

Block encoding of the three-dimensional heterogeneous Poisson equation with application to fracture flow

Austin Pechan,^{1,*} John Golden^{2,†} and Daniel O'Malley³

¹Theoretical Division, *Los Alamos National Laboratory*, Los Alamos, New Mexico 87545, USA

²CCS-3, *Los Alamos National Laboratory*, Los Alamos, New Mexico 87545, USA

³EES-16, *Los Alamos National Laboratory*, Los Alamos, New Mexico 87545, USA



(Received 12 August 2025; revised 16 December 2025; accepted 3 March 2026; published 15 April 2026)

Quantum linear system (QLS) algorithms offer the potential to solve large-scale linear systems exponentially faster than classical methods. However, applying QLS algorithms to real-world problems remains challenging due to issues such as state preparation, data loading, and efficient information extraction. In this work, we study the feasibility of applying QLS algorithms to solve discretized three-dimensional (3D) heterogeneous Poisson equations, with specific examples relating to groundwater flow through geologic fracture networks. We explicitly construct a block encoding for the 3D heterogeneous Poisson matrix by leveraging the sparse local structure of the discretized operator. While classical solvers benefit from preconditioning, we show that block encoding the system matrix and preconditioner separately does not improve the effective condition number that dominates the QLS run-time. This differs from classical approaches where the preconditioner and the system matrix can often be implemented independently. Nevertheless, due to the structure of the problem in three dimensions, the quantum algorithm achieves a run-time of $O(N^{2/3} \text{polylog } N \cdot \log(1/\epsilon))$, outperforming the best classical methods (with run times of $O(N \log N \cdot \log(1/\epsilon))$) and offering exponential memory savings. These results highlight both the promise and limitations of QLS algorithms for practical scientific computing, and point to effective condition-number reduction as a key barrier in achieving quantum advantages.

DOI: [10.1103/mx5c-8vqk](https://doi.org/10.1103/mx5c-8vqk)

I. INTRODUCTION

The three-dimensional (3D) heterogeneous Poisson equation is a foundational model in science and engineering. This partial differential equation (PDE) describes steady-state diffusion processes in media with spatially varying properties. Applications range from modeling fluid dynamics in porous media and heat transfer in inhomogeneous materials to simulating electrical currents in biological tissues. While the underlying physics is often well understood, solving these equations becomes computationally challenging when the medium's properties, such as permeability or conductivity, vary across many length scales.

For classical computers, the primary bottleneck in solving these problems is memory. Accurately capturing the

system's behavior requires a numerical discretization fine enough to resolve the small-scale variations in material properties. For large domains, this leads to linear systems of equations that far exceed the memory capacity of classical supercomputers. Standard simplification techniques such as coarse-graining or homogenization, which average over fine-scale details, often fail because they cannot capture critical long-range effects, such as percolation, that depend on the full, multiscale structure of the medium. Consequently, many important physical systems remain beyond the reach of full-resolution classical simulation.

Quantum linear system (QLS) algorithms offer a potential path forward. Recent algorithms have achieved a run-time complexity of $O(\kappa \log(1/\epsilon))$, which is information-theoretically optimal in the effective condition number κ and error tolerance ϵ [1,2]. This scaling suggests an exponential advantage over classical methods for certain classes of problems. However, this theoretical promise is contingent on satisfying a strict set of requirements related to data loading, state preparation, and information extraction. This paper moves beyond theoretical potential to rigorously assess the circumstances under which QLS algorithms could deliver a practical quantum advantage,

*Contact author: agp@berkeley.edu

†Contact author: golden@lanl.gov

Published by the American Physical Society under the terms of the [Creative Commons Attribution 4.0 International](https://creativecommons.org/licenses/by/4.0/) license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

particularly in the case of the 3D heterogeneous Poisson equation. Our key contributions are as follows.

- (1) An explicit description of a block encoding of the 3D heterogeneous Poisson matrix (discretization of $\nabla \cdot (\mathbf{k}(\mathbf{r}) \nabla h(\mathbf{r}))$).
- (2) A proof that separately block encoding a matrix and its preconditioner can never improve the effective condition number of the system, and thus never improve the QLS run-time.
- (3) A modest quantum speedup of $O(N^{2/3} \text{polylog } N \cdot \log(1/\epsilon))$, in comparison to classical algorithms that scale as $O(N \log N \cdot \log(1/\epsilon))$, for solving discretized 3D heterogeneous Poisson equations.
- (4) An explicit example of our algorithm as applied to the problem of liquid flowing through geologic fracture networks, systematically addressing state preparation, data loading, condition number, and information extraction for this real-world problem.

While many of our techniques are applicable to a wide range of problems, a fundamental goal of this work is to demonstrate that applying QLS algorithms to real-world problems requires not only deep expertise in quantum algorithms but also significant domain-specific knowledge. On the one hand, domain experts with limited exposure to quantum computing may overestimate how easily QLS algorithms can be applied to their problems, unaware of the inherent quantum limitations. On the other hand, quantum computing specialists may be overly pessimistic, underestimating how small adjustments to real-world problems can bypass these quantum-specific challenges while still delivering meaningful results. These domain-specific features not only influence the design and implementation of the relevant quantum algorithms, but also play a critical role in determining their feasibility and potential to outperform classical methods. Close collaboration between domain and quantum experts is therefore essential to accurately assess the true potential and practicality of QLS algorithms for solving real-world problems.

Several prior works have studied quantum algorithms for solving linear systems, particularly in the context of preconditioning and discretized PDEs. Tong *et al.* [3] introduced a framework for improving the condition-number dependence of quantum linear-system solvers from $O(\kappa^2)$ to $O(\kappa)$, alongside techniques for Green's function computation and matrix-function evaluation. While this was an important conceptual advance, subsequent work has established algorithms that achieve optimal $O(\kappa \log(1/\epsilon))$ scaling under various assumptions. In particular, Costa *et al.* [1] proved such optimal scaling based on a discrete adiabatic theorem, which applies when the solution norm $\|x\|$ is known or can be efficiently estimated. Later numerical analysis by Costa *et al.* [4] showed that in practical settings the effective constant factor in that

complexity bound is over $1200\times$ smaller than the analytic upper bound reported in [1]. In addition, Low and Su [5] created a solver that reduces the number of oracle calls for state preparation, but in our case (since we assume the cost of state preparation is small) this advance is not very helpful. Dalzell [2] further simplified the discrete adiabatic approach and provided explicit constants in the query complexity, assuming the solution norm is available. We adopt Dalzell's formulation as the basis for our qubit and gate-count estimates in Sec. VII.

Other recent efforts have explored more structured problem classes, achieving potential quantum speedups. For example, Deiml and Peterseim [6] proposed a quantum framework for the finite element method that shows promising performance, though its asymptotic behavior in higher dimensions needs further investigation. In addition, Orsucci and Dunjko [7] showed that under certain structural and oracle assumptions (such as access to a Cholesky-like factorization), one can reduce the condition-number dependence from $O(\kappa)$ to $O(\sqrt{\kappa})$. Lapworth and Sunderhauf [8] also analyzed a number of different preconditioning techniques for implementing sparse approximate inverse (SPAI), Toeplitz, and circulant preconditioners on a quantum computer. They showed empirical results that separately block encoding the preconditioner and system matrix is ineffective, and found that none of their techniques reduced the asymptotic scaling of the effective condition number. Our proof transforms this empirical result into a rigorous mathematical result.

II. DISCRETIZED HETEROGENEOUS ELLIPTIC PDES

A foundational class of problems in science and engineering involves modeling steady-state diffusion processes in heterogeneous media. These are described by 3D heterogeneous elliptic PDEs. A general form of this equation is

$$\nabla \cdot (\mathbf{k}(\mathbf{r}) \nabla h(\mathbf{r})) = f(\mathbf{r}), \quad (1)$$

where $h(\mathbf{r})$ is the field of interest (e.g., temperature, electrical potential, or pressure), $\mathbf{k}(\mathbf{r})$ is a spatially varying coefficient representing a material property (e.g., thermal conductivity, electrical conductivity, or permeability), and $f(\mathbf{r})$ is a source term. This equation models a steady-state condition where the flux of a quantity into any region is balanced by the flux out, plus any sources or sinks within the region.

As with most PDEs, Eq. (1) is often discretized using methods such as finite differences, turning it into a large, sparse system of linear equations, which we will denote

$$Gh = f. \quad (2)$$

If the domain is discretized into N cells, then G is an $N \times N$ matrix that encodes the geometry of the domain and the material property k , while h and f are vectors of length N that encode the field values and source terms at each cell, respectively.

For a concrete example, consider a domain discretized into a $N^{1/3} \times N^{1/3} \times N^{1/3}$ 3D grid of cubic cells. If the total domain has side length L , the cells will be of size $(\Delta x)^3 = L^3/N$. The flux from cell $(i, j, k-1)$ into an adjacent cell (i, j, k) is proportional to the difference in the field h between them and the material property k on the interface between them:

$$(k \nabla h)_{i,j,(k-1) \rightarrow k} = k_{i,j,k-1}^{i,j,k-1} \frac{h_{i,j,k-1} - h_{i,j,k}}{\Delta x}. \quad (3)$$

Here, $k_{i,j,k-1}^{i,j,k-1}$ denotes the material property on the interface, which is often calculated as the geometric or harmonic mean of the property values in the adjacent cells. Summing the flows from all six neighbors for a central cell (i, j, k) (as visualized in Fig. 1) yields the discrete version of Eq. (1). The formula for $f_{i,j,k}$ is as follows:

$$\begin{aligned} & \frac{k_{i,j,k}^{i-1,j,k} (h_{i-1,j,k} - h_{i,j,k})}{(\Delta x)^2} + \frac{k_{i,j,k}^{i,j,k-1} (h_{i,j,k-1} - h_{i,j,k})}{(\Delta x)^2} \\ & + \frac{k_{i,j,k}^{i,j,k} (h_{i+1,j,k} - h_{i,j,k})}{(\Delta x)^2} + \frac{k_{i,j,k}^{i,j,k+1} (h_{i,j,k+1} - h_{i,j,k})}{(\Delta x)^2} \\ & + \frac{k_{i,j,k}^{i-1,j,k} (h_{i-1,j,k} - h_{i,j,k})}{(\Delta x)^2} + \frac{k_{i,j,k}^{i,j,k-1} (h_{i,j,k-1} - h_{i,j,k})}{(\Delta x)^2} \\ & + \frac{k_{i,j,k}^{i,j,k} (h_{i+1,j,k} - h_{i,j,k})}{(\Delta x)^2} + \frac{k_{i,j,k}^{i,j,k+1} (h_{i,j,k+1} - h_{i,j,k})}{(\Delta x)^2}. \end{aligned} \quad (4)$$

The three positive terms represent inflow from the back ($i-1$), left ($j-1$), and bottom ($k-1$) neighbors, and the three negative terms represent outflow to the front ($i+1$), right ($j+1$), and top ($k+1$) neighbors.

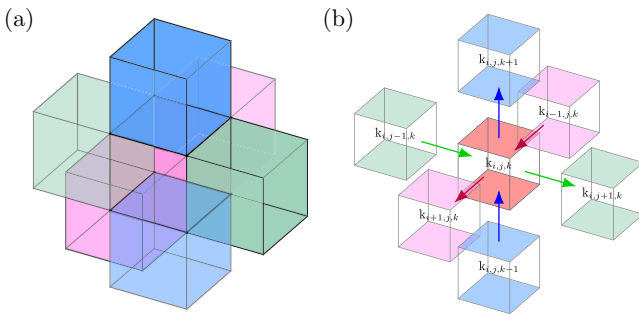


FIG. 1. (a) A 3D spatial grid representing a discretized domain of interest. (b) A visualization of a discretized heterogeneous elliptic PDE defined on the same 3D grid, where the cells are separated for ease of viewing. Each cell is given a spatially varying coefficient $k_{i,j,k}$ (e.g., diffusivity, conductivity, or permeability) and a source term $f_{i,j,k}$. The goal is to determine the field $h_{i,j,k}$ that governs the PDE at each grid point.

Following a row-by-row method for encoding the $N^{1/3} \times N^{1/3} \times N^{1/3}$ 3D grid into a vector of length N gives

$$\begin{array}{cc} \text{Layer 1 } (i=0) & \text{Layer 2 } (i=1) \\ \begin{array}{|c|c|} \hline f_{0,0,0} & f_{0,1,0} \\ \hline f_{0,0,1} & f_{0,1,1} \\ \hline \end{array} & \begin{array}{|c|c|} \hline f_{1,0,0} & f_{1,1,0} \\ \hline f_{1,0,1} & f_{1,1,1} \\ \hline \end{array} = \begin{pmatrix} f_{0,0,0} \\ f_{0,0,1} \\ f_{0,1,0} \\ f_{0,1,1} \\ f_{1,0,0} \\ f_{1,0,1} \\ f_{1,1,0} \\ f_{1,1,1} \end{pmatrix} \end{array}$$

This implies that $f_{i,j,k} = f_{k+jN^{1/3}+iN^{2/3}}$. If we define $a \equiv k + jN^{1/3} + iN^{2/3}$, then Eq. (4) can be rearranged such that we get the following formula for f_a :

$$\begin{aligned} & h_{a-N^{2/3}} \frac{k_{i,j,k}^{i-1,j,k}}{(\Delta x)^2} + h_{a-N^{1/3}} \frac{k_{i,j,k}^{i,j,k-1}}{(\Delta x)^2} + h_{a-1} \frac{k_{i,j,k}^{i,j,k}}{(\Delta x)^2} \\ & + h_a \left(-\frac{k_{i,j,k}^{i-1,j,k} + k_{i+1,j,k}^{i,j,k}}{(\Delta x)^2} - \frac{k_{i,j,k}^{i,j,k-1} + k_{i,j,k+1}^{i,j,k}}{(\Delta x)^2} \right) \\ & + h_{a+1} \frac{k_{i,j,k+1}^{i,j,k}}{(\Delta x)^2} + h_{a+N^{1/3}} \frac{k_{i,j,k+1}^{i,j,k}}{(\Delta x)^2} + h_{a+N^{2/3}} \frac{k_{i+1,j,k}^{i,j,k}}{(\Delta x)^2}. \end{aligned}$$

Combining the linear equations of this form from each cell produces a system of equations defined by the matrix G . G is given by the following formula, where we removed a multiplicative factor of $(\Delta x)^{-2}$ from each condition:

$$G_{a,b} = \begin{cases} -k_{i,j,k}^{i-1,j,k}, & b = a - N^{2/3}, \\ -k_{i,j,k}^{i,j,k-1}, & b = a - N^{1/3}, \\ -k_{i,j,k}^{i,j,k}, & b = a - 1, \\ \left(k_{i,j,k}^{i-1,j,k} + k_{i,j,k}^{i,j,k-1} + k_{i,j,k}^{i,j,k} \right. \\ \quad \left. + k_{i+1,j,k}^{i,j,k} + k_{i,j,k+1}^{i,j,k} + k_{i,j,k+1}^{i,j,k} \right), & b = a, \\ -k_{i,j,k+1}^{i,j,k}, & b = a + 1, \\ -k_{i,j,k+1}^{i,j,k}, & b = a + N^{1/3}, \\ -k_{i+1,j,k}^{i,j,k}, & b = a + N^{2/3}, \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

where we have added an overall negative sign to make G positive definite. Note that this formula is only valid for $1 < i, j, k < N^{1/3}$, for terms on the boundary of the grid additional elements of $G_{a,b}$ will also be 0. In many physical systems, Dirichlet boundary conditions are imposed, which involves setting certain rows of G to be 1 on the

diagonal and 0 elsewhere, with corresponding adjustments to f . The resulting matrix G is sparse, Hermitian, and positive definite. Classically, systems of this type are often solved using iterative methods such as the conjugate gradient (CG) method. The CG method scales with a run-time of $O(N\sqrt{K(G)}\log(1/\epsilon))$, where $K(G)$ is the condition number of G . In practice, however, the convergence rate of the CG algorithm with appropriate preconditioning is $O(N)$ or $O(N \log N \cdot \log(1/\epsilon))$ [9].

Even with these advanced classical techniques, state-of-the-art methods struggle to model real-world domains with more than 3 or 4 orders of magnitude of resolution [10]. This is generally because of memory rather than speed constraints. Coarse-graining is often effective at skirting such memory demands by not considering all length scales at once. However, this approach fails for problems where there are complex interactions the span many different length scales. A key motivating example is modeling the flow of underground fluids (e.g., water, oil) through porous and fractured media. In this case, the governing PDE is known as the groundwater flow equation, where h is the hydraulic head (a measure of fluid pressure), k is the permeability of the medium, and f represents fluid sources or sinks. For these problems, critical physical properties such as percolation depend on simultaneous interactions across fractures of all length scales [11]. Capturing this behavior can require modeling kilometer-sized domains with fractures down to 1 cm in scale. The resulting matrix would contain $O(10^{15})$ cells (and nonzero entries) in three dimensions, requiring $O(100)$ petabytes of memory, beyond the limits of modern computing systems. Therefore, accurate modeling of these systems requires fine-scale resolution beyond the reaches of classical high-performance computing.

III. SOLVING DISCRETIZED POISSON PDES WITH QLS

The first QLS algorithm, by Harrow, Hassidim, and Lloyd [12], solves the system $Ax = b$ by preparing a state $|x\rangle = |A^{-1}b\rangle$ to accuracy ϵ in time $O(\kappa^2 \log N / \epsilon)$ where $A \in \mathbb{R}^{N \times N}$ (generally $A \in \mathbb{C}^{N \times N}$, but throughout this paper we are only considering real system matrices) and has effective condition number κ . While this provides a theoretical exponential improvement over naive classical methods (which scale as $O(N^3)$), the algorithm relies heavily on quantum phase estimation and amplitude amplification techniques, making it complex to implement in practice.

More recent advancements have improved both the complexity and practicality of QLS algorithms. Notably, Costa *et al.* [1] introduced a quantum algorithm that builds on qubitization methods and employs a discrete adiabatic theorem, achieving a complexity of $O(\kappa \log(1/\epsilon))$ and saturating known lower bounds in both κ and ϵ . This method avoids the need for variable-time amplitude

amplification or truncated Dyson series, which were common in previous methods, making it more efficient and easier to implement. More recently, Dalzell [2] created a QLS algorithm that eliminates the need for ansatz state preparation, as used in Costa *et al.* [1]. This makes it even easier to analyze/implement and also saturates the known lower bound, now with improved constant factors. Despite these advances, several significant hurdles must be cleared before this (or any future) QLS algorithm can be employed to actually solve a linear system.

A. Efficient state preparation of $|b\rangle$

The QLS algorithm requires the source vector b to be prepared as a quantum state, $|b\rangle$. This preparation must be efficient, meaning its cost should not dominate the overall run-time. For many physical problems, including our groundwater flow example, the source terms f are often sparse, representing localized inputs such as injection or extraction wells, and can be encoded efficiently [13].

B. Efficient block encoding of the matrix A

State-of-the-art QLS algorithms require a block encoding U_A , which is often highly nontrivial. Fortunately, matrices arising from the discretization of local PDEs via finite difference methods are highly structured and sparse, which often permits an efficient block encoding, as we describe in Sec. IV.

C. The condition number κ must be manageable

The run-time of all known QLS algorithms depends at least linearly on the effective condition number, κ . For 3D Poisson-type problems, κ scales polynomially with the number of grid cells N (typically as $O(N^{2/3})$). This scaling can severely degrade the quantum speedup, potentially negating any exponential advantage over classical methods. While classical solvers rely on preconditioning to reduce κ , applying preconditioners in a quantum context is a significant challenge, as we discuss in Sec. V A.

D. Efficient extraction of a useful answer

A QLS algorithm produces a quantum state $|x\rangle$ proportional to the solution vector x . Reconstructing the full classical vector x from this state would require a number of measurements that scales with N , eliminating any quantum advantage. Therefore, the desired solution must be a specific, efficiently extractable property of $|x\rangle$. In physical problems, we are often interested in local observables or global properties, which can be estimated efficiently from $|x\rangle$ using a small number of measurements. This is explored further in Sec. VI.

The following sections address these requirements in detail, using the groundwater flow problem as a concrete

test case to assess the viability and remaining challenges of a quantum solution.

IV. BLOCK ENCODING

In this section we present a general block-encoding strategy for the discrete 3D heterogeneous Poisson system, extending the structured-oracle framework of [14] from the two-dimensional (2D) Laplacian to fully 3D problems. Given any coefficient field that can be specified by an arithmetic rule or lookup table with $D = O(\text{polylog } N)$ distinct values, the block encoding has cost $O(\text{polylog } N)$. This structural assumption plausibly holds for a wide class of diffusion, conductivity and elasticity models, and we discuss its implementation in detail for the groundwater flow problem.

A block encoding of a matrix A is a unitary matrix U_A that contains A/α as a submatrix in its upper-left corner:

$$U_A = \begin{pmatrix} A/\alpha & * \\ * & * \end{pmatrix}, \quad (6)$$

where $\alpha \geq \|A\|$ (throughout this paper we use $\|\cdot\|$ to denote the spectral norm) is required to ensure that the scaled matrix A/α has spectral norm at most 1. The asterisks $*$ represent arbitrary entries (of arbitrary sizes) needed to satisfy the unitarity of U_A .

To construct a block encoding of the discretized Poisson operator G , we first rescale $G \rightarrow G'$ where $\|G'\| \leq 1$. Applying the Gershgorin circle theorem, we can bound the spectral norm by

$$\|G\| \leq 12k_{\max}/(\Delta x)^2 = 12k_{\max} \cdot \frac{N^{2/3}}{L^2}, \quad (7)$$

where $k_{\max}$ is the largest element of k . This value comes from a row in G corresponding to a cell with $k_{\max}$ surrounded by other cells with $k_{\max}$. For most physical systems governed by elliptic PDEs, $k_{\max}$ is a constant that depends on the intrinsic material property of the domain (diffusivity, permeability, conductivity, etc.), and is independent of how the PDE is discretized; that is, independent of the mesh resolution (i.e., the total number of grid cells N).

In the groundwater flow problem, for example, fracture permeability is often modeled as proportional to the length of the fracture to some power β , where β depends on the type of fractured material [15]. We assume that the largest fracture in R is contained entirely within R , that is, it has length $\leq L$, and so $k_{\max} \leq L^\beta$. More generally, the size of the largest fracture will be limited (e.g., by the size of the Earth), making $k_{\max}$ a constant.

We can then define a constant $a_{\max} = 12L^{-2}k_{\max}$ and normalize G as $G' = G/(a_{\max} \cdot N^{2/3})$. This gives

$$\|G'\| = \|G/(a_{\max} \cdot N^{2/3})\| \leq 1 \quad (8)$$

as required.

Structurally, G closely resembles the discrete Laplacian operator. In particular, Eq. (5) reduces to the Laplacian Δ in the case of constant k , where we have (modulo constant factors)

$$\Delta_{a,b} = \begin{cases} -1, & b = a \pm N^{2/3}, b = a \pm N^{1/3}, b = a \pm 1, \\ 6, & b = a, \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

Recently, an exact block encoding of the 2D Laplacian was introduced by Sunderhauf *et al.* [14], which, with some modifications, can be used to block encode the 3D Laplacian. We will not explicitly do it in this paper as it is a special case of the matrix G' , and thus follows from our block encoding for G' in Appendix E. Sunderhauf's algorithm is designed for matrices A that contain structured data with relatively few distinct nonzero elements, e.g., Toeplitz or binary tree matrices. The primary components of this block encoding are a data-loading oracle, which encodes matrix values, and structure oracles, which specify the positions of each nonzero entry. The data-loading oracle has the form

$$O_{\text{data}} = \sum_{d=0}^{D-1} R_X (2 \arccos G'_d) \otimes |d\rangle \langle d|, \quad (10)$$

where D is the total number of distinct entries in G' . This data are structured using transposition and column oracles O_t and O_c , which are defined as

$$O_t |d\rangle |m\rangle = |d\rangle |m'\rangle, \quad O_c |d\rangle |m\rangle = |j\rangle |s_c\rangle. \quad (11)$$

The transposition oracle allows us to map the m th occurrence of the d th distinct value to its symmetric counterpart m' , related to m by transposition. In addition, the column oracle returns the column index j and sparsity label s_c that corresponds to the m th appearance of the d th distinct element. Since the 3D heterogeneous Poisson system matrix is sufficiently structured, such oracles can be efficiently implemented on a quantum computer. In Appendix E, we provide an explicit construction of G' and prove it has gate complexity $O(\log N + D \log D)$. Note that neither this nor our final QLS run-time has any dependence on the matrix sparsity. In Appendix E, we show that the maximum sparsity of any row/column is bounded by 7, so this reliance on sparsity is removed when we are considering asymptotic notation.

To bound D , recall that the nonzero terms in Eq. (5) only depend on the k values for adjacent cells. Suppose the domain is discretized such that there are at most F distinct values that k can take across all cells. Then, using geometric or harmonic means, the number of possible interface values (i.e., the values assigned to the permeability at each face between neighboring cells) is at most F^2 , corresponding to all possible pairwise combinations of adjacent cell coefficients.

Since the discrete 3D Poisson matrix includes up to six off diagonal entries per row (corresponding to the six face-adjacent neighbors), and each such term is defined by one of the F^2 interface values, the total number of distinct patterns of nonzero entries per row is upper-bounded by the number of 6-tuples over F^2 values. Therefore, the number of distinct nonzero values in G satisfies

$$D = O((F^2)^6) = O(F^{12}). \quad (12)$$

This provides an upper bound on the number of unique matrix entries that need to be block encoded.

As an example, in Appendix B we show that for groundwater flow, F scales as $\text{polylog}(N)$ for fracture patterns with arithmetic descriptions. As an example of typical values for F , (Greer *et al.* '22) found that in a real-world study from the Topopah Spring Aquifer in Nevada, fracture sizes range from 1.5 to 125 m in a 250 by 250 by 100 m domain.

V. EFFECTIVE CONDITION NUMBER

State-of-the-art QLS algorithms [1,2] require the system matrix A to be rescaled to a matrix $A' = A/\alpha_A$ such that $\alpha_A \geq \|A\|$. The effective condition number is then $\kappa = \|(A')^{-1}\| = \alpha_A \cdot \|A^{-1}\| \geq K(A)$, where $K(A) = \|A\| \cdot \|A^{-1}\|$ is the condition number of A . In the next subsection, we examine the implications of this rescaling.

In Sec. IV we showed that $G' = G/O(N^{2/3})$, so the effective condition number is

$$\kappa = \|(G')^{-1}\| = O(N^{2/3}) \cdot \|G^{-1}\|.$$

To upper bound $\|G^{-1}\|$, we can use the discrete Poincaré inequality to get $\|G^{-1}\| = O(1)$, as shown in Appendix A. This gives us that κ scales as $O(N^{2/3})$. An example of this scaling for groundwater flow can be seen in Fig. 2. Combined with the other findings in this paper, we get the quantum run-time of $O(N^{2/3} \text{polylog } N \cdot \log(1/\epsilon))$, which is a speedup over classical approaches that scale as $O(N \log N \cdot \log(1/\epsilon))$. The hope is to further reduce κ (and thus reduce the complexity of running QLS algorithms), but this is challenging. The next subsection explains some challenges with reducing the effective condition number.

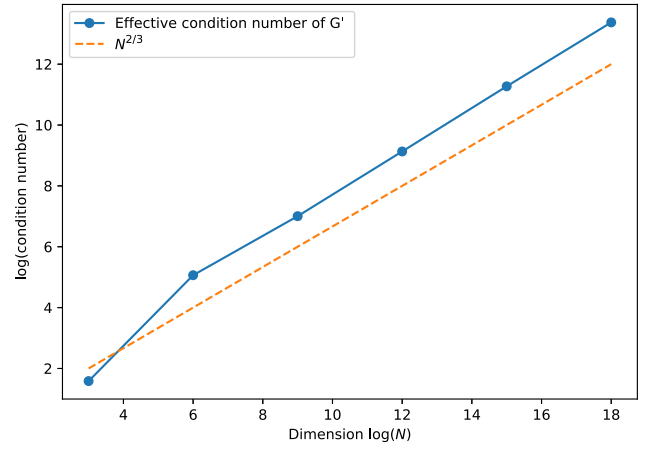


FIG. 2. Empirical scaling of the condition number (equivalent to the effective condition number of the system prior to preconditioning) for solving the groundwater flow equation. The simulation uses the 3D pitchfork fracture network described in Fig. 5. As the dimension N increases, the condition number scales as $\kappa = O(N^{2/3})$, consistent with the theoretical analysis in the beginning of Sec. V.

A. Reducing the effective condition number

While current QLS algorithms theoretically achieve optimal scaling with respect to the effective condition number κ , practical applications hinge on our ability to reduce κ efficiently. In this section, we show that separately block encoding preconditioners and their system matrix, even with singular value amplification, does not work to achieve asymptotic quantum speedups. As a result, more advanced techniques are needed to further improve the scaling of QLS algorithms for solving 3D heterogeneous Poisson equations.

To start, preconditioning is a well-established technique in the context of classical linear-system solvers, where the primary objective is to reduce the condition number of the system, thereby improving the convergence rate of iterative methods. When the condition number of the system matrix A scales poorly with the problem size N , a preconditioner matrix M can be introduced such that the condition number of the product MA exhibits more favorable scaling with N . Instead of directly solving the original system $Ax = b$, the preconditioned system $MAx = Mb$ is solved, where the improved condition number leads to faster convergence.

Block encoding techniques provide a natural and efficient way to implement preconditioners for QLS. In the block encoding framework, given block encodings U_M for the preconditioner M and U_A for the system matrix A , their product $U_M U_A$ directly yields a block encoding of the matrix product MA . As shown in Eq. (6), this product will result in a normalization factor of $\alpha_A \cdot \alpha_M$. The total cost of constructing this block encoding is then simply the sum of the costs associated with U_M and U_A .

As an example, in a previous work [11], we argued that in two dimensions the inverse Laplacian Δ^{-1} is an effective preconditioner for the groundwater flow matrix G , and that it reduces the scaling of the condition number from $O(N)$ to approximately $O(\sqrt{N})$ for multiscale fractal fracture networks. We show in Appendix C that Δ^{-1} can be implemented in $O(\log N)$ time.

Despite having good preconditioners that are easy to block encode, block encoding A and M separately does not enable QLS to benefit from the improved condition number. The effective condition number of the system resulting from separately block encoding the preconditioner and system matrix (i.e., $U_M U_A = U_{MA}$) is $\kappa = \alpha_M \cdot \alpha_A \cdot \|A^{-1} M^{-1}\|$, and QLS does not benefit from the smaller condition number $K(MA)$.

To start, recall that the spectral norm is submultiplicative. This means that for any matrices B and C , $\|BC\| \leq \|B\| \cdot \|C\|$. Using this, we get that for any invertible matrices B and C that

$$\|B^{-1} C^{-1}\| \cdot \|C\| \geq \|B^{-1}\| \iff \|B^{-1} C^{-1}\| \geq \frac{\|B^{-1}\|}{\|C\|}. \quad (13)$$

Now, combining this bound along with the fact that $\alpha_M \geq \|M\|$ and $\alpha_A \geq \|A\|$, we get a lower bound on κ :

$$\begin{aligned} \kappa &= \alpha_M \cdot \alpha_A \cdot \|A^{-1} M^{-1}\| \\ &\geq \|M\| \cdot \alpha_A \cdot \|A^{-1} M^{-1}\| \\ &\geq \|M\| \cdot \alpha_A \cdot \frac{\|A^{-1}\|}{\|M\|} \\ &= \alpha_A \cdot \|A^{-1}\| \\ &= \kappa(A). \end{aligned}$$

Thus, for any QLS algorithm, if you block encode the system matrix and the preconditioner separately, then multiply them together to get a preconditioned matrix, you can never reduce the effective condition number. By using this approach, there is no way to achieve a quicker QLS run-time.

Applying singular value amplification (SVA) to fix this issue also fails to improve the asymptotic quantum run-time. SVA would improve the block encoding normalization factors and transform the embedding of the matrix from $MA/\alpha_A \cdot \alpha_M \rightarrow MA/\|MA\|$. As shown in Lapworth and Sunderhauf [8], the additional complexity this process incurs outweighs the benefits. You might be able to get constant time speedups, but no asymptotic changes in the complexity. They also show a way of reducing the subnormalization factor of the composed block encoding U_{MA} from $\alpha_A \alpha_M$ to $\alpha_A \alpha_M / \gamma_A \gamma_M$, where we can choose γ_A and γ_M . This does not improve the scaling as it requires individually decreasing the subnormalization factor of U_A

from α_A to α_A/γ_A and the subnormalization factor of U_M from α_M to α_M/γ_M . Due to block encoding restrictions, $\alpha_A/\gamma_A \geq \|A\|$ and $\alpha_M/\gamma_M \geq \|M\|$, so the best subnormalization factor after this process is $\alpha_A \alpha_M / \gamma_A \gamma_M = \|A\| \cdot \|M\|$. Therefore, applying SVA will not improve the overall scaling of QLS algorithms.

This demonstrates the ineffectiveness of block encoding preconditioners separately from the matrix A . Despite having good preconditioning matrices, by simply block encoding them we can achieve no scaling improvement. Lapworth and Sunderhauf [8] also tried a number of other preconditioning methods that would be applicable for discretized Poisson PDEs, but again they only found constant time improvements in the effective condition number.

Due to the structure of the system matrix in three dimensions, we are able to achieve a quantum speedup; however, for 2D discretized Poisson PDEs, $K(G)$ scales as $O(N)$. This results in a quantum run-time of $O(N \text{ polylog } N \cdot \log(1/\epsilon))$, performing worse than classical algorithms that scale as $O(N \log N \cdot \log(1/\epsilon))$. For discretized Poisson PDEs, if we want to improve our quantum speedup in three dimensions, or achieve a speedup in two dimensions, we must find better preconditioning techniques.

Several recent works offer promising ideas that may help improve the asymptotic scaling of the effective condition number for our QLS algorithm. Deiml and Peterseim [6] introduced a quantum realization of the finite element method that is promising. They importantly used a combined preconditioning approach that does not conflict with our separate preconditioning bound. In Theorem 6.5 of their paper, they achieved a run-time of $O(\frac{1}{\epsilon} \text{ polylog } \epsilon)$, ignoring block encoding costs. They also made the same assumption about an efficient lookup table for the values in the system matrix. With a standard block encoding cost, such as that presented in Appendix E, their algorithm would achieve a total run-time of $O(\frac{\text{polylog } N}{\epsilon} \text{ polylog } \epsilon)$. In Appendix D, however, we show that in practical examples $1/\epsilon$ can scale as $O(N)$. This would result in a worse asymptotic bound than what is shown in our algorithm, but highlights that joint preconditioning is possible in certain scenarios. Future works are needed to further explore the possibility of these ideas, though. Separately, Orsucci and Dunjko [7] developed techniques for solving classes of positive-definite QLSs, and most notably showed that if the system matrix admits an efficiently implementable decomposition $G = LL^\dagger$, akin to the classical Cholesky decomposition, then you can achieve an effective condition number of $\sqrt{\kappa(G)}$. By efficiently implementable they meant that G can be expressed as a sum of local, positive-definite Hamiltonians, each acting nontrivially on only $\text{polylog}(N)$ qubits. These results suggest that preconditioning techniques that exploit problem-specific structure (such as that arising from discretized 3D heterogeneous Poisson equations) may enable further QLS speedups.

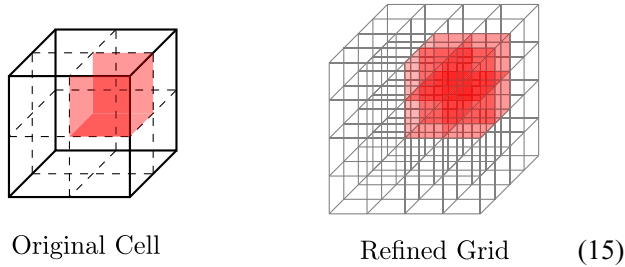
VI. INFORMATION EXTRACTION

As discussed in [13], the Hadamard test is a straightforward way to determine the average value of x in specific regions of the domain (e.g., corresponding to a well screen where a physical measurement could be taken in the fracture flow context). In a real-world case the observable of interest is coarse-grained, e.g., for groundwater flow the average pressure in a cylindrical subregion of R with diameter typically on the order of 10 cm and height on the order of 1 m (i.e., the well diameter and the screen height). That means that refining the grid past 0.1 m resolution will not significantly affect the necessary precision for information extraction. Furthermore, it is straightforward to prepare a state $|\phi\rangle$ that captures the same physical subregion regardless of grid refinement. To show this, we take $N = 2^{3\ell}$, and let $n = N^{1/3} = 2^\ell$, and assume we are interested in the physical region corresponding to the single cell at entry (i, j, k) . This corresponds to the r th element of x , where $r = k + nj + n^2i$, and means we want to measure

$$\langle \phi^{(n)} | x^{(n)} \rangle, \quad (14)$$

where $|\phi^{(n)}\rangle = |r\rangle$ is the 3ℓ -bit binary representation of the integer r , and $x^{(n)}$ is the solution to groundwater flow equation with an $n \times n \times n$ grid.

Now let us consider what happens when we double the grid resolution and want to measure the total pressure in the same physical area. Since we have doubled the resolution, the area of interest is now represented by eight cells:



where the areas shaded in red represent the physical region of interest. In the refined mesh, the row and columns of these shaded cells are given by

$$(i, j, k) \rightarrow \{(2i, 2j, 2k), (2i, 2j, 2k + 1), (2i, 2j + 1, 2k), \\ (2i, 2j + 1, 2k + 1), (2i + 1, 2j, 2k), \\ (2i + 1, 2j, 2k + 1), \\ (2i + 1, 2j + 1, 2k), (2i + 1, 2j + 1, 2k + 1)\},$$

which corresponds to the indices

$$\{r' + 4n^2 + 2n + 1, r' + 4n^2 + 2n, r' + 4n^2 + 1, \\ r' + 4n^2, r' + 2n + 1, r' + 2n, r' + 1, r'\} \quad (16)$$

for $r' = 2k + 2n(2j + 2n \cdot 2i)$. This means that for the refined system $G^{(2n)}x^{(2n)} = f^{(2n)}$, we want to measure

$$\langle \phi^{(2n)} | x^{(2n)} \rangle, \quad (17)$$

where

$$|\phi^{(2n)}\rangle = \frac{1}{\sqrt{8}}(|r'\rangle + |r' + 1\rangle + |r' + 2n\rangle + |r' + 2n + 1\rangle \\ + |r' + 4n^2\rangle + |r' + 4n^2 + 1\rangle \\ + |r' + 4n^2 + 2n\rangle + |r' + 4n^2 + 2n + 1\rangle).$$

In the binary representation of r with 3ℓ qubits/digits, the first ℓ digits represent i , the next ℓ digits represent j , and the last ℓ digits represent k . Then, in the binary representation of $r' = 2k + 2n(2j + 2n \cdot 2i) = 2k + 2^{\ell+2}j + 2^{2\ell+3}i$ with $3\ell + 3$ qubits, the first $\ell + 1$ digits will be the binary representation of k shifted to the left by one, the next $\ell + 1$ digits represent j shifted to the left by one, and similarly, the last $\ell + 1$ digits represent i shifted one to the left. In other words, if the binary representation of r has ones at locations $\{l_a\}$, then r' will have ones at locations

$$l'_a = \begin{cases} l_a + 1, & a < \ell, \\ l_a + 2, & \ell \leq a < 2\ell, \\ l_a + 3, & a \geq 2\ell. \end{cases} \quad (18)$$

Therefore, X gates on the qubits $\{l'_a\}$ will prepare the state $|r'\rangle$. Note that for a number s whose binary representation in the i th digit is zero,

$$H^{(i)}|s\rangle = \frac{1}{\sqrt{2}}(|s\rangle + |s + 2^i\rangle), \quad (19)$$

where $H^{(i)}$ means applying a Hadamard to the i th qubit. Since the binary representation of r' has zeros at the 0th, $(\ell + 1)$ th, and $(2\ell + 2)$ th digits, we have

$$|\phi^{(2n)}\rangle = H^{(2\ell+2)}H^{(\ell+1)}H^{(0)}|r'\rangle. \quad (20)$$

Therefore doubling the grid resolution requires adding only three qubits and three H gates for the information extraction.

VII. EXPLICIT GATE AND QUBIT COUNT

In this section we provide an explicit estimate of the gate and qubit resources required to apply the adiabatic-inspired linear-system algorithm of Dalzell [2] to the block encoding of G' constructed in Appendix E.

A. Mapping of the block encoding to Dalzell's complexity model

In this analysis we will focus on Dalzell's QLS algorithm [2] as it exhibits better constant factors than that in Costa *et al.* [1]. First, let G_{total} represent the total number of elementary gates used in Dalzell's algorithm. Theorem 4 of his paper shows

$$Q \leq 56.0\kappa + 1.05\kappa \ln \left(\frac{\sqrt{1-\varepsilon^2}}{\varepsilon} \right) + 2.78 \ln(\kappa)^3 + 3.17, \quad (21)$$

and the QLS algorithm makes an expected Q total queries to $U_{G'}$, $U_{G'}^\dagger$ and their controlled versions. It also makes $2Q$ queries to U_b , $U_b^\dagger$, and their controlled versions (where U_b is the unitary that generates the state $|b\rangle$).

The controlled versions of these unitaries are going to be computationally more expensive than the noncontrolled versions. Thus, for an upper bound we can assume we only makes calls to the controlled versions. In addition, we can bound this further by noticing that to go from the normal to controlled version we just need to convert every gate to its controlled version. From Lubinski *et al.* [16], the most computationally intensive gate to do this with is converting a CNOT gate to a Toffoli gate, which adds a multiplicative factor of 16. Thus, we can write G_{total} as

$$G_{\text{total}} \leq 16Q \cdot g_{U_{G'}} + 32Q \cdot g_{U_b}, \quad (22)$$

where $g_{U_{G'}}$ and g_{U_b} denote the gate counts for a single query of $U_{G'}$ and U_b , respectively.

B. Exact gate count for the block encoding of G'

From the explicit circuit in Appendix E, the block encoding $U_{G'}$ uses the following registers:

d^{ind} : 2 qubits,

d^{val} : $\lceil \log_2 D' \rceil$ qubits,

m^{hi} : 1 qubit,

m^{lo} : $3 \log_2(N^{1/3}) = \log_2 N$ qubits,

$\text{del}, \text{in_range}$: 2 qubits,

scratch : a small constant number of ancilla.

Hence the total ancilla requirement is

$$a = 2 + \lceil \log_2 D' \rceil + 1 + \log_2 N + 2 + a_{\text{scratch}}. \quad (23)$$

C. Decomposition and per-call gate counts

Each call of $U_{G'}$ is composed of (1) $n_{\text{small_adds}} = 4$ additions/subtractions acting on $\ell = \log_2(N^{1/3})$ -bit registers,

(2) $n_{\text{big_adds}} = 2$ additions/subtractions acting on the full $\log_2 N = 3\ell$ -bit registers, (3) one out-of-range oracle O_{rg} , (4) a small overhead from O_t , Z , and the Hadamard layers, and (5) a data-loading step.

1. Adder gate counts

For an n -bit addition, the Remaud adder in Remaud '24 [17] requires exactly

$$\text{CNOT count} = 14n - 10, \quad \text{T-gate count} = 10n - 3. \quad (24)$$

Using these formulas, the total adder costs for one block-encoding call are

$$\begin{aligned} \text{CNOT: } n_{\text{small_adds}}(14\ell - 10) + n_{\text{big_adds}}(14 \cdot 3\ell - 10) \\ = 140\ell - 60, \\ \text{T: } n_{\text{small_adds}}(10\ell - 3) + n_{\text{big_adds}}(10 \cdot 3\ell - 3) \\ = 100\ell - 18. \end{aligned}$$

2. O_{rg} cost

The out-of-range oracle O_{rg} implements five control conditions, four of which flip both the `del` and `in_range` qubits, and one of which flips only `del` conditioned on `in_range` = 0 and the value of d^{val} . We will use the recent result by Silva and Silva [18] that showed you can implement a t -qubit relative phase Toffoli gate using $6t + 4$ CNOT gates and zero ancilla. Note that the relative phase does not matter as we are only measuring the `del` and `in-range` qubits in the standard basis. Some of these multicontrolled Toffolis control on 0 instead of 1, so these also require two additional X-gates per 0-ctrl. With this, the formula for the gate count for each condition of the out-of-range oracle, denoted as O_{rg}^i , is

$$6(\# \text{ of gates controlled on}) + 4 + 2 \cdot (\# \text{ of 0-ctrls}). \quad (25)$$

Since the first four conditions are two Toffolis, one on the `del` qubit and one on the `in-range` qubit, they will incur a multiplicative factor of 2. Finally, we get the gate counts for each condition:

$$\begin{aligned} O_{\text{rg}}^1 &= 2 \cdot (6 \cdot 3 + 4 + 2 \cdot 2) = 52, \\ O_{\text{rg}}^2 \text{ and } O_{\text{rg}}^3 &= 2 \cdot (6(1/3 \log N + 2) + 4 \\ &\quad + 2(1/3 \log N + 1)) \\ &= \frac{16}{3} \log N + 36, \end{aligned}$$

$$\begin{aligned}
O_{\text{rg}}^4 &= 2 \cdot (6(1/3 \log N + 2) + 4 + 2(1/3 \log N)) \\
&= \frac{16}{3} \log N + 32, \\
O_{\text{rg}}^5 &\leq (D - D')(6(\lceil \log(D - D') \rceil + 1) + 4 \\
&\quad + 2(\lceil \log(D - D') \rceil + 1)) \\
&= (D - D') \cdot (8\lceil \log(D - D') \rceil + 12),
\end{aligned}$$

where the definitions for D and D' can be found in Appendix E. Note that for O_{rg}^5 , it is an inequality because we have to make an assumption on the number of 0-ctrls as it is an unknown. Then putting everything together,

$$\begin{aligned}
O_{\text{rg}} \text{ cost} &= \frac{32}{3} \log N + (D - D') \cdot (8\lceil \log(D - D') \rceil + 12) \\
&\quad + 84.
\end{aligned} \tag{26}$$

3. O_t , Z , and Hadamard layer cost

The O_t oracle is one Toffoli plus two Toffolis with a control on $|0\rangle$ instead of on $|1\rangle$. Using the 16-elementary-gate Toffoli construction mentioned above, we get

$$O_t \text{ cost} = 16 + 2 \cdot 18 = 52 \text{ elementary gates.} \tag{27}$$

The Z -gate is just a single rotation costing 1 gate. Then, the two Hadamard layers each contain $\log(2D)$ Hadamard gates, giving

$$\text{Hadamard layer cost} = 2 \log(2D) \text{ elementary gates.} \tag{28}$$

4. Data-loading cost

From Appendix A of Sunderhauf *et al.* [14], we find that the data-loading step requires D CNOT gates and D single-qubit rotation gates. This is because we cleverly manipulated D to always be a power of 2. Thus, with this,

$$\text{Data-loading cost} = 2D \text{ elementary gates.} \tag{29}$$

5. Final gate count

Combining everything from the previous sections we get that

$$\begin{aligned}
g_{U_{G'}} &\leq \frac{272}{3} \log N + 2 \log(2D) + 2D \\
&\quad + (D - D') \cdot (8\lceil \log(D - D') \rceil + 12).
\end{aligned} \tag{30}$$

D. Numerical comparison with classical solvers

To assess the practicality of our QLS algorithm, we compare the explicit gate-count estimates in Sec. VII with the classical solver benchmarks of Greer *et al.* [19], who evaluated linear solvers for 3D discrete fracture network (DFN) models of subsurface flow.

1. Classical solver benchmarks

Greer *et al.* [19] compared direct and iterative solvers for 3D DFN flow problems on networks up to the sizes tested in their paper. For the representative test studied here (three fracture sets, 5512 fractures; conforming mesh with $N = 7\,758\,411$ nodes and $15\,699\,327$ triangles; minimum feature size $h = 0.025$ m) they report that sparse Cholesky factorization is the fastest method overall, and that CG preconditioned by algebraic multigrid (CG + AMG) is the most competitive iterative fallback. On their single-workstation experiments (128 GB RAM, Intel i9 class CPU), wall times for the best methods on this problem scale lie in the envelope $\sim 10^2$ – 10^3 s; Cholesky completed the 7.76 million-node case, but its memory use approaches practical machine limits as N and factor fill grow.

2. Quantum resource estimate

Using Dalzell's QLS bound [2] applied to the explicit block encoding $U_{G'}$ (parameters chosen for concreteness: $D = 5,512$, $D' = \lceil D/2 \rceil = 2,756$, $\kappa = 2^{20}$, $\varepsilon = 10^{-3}$), we obtain the following conservative logical resources for solving the same linear system with $N = 7\,758\,411$: the Dalzell query bound evaluates to

$$Q \approx 1.31 \times 10^8,$$

the per-call gate cost from the block-encoding formulas is

$$g_{U_{G'}} \approx 3.09 \times 10^5 \text{ elementary logical gates,}$$

(a dominant contribution from the $(D - D')$ oracle term), a conservative segmented state-prep model gives

$$g_{U_b} \approx 1.10 \times 10^4,$$

and, assuming worst-case controlled-call promotion (factor 16) for all queries, the total logical gate budget is

$$G_{\text{total}} \lesssim 16 Q g_{U_{G'}} + 32 Q g_{U_b} \approx 6.9 \times 10^{14}.$$

The register/ancilla accounting yields $n_{\text{logical}} \approx 70$ – 85 logical qubits (we conservatively quote ~ 80).

Converting the total logical gate count into a fault-tolerant wall-time estimate requires modeling execution in terms of repeated quantum error correction (QEC) cycles. Elementary physical gate times on state-of-the-art superconducting hardware are on the order of $t_{\text{phys}} \sim 20$ ns. For example, Arute *et al.* [20] achieved a maximum gate speed of 25 ns. However, in a surface-code architecture, wall-clock time is not set directly by physical gate duration. Instead, execution is synchronized to repeated QEC cycles that interleave entangling gates, stabilizer measurement, resonator reset, readout, and classical processing. Recent experimental demonstrations report QEC cycle times of

approximately $t_{\text{cycle}} \approx 1.1 \mu\text{s}$, together with an average real-time decoder latency of approximately $63 \mu\text{s}$ [21].

Importantly, the decoder latency does not multiply the total number of QEC cycles. Using Pauli-frame tracking, Pauli correction operations are recorded in classical memory rather than physically applied to qubits, allowing syndrome extraction and decoding to proceed in parallel with subsequent QEC cycles [22]. In this execution model, the decoder latency contributes only an additive overhead rather than a per-cycle multiplicative cost. Consequently, the dominant contribution to wall-clock run-time is the QEC cycle time itself, not the classical decoding latency.

Logical gates must span multiple QEC cycles to suppress logical error rates below threshold. Denoting by c_{gate} the number of QEC cycles required per logical operation (which depends on code distance and gate type), the effective wall-time cost per logical gate is

$$t_{\text{logical}} \approx c_{\text{gate}} t_{\text{cycle}}.$$

Conservatively taking $c_{\text{gate}} \sim 10$, the effective logical gate duration becomes $11 \mu\text{s}$, despite underlying physical gates being 2 orders of magnitude faster. Applying this model to our estimated logical gate count $G_{\text{total}} \approx 6.9 \times 10^{14}$, the total run-time is on the order of decades on current fault-tolerant superconducting platforms. The additional $\sim 63 \mu\text{s}$ decoder latency contributes only a constant overhead and does not change this asymptotic estimate.

Several foreseeable advances could significantly reduce this run-time. Architectural improvements such as faster readout, reduced measurement latency, and improved qubit coherence could reduce QEC cycle times from the current $\sim 1 \mu\text{s}$ toward the $\sim 100 \text{ ns}$ regime. Improved scheduling and parallelism across logical qubits, block-encoding oracles, and state preparation routines could reduce effective circuit depth relative to the total logical gate count. On the algorithmic side, tighter gate analysis and improved QLS constructions with reduced constant factors could decrease G_{total} by 1 or more orders of magnitude. Taken together, such developments suggest that cumulative constant-factor reductions of several orders of magnitude in wall time are plausible over longer time horizons. Nevertheless, even under optimistic assumptions, the quantum run-time remains substantially larger than that of classical solvers for the instances considered here. Achieving a practical quantum advantage will therefore require not only hardware and compilation improvements, but also better preconditioning methods to reduce the run-time for large, ill-conditioned linear systems.

3. Modeling assumptions and limitations

The classical and quantum columns differ in both modeling and resource semantics, and these differences shape the comparison. Greer *et al.* [19] solved the fracture-only

DFN (flow confined to lower-dimensional fracture planes), which reduces the degrees of freedom by not considering the underlying rock as being permeable. On the other hand, our quantum model can be adapted to represent more general coupled fracture-matrix operators but at increased oracle complexity (reflected in the $(D - D')$ term). The quantum query complexity scales approximately linearly with κ (and logarithmically with $1/\epsilon$), while the classical direct method (Cholesky) is essentially independent of κ for run-time but sensitive to memory and fill-in. Finally, the quantum column reports logical resources (gates, logical qubits) that must be converted to physical wall time via a fault-tolerance model; direct wall-time comparison without that mapping is therefore not meaningful.

4. Summary

Anchored to the same DFN test ($N = 7758411$), Greer *et al.* demonstrated that optimized classical solvers (Cholesky; CG + AMG) solve the problem in minutes on commodity hardware using tens to hundreds of gigabytes of memory. The Dalzell QLS applied to the explicit block encoding requires $\mathcal{O}(10^{14} - 10^{15})$ logical gates under conservative assumptions and $\mathcal{O}(10^1 - 10^2)$ logical qubits. Therefore, the quantum approach offers exponentially compressed memory but incurs a very large logical-depth cost dominated by κ and oracle structure. Practical quantum competitiveness at this problem scale thus requires (1) significant reduction of κ (for example by embedding effective preconditioners into the block encoding), (2) oracle compression so $D' \approx D$ (reducing the heavy $(D - D')$ term), and (3) lower controlled-call overhead via optimized multicontrol and compilation strategies.

VIII. CONCLUSION

Finding useful, real-world applications of QLS algorithms remains an outstanding—and daunting—task. In this work, we have demonstrated that many of the critical components required for applying QLS algorithms can be realized efficiently in the case of 3D discretized heterogeneous elliptic PDEs, with explicit examples for the case of 3D fracture flow. Preparing the input state $|b\rangle$, which represents a source or boundary condition vector, is often sparse or structured in practice and can therefore be efficiently implemented. The system matrix G is highly structured and often has a nice classical description. Under the assumption that the number of distinct nonzero elements in the matrix is polylogarithmic in N , we can block encode G with $\mathcal{O}(\text{polylog } N)$ gate complexity. Furthermore, for many applications we are interested in global properties or local observables of our solution state, which can be extracted from $|x\rangle$ with low overhead. Combining all of these results with recent QLS algorithms [1,2], solving the system scales

as $O(\kappa \log(1/\epsilon)) = O(N^{2/3} \text{polylog } N \cdot \log(1/\epsilon))$. This is better than classical run-time of $O(N \log N \cdot \log(1/\epsilon))$, and importantly, it comes with an exponential reduction in memory. This opens the door to gaining a quantum advantage for simulation of 3D heterogeneous Poisson PDEs, but much work remains to be done to make these algorithms practically useful.

However, our analysis also uncovers a critical limitation: reducing the effective condition number κ remains a major obstacle to achieving more substantial speedups, which may be necessary for this to be useful on future large-scale, fault-tolerant hardware. Despite having a preconditioner that improves the scaling of the condition number, by block encoding the two matrices separately, we can never reduce the asymptotic run-time of QLS algorithms. While we mention several recent works that offer promising ideas for quantum-compatible preconditioning, further analysis is needed to assess their effect in settings such as ours. This suggests that reducing the effective condition number is highly restrictive in the quantum context, and must be treated as a significant limitation in any practical QLS application.

Looking ahead, quantum-compatible strategies for mitigating effective condition-number scaling are important. Our results highlight how a carefully tailored, interdisciplinary approach bridging quantum algorithms and domain-specific structure can result in a quantum advantage, albeit small in this case. Without improved quantum preconditioning methods, the run-time of our algorithm is dominated by the condition number of the system matrix. If the matrix is naturally well conditioned, such as in the 3D groundwater flow case, we can achieve a quantum advantage.

ACKNOWLEDGMENTS

A.P. and D.O. gratefully acknowledge support from the Department of Energy, Office of Science, Office of Basic Energy Sciences, Geoscience Research program under Award No. LANLECA1. D.O. and J.G. acknowledge support from Los Alamos National Laboratory's Laboratory Directed Research and Development program through project 20220077ER. This work was supported by the U.S. Department of Energy through the Los Alamos National Laboratory, LA-UR-22-24431. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy (Contract No. 89233218CNA000001).

DATA AVAILABILITY

The data that support the findings of this article are openly available [25].

APPENDIX A: PROOF $\|G^{-1}\| = O(1)$

To start, since G is positive definite, square, and symmetric, we can say that $\sigma_{\min}(G) = \lambda_{\min}(G)$ (where $\lambda_{\min}(G)$ is the smallest eigenvalue of G). With this, the discrete Poincaré inequality in bounded domains with suitable boundary conditions (e.g., Dirichlet) for our equation of interest $Gh = f$ is

$$\|h\|_{L^2} \leq C \cdot \|\nabla_d h\|_{L^2}. \quad (\text{A1})$$

Here, ∇_d is the discrete gradient operator and C is a constant that depends on our chosen boundary conditions and the physical domain we are considering (in this case an $L \times L \times L$ section of material). Thus, C is a constant that does not grow with the system size N .

We can write the smallest eigenvalue using the Rayleigh quotient as

$$\lambda_{\min}(G) = \min_{u \neq 0} \frac{u^T G u}{u^T u}. \quad (\text{A2})$$

Now expanding $u^T G u$ gives

$$\begin{aligned} u^T G u &= \sum_{a,b=1}^N u_a \cdot G_{ab} \cdot u_b \\ &= \sum_{a=1}^N u_a^2 \cdot G_{aa} + \sum_{a \neq b} u_a \cdot G_{ab} \cdot u_b \\ &= \sum_{a=1}^N \sum_{b \in \text{nbhd}(a)} \frac{k_{ab}}{(\Delta x)^2} \cdot u_a^2 - \sum_{a \neq b} \frac{k_{ab}}{(\Delta x)^2} \cdot u_a u_b \\ &= \sum_{\{a,b\} \text{ neighbors}} \frac{k_{ab}}{(\Delta x)^2} (u_a^2 - 2u_a u_b + u_b^2) \\ &\geq k_{\min} \cdot \sum_{\{a,b\} \text{ neighbors}} \frac{(u_a - u_b)^2}{(\Delta x)^2} \\ &= k_{\min} \cdot \|\nabla_d u\|_{L^2}^2, \end{aligned}$$

where $\text{nbhd}(a) = \{b \in \{1, \dots, N\} \setminus \{a\} : G_{ab} \neq 0\}$ and $\sum_{\{a,b\} \text{ neighbors}}$ is a sum is over unordered pairs. Thus, by now applying the discrete Poincaré inequality we get that

$$\begin{aligned} \lambda_{\min}(G) &= \min_{u \neq 0} \frac{u^T G u}{u^T u} \\ &\geq \frac{k_{\min} \cdot \|\nabla_d u\|_{L^2}^2}{u^T u} \\ &\geq \frac{k_{\min}}{C^2} \cdot \frac{\|u\|_{L^2}^2}{u^T u} \\ &= \frac{k_{\min}}{C^2}. \end{aligned}$$

Now putting everything together,

$$\|G^{-1}\| = \frac{1}{\lambda_{\min}(G)} \leq \frac{C^2}{k_{\min}} = O(1),$$

since both C and $k_{\min}$ are constants.

APPENDIX B: BOUND ON D FOR GROUNDWATER FLOW

In practice, we are discretizing all the fracture scales, which can provide an advantage over classical approaches that truncate the fracture scales and only model large fractures.

In most classical simulations of groundwater flow, the locations of very large fractures are known *a priori*, and smaller fractures are placed in the grid by sampling from a distribution chosen to match observational data. Instead we propose using fractal patterns for adding small fractures; for examples see Fig. 3. Such fractal patterns, which are based on simple arithmetic formulas and thus can be efficiently encoded in a quantum circuit, could be fine-tuned to match the characteristics and existing data for a given region. In some example fractal patterns following simple arithmetic formulas, we also get significantly better scaling than the theoretical upper bound of $O(F^{12})$. For the 3D pitchfork pattern in Fig. 4 we appear to get a linear relationship between the fracture scale and number of distinct elements, which is a significant speedup for practical uses of this algorithm. Geologic fracture networks are often modeled as fractals via an effective permeability [23,24], which loses critical connectivity information that could put the fracture network above or below a percolation threshold. Our approach overcomes this limitation and represents all the fracture scales explicitly.

Thus, as long as the fracture pattern has an arithmetic description, and the number of distinct fracture lengths F , i.e., distinct values for $k_{i,j,k}$, scales logarithmically with the number of cells N , then the block encoding circuit we showed in Appendix E 4 gives a block encoding $U_{G'}$ with scaling $O(\log N + D \log D) = O(\text{polylog } N)$.

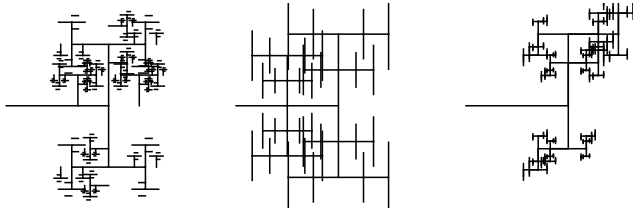


FIG. 3. Two-dimensional cross sections of different fractal models of fine-grained fracture structure with simple arithmetic formulas. These are examples of fracture structures that could lead to efficient block encodings for the groundwater flow matrix G' .

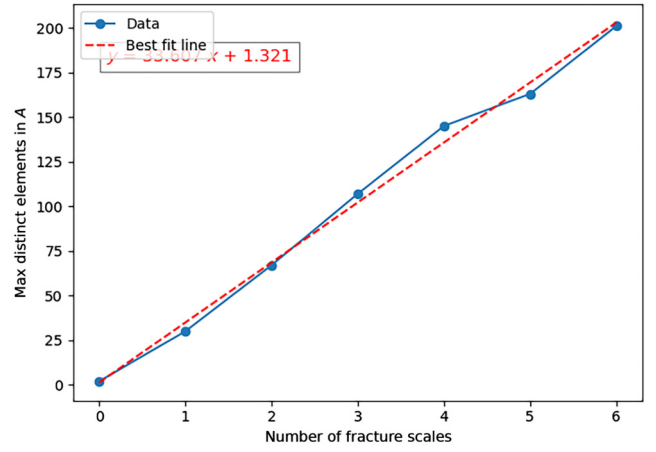


FIG. 4. For the 3D pitchfork pattern shown in Fig. 5, we observe that the number of distinct elements in the groundwater flow matrix G' grows roughly linearly with the number of fracture scales F . This suggests that the theoretical bound of $O(F^{12})$ from Eq. (12) may be loose for some practical examples of 3D heterogeneous Poisson problems.

APPENDIX C: BLOCK ENCODING Δ^{-1}

The key to constructing an efficient block encoding for Δ^{-1} is the fast inversion technique introduced by Tong *et al.* [3]. Specifically, they gave an algorithm for calculating A^{-1} for normal matrices A . This algorithm requires an efficient oracle O_D for a diagonal matrix D and a unitary V , where $A = VDV^\dagger$. Given these, then one can calculate D^{-1} and thus A^{-1} using O_D , $O_D^\dagger$, V , and $V^\dagger$ each exactly once. In the case of the discrete Laplacian, the necessary oracles for implementing the fast inverse are relatively simple and efficient. For a 2D discrete Laplacian on an $n \times n$ grid, the eigenvalues are given by

$$\lambda_{k_x, k_y} = -\frac{1}{2 \sin^2\left(\frac{\pi}{2(n+1)}\right)} \left(\sin^2\left(\frac{k_x \pi}{2(n+1)}\right) + \sin^2\left(\frac{k_y \pi}{2(n+1)}\right) \right), \quad (C1)$$

where k_x, k_y are the wave numbers associated with the Fourier modes and the eigenvalues have been rescaled so that $\|\Delta^{-1}\| = 1$. These eigenvalues can be computed efficiently, which means that the cost of implementing the oracle O_D is $O(\log N)$. In addition, we need access to the unitary matrices V and $V^\dagger$, which in this case correspond to the quantum Fourier transform (QFT) and its inverse. Since QFT can also be implemented with a gate complexity of $O(\log N)$, the block encoding for the inverse Laplacian $U_{\Delta^{-1}}$ also has a total cost of $O(\log N)$.

APPENDIX D: SCALING IN ϵ

In order to properly determine the scaling of our QLS algorithm, we must also determine how the necessary accuracy ϵ scales with N . Rescaling G and possibly applying preconditioning will not change that the final state prepared by the QLS algorithm:

$$|x\rangle = |G^{-1}b\rangle = G^{-1}b/\|G^{-1}b\|. \quad (\text{D1})$$

Ideally, we would choose ϵ such that it does not change the asymptotic run-time of our algorithm. For example, if we pick $\epsilon = \text{poly}(1/N)$, then $\log(1/\epsilon) = \text{polylog } N$, giving us a run-time of $O(\kappa \cdot \log(1/\epsilon)) = O(N^{2/3} \text{polylog } N)$, which aligns with the desired scaling.

A natural choice of ϵ , however, is to pick it such that it is on the order of the element of $|x\rangle$ with the smallest nonzero value. This would look like

$$\epsilon = O\left(\frac{\min_i^{>0} |(G^{-1}b')_i|}{\|G^{-1}b'\|}\right). \quad (\text{D2})$$

If ϵ were larger than this, we lose confidence in the accuracy of the smallest components of the solution, where even the sign could be wrong for a larger epsilon. This quantity in Eq. (D2) is highly problem dependent and depends heavily on the exact form of the decomposition of b in the eigenbasis of G . It can, however, be estimated through numerical studies of the system we care about. Numerical studies for groundwater flow, using the

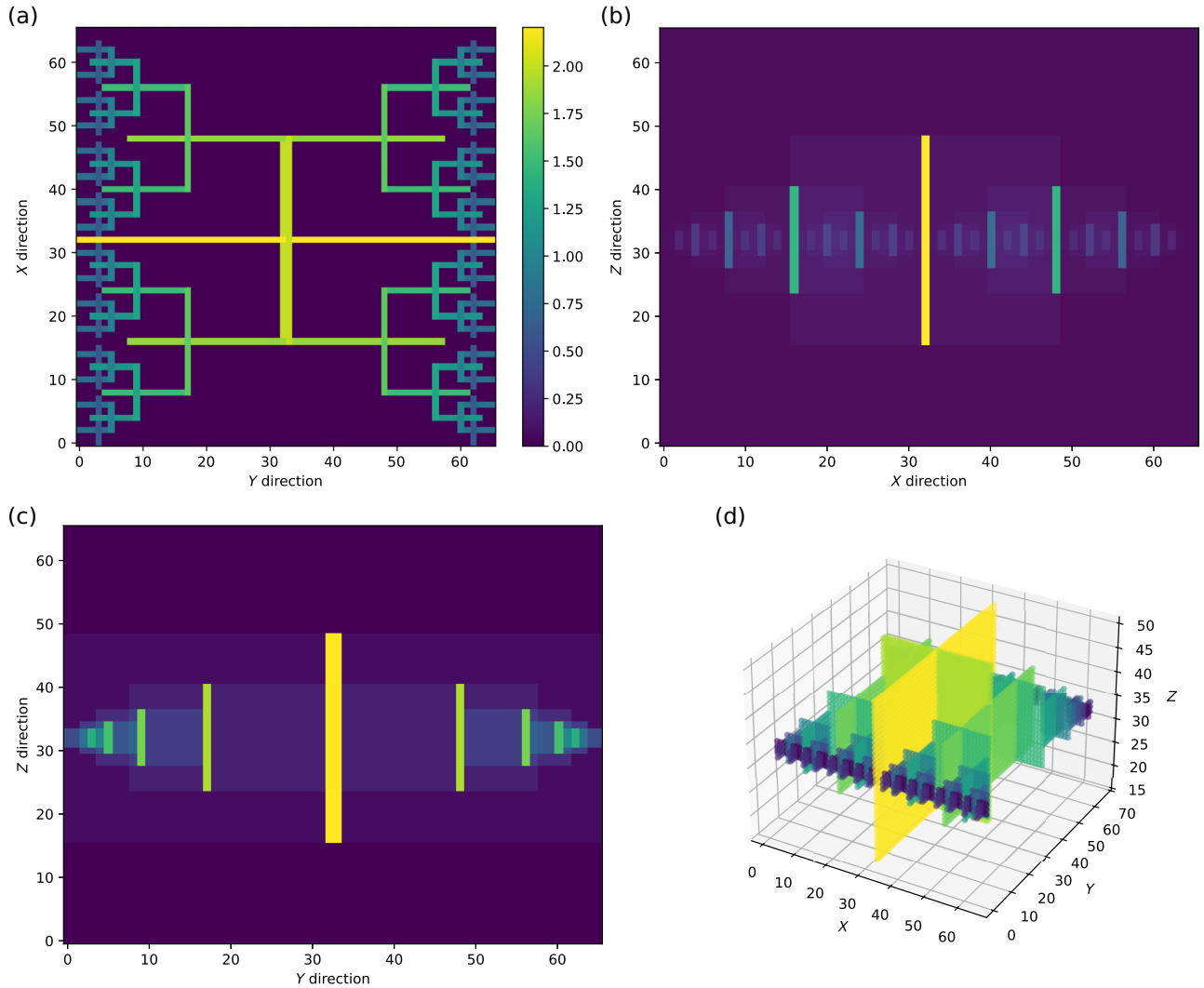


FIG. 5. Extension of the 2D pitchfork fracture pattern introduced in [11] to three dimensions. The 2D pitchfork fracture is started in opposite directions of the XY plane, where each fracture scale is assigned a depth based on its length (the smaller the fracture scale, the smaller the depth). The different colors represent different cell permeability values, and in this particular example, we have five fracture scales. (a) is the XY slice (centered at Z) of the fracture pattern. (b) is the XZ slice, where the values are averaged across Y . (c) is the YZ slice, where the values are averaged across X . (d) is the full fracture network.

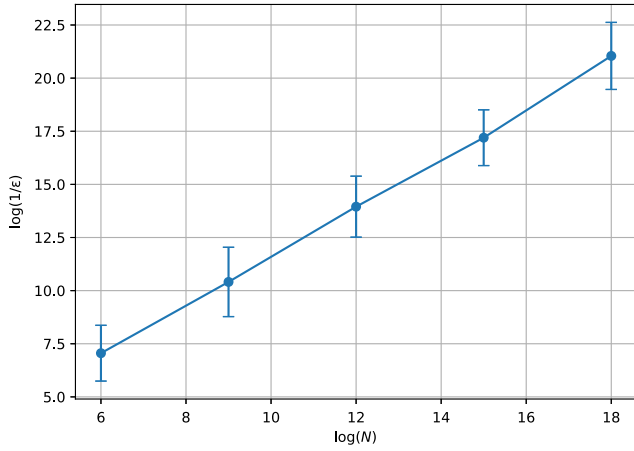


FIG. 6. Scaling of the logarithmic error $\log(1/\epsilon)$ with system size N . Here ϵ is defined as the smallest (nonzero) magnitude of an element in $G^{-1}b/\|G^{-1}b\|$. G is modeled by the modified pitchfork fracture system (shown in Fig. 4) and $|b\rangle$ represents 20 randomly placed injection/extraction sites. The error bars represent one standard deviation of results from 100 different choices of $|b\rangle$.

3D pitchfork fractal fracture network shown in Fig. 5, indicate favorable scaling of the required accuracy parameter ϵ . Using the definition of ϵ shown in Eq. (27), we observed that $\log(1/\epsilon)$ scales as $O(\text{polylog } N)$ in these simulations. Full results and parameter details are provided in Fig. 6.

APPENDIX E: BLOCK ENCODING G'

First, it will be useful to restate some important definitions that will be used frequently throughout this section:

- (1) D is the number of distinct, nonzero values in G' , and $d \in \{0, \dots, D-1\}$ is a label indicating a specific value;
- (2) M is the maximum multiplicity of any nonzero element in G' , and $m \in \{0, \dots, M-1\}$ is a label that distinguishes different elements with the same value;
- (3) S is the maximum number of nonzero elements per row or column (in the 3D groundwater flow case, both the column sparsity and row sparsity are 7, making $S = 7$), and $s_c \in \{0, \dots, S\}$ indicates a specific nonzero element in a column;
- (4) $i, j \in \{0, \dots, N\}$ are row and column labels, respectively.

Now, in addition to O_c and O_t , described in Sec. IV, we also need to define an out-of-bounds oracle O_{rg} that does the following:

$$O_{rg} |d\rangle |m\rangle |0\rangle = \begin{cases} |d\rangle |m\rangle |0\rangle & \text{if } G'_{i(d,m),j(d,m)} = G'_d \\ |d\rangle |m\rangle |1\rangle & \text{if } G'_{i(d,m),j(d,m)} = 0. \end{cases}$$

Here, $i(d,m)$ and $j(d,m)$ denote the row and column indices determined by (d,m) . Following the symmetric block-encoding circuit from [14], we additionally require $MD = NS$ to ensure that the block encoding we construct is Hermitian. This can be achieved by padding both M and S .

Before we start, we define distinct elements in a way that simplifies the block encoding. There are four distinct sections of G' as given in Eq. (5): Section 00: $a = b$, Section 01: $b = a \pm 1$, Section 10: $b = a \pm N^{1/3}$, and Section 11: $b = a \pm N^{2/3}$. With this, we label every item as $d = d^{\text{ind}} || d^{\text{val}}$ where $d^{\text{ind}} \in \{0, 1\}^2$ is an indicator of which section d is a part of and d^{val} is the label indicating which distinct value d is. From this, we get that if two elements in G' , d_1 and d_2 , have the same value but are in different sections, then they will be treated as distinct values under this labeling.

Thus, if we let D^{init} be the actual number of distinct elements in G' and D' be the number of distinct elements after relabeling every d as $d^{\text{ind}} || d^{\text{val}}$, then we get that $4 \leq D' \leq 4D^{\text{init}}$.

With this labeling scheme, the maximum multiplicity is at most $2(N-1)$, which we will pad to $M = 2N$. Along with that, the number of distinct elements must be a power of 2 (since we need to be able to represent each distinct element as a binary string), so we will finally let $D = 2^{\lceil \log_2(D') \rceil}$. This gives us $2D \geq 8 > \max \text{ sparsity}$, which is 7 in this case. Thus, by padding $S = 2D$, we get $MD = NS = 2ND$ as needed. These extra padded qubits will not create a problem as the out-of-range oracle will handle any cases where our current state does not correspond to a valid nonzero entry in G' .

Now, we are ready to construct O_c , O_t , and O_{rg} . These constructions closely resemble the 2D Laplacian example from [14], with a couple of changes to deal with the increased number of data elements and slightly different structure. This structure change is due to the fact that G' is closely related to the 3D and not 2D Laplacian. Once we have created explicit circuits for these oracles, by following Fig. 7 we will get a block encoding for G' as shown in Fig. 8.

1. Transposition oracle O_t

The transposition oracle takes a value d and a multiplicity label m and outputs (d, m') where m' is the element

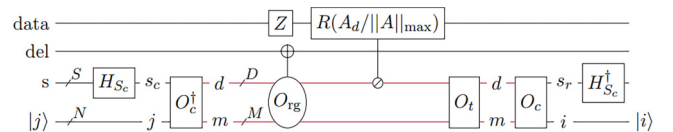


FIG. 7. Block-encoding circuit for symmetric matrices as given in Eq. (36) of [14]. This circuit serves as the template for our block encoding of the discretized 3D heterogeneous Poisson matrix G' .

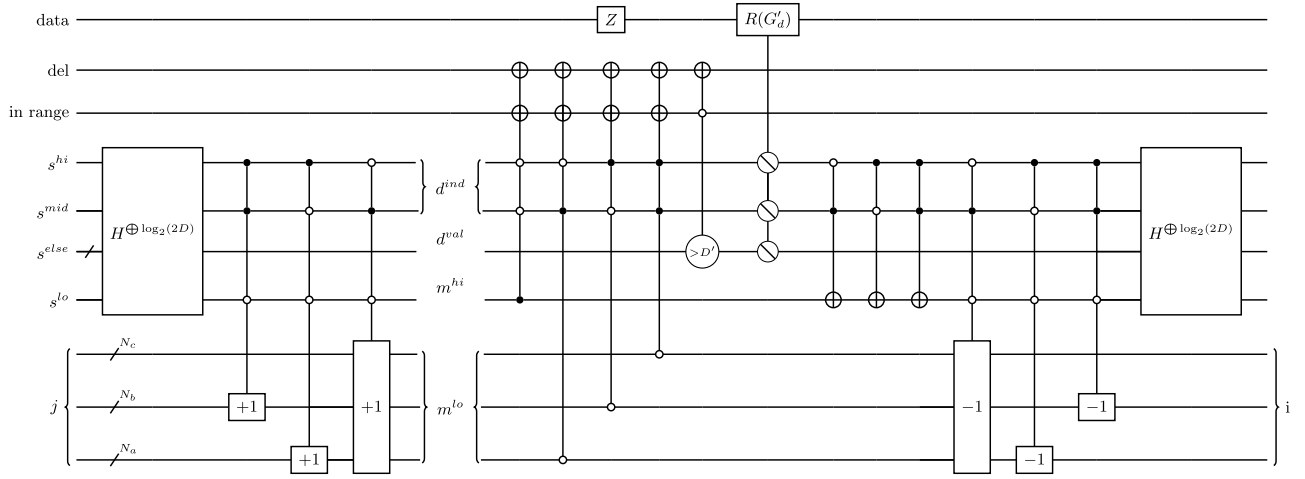


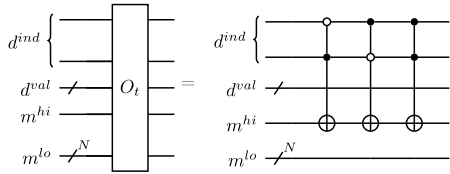
FIG. 8. Full block encoding for the 3D heterogeneous Poisson equation, following the technique of block encoding symmetric matrices shown in Fig. 7.

related to m by transposition. Note that there always exists an element m' as our matrix G' is symmetric.

Following from [14], we denote the first bit of m as m^{hi} and the last $\log_2 N$ bits as m^{lo} . We then let $m^{\text{hi}} = 0$ denote we are in the lower-left triangular matrix (including the diagonal), and let $m^{\text{hi}} = 1$ denote we are in the upper triangular matrix. In the lower triangular matrix we choose m^{lo} to be the row index, and in the upper triangular matrix we choose m^{lo} to be the column index. With this, the transposition oracle will do the following:

$$O_t(d, m^{\text{hi}}, m^{\text{lo}}) = \begin{cases} (d, m^{\text{hi}}, m^{\text{lo}}) & \text{if } d^{\text{ind}} = 00, \\ (d, 1 - m^{\text{hi}}, m^{\text{lo}}) & \text{if } d^{\text{ind}} \in \{01, 10, 11\}, \end{cases}$$

which as a quantum circuit is



Note that the “/” symbol on the quantum wires represents that it contains more than one qubit. If there is a number K attached to the “/” then that wire contains $\log_2 K$ qubits; however, since in the case of d^{val} the size of the channel varies based on the fracture network, we attach no number to it.

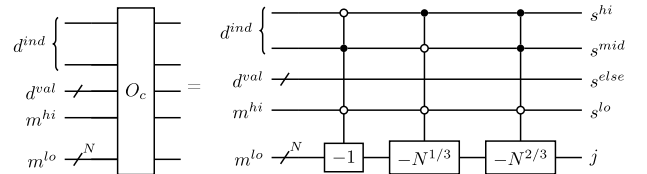
2. Column oracle O_c

The column oracle takes a value d and a number m and outputs the column j and sparsity index s_c corresponding to the m^{th} occurrence of the value d . When we apply the column oracle, we will update the m^{lo} register to be the column j and then concatenate the remaining outputs to

make the sparsity index s_c . For simplicity, we just describe $O_c : (d, m) \rightarrow j$ below:

$$O_c(d, m^{\text{hi}}, m^{\text{lo}}) = \begin{cases} m^{\text{lo}} & \text{if } d^{\text{ind}} = 00, \\ m^{\text{lo}} & \text{if } m^{\text{hi}} = 1, \\ m^{\text{lo}} - 1 & \text{if } m^{\text{hi}} = 0 \text{ and } d^{\text{ind}} = 01, \\ m^{\text{lo}} - N^{1/3} & \text{if } m^{\text{hi}} = 0 \text{ and } d^{\text{ind}} = 10, \\ m^{\text{lo}} - N^{2/3} & \text{if } m^{\text{hi}} = 0 \text{ and } d^{\text{ind}} = 11. \end{cases}$$

As a quantum circuit this is



Here the subtraction gates indicate that we add that amount to the integer representation of m^{lo} . Thus, if we apply the -1 gate then $m^{\text{lo}} \rightarrow m^{\text{lo}} - 1 \pmod{N}$, and similarly for the other two.

As mentioned previously, and shown in the circuit diagram, the m^{lo} register is updated to represent the column j , and the sparsity index is a concatenation of the remaining outputs. In particular, we relabel the outputs of the column oracle in the following way: $s_c = d^{\text{ind}} || d^{\text{val}} || m^{\text{hi}} = s^{\text{hi}} || s^{\text{mid}} || s^{\text{else}} || s^{\text{lo}}$. Since G' has at most seven elements in each column, we need at most three qubits to represent the sparsity index; however, s_c may contain more than three qubits due to the fact that it was padded. Thus, we have $s^{\text{hi}} || s^{\text{mid}} || s^{\text{lo}}$ is the part of s_c that actually tells us what the sparsity index is, while s^{else} is simply a by-product of padding S that has no relevance when deciphering the sparsity index.

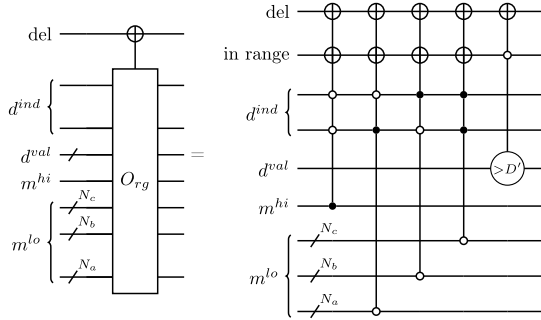
3. Out-of-range oracle O_{rg}

Since D , M , and S were padded to ensure $DM = NS$, there are assignments to (d, m) that do not correspond to actual, nonzero values in G' (we refer to these assignments as out-of-range entries and refer to valid assignments to (d, m) as in-range entries). Thus, out-of-range oracle takes as input $|d\rangle |m\rangle |0\rangle$ and outputs $|d\rangle |m\rangle |1\rangle$ if (d, m) is an out-of-range entry and $|d\rangle |m\rangle |0\rangle$ if (d, m) is an in-range entry. The 0/1 indicator qubit appended to the end of $|d\rangle |m\rangle$ we label as “del.” With this, the out-of-range indices $(d, m) = (d^{ind}, d^{val}, m^{hi}, m^{lo})$ are the following:

- (1) $d^{ind} = 00$ and $m^{hi} = 1$;
- (2) $d^{ind} = 01$ and $m^{lo} = 0 \pmod{N^{1/3}}$;
- (3) $d^{ind} = 10$ and $\lfloor m^{lo} \pmod{N^{2/3}} / N^{1/3} \rfloor = 0$;
- (4) $d^{ind} = 11$ and $m^{lo} < N^{2/3}$;
- (5) $d^{val} > D'$ and $(d^{ind}, m^{hi}, m^{lo})$ does not satisfy any of the above conditions.

The constraint on condition (5) that “ $(d^{ind}, m^{hi}, m^{lo})$ does not satisfy any of the above conditions” is added so all five conditions are dependent on each other. We need to ensure this so there is not a case where we accidentally flip the delete qubit twice.

To simplify the implementation of the quantum circuit, we split m^{lo} into three registers N_a , N_b , and N_c , all with $\log_2 N^{1/3}$ qubits. We define these registers such that if $N_a = a$, $N_b = b$, and $N_c = c$ for any $a, b, c \in \{0, 1\}^{\log_2 N^{1/3}}$, then we can write $m^{lo} = a + N^{1/3}b + N^{2/3}c$. With this, the quantum circuit for O_{rg} is the following:



Here, the circle containing $> D'$ (which we will refer to as control- D') is a sequence of control gates that indicates whether $d^{val} > D'$. Also, notice that we added another qubit labeled “in range” that indicates if $(d^{ind}, m^{hi}, m^{lo})$ is a valid in-range entry or not. We then control on this register not being flipped to satisfy the second constraint on condition (5).

The construction for control- D' is dependent on the structure of the fracture network (e.g., the number of fracture scales) and thus cannot be explicitly described here; however, the gate complexity of control- D' will be the number of out-of-range values, which is $D - D' =$

$2^{\lceil \log_2(D') \rceil} - D' = O(D')$. Each gate will also only need to control on $O(\log_2 D')$ qubits as we only need to consider operations on the last $\lceil \log_2(D - D') \rceil$ bits of the input. Thus, the total gate complexity of the out-of-range oracle is $O(D' \log_2 D')$.

4. Full block-encoding circuit

We now have all of the tools we need to block encode G' . By using the symmetric block-encoding circuit in [14] (shown in Fig. 7) and the three oracles defined above, the full quantum circuit for the fracture flow matrix is shown in Fig. 8.

Note that the notation of a slash in the control on $R(G'_d)$ indicates that the gate controls on several values. In particular, it applies a rotation according to what the d -register of the circuit is. In addition, the template in Fig. 7 requires the rotation gate to be of the form $R(A_d / \|A\|_{\max})$ (where $\|\cdot\|_{\max}$ is the maximum absolute value of any element in A). The actual requirement is that for all elements A_{ij} in A , they must satisfy $A_{ij} \in [-1, 1]$. We already satisfy this constraint by our definition of G' . Using Eq. (8), we have that for all elements G'_{ij} in G' , $|G'_{ij}| \leq \|G'\| \leq 1$. Thus, it suffices to just rotate about G'_d . Finally, the gate complexity is dominated by the additions and out-of-range oracle. The additions require $O(\log_2 N + \log_2 N^{1/3})$ gates and the out-of-range oracle requires $O(D' \log_2 D')$ gates, giving a total gate complexity of $O(\log_2 N + D' \log_2 D') = O(\log_2 N + D^{\text{init}} \log_2 D^{\text{init}})$.

All constructions above assume that we can efficiently identify the value of any matrix element. That is, given a position label (i, j) or an index pair (d, m) , we must be able to determine the corresponding entry G'_{ij} and associate it with the correct value label $d \in \{0, \dots, D - 1\}$. This requires an efficient mapping from matrix coordinates (or multiplicity index) to value class, and we assume that this mapping is computable in polylogarithmic time with respect to N and D . In practice, this can be implemented either by precomputing a lookup table for all nonzero entries or by using a deterministic rule based on the underlying physical model. For example, the fractal fracture idea mentioned in Appendix B gives an efficient way to do this for groundwater flow.

- [1] Pedro C. S. Costa, Dong An, Yuval R. Sanders, Yuan Su, Ryan Babbush, and Dominic W. Berry, Optimal scaling quantum linear-systems solver via discrete adiabatic theorem, *PRX Quantum* **3**, 040303 (2022).
- [2] Alex Dalzell, A shortcut to an optimal quantum linear system solver, *arXiv:2406.12086*.
- [3] Yu Tong, Dong An, Nathan Wiebe, and Lin Lin, Fast inversion, preconditioned quantum linear system solvers, fast Green’s-function computation, and fast evaluation of matrix functions, *Phys. Rev. A* **104**, 032422 (2021).

- [4] Pedro C. S. Costa, Dong An, Ryan Babbush, and Dominic Berry, The discrete adiabatic quantum linear system solver has lower constant factors than the randomized adiabatic solver, [Quantum](#) **9**, 1887 (2025).
- [5] Guang Hao Low and Yuan Su, Quantum linear system algorithm with optimal queries to initial state preparation, [Quantum](#) **10**, 2041 (2026).
- [6] Matthias Deiml and Dirk Peterseim, Quantum realization of the finite element method, [Am. Math. Soc.](#) **95** (2024).
- [7] Davide Orsucci and Vedran Dunjko, On solving classes of positive-definite quantum linear systems with quadratically improved runtime in the condition number, [Quantum](#) **5**, 573 (2021).
- [8] Leigh Lapworth and Christoph Sünderhauf, Preconditioned block encodings for quantum linear systems, [Quantum Sci. Technol.](#) **10**, 045064 (2025).
- [9] Gilbert Strang, *Computational Science and Engineering* (Wellesley-Cambridge Press, Wellesley, MA, 2007).
- [10] Jeffrey D. Hyman, Satish Karra, Natalia Makedonska, Carl W. Gable, Scott L. Painter, and Hari S. Viswanathan, dfn-Works: A discrete fracture network framework for modeling subsurface flow and transport, [Comput. Geosci.](#) **84**, 10 (2015).
- [11] John Golden, Daniel O'Malley, and Hari Viswanathan, Quantum computing and preconditioners for hydrological linear systems, [Sci. Rep.](#) **12**, 22285 (2022).
- [12] Aram W. Harrow, Avinatan Hassidim, and Seth Lloyd, Quantum algorithm for linear systems of equations, [Phys. Rev. Lett.](#) **103**, 150502 (2009).
- [13] Jessie M. Henderson, John Kath, John K. Golden, Allon G. Percus, and Daniel O'Malley, Addressing quantum's "fine print": State preparation and information extraction for quantum algorithms and geologic fracture networks, [Sci. Rep.](#) **14**, 3592 (2024).
- [14] Christoph Sünderhauf, Earl Campbell, and Joan Camps, Block-encoding structured matrices for data input in quantum computing, [Quantum](#) **8**, 1226 (2024).
- [15] J. D. Hyman, G Aldrich, H Viswanathan, Natalia Makedonska, and Satish Karra, Fracture size and transmissivity correlations: Implications for transport simulations in sparse three-dimensional discrete fracture networks following a truncated power law distribution of fracture size, [Water Resour. Res.](#) **52**, 6472 (2016).
- [16] Thomas Lubinski, Sonika Johri, Paul Varosy, Jeremiah Coleman, Luning Zhao, Jason Necaie, Charles Baldwin, Karl Mayer, and Timothy Proctor, Application-oriented performance benchmarks for quantum computing, [IEEE Trans. Quantum Eng.](#) **4**, 1 (2023).
- [17] Maxime Remaud, in *Proceedings of Recent Advances in Quantum Computing and Technology, ReAQCT '24* (Association for Computing Machinery, New York, NY, USA, 2024), pp. 56–61.
- [18] Jefferson D. S. Silva and Adenilton J. da Silva, Logarithmic depth decomposition of approximate multi-controlled single-qubit gates without ancilla qubits, [arXiv:2507.00400](#) [quant-ph].
- [19] Sarah Greer, Jeffrey Hyman, and Daniel O'Malley, A comparison of linear solvers for resolving flow in three-dimensional discrete fracture networks, [Water Resour. Res.](#) **58**, e2021WR031188 (2022).
- [20] Frank Arute, Kunal Arya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Rami Barends, Rupak Biswas, Sergio Boixo, Fernando G. S. L. Brandao, David A. Buell *et al.*, Quantum supremacy using a programmable superconducting processor, [Nature](#) **574**, 505 (2019).
- [21] Google Quantum AI and Collaborators, Quantum error correction below the surface code threshold, [Nature](#) **638**, 920 (2025).
- [22] L. Riesebo, X. Fu, S. Varsamopoulos, C. G. Almudever, and K. Bertels, in *Proceedings of the 54th Annual Design Automation Conference 2017, DAC '17* (Association for Computing Machinery, New York, NY, USA, 2017).
- [23] Philippe Davy, Olivier Bour, J.-R. De Dreuzy, and Caroline Darcel, Flow in multiscale fractal fracture networks, [Geological Society, London, Special Publications](#) **261**, 31 (2006).
- [24] Alireza Jafari and Tayfun Babadagli, Estimation of equivalent fracture network permeability using fractal and statistical network properties, [J. Petrol. Sci. Eng.](#) **92**, 110 (2012).
- [25] Austin Pechan, Code for: Block encoding the 3D heterogeneous Poisson equation with application to fracture flow, <https://github.com/Austin-Pechan/Block-encoding-the-3D-heterogeneous-Poisson-equation-with-application-to-fracture-flow>, 2024.