Quantum Information Processing (2022) 21:137
https://doi.org/10.1007/s11128-022-03478-w

Quantum Approach to Accelerate Finite Volume Method on Steady Computational Fluid Dynamics Problems

Zhao-Yun Chen[1] · Cheng Xue[1] · Si-Ming Chen[1] · Bing-Han Lu[1] · Yu-Chun Wu[1] · Ju-Chun Ding[2] · Sheng-Hong Huang[2] · Guo-Ping Guo[1]

Received: 5 August 2021 / Accepted: 2 March 2022 / Published online: 31 March 2022

Abstract
Computational fluid dynamics (CFD) is a branch of fluid mechanics that solves fluid flows by numerical methods. Recently, quantum computing has been proven to outperform a classical computer on specific computational tasks. However, using a quantum computer to accelerate the CFD solver remains a challenge. Existed quantum differential equation solvers, which are limited to the linearity of the equation, cannot be applied in the CFD because the fluid equations are highly nonlinear. Here, we propose a quantum approach to accelerate the finite volume method, which is typical in the classical CFD domain. We focus on how a quantum computer handles classical input and output, designing a specific quantum data structure that allows fast memory update throughout the calculation, resulting in an exponential speedup over the classical counterpart. Numerical tests show that this algorithm adapts to various CFD problems, including problems with high nonlinearity. This approach complements existed quantum methods for nonlinear differential equations and allows new frontiers of the CFD by allowing a breakthrough of the cell number and the solution speed.

Keywords Quantum computing · Quantum algorithm · Quantum random access memory · Partial differential equation · Quantum data structure · Computational fluid dynamics

✉ Yu-Chun Wu
wuyuchun@ustc.edu.cn

✉ Sheng-Hong Huang
hshnpu@ustc.edu.cn

✉ Guo-Ping Guo
gpguo@ustc.edu.cn

1 Key Laboratory of Quantum Information, CAS, University of Science of Technology of China, Hefei, China

2 Department of Modern Mechanics, USTC, Hefei, China

1 Introduction

Computational fluid dynamics (CFD) is about solving partial differential equations (PDEs) to obtain the physical properties of fluids, which plays a vital role in industrial design. One of the most significant equations, the Navier–Stokes equation, models the evolution of flow fields on the properties of density, momentum, and energy [1]. Solving such PDEs is a complicated task; thus, numerical methods are often applied to provide approximate results.

Using the quantum computing approach to achieve a faster numerical solution is promising. Quantum computing is a new computing paradigm that offers exponential acceleration over classical computing approaches. Many quantum algorithms, including quantum factorization [2], quantum simulation [3–6], and the linear system solvers, [7–9] have already appeared to prove this idea. The quantum computing hardware is also experiencing fast development. Tens of qubits are made in the superconducting system [10, 11] and ion trap [12]. The quantum error correction and topological quantum computing theories have also been developed to create a fault-tolerant quantum computer [13–16].

The numerical solution of the fluid flow involves two kinds of problems: transient problems, which require the system state at any given time, and steady problems, which require the steady state after the system evolving an infinitely long time. In this paper, we focus on the quantum speedup on steady problems. A no-go theorem has proved that a nonlinear PDE cannot be solved efficiently even with quantum computing approaches where time precision is considered, and the nonlinearity is large. This theorem prohibits a fast quantum algorithm from solving a general range of fluid flows, where the nonlinear behaviors are common. Instead, the steady problems require to output a steady solution of the fluid flow. Thus, the time precision becomes unimportant in steady problems, and the quantum speedup becomes expectable.

In steady problems, the first-order time-stepping schemes are usually applied. Such schemes start from an arbitrary initial state, evolve with pseudo-time and targets to efficiently converge the system to the stable state solution. In the first-order scheme, one can either evaluate the variable at time t^n or t^{n+1}, named by the "explicit" or "implicit" method, respectively. The explicit method can directly compute the next step's variable from this step, and the implicit method should solve a linear equation to compute that. In practice, the explicit method is more suitable for parallel computing, and the implicit method can converge faster [17].

The finite volume method (FVM) is a typical numerical method in the classical CFD domain [18, 19]. In FVM, the computational space is split into small cells, and from an initial state, one performs discrete time-stepping to evolve the system. The time complexity of FVM is mainly dominated by the time of performing the time-stepping, which involves solving large-scale linear algebra problems. For a sparse matrix linear solver, Krylov's subspace method is widely used in practice. The complexity can be linear to the problem size (the number of grid cells) [20]. When the problem size is large, the computing resource (including hardware resources and computing time) will become expensive, for example, implementing large-scale CFD problem on supercomputer and CPU/GPU clusters [21–23].

Intuition is that applying the quantum linear solver (QLS) in the implicit method exponentially accelerates the FVM.[1] However, applying QLS to practical problems is not apparent. It has been pointed out that the conversion between the classical and the quantum data could become a bottleneck [26], which will fail to demonstrate any quantum advantage.

This paper studies the case of using the implicit method to solve steady problems. We propose a full recipe to accelerate FVM with the quantum approach. By only assuming that the input and output are all classical data, we can still obtain exponential speedup on the steady problem cases. To achieve this, we apply quantum random access memory (QRAM) [27, 28] and design a quantum memory layout. At the input stage, the memory layout helps to implement quantum inputs required by the QLS. At the output stage, we sample the output state and sparsely update the memory. We show that these two processes, which act as the interface between classical and quantum data, can both run in polylogarithmic time. As a result, they enable us to integrate the quantum linear solver submodule into the classical FVM to achieve exponential speedup.

The QLS and the l_∞ tomography [24] method create a sparse update vector to enable efficient time-stepping. Although such an update differs significantly from the classical algorithm, the convergence stability is maintained if choosing appropriate quantum error tolerance. To prove this, we analyze the error model resulting from the quantum process and perform numerical experiments on various test cases. The result shows that this method has broad applicability to general types of CFD problems with the quantum advantage being kept.

There were some previous works about solving differential equations and partial differential equations with the quantum computer [29–36]. The comparison and relation between this paper and the previous works are discussed in Appendix section 1.

2 Motivation: quantum algorithm with classical input and output

When we use a quantum computer to cope with a practical problem, we should always and only expect classical inputs and outputs. Many quantum algorithms have been proposed and claimed to be faster (exponentially or polynomially) than their classical counterparts. However, a large portion of them only beats classical algorithms under some theoretical limitations. A famous example is the quantum linear system (QLS) algorithm: Harrow–Hassidim–Lloyd (HHL) algorithm [7], which can prepare the state $|x\rangle$, encoding the solution of the linear equation $\boldsymbol{Ax} = \boldsymbol{b}$. This algorithm uses $\mathcal{O}(\log N)$ calls to linear equation oracles, where the classical counterpart has to perform at least $\mathcal{O}(N)$ calls. Based on this work, many quantum machine learning algorithms were proposed and claimed to have exponential speedup over their classical counterparts. However, most of these algorithms did not answer how to deal with

[1] Although it is also viable to accelerate the explicit method quantumly, this will not provide a substantial speedup against accelerating the implicit method. The reason is that the time complexity of QLS and quantum matrix multiplication are both polylogarithmic [24, 25]. Moreover, we can better enjoy the stability benefit of the implicit method without extra sacrificing the time complexity.

real-world data to realize such oracles. Meanwhile, they did not answer how to output the classical vector. In [26], the authors raised a series of obstacles for applying the QLS on quantum machine learning algorithms with real-world data. The main problems include inputting the classical data into the quantum computer and extracting information from the output state given by the QLS. If we hope to preserve quantum speedup, two operations are forbidden. One is to prepare the input state $|b\rangle$ with an encoded quantum circuit, where even reading all data entries requires $\mathcal{O}(N)$ time. The other is to perform sampling on the output state to extract the state to a classical vector with $\mathcal{O}(N)$ times measurements.

We believe the obstacles that appeared in "QLS-based" quantum machine learning algorithms that are also challenging if we want to accelerate the FVM for CFD problems quantumly. The physical variable should be updated at every time step. The time complexity will become the multiplication of the T_x (the time complexity for preparing $|x\rangle$) and M (the number of copies required for sampling). The M should always be sublinear to the problem size N; otherwise, we cannot achieve quantum speedup. Our proposal will consider these obstacles, only assuming that the input and output of this algorithm are all classical data.

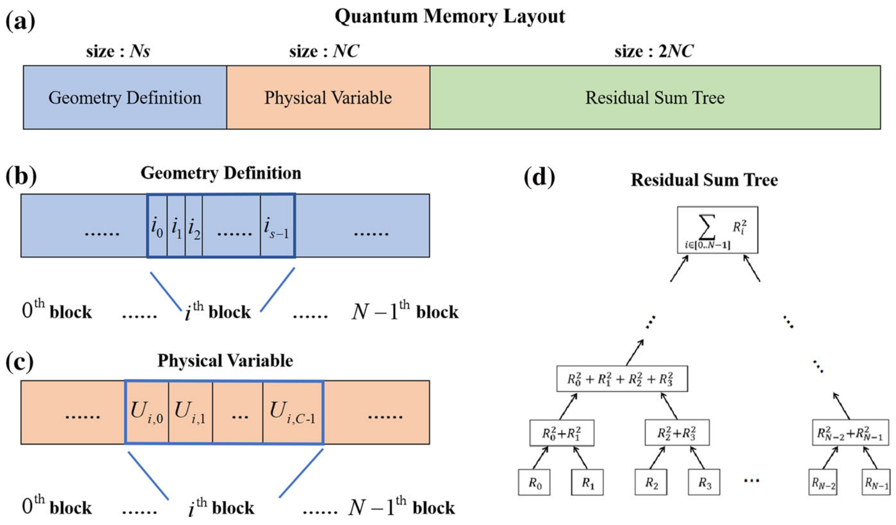
3 Methods

3.1 Designated quantum data structure for the FVM

We store the interval data into the QRAM to allow quantum access. The quantum random access memory (QRAM) is the particular storage device used by a quantum computer. It stores classical data, which can be retrieved in quantum superposition. Detailed descriptions of QRAM and our assumptions are shown in Appendix section 1.

Here we design a quantum data structure for integrating quantum approaches with the FVM, as shown in Fig. 1. The data structure includes the geometry definition, the physical variable $\boldsymbol{U}$, and the residual vector $\boldsymbol{R}$, which are identical to what the classical FVM solver stores. All data in the QRAM are continuous storage, which allows it to locate any of them with simple computations. In the FVM, each cell is represented by a C-sized tuple of several physical properties, such as a five-tuple $(\rho, \rho u, \rho v, \rho w, \rho E)$ in the compressible three-dimensional flow. (A detailed information about the FVM is shown in Appendix 1.) Thus, we locate each element with an address pair (i, k). The first is the node's number, and the second locates the position in the tuple. For example as shown in Fig. 1(c), $U_{i,k}$ represents the k^{th} variable of the i^{th} node.

Besides those, we design a vital component for the quantum input and output, which is the residual sum tree, depicted in Fig. 1d. We precompute the sum of the l_2 norm of the residual vector and its subvector and store them into the sum tree. Every node of the binary tree stored the sum of two children nodes, except that the second bottom layer stored the sum of the square of two children. This structure is maintained throughout the whole FVM computing. Whenever the physical variable's elements are changed, the residual sum tree is also re-computed. The detailed method for updating the sum tree is introduced in the following section (Sect. 3.3).



(a)
Quantum Memory Layout
size : Ns
size : NC
size : 2NC
Geometry Definition
Physical Variable
Residual Sum Tree
(b)
Geometry Definition
i_0 i_1 i_2 ...... i_{s-1}
0th block ...... ith block ...... N−1th block
(c)
Physical Variable
$U_{i,0}$ $U_{i,1}$ ... $U_{i,C-1}$
0th block ...... ith block ...... N−1th block
(d)
Residual Sum Tree
$\sum_{i\in[0..N-1]} R_i^2$
$R_0^2+R_1^2+R_2^2+R_3^2$
$R_0^2+R_1^2$
$R_2^2+R_3^2$
$R_{N-2}^2+R_{N-1}^2$
R_0 R_1 R_2 R_3 R_{N-2} R_{N-1}

Fig. 1 Schematic of the quantum data structure. **a** Three memory areas of the QRAM: geometry definition area, which holds the input of the problem; physical variable area holding the $\boldsymbol{U}^n$; and the residual sum tree. **b** The linear structure of the geometry definition area. This area is formed with N blocks. The i^{th} block holds s related indices where each of $i' = i_k$ $(0 \leq k < s)$ satisfies $C_{i,i'} = 1$. **c** The linear structure of the physical variable area. Each block of the physical variables $\boldsymbol{U}_i$ at the cell i. **d** The binary tree structure of the residual sum tree. The tree's leaves are the components of the residual vector $\boldsymbol{R}^n$. Then for each level, we sum up the square of every two nodes. The tree root is $\|\boldsymbol{R}\|^2$.

Now we introduce the quantum functions enabled by such data structures. The physical variable and the residual vector are common vectors. They can be queried by inputting the addresses (or the indices) in a quantum superposition, denoted by $\mathcal{P}_U$:

$$\mathcal{P}_U|i,k\rangle|0\rangle = |i,k\rangle|U_{i,k}\rangle. \tag{1}$$

All tree nodes are also stored continuously. We define each node's address as $a_r(p)$ from the top of the sum tree. p is a binary string where every digit represents the left/right branch with 0/1. For example, $a_r(0)$ is the address of the left child of the root; $a_r(0,1)$ is the right child of the node at $a_r(0)$. Specially, we directly use a_r to represent the root's address. The binary tree's size is determined, so that any address $a_r(p)$ can be computed efficiently. The data contained in address $a_r(p)$ are denoted by $S_R(p)$. We can perform such unitary transform:

$$\mathcal{P}_R|a_r(p)\rangle|0\rangle = |a_r(p)\rangle|S_R(p)\rangle. \tag{2}$$

The geometry definition stores the geometry input of the CFD problem. From this, we can query the position and the connection of all cells in a quantum parallel. These are constant during the calculation and are used for computing the elements of the equation. This part enables this unitary:

$$\mathcal{P}_s|i\rangle|j\rangle = |i\rangle|i_j\rangle, \tag{3}$$

which is essential for the quantum linear solver.

To summarize, this data structure enables these following processes:

1 *Initialization* Initialize the residual sum tree will classically access the QRAM $\mathcal{O}(N)$ times;
2 *Quantum Input* With access to the QRAM, one can prepare $|R\rangle$ in $\mathcal{O}(\log^2 N)$ time.
3 *Quantum Output* The quantum solver outputs the classical solution which sparsely updating the physical variable $\boldsymbol{U}$ no more than $\mathcal{O}(\epsilon^{-2}\log N)$ times.
4 *Result* Time-stepping from $\boldsymbol{U}^n$ to $\boldsymbol{U}^{n+1}$ will cost $\tilde{\mathcal{O}}((s^3+\log N)s\kappa\epsilon^{-2}\log^3 N)$ time.

In the following text, we will introduce how quantum input and output are realized. Then, we analyze the time complexity of all the above processes in the "run-time analysis" section to prove these results.

3.2 Quantum input: constructing quantum subprocedures from the QRAM

The time-stepping of $\boldsymbol{U}$ is realized by solving linear equations while using an implicit Euler time-stepping scheme. The linearization scheme of the finite volume method is introduced in Appendix Sect. 1, where Eq. (18) is the typical form. In general, the implicit Euler method derives

$$\boldsymbol{A}\Delta\boldsymbol{U}=\boldsymbol{R}. \tag{4}$$

In this equation, $\boldsymbol{A}$ and $\boldsymbol{R}$ are the coefficient matrix and the vector, which subjects to the $\boldsymbol{U}$ at a certain step. Solving this equation, we obtain $\Delta\boldsymbol{U}$, which is the update vector to the $\boldsymbol{U}$. The Euler method is a first-order timing scheme; thus,

$$\boldsymbol{U}^{n+1}=\boldsymbol{U}^n+\Delta\boldsymbol{U}^n \tag{5}$$

performs a stepping from n to $n+1$. Here we use the superscript n to denote the n^{th} iteration step. The main idea is to use the quantum approach to accelerate the linear solver to achieve exponential speedup.

We introduce the quantum linear solver in Appendix Sect. 1. The input of the quantum linear solver consists of three quantum subprocedures. For a general sparse linear equation (like 4), we have the following three quantum processes which encode the information of the linear equation:

$$\mathcal{P}_A|i,k,i',k'\rangle=|i,k,i',k'\rangle|A_{i,k}^{i',k'}\rangle, \tag{6}$$

which encodes the matrix's element, and

$$\mathcal{P}_R|i,k\rangle=|i,k\rangle|R_{i,k}\rangle, \tag{7}$$

which encodes the vector's element, and

$$\mathcal{P}_l|i,p\rangle=|i,C_i(p)\rangle, \tag{8}$$

which encodes the p^{th} related cell in the difference scheme.

In the original proposal of the quantum linear solver, these three processes are viewed as "black-boxes". They did not evaluate the time complexity for solving a certain linear equation. Instead, they evaluated the query complexity to these black-boxes, namely how many times the quantum linear solver calls these three processes. Our task is to construct the input of the quantum linear solver by utilizing the quantum data structure introduced above.

Using the data stored in QRAM to construct $\mathcal{P}_A$ and $\mathcal{P}_l$ is straightforward. Construction of $\mathcal{P}_l$ is equivalent to query the geometry definition data, which has been prepared beforehand. To construct $\mathcal{P}_A$, we need to compute the value of a matrix element with its position as the input (row and column). Constructing $\mathcal{P}_A$ requires at most $O(s)$ queries. First, query $\mathcal{P}_l$ to find all related nodes, then query the physical variables of these nodes, where only $O(s)$ nodes are related. Computing the matrix element is also gate-efficient because these computations are also efficient in classical computing.

Constructing $\mathcal{P}_b$ requires the data in the residual sum tree. Inspired from [37], if we use a sum tree storing all precomputed of a real-value vector in the QRAM, then with the method introduced in [38], we can prepare such vector efficiently, time complexity being $O(\log N)$. Using this method, $\mathcal{P}_b$ can be constructed with $O(\log N)$ times of queries to the QRAM. A detailed implementation process is written in Appendix Sect. 1.

Using this approach, we can also append a preconditioner to the linear equation. The classical Krylov subspace method and the quantum linear solver are faced with condition number problems. Both complexities have a linear dependency on the condition number. One can use preconditioner, denoted by $\boldsymbol{P}$, to construct a new equation $\boldsymbol{PAx} = \boldsymbol{Pb}$. If $\boldsymbol{PA}$ has a smaller condition number than $\boldsymbol{A}$, and such preconditioning processes can be implemented efficiently; then, the time complexity can be reduced. Preconditioners are widely used in classical CFD solvers. There have also been proposed several preconditioned quantum linear solvers [39–41]. Here, we integrate Jacobi preconditioner in our method. The time complexity of constructing $\mathcal{P}'_A$ and $\mathcal{P}'_R$ multiplies by an $O(\text{poly} s)$ which is preconditioned version $\boldsymbol{A}' = \boldsymbol{PA}$ and $\boldsymbol{A}' = \boldsymbol{PR}$ correspondingly. The data stored in the residual sum tree are modified to its preconditioned version. A detailed implementation process of the quantum Jacobi preconditioner is written in Appendix section 1.

3.3 Quantum output: sampling the solution state and update the QRAM

With the quantum inputs, QLS outputs a solution $|u\rangle = |\frac{\Delta \boldsymbol{U}^{n+1}}{\|\Delta \boldsymbol{U}^{n+1}\|}\rangle$, a normalized solution of the linear equation, where $\|\cdot\|$ represents l_2 norm of a vector in this paper. Now we name the quantum linear solver process as $\mathcal{A}_u$. $\mathcal{A}_u$ can prepare the $|u\rangle$ within sublinear time. Our task is to call $\mathcal{A}_u$ sublinear time to update the physical vector and move to next iteration step.

Using l_∞ tomography algorithm [42], we can convert the quantum state into a classical vector within sublinear time. Taking $\mathcal{A}_u$ as the input, running $\mathcal{A}_u$ and its controlled version by $\mathcal{O}(\epsilon^{-2} \log N)$ many times, we obtain a classical vector $\tilde{u}$ which is

ϵ-close to the quantum solution $\boldsymbol{u}$. We introduce l_∞ tomography algorithm in Appendix Sect. 1. As a result, obtaining a classical vector $\tilde{\boldsymbol{u}}$ requires sublinear time.

The tomography algorithm only produces a normalized vector $\tilde{\boldsymbol{u}}$. We should also obtain all the normalized factors in the algorithm to get the actual update vector of $\boldsymbol{U}$. QLS produces two factors. First is c_R, which is generated when preparing $|R\rangle$ and can be obtained from the residual sum tree described above. The second is c_l, which is derived from the non-unitarity of the matrix inversion and the normalization factor of the solution. With amplitude estimation [43] (also introduced in Appendix Sect. 1), we can compute the probability p_l and then obtain the factor by $c_l = \alpha\sqrt{p_l}$, where α is a constant in the QLS. Obtaining the normalization factors will not affect the asymptotic time complexity of the algorithm. Combining these two factors $c = c_R c_l$, we obtain the norm of solution $\|\Delta\boldsymbol{U}\|$, which implies the variation updated on the target vector $\boldsymbol{U}$ in the CFD solver.

To perform the time-stepping, we update the physical variable $\boldsymbol{U}$ with $\Delta\boldsymbol{U}$. Updating the QRAM from the sampled vector $\tilde{\boldsymbol{u}}$ is also efficient. The l_∞ tomography algorithm produces a sparse classical vector with not more than $\mathcal{O}(\epsilon^{-2}\log N)$ nonzero elements, which means that the update of QRAM will be performed for less than $\mathcal{O}(\epsilon^{-2}\log N)$ times to update the physical variable.

Same as the computing $\boldsymbol{R}$ and its sum from $\boldsymbol{U}$, when updating any element of the physical variable, only the residual on the related cells would change. From the tree leaves, we update the all residual $R_{i',k'}$ related to the updated $U_{i,k}$ with $C_{i,i'} = 1$. After these residual vector entries change, we again compute the sum tree from the leaves to the root and update correspondingly. The number of updated nodes will not exceed the number of the multiplication of the related residual entries $\mathcal{O}(s)$ and the number of layers of the sum tree $\log N$.

As a result, the cost of update one entry of $\boldsymbol{U}$ is less than $\mathcal{O}(s\log N)$. Thus, both sampling and updating processes can run in sublinear time.

For a steady problem, the computing stops in two cases. One is when the residual is smaller than the convergence limit ϵ, which can be extracted from the top of the tree. Another is when reaching the maximum iteration steps. After stopping, the output of this algorithm is the classical vector stored in the physical variable area.

4 Run-time analysis

The time cost for the quantum approach has two main contributions. One is the cost of initializing the data structures (initialization cost); the other is the time complexity between two iteration steps (time-stepping cost).

4.1 Initialization cost

The initialization process fills the QRAM following the memory layout, which is entirely a classical process. The first step is to write in the initial physical variable $\boldsymbol{U}$ and fill the tree with the wanted sum. Along with these data, we should also fix the memory layout to quickly obtain the memory address of every data entry in constant time.

There may be some concentrations about whether such $\mathcal{O}(N)$ preparation time will cause the vanishment of this algorithm's speedup. However, after considering the time consumption of initialization, the quantum speedup is still preserved. To support this, we analyze the three things that contribute to initialization time.

The first is the calculation of the residual vector $\boldsymbol{R}$. The calculation of this vector exists at every step of the classical FVM in a CFD problem. Even in classical algorithms, this part is not the bottleneck of time. Our algorithm only calculates the residual once initially, which will consume much less than a classical algorithm does.

The second is the fill of the sum tree. To fill a sum tree only requires repeatedly adding the sum of the square of the residual vector. Therefore, it is natural to think this process takes a shorter time than calculating the residual vector.

The third is about the cost of accessing the QRAM classically. As we have mentioned in Appendix Sect. 1, we assume the QRAM has the near capability of RAM, which allows the access to be performed in constant time.

As a result, under the assumption about the QRAM's capability, the initialization cost would not cost much more than the preprocessing stage of the classical FVM. Also, the initialization of quantum memory only processes once, and we believe this cost would not become the time's bottleneck.

4.2 Complexity of time-stepping

The evolution cost is the time complexity at every evolution stage. In [9], the authors provided a linear solver algorithm with logarithmic dependence on precision. They show that the query complexity of this algorithm of O_A, O_l and O_b is $O\left(s\kappa \text{polylog}(\frac{s\kappa}{\epsilon})\right)$. Now we start to analyze the time complexity of constructing these subprograms from the initial problem settings.

According to the results in the previous sections, the number of queries to QRAM for implementing O_A, O_l, and O_b is $\mathcal{O}(s)$, $\mathcal{O}(1)$, and $\mathcal{O}(\log N)$, correspondingly. The time complexity of preconditioned O'_A has a multiplier of $\mathcal{O}(s^3)$ contributed by computing the inverse of the diagonal blocks of (A). The preconditioned O'_b has the same complexity as O_b.

Now consider the time cost of sampling and updating. We run the QLS with $\mathcal{O}(\epsilon^{-2}\log N)$ times to obtain an l_∞-close classical vector. This becomes another multiplier to the time complexity of the quantum procedure.

The last multiplier is the cost of querying the QRAM. As we have assumed, the QRAM use $\mathcal{O}(\log N)$ time to perform one query. By composing these results, the time complexity of the quantum procedure is

$$\mathcal{O}\left((s^3 + \log N)s\kappa \log^3 N\epsilon^{-2}\text{polylog}(s\kappa/\epsilon)\right). \tag{9}$$

The final step is to update the sum tree. Updating a preconditioned residual tree has two steps. One is to compute the preconditioned residual, where each term will involve in another inversion of the matrix A, which is $\mathcal{O}(s^3)$; the other is to update the tree from bottom to the top, which involves $\mathcal{O}(\log N)$ times for one change in the bottom of the tree. While at most $\mathcal{O}(\epsilon^{-2}\log N)$ terms of $\boldsymbol{U}$ changes, the time cost of updating the tree is $\mathcal{O}(s^3\log^2 N\epsilon^{-2})$. The total time complexity is the addition of the quantum and the classical procedure. Because the quantum procedure's complexity is asymptotically greater than the classical's, we conclude that the evolution time cost has the time complexity shown in Eq. (9).

The classical counterpart's time complexity is $\mathcal{O}(Ns\kappa\log 1/\epsilon)$ when using CG as the linear solver. Our algorithm outperforms the classical algorithm on the problem size's dependency but has worse performance when the problem requires high precision. When the problem size N and the requirement of the precision ϵ has such relation $N \gg \epsilon^{-2}$, the quantum algorithm will potentially have better performance on time.

5 Error analysis

The time complexity of the quantum approach has better performance on the number of grid cells N but worse on the precision ϵ, which implies that the problem size should be large enough to show the quantum advantage. On the other side, the numerical experiment shows that the precision should be small. Otherwise, the time integration will not converge. There is the problem: if the precision requirement has some dependency on the problem size, the quantum acceleration will decrease or even vanish. In this section, we will provide evidence that the error threshold will not grow with the problem size.

First, we calculate the total error generated by the quantum sampling with error bound ϵ specified. At one step, we define the physical variable $\boldsymbol{U}$ and its update $\Delta\boldsymbol{U}$. In our proposal, the quantum process outputs a quantum state $|u\rangle$ which is proportional to $\Delta\boldsymbol{U}$,

$$\Delta\boldsymbol{U} = u\|\Delta\boldsymbol{U}\|. \tag{10}$$

The l_∞ tomography outputs a classical vector $\tilde{u}$ which is ϵ-close to u. At any index i, we have

$$u_i = \tilde{u}_i + e_i, \tag{11}$$

where the error term $|e_i| < \epsilon$.

Now we consider the amplitude of the e_i. When performing l_∞ tomography, the output vector is a sample from the multinomial distribution where the sampling number $M = C\log N/\epsilon^2$ and the probability distribution $(|u_0|^2, |u_1|^2, ... |u_{N-1}|^2)$. At any term, the standard error of such sample is: $\sigma_i = \sqrt{M|u_i|^2(1-|u_i|^2)}$. When $|u_i|$ is small enough, we have $\sigma_i \sim |u_i|\sqrt{N}$. Now we assume the error e_i is approximately linear dependent on the standard error σ_i, and thus, we have

$$e_i = \mathcal{O}(\sigma_i) \sim \mathcal{O}(|u_i|\sqrt{N}). \tag{12}$$

The update vector output by the quantum linear solver should be multiplied by $\|\Delta\boldsymbol{U}\|$. As a result, the total error will be amplified by this coefficient.

$$E_i = \|\Delta U\|e_i. \tag{13}$$

Compare two cases describing the same problem where one has N cells and the other has kN (mark the variables with extra prime, e.g., u'). We can assume the distribution of $\Delta \boldsymbol{U}$ and $\Delta \boldsymbol{U}'$ is the same because the physical characteristic does not change. From this, we have

$$\|\Delta U'\|^2 = k\|\Delta U\|^2, \tag{14}$$

because only the vector size changes to k times. From the definition of u (equation (10)), this results in the decrease of the amplitude of the u, i.e.,

$$u_i' = \frac{1}{\sqrt{k}} u_i. \tag{15}$$

Combining Eqs. (12), (13), and (15), we obtain that $E_i = \mathcal{O}(U_i)$. This result implies that the total error generated by the quantum sampling will not change over the problem size N.

6 Numerical experiment

6.1 Numerical methods

The open-source classical CFD software, SU2 [44–46], is selected as the base CFD solver. The SU2 mainly implements the finite volume method solver for unstructured grids. The SU2 is highly configurable, supporting sorts of equations and algorithms.

Concentrating on the implicit Euler solver, we inserted codes to support the simulations of quantum error. This part of the code is written around the original linear solver to emulate the errors introduced by quantum processes.

Here we ignored the error from the QLS and focused on simulating the error generated by l_∞ tomography. As introduced in Appendix Sect. 1, the time complexity increases polylogarithmic to the precision, which implies that we can set a sufficiently high precision without much decelerating this process. In fact, the slowdown effect on the quantum error complexity (see Run-Time Analysis Section) results from the sampling process, namely the error setting from the l_∞ tomography.

After the linear equation is generated, we classically solve this equation to obtain an unnormalized output. Then, we divide this vector by its l_2 norm to emulate the output of the quantum linear solver. Then, we take this normalized vector as the input of a classical sampling algorithm. The algorithm samples the probability distribution $C\epsilon^{-2}\log N$ times for the vector size N, error ϵ, and $C = 32$(consistent with the original l_∞ algorithm). The sample vector is finally multiplied by the above l_2 norm to obtain a quantum version solution. The whole computation process uses the quantum solution instead of the classical solution, while other processes remain unchanged.

We chose a series of example test cases, appended different quantum errors, output the evolution history and flow field. By comparing the quantum error biased results to the classical solver, we study whether the quantum error effect will damage the availability of the FVM.

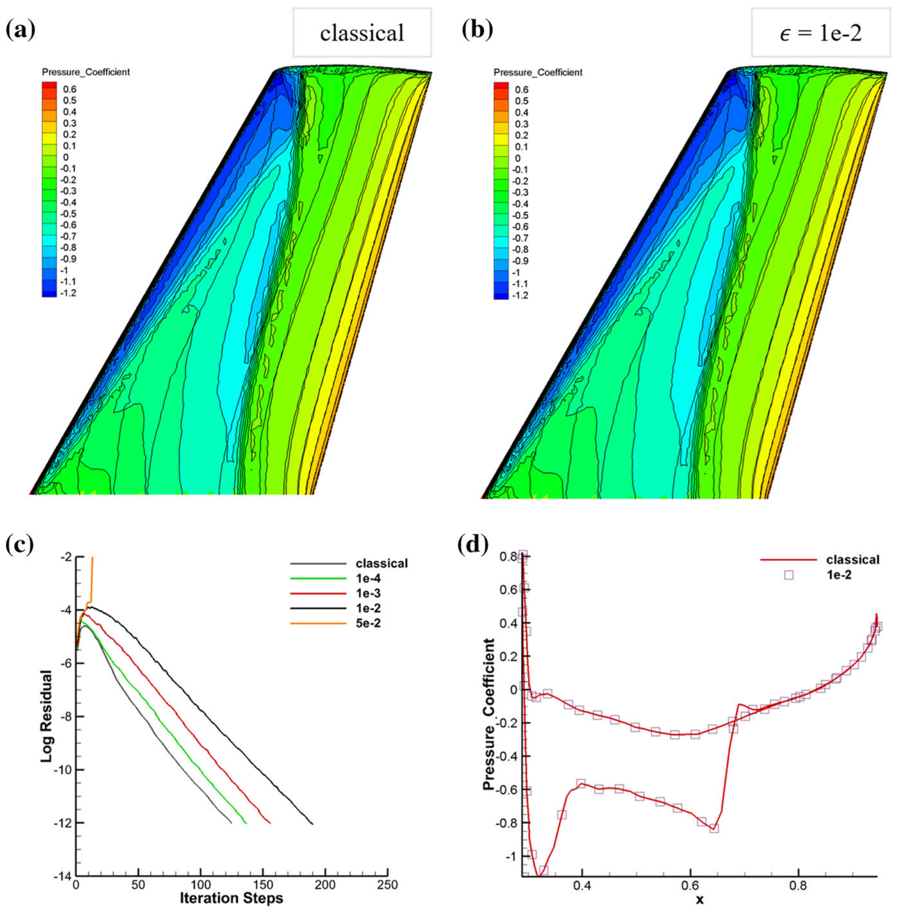


Fig. 2 Results of the numerical experiment. The test case is the three-dimensional inviscid flow around the Onera M6 airfoil. **a** Pressure coefficient around the airfoil. This result is calculated by the original classical solver. **b** The simulation result of the quantum approach. The error is set to 1e-2. The quantum solver also solves the flow field correctly. **c** Convergence history of the test case of the quantum solver with different error settings, compared to the classical solver as the baseline. The error is set from 5e-2 to 1e-4. Except ϵ=5e-2, all other cases converge correctly. The maximum stable error is between 5e-2 and 1e-2. **d** A cross line with y=0.5 solution output by the classical solver and the quantum solver of error 1e-2.

6.2 Inviscid transonic flow

We select the inviscid flow around Onera M6 airfoil as the test case. Onera M6 airfoil is a fundamental test case for a CFD solver. In this case, the problem is a three-dimensional case with 108396 grid points.

We focus on comparison among the classical result with different error settings: from 5e-2 to 1e-4. The definition of error settings is described above. A general view of the convergence history is demonstrated in Fig. 2c. We can find that the computation converges correctly when the error is smaller than 1e-2 and diverges quickly at 5e-2.

The comparison of the flow field is shown in Fig. 2a and b, displayed by the pressure coefficient of the surface flow on the airfoil. The “lambda”-shaped shock wave on the airfoil is computed correctly with both classical and the quantum-accelerated solvers with a certain amount of error. In Fig. 2, we pick a horizontal cross line on half of the airfoil ($y = 0.5d$, where d is the length of the wing), showing that the result of error 1e-2 correctly matches the classical result. We can conclude that the result is correct as long as the computation converges, even with a high quantum error closing to the convergence threshold.

The convergence history of test cases implies an effective band in terms of the quantum error settings. The lower bound is the maximum stable error, defined by the maximum error setting where the computation can converge. The upper bound is $\epsilon^{-2} < N$, defined by the minimum error where the quantum advantage is preserved. The quantum approach will overperform the classical when we set the error on this band.

In this case, the maximum stable error is between 5e-2 and 1e-2, satisfying $N \gg \epsilon^{-2}$, which suggests the existence of quantum advantage in cases like this.

6.3 Turbulent viscous flow

The turbulent flow, in contrast to laminar flow, is a complicated type of flow motion. Due to its chaotic behavior, turbulence is hard to compute directly from the NS equations. Instead, Reynolds-averaged Navier–Stokes (RANS) equations are used to describe the turbulent flow, which provides approximate time-averaged solutions to the Navier–Stokes equations [47].

The finite volume method with the implicit Euler time-stepping scheme is also applied to the RANS equation. Similarly, the system evolves by solving a linear equation. SU2 code also supports setting RANS as the governing equation. We performed experiments again on the Onera M6 airfoil problem where the Reynolds number is set to 11.72E6.

The convergence histories are shown in Fig. 3a look different from those of the above case. In this case, the system cannot keep converging, where a larger error leads to a larger converged residual. We compare the converged results of 1e-2 and 5e-3, which correspond to the largest converged residual and the second largest one among all of our results, as demonstrated in Fig. 3b. The 5e-3 case, which is even converged at a much higher residual than the classical one, closely matches the classical solution. The 1e-2 also matches well, except for $x > 0.8$.

The result also shows that our approach also adapts to the RANS equation. We believe that regardless of the type of the equation, the acceleration of FVM with the quantum approach has broad adaptability. Because the classical algorithms share a similar equation linearization method and implicit time-stepping scheme, using our approach to achieve the quantum speedup is possible if we can construct the linear equation efficiently.

The supersonic compressible flow will generate a shock wave characterized by a discontinuous change in pressure, density, and temperature. This phenomenon is visible by solving NS equations by the FVM. As shown in Fig. 4, we calculate the

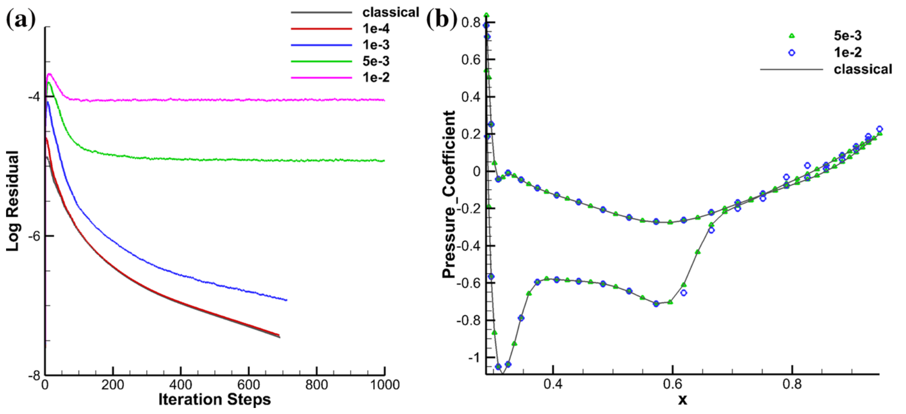


Fig. 3 Results of the supersonic wedge test case. **a** Convergence history of quantum error set from 1e-2 to 1e-4 and the classical case. These cases terminate after reaching convergence conditions or over 1000 steps. **b** The pressure coefficient on the cross line at $y = 0.5d$ of the airfoil where d is the length of the wing. The green triangle and the blue circle are plotted of the cases 1e-2 and 5e-3 correspondingly. The 5e-3 case matches the classical correctly. The 1e-2 case matches the classical at most places but slightly biases at $x > 0.8$.

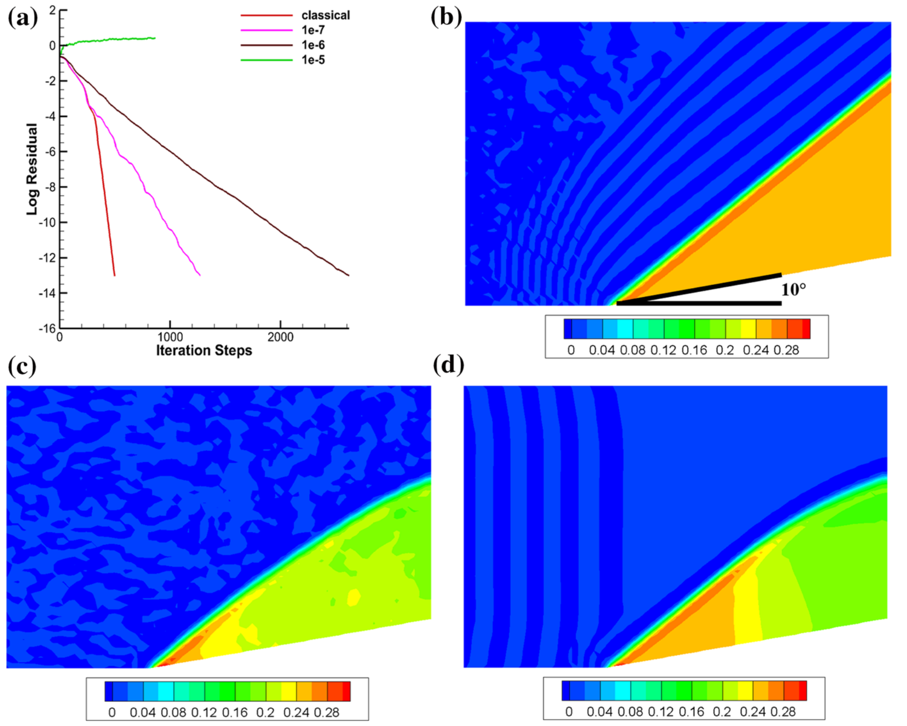


Fig. 4 Results of the supersonic wedge. **a** Convergence history of quantum error set from 1e-5 to 1e-7 and the classical case. **b** The pressure coefficient of this case after convergence, where a shock wave appears at an angle over the wedge (i.e., the sudden change of pressure from the right bottom to left top). **c** The flow field with error 1e-6 after 100 iteration steps. **d** The flow field of the classical case after 100 iteration steps.

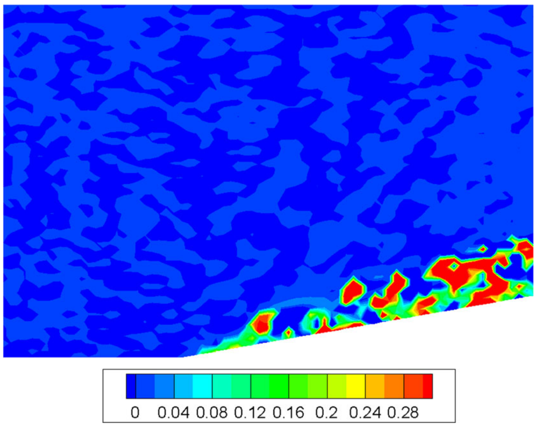


Fig. 5 Result of supersonic wedge with error set to 1e-5 after 100 iteration steps

two-dimensional supersonic flow over a wedge. The initial flow is set horizontal with Mach number 2.0, and the wedge is 10 degrees. After converged to the stable state, the NS equation predicts that there will be a shock with certain angles above the wedge.

6.4 Inviscid supersonic flow

We compared the classical result with error set to 1e-5, 1e-6 and 1e-7, whose convergence histories are displayed in Fig. 4a. In subfigure (b), we show the pressure coefficient of the space with quantum error set to 1e-6. The right bottom area is the wedge. The discontinuous change from blue (low pressure) to orange (high pressure) forms an interface that is consistent with the theoretical prediction.

In this case, we tried to evaluate the effect of quantum error physically. Subfigures (c) and (d) show the flow field after 100 steps of ϵ=1e-6 and the classical case correspondingly. When both cases are at the early stage of computing, a small quantum error does not affect the solver's ability to capture the properties of the fluid. In this test case, the discontinuous interface still appears even when the error is close to the threshold (between 1e-5 and 1e-6). Instead, quantum error generates random fluctuations at the blank space, such as the noise pattern at the left-upper space. This becomes the main reason why quantum error will slow down the convergence. We also plot the field at 100 steps of the 1e-5 case, shown in Fig. 5. The error over the threshold will hinder capturing of the shock wave. These results provide possible optimizations to our proposal, including the following two strategies.

First, use different quantum errors at different stages. The error analysis section mentioned that the actual error has an extra multiplier to the l_2 norm of the residual vector due to the normalization of the quantum state. At the beginning of the computation, the system's residual is large, where a small quantum error can be applied. Thus, the system can successfully capture the character of the flow. When the system is converging and the residual decreases, we can then set a larger quantum error to perform time-stepping rapidly. This strategy could potentially have higher stability and faster convergence.

Second, treat different areas with different errors. The blank space highly contributes to the global error; however, they are not crucial in determining the computation's correctness and stability. For example, if we concentrate on a small area instead of the whole space (such as the airfoil test cases), the quantum error can be set higher on this area and lower at others.

7 Conclusion

This paper developed a quantum approach for accelerating the classical FVM and achieved exponential speedup on the time-stepping. The initial intuition is to use the quantum linear solver to replace the classical one. The main problem is that the quantum linear solver only accepts the quantum form of input and outputs a quantum state. The conversion between the classical and the quantum data is the bottleneck, and our paper mainly studied this. We apply the QRAM to store the interval data. A data structure for this problem is specially designed. The most critical part is that we always maintain a residual sum tree corresponding to the physical variable. On the one hand, this tree can help the efficient preparation of the quantum input. On the other hand, the tree can be efficiently updated throughout the time-stepping of the FVM. As the result, the time complexity of performing a first-order implicit Euler time-stepping is $\mathcal{O}\left(\frac{(s^3+\log N)s\kappa \log^3 N}{\epsilon^2}\text{polylog}(s\kappa/\epsilon)\right)$. The time complexity depends logarithmically on the grid points N, exponentially faster than the best classical case $O(N)$.

The quantum error is additionally introduced. The quantum approach has a quadratic slowdown in terms of quantum error. We analyzed the effect of the error, concluding that a sufficiently small quantum error will not affect the correctness of the quantum-accelerated solver. Numerical experiments are conducted on various test cases to check the correctness of our approach. The error analysis and the numerical results imply the possibility that our approach can be practically applied to large-size CFD problems and demonstrates the quantum advantage.

Our future work will focus on optimizing the error setting on different stages and different computation areas. We believe that the quantum computer will show its advantage in solving a more complex CFD problem shortly.

Acknowledgements This work was supported by the National Natural Science Foundation of China (Grants Nos. 11625419), the National Key Research and Development Program of China (Grant No. 2016YFA0301700), the Strategic Priority Research Program of the Chinese Academy of Sciences (Grant No. XDB24030600), and the Anhui Initiative in Quantum Information Technologies (Grants No. AHY0800000).

Code and data availability The code that can reproduce the results is open source at GitHub. https://github.com/Agony5757/SU2-Quantum

Declarations

Conflict of interest The authors declared that they have no conflicts of interest to this work.

A Finite volume method

The typical physical governing equations (Euler, NS, RANS) have to be linearized to apply to the FVM. In this paper, we do not focus on the detail of the linearization. Instead, we apply the identical linearization method to the classical algorithm and analyze the relationships among the equation variables.

Here we take a two-dimensional NS equation with compressible flow as an example. First write down the differential form of the NS equation:

$$\frac{\partial}{\partial t}\int_{\Omega} U dV + \oint_{\partial\Omega} \boldsymbol{F} \cdot d\boldsymbol{S} = 0, \tag{16}$$

where

$$U = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{bmatrix} \boldsymbol{F}_x = \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho u v \\ \rho u H \end{bmatrix} \boldsymbol{F}_y = \begin{bmatrix} \rho v \\ \rho u v \\ \rho v^2 + p \\ \rho v H \end{bmatrix}, \tag{17}$$

for any volumne Ω and its boundary $\partial\Omega$.

To discretize it spatially and timely, we split the space and time into small grid cells. An example of the space discretization is shown in Fig. 6. At the cell i and time step n, the NS equation can be discretized to

$$\frac{\Omega_i}{\Delta t}\left(U_i^{n+1} - U_i^n\right) = -\sum_{\partial\Omega} \boldsymbol{F}_{i,\partial\Omega}^{n+1} \cdot \Delta \boldsymbol{S}_{i,\partial\Omega}^{n+1}, \tag{18}$$

where the implicit Euler method is applied. We define the right-hand side of Eq. (18) as the residual of this point, denoted by R_i^{n+1}. The $\boldsymbol{F}_i$ is defined by the difference scheme, which is calculated by variables $\boldsymbol{U}$ in the surrounding cells. The difference scheme gives a relation between nodes. In this paper, we define a matrix C which has

$$C_{i,i'} = 1 \tag{19}$$

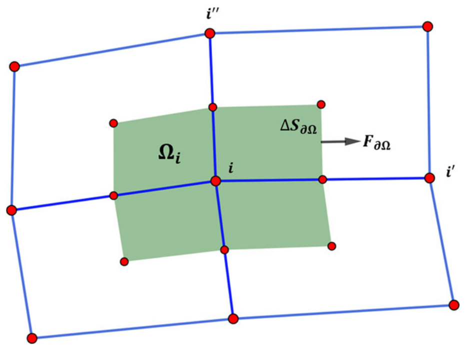


Fig. 6 The grid cell around i^{th} point. $\boldsymbol{F}_{\partial\Omega}$ is the flux at the certain boundary; $\Delta S_{\partial\Omega}$ is the area. Ω_i is the volume of this cell.

if i and i' are related in the difference scheme. In other words, $C_{i,i'} = 1$ means calculating the residual at i^{th} node uses the variables in j^{th} node. Specially, we always have $C_{i,i} = 1$.

Let $\Delta \boldsymbol{U}^{n+1} = \boldsymbol{U}^{n+1} - \boldsymbol{U}^n$, we have

$$\left(\frac{\Omega_i}{\Delta t} \delta_{i,i'} + \left. \frac{\partial R_{i,k}}{\partial U_{i',k'}} \right|_{U=U^n} \right) \Delta U^n_{i',k'} = -R^n_{i,k}. \tag{20}$$

Simply replacing $\boldsymbol{A} = \left(\frac{\Omega_i}{\Delta t} \delta_{i,i'} + \left. \frac{\partial R_{i,k}}{\partial U_{i',k'}} \right|_{U=U^n} \right)$, we obtain a linear equation whose solution implies the time evolution of the physical variable $\boldsymbol{U}$.

The coefficient matrix $\boldsymbol{A}$ is a sparse matrix. From Eq. (18), the $A_{i,k,i',k'}$ is nonzero when i and i' are related in the difference scheme ($C_{i,i'} = 1$). The sparse number (number of nonzero element in a row or column) is fixed by how we select the difference scheme, denoted by s.

Regardless of the physical governing equation, the discretization and the linearization following the classical FVM method do not change. We will finally show that the spatial or time difference scheme does not affect how this algorithm works, and only the constant coefficient will change in the analysis of the time complexity.

B Quantum preliminaries

Quantum computing is a novel computational paradigm. We will briefly introduce some basic ideas of quantum computing in this section, and we suggest the reader refers to [48] for a complete overview of quantum computing.

B.1 Quantum states and register

Quantum states are the data representation in the quantum computer. The quantum registers are the physical container of the quantum data. Similar to the classical registers, several binary digits are used to represent a number. The main difference between quantum computing and classical computing is that quantum data can stay at a superposition state. For example, for a 1-bit register, the classical data stores 0 or 1, while the quantum data can be:

$$\alpha|0\rangle + \beta|1\rangle,$$

where α and β are two complex numbers. When the register is n-bit, the quantum data can store all the superpositions from 0 to $2^n - 1$ (11...1 in binary form) with each case has a complex coefficient. In a mathematical notion, the quantum state is a linear combination of basis vectors in the Hilbert space.

B.2 Quantum unitaries and gates

Quantum unitaries (a.k.a, quantum gates or quantum operations) operate the data in the quantum registers. It has been proved that a universal quantum computer could cover all computations that a classical computer can perform. This implies that any arithmetic operation is available in quantum computers.

Operating quantum states also follow the rules of linear transformations in the Hilbert space, which means the quantum operation applies equally to all superposition states and generates a superposition output. Suppose we have a quantum register stored a superposition of two integers $\alpha|x\rangle + \beta|y\rangle$. After we perform an increment operation on this register, we can obtain $\alpha|x + 1\rangle + \beta|y + 1\rangle$, where all data get increased in parallel.

B.3 Quantum measurements

Quantum measurement is a way to read out the data in the quantum register. When the data are superposition states, the measurement result is random. The probability of one outcome equals the square of the magnitude of the coefficient. For example, when the quantum register stores $\alpha|x\rangle + \beta|y\rangle$, the measurement outcome will be x with $|\alpha|^2$ probability, or y with $|\beta|^2$ probability.

C Quantum algorithms used in this paper

In the following part, we will introduce several quantum algorithms which are used in this paper.

C.1 Amplitude estimation and amplification

Producing a probabilistic result is common in many quantum algorithms. In these algorithms, there will be a workspace and an ancillary register. The answer of the workspace is correct only when the ancillary register is measured to 0. We denote an algorithm as A, so we have:

$$A|0\rangle_{work}|0\rangle_{anc} = \alpha|\psi_{good}\rangle_{work}|0\rangle_{anc} \tag{21}$$

$$+\sqrt{1-|\alpha|^2}|\psi_{bad}\rangle_{work}|1\rangle_{anc}. \tag{22}$$

The probability of producing a good state is $|\alpha|^2$. Amplitude estimation is the algorithm that can estimate this probability, and amplitude amplification can produce the good state at a sufficiently high probability with a known probability.

Theorem 1 *(Amplitude estimation [43]) For any positive integer k, the algorithm* EstAmp(A, χ, M) *outputs* $\tilde{a}$ $0 \le \tilde{a} \le 1$ *such that*

$$|\tilde{a} - a| \leq 2\pi k \frac{\sqrt{a(1-a)}}{M} + k^2 \frac{\pi^2}{M^2}$$

with probability at least $\frac{8}{\pi^2}$ when $k = 1$ and with probability greater than $1 - \frac{1}{2(k-1)}$ for $k \geq 2$. It uses exactly M evaluations of A. If $a = 0$, then $\tilde{a} = 0$ with certainty, and if $a = 1$ and M is even, then $\tilde{a} = 1$ with certainty.

Theorem 2 *(Amplitude amplification [43]) Let A be any quantum algorithm that uses no measurements. There exists a quantum algorithm that, given the initial success probability $a > 0$ of A, finds a good solution with certainty using a number of applications of A and $A^\dagger$ which is in $\Theta(\frac{1}{\sqrt{a}})$ in the worst case.*

These two algorithms take the raw algorithm A as the input. With these two algorithms, we are able to produce the good state (or the target state) with amplified probability by iteratively calling the algorithm $\mathcal{O}(1/\sqrt{a})$ times.

C.2 The l_∞ tomography

Quantum tomography is a set of measurements taken on an unknown quantum state to extract its information to a classical vector. It is required that the classical vector $\tilde{x}$ should be close enough to the quantum state $|x\rangle$. Given a small error bound, we have the following relation:

$$\|x - \tilde{x}\| < \epsilon.$$

In the work of [49], producing a classical vector that is l_2-close to the quantum state requires $\mathcal{O}(N)$ measurements, while N is the dimension of the Hilbert space of the quantum state. Here we will use an exponentially faster version which will produce l_∞-close classical version in logarithm time.

Theorem 3 *(l_∞ tomography [42]) Given U such that $U|0\rangle = |x\rangle = \sum_i x_i |i\rangle$ and its controlled version in time $T(U)$, there exists a tomography algorithm with time complexity $\mathcal{O}(T(U)\frac{\log d}{\delta^2})$ that produces a unit vector $\tilde{x} \in \mathbb{R}^d$ such that $\|\tilde{x} - x\| < (1+\sqrt{2})\delta$ with probability at least $1 - \frac{1}{\mathrm{poly}(d)}$.*

C.3 Quantum linear solver

Using a quantum linear solver (QLS) is the core procedure to distinguish the quantum algorithm from the classical one. The classical solver for an N dimensional matrix and vector, even given in the sparse form, will definitely require $\mathcal{O}(N)$ time complexity. The QLS uses sublinear time complexity, usually given in $\mathcal{O}(\mathrm{polylog}N)$.

In this paper, we reference the version of Andrew Childs et al. [9], which provides the fastest of state-of-the-art dependence on precision. The authors provided two ways to realize a QLS, the Fourier approach, and the Chebyshev approach. With the variable-time amplitude amplification [8] and gapped phase estimation [9], we can build the algorithm which has a linear dependence on the conditional number κ, same as the classical linear solver.

For linear equation $Ax = b$, such three oracles should be defined as the input of the algorithm:

$$O_A|i\rangle|j\rangle \to |i\rangle|j\rangle|A_{i,j}\rangle,$$
$$O_l|i\rangle|j\rangle = |i\rangle|g(i,j)\rangle,$$

and

$$O_b|i\rangle \to |i\rangle|b_i\rangle,$$

where i and j are indexes of the matrix or vector. $g(i, j)$ represents the j^{th} nonzero element in row i. We have the following theorem.

Theorem 4 *(Quantum linear solver with linear dependence on the condition number [9]) Given oracles O_A, O_l, O_b (as stated above), there exists a quantum linear solver whose query complexity is $\mathcal{O}(d\kappa \text{poly}(\log(d\kappa/\epsilon)))$, and gate complexity exceeds the query complexity by a multiplicative factor of $\mathcal{O}(\log N + \log^{2.5}(\kappa d/epsilon)$.*

D Quantum random access memory

Quantum random access memory is the storage device for the quantum computer. As the quantum analog of RAM, QRAM allows a quantum computer to obtain classical data with given addresses in quantum parallel. In other words, QRAM could perform such unitary transformation:

$$U_{\text{QRAM}}|i\rangle_A|0\rangle_D = |i\rangle_A|d_i\rangle_D, \tag{23}$$

where A and D denote the address and the data registers. d_i is a classical data entry stored at the address i.

A seminal architecture called "bucket-brigade" provides an efficient way for querying. There have been many proposed physical implementations of such architecture, such as optical system [27, 28], acoustics system [50], and circuit quantum electrodynamics [51]. Our work is based on the QRAM with architecture implemented by any of the physical systems. To eliminate the difference in understanding the availability of the QRAM, we list all assumptions when we apply the QRAM to our algorithm.

First, the QRAM is general to all input addresses and their superpositions, namely $\sum c_i|i\rangle$. The QRAM should be an arbitrary data loader rather than only allowing to prepare the $\sum |i\rangle|d_i\rangle$ state.

Second, if the address register has been prepared, performing one query costs $\mathcal{O}(\log N)$ time where the full data length is N.

Third, we assume that a QRAM has at least a classical RAM capability, enabling access to a single entry or overlaying it to another value with constant time. Meanwhile, the QRAM should be compatible with a classical computer. A classical computer can read the data in QRAM without extra cost.

Even though a real physical implementation of QRAM is hard, these assumptions are reasonable because they do not exceed the capabilities of the previous physical implementations.

We claim our algorithm as "classically input and output" under the sense that the input and output of the algorithm are stored in the QRAM. Because we believe in the compatibility of QRAM and classical computer, the problem definition, data initialization, and post-processing of the calculation results can all be performed in a connected classical computer.

E Preparation of the residual vector state

According to the method described in [38], the state $|R\rangle$ could be efficiently prepared because we have access to all wanted sums of the vector. We precompute them in the sum tree, so the preparation can be realized by querying the sum tree.

The first step is to query the tree root and its left child node and then calculate the rotating angle at this step:

$$|a_r(0)\rangle|S_R(0)\rangle|a_r\rangle|S_R\rangle|\theta\rangle|0\rangle, \tag{24}$$

where $S_R(0)=\sum_{i\in[0..N/2-1]} R_i^2$, $S_R=\sum_{i\in[0..N-1]} R_i^2$, $\theta=\arccos\frac{S_R(0)}{S_R}$.

Now performing a conditional rotation and uncomputing, we have:

$$\cos\theta|0\rangle+\sin\theta|1\rangle. \tag{25}$$

Add another qubit, perform Hadamard gate on it, we have

$$(\cos\theta|0\rangle+\sin\theta|1\rangle)\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle) \tag{26}$$

$$=\frac{1}{\sqrt{2}}(\cos\theta|00\rangle+\sin\theta|10\rangle) \tag{27}$$

Then we iteratively perform the query, computing the rotating angles and conditional rotation. At k^{th} step, we have the state

$$\sum c_i^k|i\rangle\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle). \tag{28}$$

Computing the addresses of $|i,0\rangle$, $|i,1\rangle$, we obtain the real addresses $a_r(i,0)$ and $a_r(i,1)$ in the QRAM, that is

$$\sum c_i^k|i\rangle\frac{1}{\sqrt{2}}(|0\rangle|a_r(i,0)\rangle+|1\rangle|a_r(i,1)\rangle, \tag{29}$$

then query to the $\mathcal{P}_R$ to obtain the rotating angles and uncompute extra registers. Finally, after performing conditional rotation, we step to

$$\sum c_i^{k+1}|i\rangle \tag{30}$$

Repeatedly performing this process, we can efficiently prepare the residual state $|R\rangle = -\sum R_i|i\rangle/\sqrt{\sum_{j\in[0..N-1]} R_j^2}$ with the help of the sum tree.

F Implementation of quantum preconditioner

The condition number of the linear equation represents to what extent the solution can be affected by the perturbation on the right-hand-side vector. The condition number is defined as:

$$\kappa(\boldsymbol{A}) = \frac{|\lambda_{max}|}{|\lambda_{min}|}, \tag{31}$$

where $|\lambda_{max}|$ and $|\lambda_{min}|$ is the maximum/minimum absolute of the eigenvalues of $\boldsymbol{A}$. When the condition number is large, we say the equation is ill-conditioned, requiring high precision and time complexity to solve. The time complexity of the classical sparse linear solver has a dependency on the condition number. For example, the time complexity of the conjugate gradient method is $\mathcal{O}(\kappa s N \log 1/\epsilon)$. The QLS used in our algorithm also has a linear dependency on the condition number.

Preconditioner is a preprocessing method that can reduce the condition number of the equation. If we have a matrix P such that $\kappa(\boldsymbol{PA}) < \kappa(\boldsymbol{A})$, we can transform this equation as:

$$\boldsymbol{Ax} = \boldsymbol{b} \Leftrightarrow \boldsymbol{PAx} = \boldsymbol{Pb}. \tag{32}$$

Preconditioners are constructed from the original equation, and there have been many types of preconditioners. However, not all classical preconditioners could be directly transplanted to quantum versions. First, the matrix multiplication by the preconditioner should be computed efficiently, namely within $\mathcal{O}(\text{polylog}(N))$ time. Second, the preconditioned matrix should also be sparse; otherwise, it cannot be efficiently solved by the QLS. Some preconditioners suitable for QLS have already been proposed in [39] and [40]. We here display an example preconditioner: the blockwise Jacobi preconditioner, which is widely used in the classical CFD solver. We implement the blockwise Jacobi preconditioner in our algorithm without affecting the asymptotic complexity on the problem size N.

Apply Jacobi preconditioner to subprocedures Jacobi preconditioner uses the inverse of the diagonal block. For the raw linear equation, we construct subprocedures as the input of the QLS. The preconditioned equation has a different matrix and vector; therefore, these subprocedures should be modified.

Let $\tilde{\boldsymbol{A}} = \boldsymbol{PA}$ and $\boldsymbol{R'} = \boldsymbol{PR}$, where $\boldsymbol{P}$ is the Jacobi preconditioner of the matrix $\boldsymbol{A}$. The element of the $\boldsymbol{P}$ is

$$\boldsymbol{P}_{i,k}^{i',k'} = \delta_{i,i'}\boldsymbol{B}_{i,k}^{i',k'}, \tag{33}$$

where $\boldsymbol{B}_{i,\_}^{i',\_}$ represents the inverse of the block $\boldsymbol{A}_{i,\_}^{i',\_}$.

The element of $\tilde{\boldsymbol{A}}$ is

$$\tilde{A}_{i,k}^{i',k'} = \sum_{j\in[0..N-1];l\in[0..n_{\text{var}}-1]} \boldsymbol{P}_{j,l}^{i',k'} \boldsymbol{A}_{i',k'}^{j,l}. \tag{34}$$

The Jacobi preconditioner $\boldsymbol{P}$ is blockwise diagonal. We can simplify the equation (34) as:

$$\tilde{A}_{i,k}^{i',k'} = \sum_{l\in[0..n_{\text{var}}-1]} \boldsymbol{P}_{i,l}^{i,k'} \boldsymbol{A}_{i',k'}^{i,l}. \tag{35}$$

This implies that computing a single element of $\tilde{\boldsymbol{A}}$ requires to queries n_{var} elements of $\boldsymbol{A}$. Another fact is that the sparsity matrices of $\tilde{\boldsymbol{A}}$ and $\boldsymbol{A}$ are the same when they are symmetric to the diagonal line. This is often true because in the difference scheme, i and i' are related so that $A_{i,k}^{i',k'}$ and $A_{i',k}^{i,k'}$ are all nonzero elements.

When it is efficient to implement O_A, O'_A will also be efficient to implement. That is

$$\mathcal{P}'_A|i,k,i',k'\rangle|0\rangle = |i,k,i',k'\rangle|\tilde{A}_{i,k}^{i',k'}\rangle. \tag{36}$$

First we query all elements required for computing the inverse at (i,i') block and compute the inverse, we have

$$|i\rangle|B_{i,\_}^{i,\_}\rangle_{\text{regs(A)}}. \tag{37}$$

We use $B_{i,\_}^{i',\_}$ to represent a matrix block with n_{var}^2 elements. The subscript "regs(A)" mean we require a group of quantum registers to hold this matrix, marked by A.

The corresponding block in $\boldsymbol{A}$ is also queried,

$$|i,i'\rangle|A_{i,\_}^{i',\_}\rangle_{\text{regs(B)}}, \tag{38}$$

Combining two register group A and B, we obtain the wanted element $A_{i,k,i',k'}$. Computing one element requires n_{var}^2 times of calls to the $\mathcal{P}_A$, namely proportional to $\mathcal{O}(s^2)$.

From the above derivation, the sparsity of $\tilde{\boldsymbol{A}}$ is same with the $\boldsymbol{A}$. Therefore, O_l remains unchanged. The subprocedure $\mathcal{P}_b$ should also be modified. In the original description of the data structures, the sum tree stores the precomputed residual vector. In the preconditioned version, the $\boldsymbol{R}$ is replaced by $\boldsymbol{R'}$. When the $\boldsymbol{U}$ changes according to the sampling results, we need to compute the preconditioned residual vector and update the sum tree. Computing any element of $R'_{i,k}$ is still related to all connected cells, which is

$$R'_{i,k} = \sum_{k'\in[0..n_{\text{var}}-1]} \boldsymbol{P}_{i,k'}^{i,k} R_{i,k'} \tag{39}$$

The complexity of this process is also contributed by computing the diagonal block's inverse of $\boldsymbol{A}$. From the preconditioned sum tree, constructing such P'_b is completely same with the original method.

G Relation to previous works

Linear problem is proved suitable for the quantum computer because of the linear properties of the quantum mechanics. Such as quantum algorithms for linear equations, linear differential equations, and linear PDEs. They achieved exponential speedup over their classical counterparts.

Nevertheless, nonlinear problems remain a challenge. In a recent work [36], the researchers propose an efficient algorithm for the dissipative nonlinear differential equation. Besides proposing this algorithm, they also give a theorem that there is no efficient quantum algorithm that can solve a general type of nonlinear differential equation. They use a variable to scale the nonlinearity of the problem and show that the time must scale exponentially to T if using forward time-stepping in general problems. This "no-go theorem" suggests that, even with a quantum computer, solving a nonlinear differential equation like the NS equation is hard and non-efficient.

Now compare the transient problem and the steady problem. In transient problems, we care about the history of evolution and pursue a high precision on time. In steady problems, we only want to know about the given system's stable state regardless of the detailed history of evolution.

In this work, we focus on steady problems. In CFD, the steady problems are the problems whose solution is time-independent. If we set an appropriate initial state, the state will be stable after infinity long evolution time. Because we did not obtain a high-precision time-stepping history, this work does not violate the "no-go theorem" mentioned above and is complementary to previous work about nonlinear differential equations.

References

1. Anderson, J.: Computational Fluid Dynamics?: The Basics with Applications. McGraw-Hill Science Engineering, New York (1995)
2. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM Rev. **41**(2), 303 (1999)
3. Georgescu, I.M., Ashhab, S., Nori, F.: Quantum simulation. Rev. Mod. Phys. **86**(1), 153 (2014)
4. Berry,D.W., Childs, A.M., Cleve,R., Kothari,R., Somma,R.D.: Exponential improvement in precision for simulating sparse Hamiltonians. In: *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing* (ACM, New York, NY, USA, 2014), STOC '14, pp. 283–292. 10.1145/2591796.2591854
5. Berry, D.W., Childs, A.M., Kothari,R.:Hamiltonian simulation with nearly optimal dependence on all parameters, in *2015 IEEE 56th Annual Symposium on Foundations of Computer Science* (IEEE, 2015), pp. 792–809
6. O'Malley, P., Babbush, R., Kivlichan, I., et al.: Scalable quantum simulation of molecular energies. Phys. Rev. X **6**(3), 031007 (2016)
7. 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
8. Ambainis, A.: Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations, Variable time amplitude amplification and a faster quantum algorithm for solving systems of linear equations (2010)
9. 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**(6), 1920 (2017). https://doi.org/10.1137/16M1087072

10. Frank, A., Kunal, A., Babbush, R., et al.: Quantum supremacy using a programmable superconducting processor. Nature **574**(7779), 505 (2019)
11. Gong,M., Wang, S., Zha, C., Chen,M.C., Huang,H.L., Wu,Y., Zhu, Q., Zhao,Y., Li,S., Guo, S.: Quantum walks on a programmable two-dimensional 62-qubit superconducting processor, Science (2021)
12. Pino, J.M., Dreiling, J.M., Figgatt, C., Gaebler, J.P., Moses, S.A., Allman, M.S., Baldwin, C.H., Foss-Feig, M., Hayes, D., Mayer, K., et al.: Demonstration of the trapped-ion quantum CCD computer architecture. Nature **592**(7853), 209–213 (2021). https://doi.org/10.1038/s41586-021-03318-4
13. Gottesman,D.: Stabilizer codes and quantum error correction. Ph.D. thesis, California Institute of Technology (1997)
14. Kitaev, A.Y.: Fault-tolerant quantum computation by anyons. Ann. Phys. **303**(1), 2 (1997)
15. Knill, E., Laflamme, R., Viola, L.: Theory of quantum error correction for general noise. Phys. Rev. Lett. **84**(11), 2525 (2000)
16. Fowler, A.G.: Towards large-scale quantum computation, Physics (2005)
17. F. I.,Note on the Convergence of the Implicit Euler Method, in *International Conference on Numerical Analysis and Its Applications*, vol. 8236 (Springer-Verlag New York, Inc., 2012), vol. 8236
18. Jameson, A., Schmidt,W., Turkel,E.: Numerical solution of the Euler equations by finite volume methods using Runge Kutta time stepping schemes, in *14th fluid and plasma dynamics conference* (1981), p. 1259
19. Versteeg, H.K., Malalasekera, W.: An Introduction to Computational Fluid Dynamics: The Finite Method. Pearson, London (2007)
20. Shewchuk, J.R.: An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Carnegie Mellon University, Pittsburgh (1994)
21. Wang, Y.X., Zhang, L.L., Liu, W., Che, Y.G., Xu, C.F., Wang, Z.H., Zhuang, Y.: Efficient parallel implementation of large scale 3D structured grid CFD applications on the Tianhe-1A supercomputer, Comput. Fluids (2013)
22. Wang, Y.X., Zhang, L.L., Che, Y.G., Xu, C.F., Cheng, X.H.: Efficient parallel computing and performance tuning for multi-block structured grid CFD applications on Tianhe supercomputer. Acta Electron. Sin. **43**(1), 36 (2015)
23. Xu, C., Zhang, L., Deng, X., Fang, J., Wei, L.: Balancing CPU-GPU collaborative high-order CFD simulations on the Tianhe-1A supercomputer. In: *IEEE International Parallel and Distributed Processing Symposium* (2014)
24. Kerenidis, I., Landman, Prakash, A.: Quantum algorithms for deep convolutional neural networks (2019)
25. Chakraborty,S., Gilyén, A., Jeffery, S.: The power of block-encoded matrix powers: improved regression techniques via faster Hamiltonian simulation. In: *In Proceedings of the 46th International Colloquium on Automata, Languages, and Programming (ICALP 2019)* (2018), pp. 33:1–33:14
26. Aaronson, S.: Read the fine print. Nat. Phys. **11**, 291 (2015)
27. Giovannetti, V., Lloyd, S., Maccone, L.: Architectures for a quantum random access memory. Phys. Rev. A **78**(5), 052310 (2008)
28. Giovannetti, V., Lloyd, S., Maccone, L.: Quantum random access memory. Physical Rev. Lett. **100**(16), 160501 (2008)
29. Berry, D.W.: High-order quantum algorithm for solving linear differential equations. J. Phys. A: Math. Theor. **47**(10), 298 (2012)
30. Leyton,S.K., Osborne,T.J.: A quantum algorithm to solve nonlinear differential equations, Physics (2008)
31. Zanger, B., Mendl, C.B., Schulz, M., Schreiber, M.: Quantum algorithms for solving ordinary differential equations via classical integration methods. Quantum **5**, 502 (2021). https://doi.org/10.22331/q-2021-07-13-502
32. Garcìa-Molina, P., Rodrìguez-Mediavilla, J., Garcìa-Ripoll, J.J.: Solving partial differential equations in quantum computers (2021)
33. Lloyd, S., Palma, G.D., Gokler, C., Kiani, B., Palmer, T.: Quantum algorithm for nonlinear differential equations (2020)
34. Arrazola, J.M., Kalajdzievski, T., Weedbrook, C., Lloyd, S.: Quantum algorithm for nonhomogeneous linear partial differential equations. Phys. Rev. A (2019). https://doi.org/10.1103/PhysRevA.100.032306
35. Berry, D.W., Childs, A.M., Ostrander, A., Wang, G.: Quantum algorithm for linear differential equations with exponentially improved dependence on precision. Commun. Math. Phys. **356**(3), 1057 (2017)

36. Liu, J.P., Kolden, H.I., Krovi, H.K. , Loureiro, N.F., Trivisa, K., Childs, A.M.: Efficient quantum algorithm for dissipative nonlinear differential equations (2020)
37. Kerenidis, I., Prakash., A.: Quantum recommendation systems (2016)
38. Grover, L., Rudolph, T.: Creating superpositions that correspond to efficiently integrable probability distributions (2002)
39. Clader, B.D., Jacobs, B.C., Sprouse, C.R.: Preconditioned quantum linear system algorithm. Phys. Rev. Lett. (2013). https://doi.org/10.1103/physrevlett.110.250504
40. Shao, C., Xiang, H.: Quantum circulant preconditioner for a linear system of equations. Phys. Rev. A (2018). https://doi.org/10.1103/physreva.98.062321
41. Tong, Y., An, D., Wiebe, N., Lin, L.: Fast inversion, preconditioned quantum linear system solvers, fast Green's-function computation, and fast evaluation of matrix functions. Phys. Rev. A (2021). https://doi.org/10.1103/physreva.104.032422
42. Iordanis, K., Jonas, L., Prakash, A.: Quantum algorithms for deep convolutional neural networks. In: *International Conference on Learning Representations* (2020)
43. Brassard, G., Høyer, P., Mosca, M., Tapp, A.: Quantum computation and information. Contemp. Math. (2002). https://doi.org/10.1090/conm/305/05215
44. Economon, T.D., Palacios, F., Copeland, S.R., Lukaczyk, T.W., Alonso, J.J.: SU2: an open-source suite for multiphysics simulation and design. AIAA J. **54**(3), 828 (2016). https://doi.org/10.2514/1.J053813
45. Palacios,F., Economon,T., Aranake, A., Copeland, S., Lonkar, A., Lukaczyk, T., Manosalvas-Kjono, D., Naik, K., Padrón, A., Tracey,B., Variyar, A., Alonso, J.: Stanford University Unstructured (SU2): Analysis and Design Technology for Turbulent Flows (2014). https://doi.org/10.2514/6.2014-0243
46. Palacios, F., Colonno, M., Aranake, A., Campos, A., Copeland, S., Economon, T., Lonkar, A., Lukaczyk, T., Taylor, T., Alonso, J.: Stanford University Unstructured (SU 2): an open-source integrated computational environment for multi-physics simulation and design. AIAA J. **2013**, 1 (2013)
47. Osborne, Reynolds,On the Dynamical Theory of Incompressible Viscous Fluids and the Determination of the Criterion, Philosophical Transactions of the Royal Society of London (1895)
48. Nielsen, M.A., Chuang, I.: Quantum computation and quantum information. Am. J. Phys. **70**(5), 558 (2002). https://doi.org/10.1119/1.1463744
49. Kerenidis,I., Prakash,A.: A quantum interior point method for lps and sdps (2018)
50. Hann,C.T., Zou,C.L., Zhang,Y., Chu,Y., Schoelkopf,R.J., Girvin,S.M., Jiang,L.: Hardware-Efficient Quantum Random Access Memory with Hybrid Quantum Acoustic Systems,Physical Review Letters **123**(25) (2019). 10.1103/physrevlett.123.250501. http://dx.doi.org/10.1103/PhysRevLett.123.250501
51. Naik, R., Leung, N., Chakram, S., Groszkowski, P., Lu, Y., Earnest, N., McKay, D., Koch, J., Schuster, D.: Random access quantum information processors using multimode circuit quantum electrodynamics. Nat. Commun. **8**(1), 1904 (2017)