)$ in operations, which is the same with that in Ref. [11]. However, the present circuit design is complete and much more modular, and the error propagation is easier to control.

3.2 Controlled rotation

After phase estimation, the state of $\sum_{j=1}^{2^n-1} \beta_j|\hat{\lambda}_j\rangle|u_j\rangle$ is obtained in registers B and E. Then we perform the linear map taking the state of $|\hat{\lambda}_j\rangle$ to $1\big/\tilde{\lambda}_j|\hat{\lambda}_j\rangle$, which can be accomplished by the controlled rotation. It consists of two parts: calculating the rotation angular coefficients and performing the controlled R_y operation.

The probability amplitude of $1\big/\tilde{\lambda}_j$ would be produced by implementing the controlled R_y rotation, namely $R_y(2\theta_j)|0\rangle = \cos\theta_j|0\rangle + \sin\theta_j|1\rangle$. The relationship between rotation angle θ_j and $\hat{\lambda}_j$ can be expressed as

$$\sin\theta_j = 1\big/\hat{\lambda}_j. \tag{13.a}$$

This equation can be further written as,

$$\cot\theta_j = \sqrt{\hat{\lambda}_j^2 - 1}, \theta_j \in (0, \pi\big/2). \tag{13.b}$$

Now we omit the subtrahend 1 under the square root and take $\theta_j' = \omega_j\pi$; then, Eq. (13.b) turns to

$$\omega_j = \frac{\text{arccot}\hat{\lambda}_j}{\pi}, \omega_j \in (0, 1\big/2), \tag{14}$$

where ω_j hereafter is referred to as rotation angular coefficient. The error of $1\big/\tilde{\lambda}_j$ caused by omitting the subtrahend 1 is less than 2^{-10}, which will be discussed in the next section.

In order to solve Eq. (14) efficiently, we developed the qFBE (quantum function-value binary expansion) method which could evaluate the transcendental functions in a very simple recursive way. The classical counterpart of this method was first proposed by Borwein and Girgensohn, who were inspired by the Plouffe's iteration identity for the arctan [22]. Generally, this method defines the (S) system of the following two functional equations for an unknown function f: $I \to [0, 1]$,

$$\begin{aligned} 2f(x) &= f(r_0(x)) \quad if\ x \in D_0, \\ 2f(x) - 1 &= f(r_1(x)) \quad if\ x \in D_1, \end{aligned} \tag{15.a}$$

where $D_0 \cup D_1 = I$, $D_0 \cap D_1 = 0$ and the functions r_0: $D_0 \to I$, r_1: $D_1 \to I$. Then the function value $f(x)$ at point x_0 can be computed by the following recursion.

$$\begin{aligned} &Set \quad a_0 = x_0, \quad a_{n+1} = \begin{cases} r_0(a_n) & if\ a_n \in D_0, \\ r_1(a_n) & if\ a_n \in D_1, \end{cases} \\ &Then \quad f(x_0) = f(a_0) = \sum_{a_n \in D_1, n \ge 0} \frac{1}{2^{n+1}}. \end{aligned} \tag{15.b}$$

This method outputs the binary expansion of the function value digit by digit conditioned on each iteration step of a_n. Note that each digit of the output binary string is exact, so the error of solution only comes from the truncation error of the number of bits. We adapt this method and develop the corresponding quantum version. Another manuscript (with the title of *quantum circuit design for evaluating transcendental functions based on binary expansion method*) is being prepared to discuss this method in detail.

The arc cotangent function is a member of the (S) system, so Eq. (14) is solved by the qFBE method. Specifically, the binary solution of the angular coefficient $\omega_j = (0.w_0 w_1 \cdots w_{m-1})_2$ can be calculated digit by digit using the following recursive formula,

$$w_i = \begin{cases} 0, & if\ a_i > 0 \cup \{-\infty\} \\ 1, & if\ \ others \end{cases}; \quad a_0 = \hat{\lambda}_j, \quad a_{i+1} = \begin{cases} 1/2(a_i - 1/a_i), & if\ a_i \ne 0 \\ -\infty, & if\ a_i = 0 \end{cases}. \tag{16}$$

The overall circuit for the controlled rotation is shown in Fig. 9. The arccot module in the circuit is designed to implement the operations of Eq. (16). Each arccot module represents a single recursive step. After the m arccot modules, the $|\omega_j\rangle$ state is prepared in register A. Then the controlled R_y operations take $|0\rangle$ to $\sqrt{1 - \left(C\big/\tilde{\lambda}_j\right)^2}|0\rangle + C\big/\tilde{\lambda}_j|1\rangle$, where C is a normalizing constant.

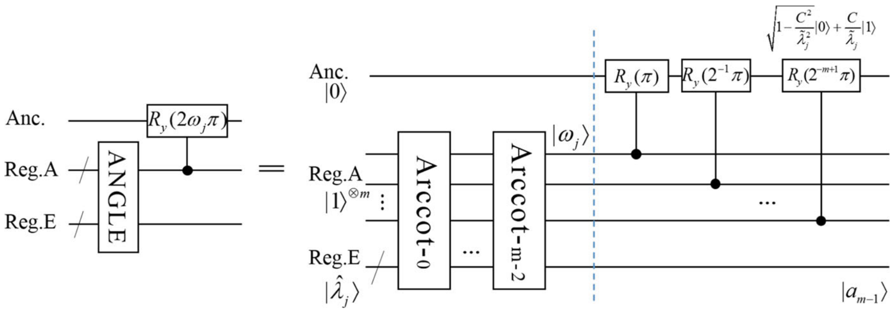


Fig. 9 Circuit for the controlled rotation. The ANGLE module is used to compute the rotation angular coefficient ω_j. The total number of qubits and operations is $m^2 + 3m$ and $34m^3 - 16m^2$, respectively

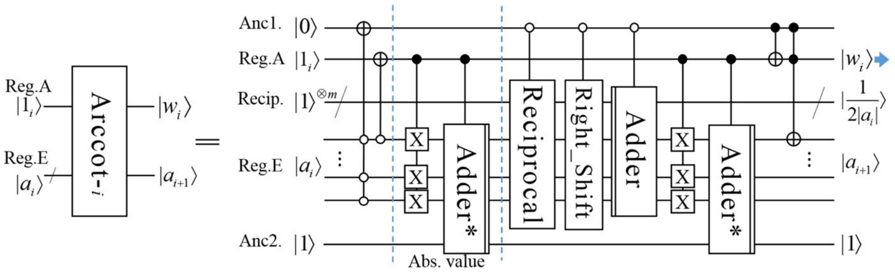


Fig. 10 Circuit of the ith arccot module used to compute each bit of $\omega_j = (0.w_0w_1 \cdots w_{m-1})_2$. The Adder and Adder* modules are the same as those in Fig. 6. The Right_Shift module is for the factor of 1/2 in Eq. (16). The total number of qubits and operations is $4m$ and $34m^2 + 18m$, respectively

Figure 10 shows the circuit of the ith arccot module, whose main cost results from the reciprocal operation. Bhaskar et al. also discussed the quantum algorithm for solving the reciprocal function based on Newton iteration method [40]. Like that in the square root, we develop a quantum digit-recurrence algorithm to calculate the reciprocal based on the non-restoring method [23]. The computing procedure of this method, as well as an illustrative example is described in "Non-restoring method for solving the reciprocal function" in Appendix 2.

Figure 11 shows the quantum circuit for the reciprocal operation based on the non-restoring method. It consists of $m-2$ Recip modules used to calculate the reciprocal value and $m-2$ Inv modules to uncompute the ancillary register. The circuits of the Recip and Inv modules are shown in Fig. 12. There are only one subtraction and one addition operation in the Recip and Inv modules. The cost of the present algorithm for computing reciprocal is $O(m)$ in qubits and $O(m^2)$ in operations, which are the same as that of multiplication operation.

To sum up, the complexity of the present algorithm for the controlled rotation operation, in fact for computing the arc cotangent function, is $O(m^2)$ in qubits and $O(m^3)$ in operations, which is the same with the *EVC* module. In Ref. [11], the complexity is $O(m^3)$ qubits and $O(m^4)$ operations. So our method reduces the complexity by one order. Furthermore, the present circuit design is complete and much more modular, and the error propagation is easier to control.

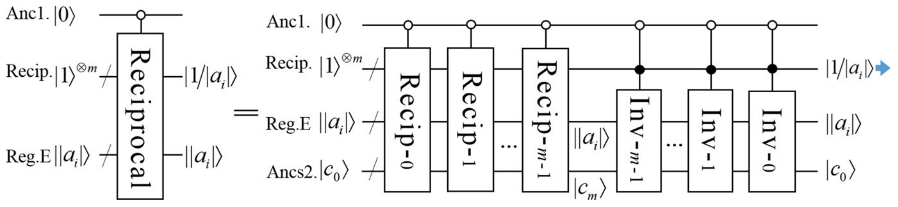


Fig. 11 Circuit for solving the reciprocal function. The initial state of Ancs2 is $|c_0\rangle = |0\rangle^{\otimes m+1}|1\rangle$. The total number of qubits and operations is $4m$ and $34m^2$, respectively

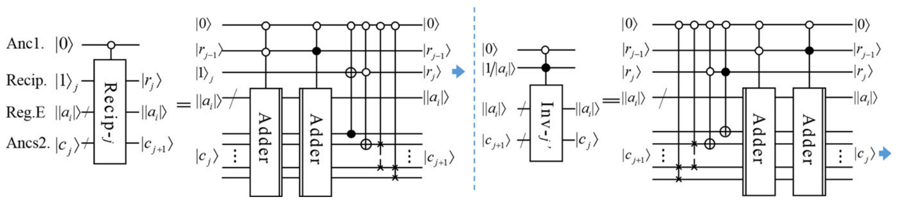


Fig. 12 Circuits of the Recip-*j* and Inv-*j'* modules. The Adder is the same as that in Fig. 6. The total number of qubits and operations is $3m$ and $34m$, respectively

After controlled rotation, the uncomputation is implemented to evolve the state of registers B, E and A back to the initial state. Finally, we perform the measurement operation. If the measurement result of the ancillary qubit is $|1\rangle$, then we know the final state is $\sum_j \beta_j \tilde{\lambda}_j^{-1}|u_j\rangle$ which is proportional to $A^{-1}|b\rangle$, namely the solution state for the Poisson equation.

3.3 Algorithm complexity and error analysis

The present algorithm for solving one-dimensional Poisson equation takes the HHL algorithm as the template. HHL uses roughly $O(\kappa^2 \log(N)/\varepsilon)$ steps to output the solution state with a success probability arbitrarily close to one, where ε is the error of the solution state and κ and N is, respectively, the discretized matrix's condition number and dimension [5]. Since we utilize the specific properties of the discretized matrix A to simulate the Hamiltonian e^{iAt}, the complexity of the present algorithm should be lower than that of HHL.

Table 1 shows the number of qubits and elementary gates used in each part of our circuit as shown in Fig. 1. In the table, only the first two significant terms of the numbers are shown for simplicity. The m and n are the number of qubits of registers E (A) and B, respectively, and satisfy $m = 2n + 2 + f$. Based on the upper bound, we assume that $q = 3m/2$ (see "Error accumulation in the iteration process of EVC module" in Appendix 3).

As can be seen from the table, the total cost of the circuit is $O(m^2)$ qubits and $O(m^3)$ operations. Note this is the cost for one computation. Using the trick of amplitude amplification [41], a number of repetitions proportional to κ could lead to a success probability arbitrarily close to one [5]. So the complexity of the present algorithm for solving one-dimensional Poisson equation is $O(m^2)$ qubits and $O(\kappa m^3)$ operations.

Table 1 The number of qubits and elementary gates used in each module of our algorithm

Modules	Qubits	Elementary gates[a]
Phase estimation		
S[b]	$n+2$	$97n^2 - 745n$
EVC	$3m(n+4)/2$	$75nm^2 + 96nm$
Phase Kickback	$3m$	$m^3/3 + 11m^2/2$
Controlled rotation		
Angle computing	$m^2 + 3m$	$34m^3 - 16m^2$
Controlled R_y	$m+1$	$4m$
Uncomputation	$m^2 + 3nm/2$	$34m^3 + 75nm^2$
Total	$m^2 + 3nm/2$	$68m^3 + 150nm^2$

[a]They include the H, X, S, R_y, *C-NOT*, *C-*R_z, *SWAP* and *TOFFOLI* gates
[b]It represents the sine transform accomplished through the $T_N^\dagger FT_{2N} T_N$ operation

Since the condition number κ of the discretized matrix A is *polynomial*(N) (see "Appendix 1" for details), the present algorithm for solving one-dimensional Poisson equation could not offer exponential speedup over classical algorithms. However, for d-dimensional Poisson equation the present algorithm achieves exponential speedup in terms of d. As mentioned in the Introduction section, the cost of any classical algorithm solving the d-dimensional Poisson equation grows exponentially with the dimensions. But here the complexity of our algorithm grows linearly with d. Specifically, the main difference of the circuits between d- and one- dimensional case is the Hamiltonian simulation in phase estimation. According to Eq. (6), the circuit simulating $e^{iA_d t}$ can be obtained by replication and parallel application of the circuit simulating e^{iAt} used in one-dimensional case. So the complexity of the present algorithm for solving d-dimensional Poisson equation is $O(dm^2)$ qubits and $O(\kappa dm^3)$ operations, which is linear with d. In terms of error ε, the complexity can be further expressed as follows:

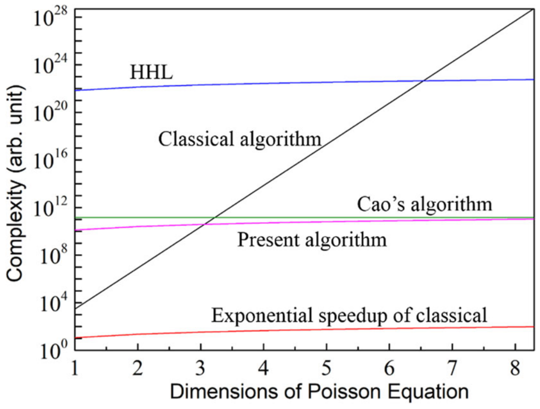
$$\begin{cases} O\left(d \log^2(\varepsilon^{-\alpha})\right) & qubits \\ O\left(\kappa d \log^3(\varepsilon^{-\alpha})\right) & operations \end{cases}. \tag{17}$$

To obtain this equation, the following two facts are used. First, $m = \Theta(n) = \Theta(\log(N))$. Second, the basic error is from the central-difference approximation of the second derivative (see Eq. (2)), and it satisfies $N = \varepsilon^{-\alpha}$, where $\alpha > 0$ is a constant depending on the smoothness of the solution. In terms of error ε, the complexity of Cao's algorithm is $\max\{d, \log(\varepsilon^{-\alpha})\} \cdot \log^2(\varepsilon^{-\alpha})$ in qubits and $\max\{d, \log(\varepsilon^{-\alpha})\} \cdot \log^3(\varepsilon^{-\alpha})$ in operations.

The variation trends of the complexities discussed above are summarized in Fig. 13. In the diagram, we assume that $\varepsilon = 2^{-23}$, namely the single precision, and use the fact that $\kappa = \varepsilon^{-2\alpha}$ (see "Appendix 1"). As can be seen from the figure, the complexity of present algorithm is lower than that of HHL, but does not achieve an exponential speedup of classical algorithm. It is intriguing that the speedup of present algorithm against the classical one would appear when the dimension of the Poisson equation is greater than three.

Here we would like to make a comparison between our algorithm and Cao's from the following two aspects. Firstly from the complexity point of view, the cost of

Fig. 13 Sketch of the complexity of different algorithms for solving the d-dimensional Poisson equation



implementing the controlled rotation of HHL in our algorithm is $O(m^3)$, which is one order lower than that in Cao's algorithm ($O(m^4)$). Therefore, our algorithm reduces the algorithm's complexity by one order in the cases of low dimensions of Poisson equation or high precisions of the solutions. Secondly from the algorithm's practicality point of view, the present work implies an important advance. Our circuit design for the algorithm is complete and could be scaled in a simple programmed way due to the novel method for arithmetic. The error propagation is easier to control along the circuit. The present circuits seem to be more intriguing for applications on near-term quantum devices because of its advantage in the case of low dimensions of Poisson equation.

Now we analyze the error of the final solution state $\sum_{j=1}^{2^n-1}\beta_j\tilde{\lambda}_j^{-1}|u_j\rangle$, namely the error of the probability amplitude $\tilde{\lambda}_j^{-1}$. We approximate the reciprocal of eigenvalues through two steps. First, we obtain $\hat{\lambda}_j$ by approximating λ_j in the phase estimation, and then we obtain $\tilde{\lambda}_j^{-1}$ from $\hat{\lambda}_j$ in the controlled rotation. So the error can be calculated by the following expression,

$$\left|\frac{1}{\tilde{\lambda}_j}-\frac{1}{\lambda_j}\right|=\left|\left(\frac{1}{\hat{\lambda}_j}-\frac{1}{\lambda_j}\right)+\left(\frac{1}{\tilde{\lambda}_j}-\frac{1}{\hat{\lambda}_j}\right)\right|. \tag{18}$$

For the first item, we have

$$\frac{1}{\hat{\lambda}_j}-\frac{1}{\lambda_j}=\frac{\lambda_j-\hat{\lambda}_j}{\hat{\lambda}_j\lambda_j}<\frac{2^{-f}}{(\hat{\lambda}_j)^2}\in[0,2^{-f-6}], \tag{19}$$

where f is the number of qubits holding the fractional part in register E. To obtain the above equation, we use the following two facts. First, the error of the approximated eigenvalues only comes from the final truncation of the results to m-bits. This is because we use q qubits to implement the intermediate iteration process of calculating eigenvalues (Fig. 6), which is adequate to guarantee that all the m-bits of output are exact (see "Error accumulation in the iteration process of EVC module" in Appendix

3). Secondly, the eigenvalue $\hat{\lambda}_j$ is no less than 8. In fact, the smallest eigenvalue is $\lambda_1(N) = 4N^2 \sin^2(\pi \big/ 2N)$ and it is a monotonic increasing function with N. So we have the inequality of $\lambda_1(N) \geq \lambda_1(2) = 8, \ N \geq 2$.

For the second item, we have

$$\frac{1}{\tilde{\lambda}_j} - \frac{1}{\hat{\lambda}_j} = \frac{1}{\sqrt{1+\hat{\lambda}_j^2}} - \frac{1}{\hat{\lambda}_j} \in (-2^{-10}, 0), \tag{20}$$

which also use the fact that $\hat{\lambda}_j$ is larger than 8. In addition, the error mainly comes from the approximation of Eq. (13) by Eq. (14). That is, the error accumulation in the recurrence process of calculating the arc cotangent function is ignored due to the fact that according to Eq. (16) the digit w_i only depends on the sign of a_i, not on the specific value of a_i. The error can be reduced further by amplifying the value of eigenvalues, i.e., shift the binary string of $\hat{\lambda}_j$ left. More details about the deduction of Eq. (20) and the method of reducing the error can be seen in "The error in controlled rotation" in Appendix 3.

4 Circuits demonstration on a quantum virtual system

We demonstrate our algorithm on a quantum virtual computing system installed on the Sunway TaihuLight supercomputer of the National Supercomputing Center in Wuxi, China. This virtual system was developed by the Origin Quantum Computing Technology Co., Ltd, in Hefei of China. It has three operating modes, which are full amplitude mode, partial amplitude mode and single amplitude mode, respectively. The full amplitude mode could output the probabilities of all the final states for a circuit with at most 43 qubits. The partial amplitude mode could output the amplitudes of partial final states for a 82-qubit circuit, while the single amplitude mode the amplitude of one final state for a 200-qubit circuit with at most 20 depths [42].

Because of the limitation of the quantum virtual machine and the computing resource, we demonstrate our algorithm in two steps. Firstly, we propose a simplified version of our circuit for the one-dimensional Poisson equation with discretized points $N = 4$ to demonstrate the general effectiveness of our algorithm. The circuit is shown in Fig. 14. Since the *EVC* (Fig. 6) and *ANGLE* modules (Fig. 9) contribute the most complicated parts of our algorithm, in the simplified circuit the two modules are replaced by the modules which would prepare directly eigenvalues and rotation angular coefficients into the registers E and A, respectively. Note the operation cost of such modules grows exponentially with the number of qubits, so they could not be used in real circuit.

Secondly, the circuits for the cosine, arc cotangent, square root and reciprocal functions (as shown in Figs. 6, 7, 8, 11, respectively) are demonstrated individually. The configuration parameters of implementing the circuits on the quantum virtual computing system are listed in Table 2. Source codes of all of the circuits are provided as the supplementary material of the present paper, which can be found in Ref. [43].

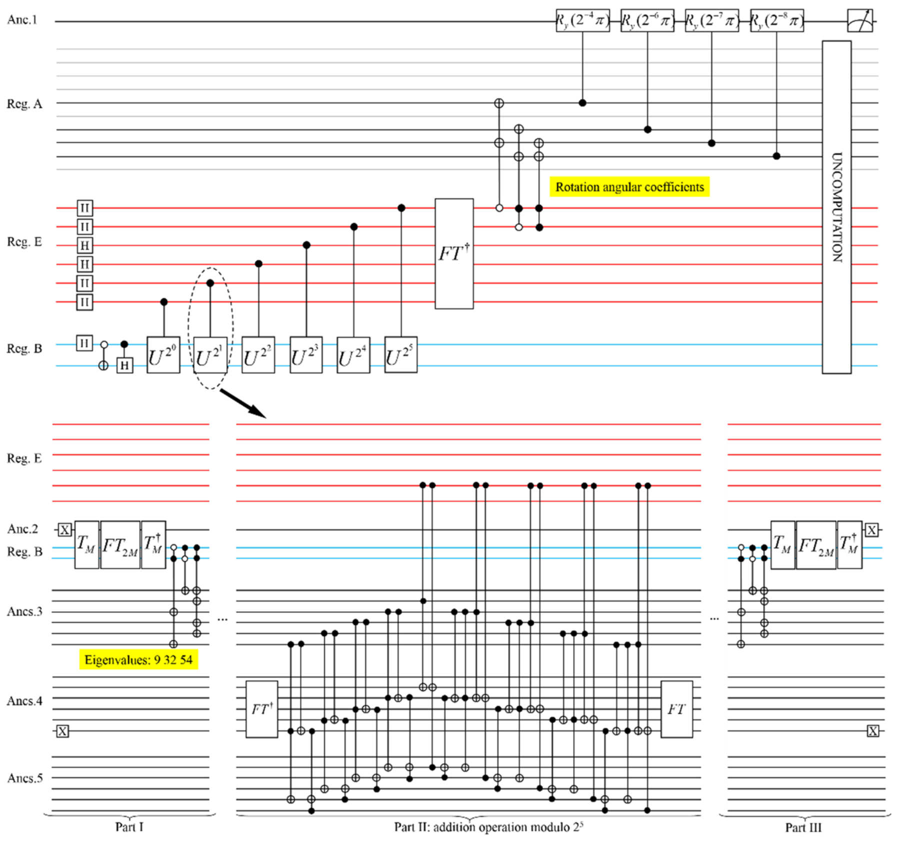


Fig. 14 Simplified circuit of our algorithm for solving Poisson equation. The upper subgraph shows the overall circuit, while the lower one shows the circuit of controlled U^{2^1} transformation. The gray lines in register A denote that such qubits could be omitted. The first three operations in register B are used to prepare the initial state of $|b\rangle = 1\big/\sqrt{2}|01\rangle + 1\big/2|10\rangle + 1\big/2|11\rangle$. The circuit for T_N transformation is shown in Fig. 4

The codes are written using the quantum assembly language of QRunes developed by the Origin Quantum Computing Company [44].

The results of our circuits, including the Poisson, cosine, arc cotangent, square root and reciprocal functions, are listed in Table 3. For the Poisson circuit, the input is an entanglement state encoding the coefficients of the right-hand side of the Poisson equation as the probability amplitudes, and the solutions are encoded into the probability amplitudes of the output states as indicated in Eq. (7). The real solution of the sample Poisson equation is (0.9053, 1.1036 0.8018), which after normalization turns to (0.553 0.674 0.490). So the error of the output solutions is less than 0.5%. The success probability of obtaining the desired state in one computation is 1.12%, which is proportional to κ^{-2}. If not use the amplitude amplification trick, $O(\kappa^2)$ repetitions of computation are needed to achieve a success probability arbitrarily close to 1.

Table 2 The configuration parameters of executing our circuits on the quantum virtual system

	Poisson[a]	Cosine	Arc cot	Square root	Reciprocal
Qubits	38	29	24	25	21
Gates	800	750	1000	310	420
Measure times	1	1	1	1	1
Least nodes[b]	1024	2	1	1	1
Operation mode	Full amp.	Full amp.	Full amp.	Full amp.	Full amp.
Run time (min)	20	15	4	2	1

[a]The simplified circuit of the present algorithm as shown in Fig. 14

[b]One node of the Sunway TaihuLight supercomputer contains 260 computing cores

Table 3 The results of our circuits implemented on the quantum virtual system

	Input states	Output states
Poisson	$\frac{1}{\sqrt{2}}\|01\rangle + \frac{1}{2}\|10\rangle + \frac{1}{2}\|11\rangle$	$0.551\|01\rangle + 0.675\|10\rangle + 0.491\|11\rangle$
Cosine	$\|00\rangle, \|01\rangle, \|10\rangle, \|11\rangle$	$\|01.000\rangle, \|00.101\rangle, \|00.000\rangle, \|11.011\rangle$
Arccot	$\|01.00\rangle$	$\|.01\rangle$
Square root	$\|0010\rangle, \|0111\rangle, \|1001\rangle, \|1111\rangle$	$\|01.01\rangle, \|10.10\rangle, \|11.00\rangle, \|11.11\rangle$
Reciprocal	$\|0010\rangle, \|0011\rangle, \|1000\rangle, \|1111\rangle$	$\|0.1000\rangle, \|0.0101\rangle, \|0.0010\rangle, \|0.0001\rangle$

The cosine circuit evaluates the function of $\cos(j\pi/N)$ with $N = 4$, and the j are encoded as the input states. The binary string of the output states represents the complement of the solution; that is, when the most significant bit of j is 1, invert each bit and plus one to obtain the final solution. The arccot, square root and reciprocal circuits evaluate the function of $\text{arccot}(x)/\pi$, $\sqrt{x}$ and $1/x$, respectively. As can be seen from the results, each bit of the binary string of the solution is exact and the error is determined totally by the number of the qubits. All the running results verify the effectiveness of our algorithms.

5 Conclusions

In the present work, we develop a quantum Fast Poisson Solver, including the algorithm and complete and modular circuit design. The algorithm is generally based on the HHL algorithm, and it achieves an exponential speedup in terms of dimension of Poisson equation. We perform the controlled rotation of HHL based on the arc cotangent function, so the rotation angles are prepared directly from the eigenvalues, instead of its reciprocals. This new way reduces the cost of the problem by one order, from $O(m^4)$ to $O(m^3)$, comparing with Cao's method, and makes the circuit design complete and more modular. A novel method of qFBE is proposed to evaluate the arc cotangent and cosine function in a simple recursive way. Based on the cosine function, we perform the eigenvalue approximation for the Hamiltonian simulation by a more modular way. In addition, quantum algorithms for solving square root and reciprocal functions are developed based on the non-restoring digit-recurrence method. All these developments make the present algorithm have lower complexity, i.e., one order lower comparing with Cao's algorithm in the cases of low dimensions of Poisson equation or high precisions of solutions, and more importantly from the practicality point of view make the circuit complete and implementable. The speedup of our algorithm against the classical method appears when the dimension of Poisson equation is greater than three. Furthermore, we remark that our qFBE method could be used to solve other kinds of trigonometric, inverse trigonometric, logarithmic functions or even the elliptic integrals of the first kind. This method should be useful in many other quantum problems, like Grover's state preparation algorithm [30], quantum analog–digital conversion algorithm [45], etc. [46].

The effectiveness of our circuits has been demonstrated on a quantum virtual computing system installed on the Sunway TaihuLight supercomputer. We think this is an important step towards the real applications of the quantum algorithm for solving Poisson equation as a fast Poisson solver in the near-term hybrid classical/quantum system. For the next step, we will first optimize the fundamental modules used in the circuit, e.g., adder, reciprocal and square root, to make the circuit model more efficient, and then study how to embed the quantum fast Poisson solver into the classical program for practical applications.

Acknowledgements We are very grateful to the National Supercomputing Center in Wuxi for the great computing resource. We would also like to thank the technical team from the Origin Quantum Computing Technology Co., Ltd, in Hefei for the professional services on quantum virtual computation. The present work is financially supported by the National Natural Science Foundation of China (Grant Nos. 61575180, 61701464, 11475160) and the Pilot National Laboratory for Marine Science and Technology (Qingdao).

Appendix 1

We take the maximum eigenvalue as the norm of the discretized matrix A. The norm of matrix A is $\lambda_{\max} = 4N^2 \sin^2 \frac{(N-1)\pi}{2N}$ as shown in Sect. 2 and that of matrix A^{-1} is $\lambda'_{\max} = (4N^2 \sin^2 \frac{\pi}{2N})^{-1}$. So the condition number of matrix A, or the ratio between A's largest and smallest eigenvalues [5], can be obtained as follows:

$$\kappa = \|A\| \cdot \left\|A^{-1}\right\| = \lambda_{\max} \cdot \lambda'_{\max} = \cot^2 \frac{1}{2N}\pi. \tag{A.1}$$

The relationship between κ and N is apparently nonlinear. Assuming $x = \pi \big/ 2N < 1$, the squared cotangent function satisfies the following equalities,

$$\begin{aligned}\cot^2 \frac{1}{2N}\pi = \cot^2 x &= \left(i\frac{e^{ix} + e^{-ix}}{e^{ix} - e^{-ix}}\right)^2 \\ &= \frac{1}{x^2} - 1 + o(x^3) < \frac{N^2}{2} - 1 + o\left(\left(\frac{\pi}{2N}\right)^3\right),\end{aligned} \tag{A.2}$$

where Taylor expansions of e^{ix} and e^{-ix} are used. The condition number of matrix A is $\kappa = O(N^2)$. In addition, we have the relationship between N and the basic error of the solutions caused by the central-difference approximation, namely $N = \varepsilon^{-\alpha}$. The α is a smoothness parameter depending on the smoothness of the solution function. (For example, when the solution function has uniformly bounded partial derivatives up to order four, α is 1/2 [26].) Therefore, the condition number of matrix A can be further expressed as $\kappa = O(N^2) = O(\varepsilon^{-2\alpha})$, which is independent of the dimension of matrix A. The additive preconditioner [47] would be used to reduce the κ.

Appendix 2

Non-restoring method for solving the square root function

The calculation procedure consists of the following five steps.

1st. Ignore the binary point of the binary number and expand the m-bits binary string to be $2m$-bits by adding 0 on the right-hand side.
2nd. Divide the $2m$-bits string into m parts in pairs from upper bit to lower.
3rd. Subtract 01 from the most left part. If the subtraction result is nonnegative, then the first bit of the solution is 1 and proceed to the next step. Otherwise, first bit of the solution is 0 and undo the subtraction operation.
4th. Expand the subtraction result of the last step by combining it with the next 2-bit part, and combine the solution obtained in the last step with 01. Then subtract the second number from the first number. If the subtraction result is nonnegative, the next bit of the solution is 1 and proceed to the next step. Otherwise, the next bit is 0 and undo the subtraction operation.
5th. Repeat step $4m-1$ times and the m-bits of the solution are obtained.

We take the square root of 01.00_2 as the example to illustrate the calculation procedure as shown in Fig. 15.

Non-restoring method for solving the reciprocal function

The calculation procedure consists of the following three steps.

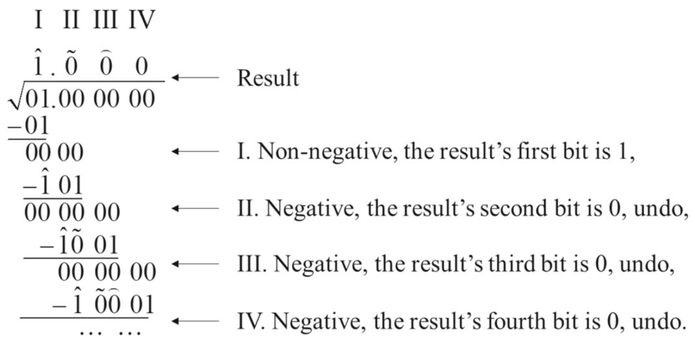
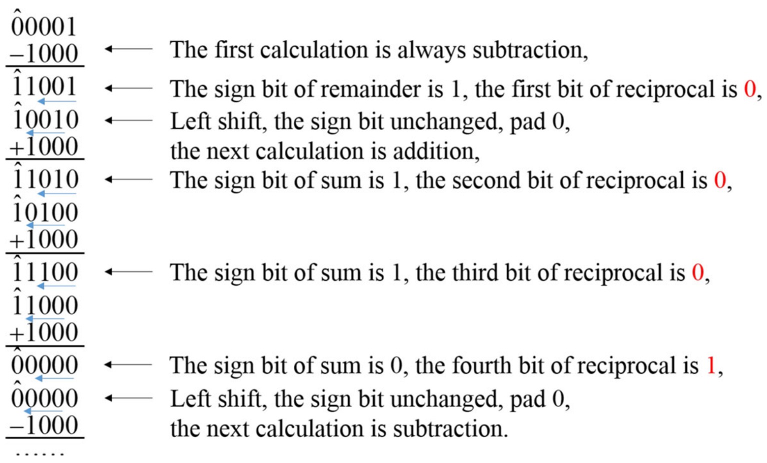

I II III IV
1.0 0 0
Result
√01.00 00 00
−01
00 00
−1 01
I. Non-negative, the result's first bit is 1,
00 00 00
−10 01
II. Negative, the result's second bit is 0, undo,
00 00 00
−1 00 01
III. Negative, the result's third bit is 0, undo,
... ...
IV. Negative, the result's fourth bit is 0, undo.

Fig. 15 Demo case of calculating square root of 01.00_2 using our revised non-restoring square root method



$\hat{0}0001$

-1000 ← The first calculation is always subtraction,

$\hat{1}1001$ ← The sign bit of remainder is 1, the first bit of reciprocal is 0,

$\hat{1}0010$ ← Left shift, the sign bit unchanged, pad 0,

$+1000$ the next calculation is addition,

$\hat{1}1010$ ← The sign bit of sum is 1, the second bit of reciprocal is 0,

$\hat{1}0100$

$+1000$

$\hat{1}1100$ ← The sign bit of sum is 1, the third bit of reciprocal is 0,

$\hat{1}1000$

$+1000$

$\hat{0}0000$ ← The sign bit of sum is 0, the fourth bit of reciprocal is 1,

$\hat{0}0000$ ← Left shift, the sign bit unchanged, pad 0,

-1000 the next calculation is subtraction.

……

Fig. 16 Demo case of calculating reciprocal of 1000_2 using the non-restoring method

1st. The highest bit of dividend 1 is the sign bit. Subtract the divisor from the dividend. If the sign bit of the result is 0, the first bit of reciprocal is 1, otherwise the first bit is 0. Then the result shifts left one bit except the sign bit with zero padding to the end to obtain the second dividend.

2nd. If the sign bit of the dividend of the last step is 0, subtract the divisor. If the sign bit is 1, add the divisor. If the sign bit of the result is 0, the next bit of reciprocal is 1, otherwise the next bit is 0. Then the result shifts left one bit except the sign bit with zero padding to the end to obtain the next dividend.

3rd. Repeat step $2m-1$ times and the m-bits of reciprocal of the divisor are obtained.

The reciprocal of 1000_2 is taken as the example as described in Fig. 16.

Appendix 3

Error accumulation in the iteration process of EVC module

As shown in Fig. 6, the *EVC* module is first initialized with q qubits to perform the intermediate iterative calculations and finally truncated to m qubits to store the approximated eigenvalues. There exists truncation error in each iteration which will be accumulated to the final result. So appropriate q should be selected to guarantee that all the m-bits of the output are exact.

According to the characteristics of the iterative formula as shown in Eq. (12), the maximum error accumulation occurs at the points of $|\cos(1/2 \pm 1/2^n)\pi|$. We take one of them, namely $j\big/N = (0.0\underbrace{11\cdots1}_{n-1})_2$, to evaluate the upper bound of the truncation error. Firstly, Eq. (12) can be expressed as

$$y(x) = \begin{cases} \sqrt{(1+x)/2}, & if \quad v_i = v_{i-1} \\ \sqrt{(1-x)/2}, & if \quad v_i \neq v_{i-1} \end{cases}, x \in [0, 1], y \in [0, 1]. \tag{C.1.1}$$

The derivative of the first iteration function is decreasing and belongs to the interval of $[1/4, \sqrt{2}/4]$, while the second is also decreasing and belongs to the interval of $[-\sqrt{2}/4, -\infty)$. The iteration process is as follows:

$$\begin{cases} a_1 = y(a_0) + 0 \\ \hat{a}_2 = y(a_1) + \varepsilon_2 \\ \hat{a}_3 = y(\hat{a}_2) + \varepsilon_3 \\ \cdots \\ \hat{a}_n = y(\hat{a}_{n-1}) + \varepsilon_n \end{cases}, \tag{C.1.2}$$

where ε_i represents the truncation error of each iteration step. The error of the first step is always zero. For $j/N = (0.011\cdots 1)_2$, the next $n-2$ steps satisfy $v_i = v_{i-1}$, so the function of $\sqrt{(1+x)/2}$ is calculated; the last step calculates $\sqrt{(1-x)/2}$. Using the fact that the derivative of $\sqrt{(1+x)/2}$ is no larger than $\sqrt{2}/4$, the error accumulation can be expressed as

$$\begin{aligned} |a_n - \hat{a}_n| &= |y(a_{n-1}) - y(\hat{a}_{n-1}) - \varepsilon_n| \\ &< \delta|a_{n-1} - \hat{a}_{n-1}| + |\varepsilon_n| \\ &< \frac{\sqrt{2}}{4}\delta|a_{n-2} - \hat{a}_{n-2}| + \delta|\varepsilon_{n-1}| + |\varepsilon_n| \\ &\cdots \\ &< \delta\left(\left(\frac{\sqrt{2}}{4}\right)^{n-3}|\varepsilon_2| + \cdots + \frac{\sqrt{2}}{4}|\varepsilon_{n-2}| + |\varepsilon_{n-1}|\right) + |\varepsilon_n| \\ &< [4\delta + 1]2^{-q}, \end{aligned} \tag{C.1.3}$$

where δ is the derivative of $\sqrt{(1-x)/2}$ satisfying $\delta = |y_n'| \in [\sqrt{2}/4, \infty)$.

Now we need to evaluate the upper bound of δ. First we have the relation of $\cos x < 1 - x^2/4$ when $x \in (0, 1)$. Then the output of the $(n-1)$th iteration satisfies

$$\hat{a}_{n-1} < a_{n-1} = \cos\frac{\pi}{2^{n-1}} < 1 - \frac{1}{4}\left(\frac{\pi}{2^{n-1}}\right)^2 < 1 - 2^{-2n+3}. \tag{C.1.4}$$

So the absolute value of the derivative at the nth iteration could be calculated,

$$|y_n'| = \frac{1}{\sqrt{2}}\frac{1}{\sqrt{1-\hat{a}_{n-1}}} < \frac{1}{\sqrt{2}}\frac{1}{\sqrt{2^{-2n+3}}} = \frac{1}{\sqrt{2^{-2n+4}}} = 2^{n-2}. \tag{C.1.5}$$

Substitute the result in Eq. (C.1.3) and we obtain

$$|a_n - \hat{a}_n| < [4\cdot 2^{n-2} + 1]2^{-q} = (2^n + 1)2^{-q}. \tag{C.1.6}$$

This is the error for $|\cos(j\pi/N)|$. We finally truncate the result of $2N^2(1-\cos(j\pi/N))$ into m qubits, and $m=2n+2+f$ (note that the n here represents the same parameter as that in, say Eq. (C.1.2)), where the $2n+2$ bits hold the integer part and the f bits hold the fractional part. In order to guarantee that all the m-bits of the output are totally exact, the q should satisfy the following inequality,

$$q \geq m+n. \tag{C.1.7}$$

Without loss of generality, we would assume that $f=2n+2$, so in such case $q=5m/4$ is adequate to make the high m-bits output being exact in theory.

The error in controlled rotation

According to Eq. (13), the probability amplitude of the quantum state after controlled rotation should be

$$\sin\theta_j = \frac{1}{\sqrt{1+\cot^2\theta_j}} = \frac{1}{\hat{\lambda}_j}, \theta_j \in (0, \pi/2). \tag{C.2.1}$$

Substituting Eq. (14) in Eq. (C.2.1), then it turns to

$$\sin\theta'_j = \frac{1}{\sqrt{1+\hat{\lambda}_j^2}}, \theta'_j \in (0, \pi/2). \tag{C.2.2}$$

Based on Eq. (14), we know that $1\big/\tilde{\lambda}_j = 1\big/\sqrt{1+\hat{\lambda}_j^2}$. So the error is

$$\begin{aligned}\varepsilon_2(0) = |\sin\theta'_j - \sin\theta_j| &= \left|\frac{1}{\sqrt{1+\hat{\lambda}_j^2}} - \frac{1}{\hat{\lambda}_j}\right| \\ &\leq \left|\frac{1}{\sqrt{1+\hat{\lambda}_{\min}^2}} - \frac{1}{\hat{\lambda}_{\min}}\right| = \left|\frac{1}{\sqrt{1+8^2}} - \frac{1}{8}\right| < 2^{-10}.\end{aligned} \tag{C.2.3}$$

We can reduce error by amplifying the approximated eigenvalues, i.e., shift the binary numeral of $\hat{\lambda}_j$ left, say, i bits. Now the error turns to

$$\begin{aligned}\varepsilon_2(i) &= \left|\frac{1}{\tilde{\lambda}_j} - \frac{1}{\hat{\lambda}_j}\right| = \left|\frac{1}{\sqrt{2^{-2i}+\hat{\lambda}_j^2}} - \frac{1}{\hat{\lambda}_j}\right| \leq \left|\frac{1}{\sqrt{2^{-2i}+8^2}} - \frac{1}{8}\right| \\ &= 2^{-3}\left(2^0 - \frac{1}{\sqrt{2^0+2^{-2i-6}}}\right) \underset{NR}{<} 2^{-3}\left(2^0 - \frac{1}{2^0+2^{-2i-7}}\right) \\ &\underset{NR}{=} 2^{-3}\left(2^0 - \sum_{j=1}^{i} 2^{-2j-7}\right) = 2^{-2i-10},\end{aligned} \tag{C.2.4}$$

where NR means the operation is calculated by non-restoring method. So the upper bound of error $\varepsilon_2(i)$ reduces exponentially with i. Shifting left one bit makes the error reduce about 2^2 times. And the state after controlled rotation turns to $\sqrt{1-\left(C'\big/2^i\tilde{\lambda}_j\right)^2}|0\rangle + C'\big/2^i\tilde{\lambda}_j|1\rangle$, where C' represents the normalizing constant. Factor $1\big/2^i$ will be contained in the normalizing constant.

References

1. Cleve, R., Ekert, A., Macchiavello, C., Mosca, M.: Quantum algorithms revisited. Proc. R. Soc. Lond. A **454**, 339–354 (1998)
2. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, Chaps. 1–6. Cambridge University Press, Cambridge (2010)
3. Shor, P.W.: Algorithms for quantum computation: discrete logarithms and factoring. In: Proceedings 35th Annual Symposium on Foundations of Computer Science, pp. 124–134 (1994)
4. Grover, L.: Quantum mechanics helps in searching for a needle in a haystack. Phys. Rev. Lett. **79**, 325–328 (1997)
5. Harrow, A.W., Hassidim, A., Lloyd, S.: Quantum algorithm for linear systems of equations. Phys. Rev. Lett. **103**, 150502 (2009)
6. Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., Lloyd, S.: Quantum machine learning. Nature **549**, 195–202 (2017)
7. Leyton, S.K., Osborne, T.J.: A quantum algorithm to solve nonlinear differential equations (2008). arXiv:0812.4423
8. Berry, W.: High-order quantum algorithm for solving linear differential equations. J. Phys. A Math. Theor. **47**, 105301 (2014)
9. Berry, W., Childs, A.M., Ostrander, A., Wang, G.: Quantum algorithm for linear differential equations with exponentially improved dependence on precision. Commun. Math. Phys. **356**, 1057–1081 (2017)
10. Arrazola, J.M., Kalajdzievski, T., Weedbrook, C., Lloyd, S.: Quantum algorithm for non-homogeneous linear partial differential equations. Phys. Rev. A **100**, 032306 (2019)
11. Cao, Y., Papageorgiou, A., Petras, I., Traub, J., Kais, S.: Quantum algorithm and circuit design solving the Poisson equation. New J. Phys. **15**, 013021 (2013)
12. White, F.M.: Fluid Mechanics, Chap. 4, 8th edn. McGraw-Hill Education, New York (2016)
13. Lukaszewicz, G., Kalita, P.: Navier–Stokes Equations: An Introduction with Applications, Chap. 2. Springer, Switzerland (2016)
14. Kosior, A., Kudela, H.: Parallel computations on GPU in 3D using the vortex particle method. Comput. Fluids **80**, 423–428 (2013)
15. Steijl, R., Barakos, G.N.: Parallel evaluation of quantum algorithms for computational fluid dynamics. Comput. Fluids **173**, 22–28 (2018)
16. Hockney, R.W.: A fast direct solution of Poisson's equation using Fourier analysis. J. Assoc. Comput. Mach. **12**, 95–113 (1965)
17. Buzbee, B.L., Golub, G.H., Nielson, C.W.: On direct methods for solving Poisson's equations. SIAM J. Numer. Anal. **7**, 627–656 (1970)
18. Swarztrauber, P.N.: The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. SIAM Rev. **19**, 490–501 (1977)
19. Ritter, K., Wasilkowski, G.W.: On the average case complexity of solving Poisson equations. Lect. Appl. Math. **32**, 677–688 (1996)
20. Childs, A.M., Liu, J.-P.: Quantum spectral methods for differential equations (2019). arXiv:1901.00961
21. Childs, A.M., Kothari, R., Somma, R.D.: Quantum algorithm for systems of linear equations with exponentially improved dependence on precision. SIAM J. Comput. **46**, 1920–1950 (2017)
22. Borwein, J.M., Girgensohn, R.: Addition theorems and binary expansions. Can. J. Math. **47**, 262–273 (1995)
23. Freiman, C.V.: Statistical analysis of certain binary division algorithms. Proc. IRE **49**, 91–103 (1961)

24. Sutikno, T.: An efficient implementation of the non restoring square root algorithm in gate level. Int. J. Comput. Theory Eng. **3**, 46 (2011)
25. Burks, A.W., Goldstine, H.H., von Neumann, J.: Preliminary discussion of the logical design of an electronic computing instrument. Princeton, Institute for Advanced Study (1947)
26. Demmel, J.W.: Applied Numerical Linear Algebra, Chap. 6. SIAM, Philadelphia (1997)
27. Aaronson, S.: Read the fine print. Nat. Phys. **11**, 291–293 (2015)
28. Biham, O., Biron, D., Grassl, M., Lidar, D.A.: Grover's quantum search algorithm for an arbitrary initial amplitude distribution. Phys. Rev. A **60**, 2742 (1999)
29. Grover, L.K.: Synthesis of quantum superpositions by quantum computation. Phys. Rev. Lett. **85**, 1334 (2000)
30. Grover, L., Rudolph, T.: Creating superpositions that correspond to efficiently integrable probability distributions (2002). arXiv:quant-ph/0208112
31. Soklakov, A.N., Schack, R.: Efficient state preparation for a register of quantum bits. Phys. Rev. A **73**, 012307 (2006)
32. Luis, A., Peřina, J.: Optimum phase-shift estimation and the quantum description of the phase difference. Phys. Rev. A **54**, 4564 (1996)
33. Weinstein, Y.S., Pravia, M.A., Fortunato, E.M., Lloyd, S., Cory, D.G.: Implementation of the quantum Fourier transform. Phys. Rev. Lett. **86**, 1889 (2001)
34. Lloyd, S.: Universal quantum simulators. Science **273**, 1073–1078 (1996)
35. Berry, D.W., Ahokas, G., Cleve, R., Sanders, B.C.: Efficient quantum algorithms for simulating sparse Hamiltonians. Commun. Math. Phys. **270**, 359–371 (2007)
36. Wickerhauser, M.V.: Adapted Wavelet Analysis: From Theory to Software, Chap. 3. Peters A.K./CRC Press, Wellesley (1994)
37. Klappenecker, A., Rotteler, M.: Discrete cosine transforms on quantum computers (2001). arXiv:quant-ph/0111038
38. Barenco, A., Bennett, C.H., Cleve, R., DiVincenzo, D.P., Margolus, N., Shor, P., Sleator, T., Smolin, J.A., Weinfurter, H.: Elementary gates for quantum computation. Phys. Rev. A **52**, 3457 (1995)
39. Vedral, V., Barenco, A., Ekert, A.: Quantum networks for elementary arithmetic operations. Phys. Rev. A **54**, 147 (1996)
40. Bhaskar, M.K., Hadfield, S., Papageorgiou, A., Petras, I.: Quantum algorithms and circuits for scientific computing. Quantum Inf. Comput. **16**, 197–236 (2016)
41. Brassard, G., Hoyer, P., Mosca, M., Tapp, A.: Quantum amplitude amplification and estimation. Contemp. Math. **305**, 53–74 (2002)
42. Zhao-Yun, Chen, Qi, Zhou, Cheng, Xue, Xia, Yang, Guang-Can, Guo, Guo-Ping, Guo: 64-qubit quantum circuit simulation. Sci. Bull. **63**, 964–971 (2018)
43. The web address for accessing our QRunes codes
44. User manual of QRunes. https://github.com/OriginQ/QPanda/tree/master/QRunes
45. Mitarai, K., Kitagawa, M., Fujii, K.: Quantum analog-digital conversion. Phys. Rev. A **99**, 012301 (2019)
46. Häner, T., Roetteler, M., Svore, K.M: Optimizing quantum circuits for arithmetic (2018). arXiv:1805.12445
47. Pan, V.Y., Ivolgin, D., Murphy, B., Rosholt, R.E., Tang, Y., Yan, X.: Additive preconditioning for matrix computations. Linear Algebra Appl. **432**, 1070–1089 (2010)