frontiers in Physics

ORIGINAL RESEARCH
published: 06 November 2020
doi: 10.3389/fphy.2020.00265

Floating-Point Calculations on a Quantum Annealer: Division and Matrix Inversion

Michael L. Rogers[1*†] *and Robert L. Singleton Jr.*[2*†]

[1] SavantX, Santa Fe, NM, United States, [2] School of Mathematics, University of Leeds, Leeds, United Kingdom

Systems of linear equations are employed almost universally across a wide range of disciplines, from physics and engineering to biology, chemistry, and statistics. Traditional solution methods such as Gaussian elimination are very time consuming for large matrices, and more efficient computational methods are desired. In the twilight of Moore's Law, quantum computing is perhaps the most direct path out of the darkness. There are two complementary paradigms for quantum computing, namely, circuit-based systems and quantum annealers. In this paper, we express floating point operations, such as division and matrix inversion, in terms of a *quadratic unconstrained binary optimization* (QUBO) problem, a formulation that is ideal for a quantum annealer. We first address floating point division, and then move on to matrix inversion. We provide a general algorithm for any number of dimensions, and, as a proof-of-principle, we demonstrates results from the D-Wave quantum annealer for 2×2 and 3×3 general matrices. In principle, our algorithm scales to very large numbers of linear equations; however, in practice the number is limited by the connectivity and dynamic range of the machine.

Keywords: quantum computing, matrix inversion, quantum annealing algorithm, linear algebra algorithms, D-wave

OPEN ACCESS

Edited by:
Marcelo Silva Sarandy, Fluminense Federal University, Brazil

Reviewed by:
Marcos César de Oliveira, Campinas State University, Brazil
Alexandre M. Souza, Centro Brasileiro de Pesquisas Físicas, Brazil

***Correspondence:**
Robert L. Singleton Jr.
r.singleton@leeds.ac.uk
Michael L. Rogers
michael.rogers@savantx.com

[†] These authors have contributed equally to this work

Specialty section:
This article was submitted to Quantum Computing, a section of the journal Frontiers in Physics

Received: *30 January 2019*
Accepted: *12 June 2020*
Published: *06 November 2020*

Citation:
Rogers ML and Singleton RL Jr (2020) Floating-Point Calculations on a Quantum Annealer: Division and Matrix Inversion. Front. Phys. 8:265. doi: 10.3389/fphy.2020.00265

1. INTRODUCTION

Systems of linear equations are employed almost universally across a wide range of disciplines, from physics and engineering to biology, chemistry, and statistics. An interesting physics application is computational fluid dynamics (CFD), which requires inverting very large matrices to advance the state of the hydrodynamic system from one time step to the next. An application of importance in biology and chemistry would include the protein folding problem. For large matrices, Gaussian elimination and other standard techniques becomes too time consuming, and faster computational methods are therefore desired. As Moore's Law draws to a close, quantum computing offers the most direct path forward; it is also perhaps the most radical path. In a nutshell, quantum computers are physical systems that exploit the laws of quantum mechanics to perform arithmetic and logical operations much faster than a conventional computer. In the words of Harrow, Hassidim, and Lloyd (HHL) [1], "quantum computers are devices that harness quantum mechanics to perform computations in ways that classical computers cannot." There are currently two complementary paradigms for quantum computing, namely, circuit-based systems and quantum annealers. Circuit-based systems exploit the deeper properties of quantum mechanics such coherence, entanglement, and non-locality, while quantum annealers mainly take advantage of tunneling between metastable states and the ground state. In [1], HHL introduces a circuit-based method by which the inverse of a matrix can be computed, and [2, 3] provide implementations of the algorithm to invert 2×2 matrices. Circuit-based methods are limited by the relatively small number of qubits that can be

entangled into a fully coherent quantum state, currently of order 50 or so. An alternative approach to quantum computing is the quantum annealer [4], which takes advantage of quantum tunneling between metastable states and the ground state. The D-Wave Quantum Annealers have reached capacities of 2000+ qubits, which suggests that quantum annealers could be quite effective for linear algebra with hundreds to thousands of degrees of freedom. In this paper, we express floating point operations such as division and matrix inversion as *quadratic unconstrained binary optimization* (QUBO) problems, which are ideal for a quantum annealer. We should mention that our algorithm provides the full solution the matrix problem, while HHL provides only an expectation value. Furthermore, our algorithm places no constraints on the matrix that we are inverting, such as a sparsity condition.

The first step in mapping a general problem to a QUBO problem begins with constructing a Hamiltonian that encodes the problem in terms of a set of "logical" qubits. Next, because of the limited connectivity of the D-Wave chip, it will be necessary to "embed" the problem onto the chip, first by mapping each logical qubit to a collection or "chain" of physical qubits and then by determining parameter settings for all the physical qubits, including the chain couplings. We have implemented our algorithms on the D-Wave 2000Q and 2X chips, illustrating that division and matrix inversion can indeed be performed on an existing quantum annealer. The algorithms that we propose should ideally scale well for large numbers of equations, and should be applicable to a matrix inversion of relatively high order (although probably not exponentially higher order as in HHL). Currently, the scaling that may be achieved is limited by the connectivity and dynamic range of the chip.

Before examining the various algorithms, it is useful to review the basic formalism and to establish some notation. The general problem starts with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the vertex set and $\mathcal{E}$ is the edge set. The QUBO *Hamiltonian* on $\mathcal{G}$ is defined by

$$H_{\mathcal{G}}[Q] \;=\; \sum_{r\in\mathcal{V}} A_r\, Q_r + \sum_{rs\in\mathcal{E}} B_{rs}\, Q_r Q_s \tag{1.1}$$

with $Q_r \in \{0, 1\}$ for all $r \in \mathcal{V}$. The coefficient A_r is called the *weight* at vertex r, while the coefficient B_{rs} is called the *strength* between vertices r and s. It might be better to call (1.1) the *objective function* rather than the Hamiltonian, as $H_{\mathcal{G}}$ is a real-valued function and not an operator on a Hilbert-space. However, it is easy to map (1.1) in an equivalent Hilbert space form,

$$\hat{H}_{\mathcal{G}} \;=\; \sum_{r\in\mathcal{V}} A_r\, \hat{Q}_r + \sum_{rs\in\mathcal{E}} B_{rs}\, \hat{Q}_r \hat{Q}_s \tag{1.2}$$

where $\hat{Q}_r|Q\rangle \;=\; Q_r|Q\rangle$ for all $r \in \mathcal{V}$, and $|Q\rangle \in \mathcal{H}$ for Hilbert space $\mathcal{H}$. The hat denotes an operator on the Hilbert space, and Q_r is the corresponding Eigenvalue of $\hat{Q}_r$ with Eigenstate $|Q\rangle$. Consequently, we can write

$$\hat{H}_{\mathcal{G}}|Q\rangle = H_{\mathcal{G}}[Q]\,|Q\rangle \tag{1.3}$$

and we use the terms *Hamiltonian* and *objective function* interchangeably. By the *QUBO problem*, we mean the problem of finding the lowest energy state $|Q\rangle$ of the Hamiltonian (1.2), which corresponds to minimizing Equation (1.1) with respect to the Q_r. This is an NP-hard problem uniquely suited to a quantum annealer. Rather than sampling all $2^{\#\mathcal{V}}$ possible states, quantum tunneling finds the *most likely* path to the ground state by minimizing the Euclidian action. In the case of the D-Wave 2X chip, the number of distinct quantum states is of order the very large number 2^{1000}, and the ground state is selected from this jungle of quantum states by tunneling to those states with a smaller Euclidean action.

The Ising model [5] is perhaps the quintessential physical example of a QUBO problem, and, indeed, it is one of the most studied systems in statistical physics. The Ising model consists of a square lattice of spin-1/2 particles with nearest neighbor spin–spin interactions between sites r and s, and when the system is immersed in a nonuniform magnetic field, this introduces coupling terms at individual sites r, thereby producing a Hamiltonian of the form

$$H_{\mathcal{G}}[J] \;=\; \sum_{r\in\mathcal{V}} B_r\, \Sigma_r + \sum_{rs\in\mathcal{E}} J_{rs}\, \Sigma_r \Sigma_s \tag{1.4}$$

where $\Sigma_r = \pm 1/2$. The Ising problem is connected to the QUBO problem by $\Sigma_r = Q_r - 1/2$.

For floating point division to R bits of resolution, the graph $\mathcal{G}$ is in fact just the fully connected graph $K_{\rm R}$. In terms of vertex and edge sets, we write $K_{\rm R} = (\mathcal{V}_{\rm R}, \mathcal{E}_{\rm R})$, and **Figure 1** illustrates K_8 and K_4. The left panel shows the completely connected graph K_8, with vertex and edge sets

$$\mathcal{V}_8 \;=\; \{0, 1, 2, \cdots, 7\} \tag{1.5}$$

$$\mathcal{E}_8 \;=\; \{\{0,1\},\{0,2\},\cdots,\{0,7\},\{1,2\},\cdots,\{1,7\},\cdots,\{6,7\}\} \tag{1.6}$$

while the right panel shows the K_4 graph,

$$\mathcal{V}_4 \;=\; \{0, 1, 2, 3\} \tag{1.7}$$

$$\mathcal{E}_4 \;=\; \{\{0,1\},\{0,2\},\{0,3\},\{1,2\},\{1,3\},\{2,3\}\}. \tag{1.8}$$

Just as 8-bit is called a *word*, 4-bit is called a *nibble*. As we will also see, the dynamic range of the D-Wave is most directly suitable to K_4, and the connectivity of K_4 consequently gives a quantum nibble.

Let us remark about our summation conventions. Rather than summing over the edges,

$$H[Q] \;=\; \sum_{r\in\mathcal{V}_{\rm R}} A_r\, Q_r + \sum_{rs\in\mathcal{E}_{\rm R}} B_{rs}\, Q_r Q_s \tag{1.9}$$

$$\;=\; \sum_{r=0}^{R-1} A_r\, Q_r + \sum_{r=0}^{R-1}\sum_{s>r}^{R-1} B_{rs}\, Q_r Q_s \tag{1.10}$$

we find it convenient to sum over all values of r and s taking B_{rs} to be symmetric. In this case, the double sum differs by a factor of two relative to summing over the edge set of the graph,

$$H[Q] \;=\; \sum_{r=0}^{R-1} A_r\, Q_r + \sum_{r=0}^{R-1}\sum_{s=0}^{R-1} \frac{1}{2}\, B_{rs}\, Q_r Q_s. \tag{1.11}$$

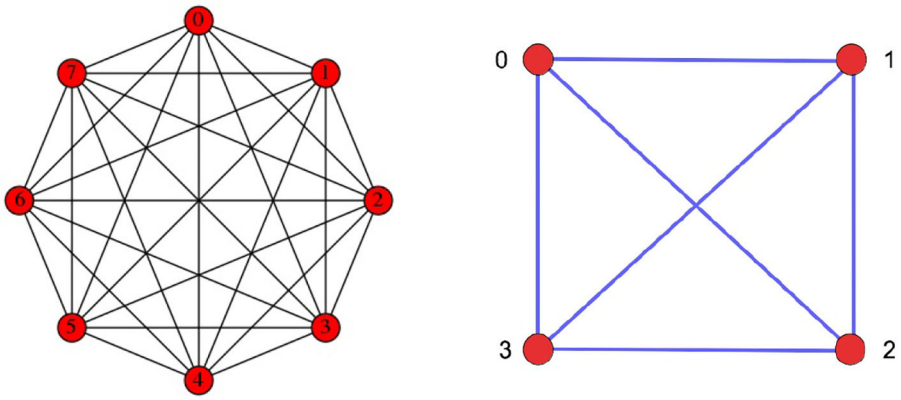

FIGURE 1 | The left panel shows the fully connected graph K_8, and the right panel shows the corresponding graph K_4. To perform a calculation to 8-bit accuracy requires the connectivity of K_8. We take the vertex and edge sets of K_8 to be $\mathcal{V}_8 = \{0, 1, 2, \cdots, 7\}$ and $\mathcal{E}_8 = \{\{0, 1\}, \{0, 2\}, \cdots, \{0, 7\}, \{1, 2\}, \{1, 3\}, \cdots, \{6, 7\}\}$. To perform a calculation to 4-bit accuracy requires K_4 connectivity, and, similarly, the vertex and edge sets for K_4 are $\mathcal{V}_4 = \{0, 1, 2, 3\}$ and $\mathcal{E}_4 = \{\{0, 1\}, \{0, 2\}, \{0, 3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$.

Furthermore, for $r = s$, there will be a linear contribution from the idempotency condition $Q_r^2 = Q_r$, so that

$$H[Q] = \sum_{r=0}^{R-1} \left[A_r + \frac{1}{2} B_{rr} \right] Q_r + \sum_{r=0}^{R-1} \sum_{s \neq r, s=0}^{R-1} \frac{1}{2} B_{rs} Q_r Q_s. \quad (1.12)$$

We can write this as

$$H[Q] = \sum_{r=0}^{R-1} \tilde{A}_r Q_r + \sum_{r=0}^{R-1} \sum_{s \neq r, s=0}^{R-1} \tilde{B}_{rs} Q_r Q_s. \quad (1.13)$$

2. FLOATING POINT DIVISION ON A QUANTUM ANNEALER

2.1. Division as a QUBO Problem

In this section we present an algorithm for performing floating point division on a quantum annealer. Given two input parameters m and y to R-bits of resolution, the algorithm calculates the ratio y/m to R bits of resolution. The corresponding division problem can be represented by the linear equation

$$m \cdot x - y = 0, \quad (2.1)$$

which has the unique solution

$$x = y/m. \quad (2.2)$$

Solving (2.1) on a quantum annealer amounts to finding an objective function $H(x)$ whose minimum corresponds to the solution of Equation (2.2). Although the form of $H(x)$ is not unique, for this work we employ the simple real-valued quadratic function

$$H(x; m, y) = \left(m x - y\right)^2 \quad (2.3)$$

where m and y are continuous parameters. For an ideal annealer, we do not have to concern ourselves with the numerical range and resolution of the parameters m and y; however, for a real machine such as the D-Wave, this is an important consideration. For a well-conditioned matrix, we require that the parameters m and y possess a numerical range that spans about an order of magnitude, from approximately 0.1–1.0. This provides about 3–4 bits of resolution: $1/2^0 = 1$, $1/2^1 = 0.5$, $1/2^2 = 0.25$, and $1/2^3 = 0.125$. The dynamic range and the connectivity both impact the resolution of a calculation.

To proceed, let us formulate floating point division as a *quadratic unconstrained binary optimization* (QUBO) problem. The algorithm starts by converting the real-valued number x in (2.3) into an R-bit binary format, while the numbers m and y remain real valued parameters of the objective function. For any number $\chi \in [0, 2)$, the binary representation accurate to R bits of resolution can be expressed by $[Q_0.Q_1 Q_2 \cdots Q_{R-1}]_2$, where $Q_r \in \{0, 1\}$ is value of the r-th bit, and the square bracket indicates the binary representation[1]. It is more algebraically useful to express this in terms of the power series in 2^{-r},

$$\chi = \sum_{r=0}^{R-1} 2^{-r} Q_r. \quad (2.4)$$

In order to represent negative number, we perform the binary offset

$$x = 2\chi - 1 \quad (2.5)$$

where $x \in [-1, 3)$. The objective function now takes the form

$$H(\chi) = 4m^2\chi^2 - 4m(m + y)\chi + (m + y)^2. \quad (2.6)$$

The constant term $(m + y)^2$ can be dropped when finding the minimum of (2.6), but we choose to keep it for completeness.

[1]Since the infinite geometric series $\sum_{r=0}^{\infty} 2^{-r}$ sums to 2, the finite series is <2. In binary form we have $[1.11\cdots]_2 = 2$ and $[1.11\cdots 1]_2 < 2$. Working to resolution R is like calculating the R-th partial sum of an infinite series.

Equation (2.4) provides a change of variables $\chi = \chi[Q]$ (where Q is the collection of the Q_r), and this allows us to express (2.3) in the form

$$H[Q] = \sum_{r=0}^{R-1} A_r\, Q_r + \sum_{r=0}^{R-1} \sum_{s\neq r, s=0}^{R-1} B_{rs}\, Q_r Q_s. \tag{2.7}$$

In the notation of graph theory, we would write

$$H[Q] = \sum_{r\in\mathcal{V}_R} A_r\, Q_r + \sum_{rs\in\mathcal{E}_R} \frac{1}{2}\, B_{rs}\, Q_r Q_s \tag{2.8}$$

where $\mathcal{V}_R = \{0, 1, 2, \cdots, R-1\}$ is the vertex set, and $\mathcal{E}_R$ is the edge set. We often employ an abuse of notation and write $rs \in \mathcal{E}_R$ to mean $\{r, s\} \in \mathcal{E}_R$. Thus, instead of $B_{\{r,s\}}$, we write B_{rs}. Since the order of the various elements of a set are immaterial, we require B_{rs} to be symmetric in r and s. Rather than summing over the edge sets $rs \in \mathcal{E}_R$, we employ the double sum $\sum_{r\neq s}$, which introduces a relative factor of two in the convention for the strengths B_{rs}. The goal of this section is to find A_r and B_{rs} in terms of m and y.

Note that we can generalize the simple binary offset (2.4) if we scale and shift $\chi \in [0, 2)$ by

$$x = c\chi - d \tag{2.9}$$

so that $x \in [-d, 2c - d)$. When $d > 0$ and $c > d/2$, the domain of x always contains a positive and negative region, and the precise values for d and c can be chosen based on the specifics of the problem. For Equation (2.9), the objective function takes the form

$$H(\chi) = 4m^2c^2\,\chi^2 - 4mc\,(m + y)\chi + (md + y)^2. \tag{2.10}$$

For simplicity of notation, this paper employs the simple binary offset (2.5), although our Python interface to the D-Wave quantum annealer employs the generalized form (2.10).

Equation (2.4) allows us to express the quadratic term in χ as

$$\chi^2 = \sum_{r=0}^{R-1}\sum_{s=0}^{R-1} 2^{-r-s} Q_r Q_s = \sum_{r=0}^{R-1}\sum_{s\neq r, s=0}^{R-1} 2^{-r-s} Q_r Q_s + \sum_{r=0}^{R-1} 2^{-2r} Q_r \tag{2.11}$$

where we have used the idempotency condition $Q_r^2 = Q_r$ along the diagonal in the last term of (2.11). Substituting the forms (2.4) and (2.11) into (2.6) yields the Hamiltonian

$$\begin{aligned} H[Q] = &\sum_{r=0}^{R-1} 4m\, 2^{-r}\Big[m\, 2^{-r} - (y + m)\Big] Q_r \\ &+ \sum_{r=0}^{R} \sum_{s\neq r, s=0}^{R-1} 4m^2\, 2^{-r-s}\, Q_r Q_s \end{aligned} \tag{2.12}$$

and the Ising coefficients in (2.7) can be read off:

$$A_r = 4m\, 2^{-r}\Big[m\, 2^{-r} - (y + m)\Big] \tag{2.13}$$

$$B_{rs} = 4m^2\, 2^{-r-s} \quad r \neq s. \tag{2.14}$$

Because of the double sum over r and s in the objective function in (2.12), the algorithm requires a graph of connectivity K_R. The special cases of K_8 and K_4 have been illustrated in **Figure 1**. To obtain higher accuracy than the K_R graph allows, we can iterate this procedure in the following manner. Suppose we start with $y_0 = y$ and are given a value y_{n-1} with $n > 1$; we then advance the iteration to y_n in the following manner:

$$\text{solve } mx_n = y_{n-1} \text{ for } x_n \text{ to } R \text{ bits} \tag{2.15}$$

$$\text{calculate the error } y_n = y_{n-1} - mx_n. \tag{2.16}$$

Now that we have the value y_n, we can repeat the process to find y_{n+1}, and we can stop the iterative procedure when the desired level of accuracy has been achieved.

2.2. Embedding K_R Onto the D-Wave Chimera Architecture

The D-Wave Chimera chip consists of coupled bilayers of micro rf-SQUIDs overlaid in such a way that, while relatively easy to fabricate, results in a fairly limited set of physical connections between the qubits. However, by *chaining* together well chosen qubits in a positively correlated manner, this limitation can largely be overcome. The process of chaining requires that we (i) embed the logical graph onto the physical graph of the chip (for example K_4 into C_8) and that we (ii) assign weights and strengths to the physical graph embedding in such as a way as to preserve the ground state of the logical system. These steps are called graph embedding and Hamiltonian embedding, respectively.

Let us explore the connectivity of the D-Wave Chimera chip in more detail. The D-Wave architecture employs the C_8 bipartate Chimera graph as its most basic unit of connectivity. This *unit cell* is illustrated in **Figure 2**, and it consists of 8 qubits connected in a 4×4 bipartate manner. The left panel of the figure uses a *column* format in laying out the qubits, and the right panel illustrates the corresponding qubits in a *cross* format, where the gray lines represent the direct connections between the qubits. The cross format is useful since it minimizes the number intersecting connections. The complete two-dimensional chip is produced by replicating C_8 along the vertical and horizontal directions, as illustrated in **Figure 3**, thereby providing a chip with thousands of qubits. The connections between qubits are limited in two ways: (i) by the connectivity of the basic unit cell C_8 and (ii) by the connectivity between the unit cells across the chip. The bipartate graph $C_8 = (\mathcal{V}_8, \mathcal{B}_8)$ is formally defined by the vertex set $\mathcal{V}_8 = \{1, 2, \cdots, 8\}$, and the edge set

$$\begin{aligned} \mathcal{B}_8 = \big\{&\{1,5\},\{1,6\},\{1,7\},\{1,8\},\{2,5\},\{2,6\},\{2,7\},\{2,8\},\\ &\{3,5\},\{3,6\},\{3,7\},\{3,8\},\{4,5\},\{4,6\},\{4,7\},\{4,8\}\big\}. \end{aligned} \tag{2.17}$$

The set $\mathcal{B}_8$ represents the connections between a given red qubit and the corresponding blue qubits in the figures. The red and blue dots illustrate the bipartate nature of C_8, as every red dot is connected to every blue dot, while none of the blue and red dots are connected to one another.

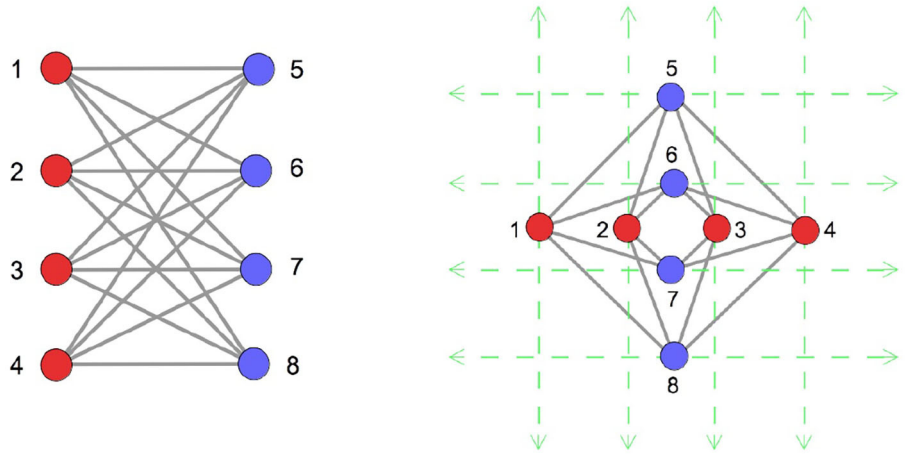

FIGURE 2 | The left panel illustrates the bipartate graph C_8 in *column* format, while the right panel illustrates the corresponding graph in *cross* format, often called a Chimera graph. The gray lines represent direct connections between qubits. The cross format is useful since it minimizes the number intersecting connections. The use of red and blue dots emphasize the bipartate nature of C_8, as every red dot is connected to every blue dot, while none of the red and blue dots are connected to one another. The vertex set of C_8 is taken to be $\mathcal{V}_8 = \{1, 2, \cdots, 8\}$ and edge set is $\mathcal{B}_8 = \{\{1,5\}, \{1,6\}, \{1,7\}, \{1,8\}, \{2,5\}, \{2,6\} \cdots \{7,8\}\}$.

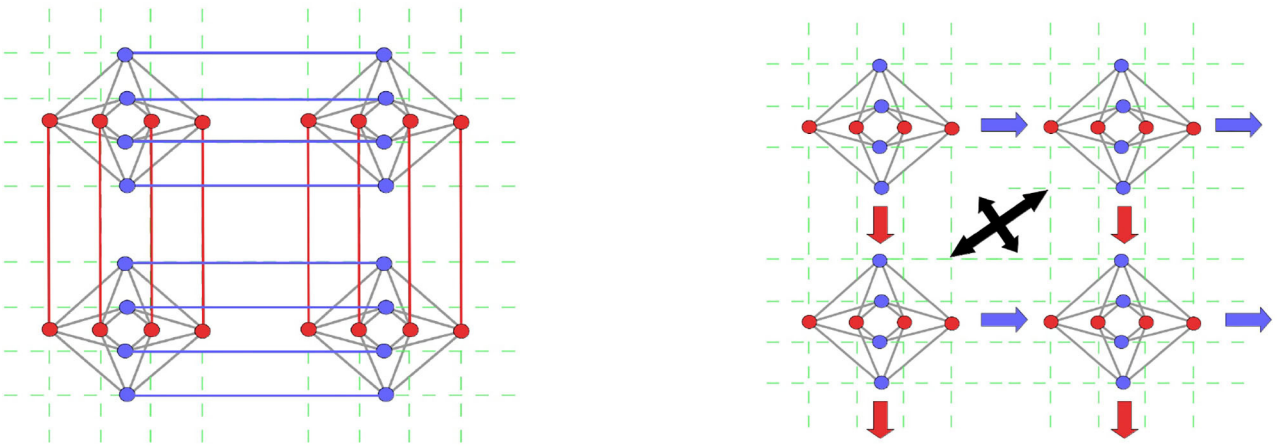

FIGURE 3 | The left panel shows the connectivity between four C_8 bipartate Chimera zones, and the right panel illustrates how multiple C_8 graphs are stitched together along the vertical and horizontal directions to provide thousands of possible qubits. A limitation of this connectivity strategy is that red and blue zones cannot communicate directly with one another, as indicated by the black crossed arrows. The purpose of *chaining* is to allow communication between the red and blue qubits.

We will denote the *physical* qubits on the D-Wave chip by q_ℓ. For the D-Wave 2000Q there is a maximum of 2,048 qubits, while the D-Wave 2X has 1,152 qubits. For the example calculation in this text, we only use 10–50 qubits. The physical Hamiltonian or objective function takes the form

$$H[q] = \sum_{\ell} a_\ell \, q_\ell + \sum_{\ell \neq m} 2 b_{\ell m} \, q_\ell q_m \tag{2.18}$$

where we have introduced a factor of 2 in the strength to account for the symmetric summation over r and s. We will call the qubits Q_r of the previous section the *logical qubits*. To write a program for the D-Wave means finding an embedding of the problem for logical qubits onto the physical collection of qubits q_ℓ. If the connectivity of the Chimera graphs were large enough, then the logical qubits would coincide exactly with the physical qubits. However, since the graph C_8 possesses less connectivity than K_4, we must resort to chaining on the D-Wave, even for 4-bit resolution. **Figure 4** illustrates the K_4 embedding used by our algorithm, where, as before, the left panel illustrates the bipartate graph in column format, and the right panel illustrates the corresponding graph in cross format.

In **Figure 4**, we have labeled the physical qubits by $\ell = 1, 2, 3 \cdots 8$, and we wish to map the problem involving logical qubits $Q_r \, Q_s \, Q_t$ onto the four physical qubits $q_5 \, q_1 \, q_6 \, q_2$. The embedding requires that we *chain* together the two qubits 1-6 and 3-8, respectively. We may omit qubits 4 and 7 entirely. As illustrated in **Figure 5**, the physical qubits q_1 and q_6 are *chained*

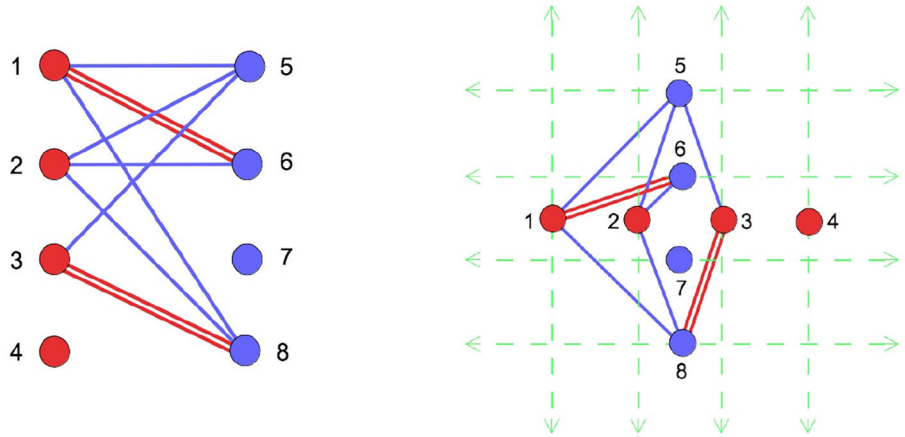

FIGURE 4 | The K_4 embedding onto C_8 used in our implementation of 4-bit of division on the D-Wave. The blue lines represent normal connections between qubits, while the red double-lines represent chained qubits, that is to say, qubits that are strictly correlated (and can thereby represent a single logical qubit at a higher level of abstraction). The qubits 1–6 are chained together, as are the qubits 3–8.

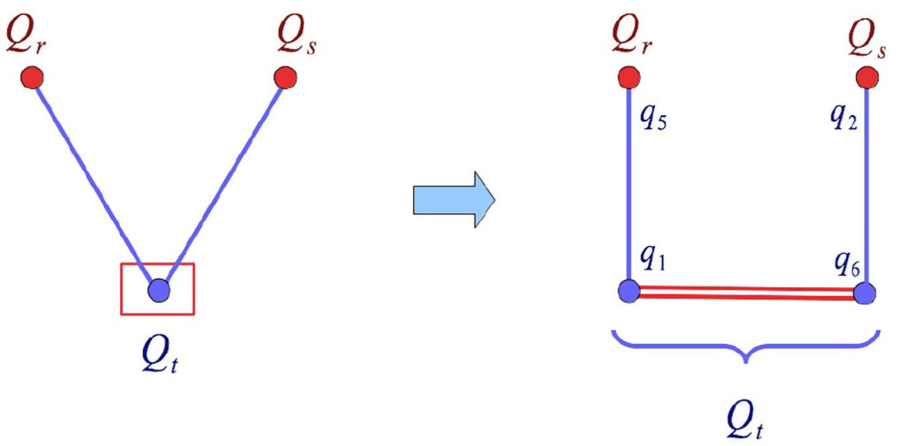

FIGURE 5 | The left panel shows three logical qubits Q_r, Q_s, Q_t with connectivity between r-t and t-s. The box surrounding qubit t means that it will be modeled by a linear chain of physical qubits, as illustrated in the right panel. The labeling is taken from **Figure 4** for qubits 5-1-6-2, where Q_r is mapped to q_5, Q_s is mapped to q_2, and Q_t is split between q_1 nd q_6. Qubits q_1 and q_6 are chained together to simulate the single logical qubit Q_t, while qubits Q_r and Q_s map directly onto physical qubits q_5 and q_2.

together to simulate a single logical qubit Q_t, while qubits q_5 and q_2 are mapped directly to the logical qubits Q_r and Q_s, respectively. Qubit q_5 is assigned the weight $a_5 = A_r$ and the coupling between q_5 and q_1 is assigned the value $b_{51} = B_{rt}$. Similarly for qubit q_2, the vertex is assigned weight $a_2 = A_s$, and strength between q_2 and q_6 is $b_{26} = B_{st}$. We must now distribute the logical qubit Q_t between q_1 and q_6 by assigning the values a_1, a_6 and b_{16}. We distribute the weight A_t uniformly between qubits q_1 and q_2, giving $a_1 = A_t/2$ and $a_6 = A_t/2$. We must now choose b_{16}. To preserve the energy spectrum, we must shift the values of the weights a_1 and a_6. We can do this by adding a counter-term Hamiltonian

$$H^{\mathrm{CT}} = a\, q_1 + a\, q_6 + 2b_{16}\, q_1 q_6 \tag{2.19}$$

to the physical Hamiltonian. The double lines in **Figures 4**, **5** indicate that two qubits are chained together. This means that the qubits are strictly correlated, i.e., when q_1 is up then q_6 is up, and when q_1 is down then q_6 is down. This is accomplished by choosing the coupling strength b_{16} to favor a strict correlation; however, to preserve the ground state energy, this also requires shifting the weights for q_1 and q_6. For $q_1 = q_6 = 0$ we have $H^{\mathrm{CT}} = 0$. We wish to preserve this condition when $q_1 = q_6 = 1$, which means $2a + 2b = 0$. Furthermore, the state $q_1 = 1$ and $q_6 = 0$ must have positive energy, which means $a > 0$. Similarly for $q_1 = 0$ and $q_6 = 1$. We therefore choose $a_1 = a_6 = \alpha > 0$ and $b_{16} = -\alpha$, where α is an arbitrary parameter. This is illustrated in **Table 1**. A more complicated case is the linear chain of N qubits as shown in **Figure 6**. The counter-term Hamiltonian

is taken to be

$$H^{\rm CT} = \sum_{m=1}^{N} a_m^t\, q_m^t + \sum_{m=1}^{N-1} b_{m,m+1}^t\, q_m^t q_{m+1}^t. \tag{2.20}$$

Note that $H^{\rm CT}$ vanishes when $q_m = 0$ for all $m = 1 \cdots N$. And conversely, we must arrange the counter-term to vanish when $q_m = 1$. The simplest choice is to take all weights to be the same and all couplings to be identical. Then, to preserve the ground state when the $q_r = 1$, we impose

$$a_r^t = \frac{A_t}{N} + \frac{2(N-1)}{N}\,\alpha \tag{2.21}$$

$$b_{r,r+1}^t = -\alpha \tag{2.22}$$

with $\alpha > 0$ and $r = 1 \cdots N$. The first term in a_r^t distributes the weight A_t uniformly across all N nodes in the chain. The second set of terms $b_{r,r+1}^t$ ensures that the qubits of the chain are strictly correlated. The counter-term energy contribution is positive when the linearly chained qubits are not correlated, therefore anti-correlation is always penalized. **Table 2** illustrates the spectrum of the counter-term Hamiltonian for three qubits. We may need to choose large values of α, of order 20 or more, to sufficiently separate the states. The uniform spectrum of 4 states with $H^{\rm CT} = a$ in **Table 2** arises from a permutation symmetry in q_1, q_2, q_3.

To review, note that a linear counter-term is represented in **Figure 6**. We add a counter-term to break the logical qubits into a chain of physical qubits that preserve the ground state. Let us consider the conditions that we place on the Hamiltonian to ensure strict correlation between the chained qubits. We adjust the values of A_r and B_{rs} to ensure that spin alignment is energetically favorable. By slaving several qubits together, we can overcome the limitations of the Chimera connectivity. As a more complex example, consider the four logical qubits of **Figure 7** connected in a circular chain by strengths B_{12}, B_{24}, B_{43}, and B_{31}. Suppose the weights are A_1, A_2 A_3, and A_4. **Figure 8** provides an example in which each logical qubit is chained in a linear fashion to the physical qubits.

TABLE 1 | For two qubits the counter-term Hamiltonian is $H^{\rm CT}(q_1, q_6) = a\,q_1 + a\,q_6 + 2b\,q_1 q_6$.

q_1	q_6	$H^{\rm CT}$
0	0	0
0	1	α
1	0	α
1	1	0

The lowest energy state is preserved for $b = -\alpha$ and $a = \alpha$ where $\alpha > 0$. We will split the weight A_t uniformly across the N chained physical qubits, thereby giving a contribution to the physical Hamiltonian $H_{16}^t = A_t/2 + \alpha\, q_1 + \alpha\, q_6 - 2\alpha\, q_1 q_6$. The energy spectrum ensures that the two qubits are strictly correlated.

3. MATRIX INVERSION AS A QUBO PROBLEM

In this section we present an algorithm for solving a system of linear equations on a quantum annealer. To precisely define the mathematical problem, let $\boldsymbol{M}$ be a nonsingular $N \times N$ real matrix, and let $\mathbf{Y}$ be a real N dimensional vector; we then wish to solve the linear equation

$$\boldsymbol{M} \cdot \mathbf{x} = \mathbf{Y}. \tag{3.1}$$

The linearity of the system means that there is a unique solution,

$$\mathbf{x} = \boldsymbol{M}^{-1} \cdot \mathbf{Y} \tag{3.2}$$

and the algorithm is realized by specifying an objective function whose ground state is indeed (3.2). The objective function is not unique, although it must be commensurate with the architecture of the hardware. If the inverse matrix itself is required, it can be constructed by solving (3.1) for each of the N linearly independent basis vectors for $\mathbf{Y}$. It is easy to construct a quadratic objective $H(\mathbf{x})$ whose minimum is (3.2), namely,

$$H(\mathbf{x}) \;=\; \big(\boldsymbol{M}\mathbf{x} - \mathbf{Y}\big)^2 = \big(\boldsymbol{M}\mathbf{x} - \mathbf{Y}\big)^{\rm T} \cdot \big(\boldsymbol{M}\mathbf{x} - \mathbf{Y}\big). \tag{3.3}$$

In terms of matrix components, this can be written as

$$H(\mathbf{x}) \;=\; \mathbf{x}^{\rm T}\boldsymbol{M}^{\rm T}\boldsymbol{M}\mathbf{x} - \mathbf{x}^{\rm T}\boldsymbol{M}^{\rm T}\mathbf{Y} - \mathbf{Y}^{\rm T}\boldsymbol{M}\mathbf{x} + \mathbf{Y}^{\rm T}\mathbf{Y}$$

TABLE 2 | For a three qubit chain the counter-term Hamiltonian is $H^{\rm CT}(q_1, q_2, q_3) = a\,q_1 + a\,q_2 + a\,q_3 + 2b\,q_1 q_2 + 2b\,q_2 q_3$, where $a = 4\alpha/3$ and $b = -\alpha$.

q_1	q_2	q_3	$H^{\rm CT}$	
0	0	0	0	0
0	0	1	$4\alpha/3$	a
0	1	0	$4\alpha/3$	a
0	1	1	$2\alpha/3$	$a/2$
1	0	0	$4\alpha/3$	a
1	0	1	$8\alpha/3$	$2a$
1	1	0	$2\alpha/3$	a
1	1	0	0	0

The degeneracy in energy of value a arises from a permutation symmetry in $q_1 \to q_2 \to q_3$ that preserves the form of the counter-term Hamiltonian.

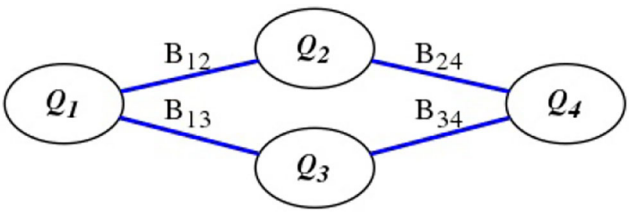

FIGURE 7 | Four logical qubits Q_1, Q_2, Q_3, Q_4 in a circular loop with connection strengths B_{12}, B_{24} B_{43}, and B_{31}.

$$= \sum_{ijk=1}^{N} M_{ki}M_{kj}\,x^i x^j - 2\sum_{ij=1}^{N} Y_j M_{ji}\,x^i + \|\mathbf{Y}\|^2. \quad (3.4)$$

Note that $\|\mathbf{Y}\|^2$ is just a constant, which will not affect the minimization. In principle all constants can be dropped from the objective function, although we choose to keep them for completeness. One may obtain a floating point representation of each component of $\mathbf{x} = (x^1, \cdots, x^N)^{\mathrm{T}}$ by expanding in powers of 2 multiplied by Boolean-valued variables $q_r^i \in \{0, 1\}$,

$$\chi^i = \sum_{r=0}^{R-1} 2^{-r} q_r^i \quad (3.5)$$

$$x^i = 2\chi^i - 1. \quad (3.6)$$

As before, the domains are given by $\chi^i \in [0, 2)$ and $x^i \in [-1, 3)$, and upon expressing $\mathbf{x}$ as a function q_r^i, we can recast (3.4) in the form

$$H[q] = \sum_{i=1}^{N}\sum_{r=0}^{R-1} a_r^i\, q_r^i + \sum_{i=1}^{N}\sum_{i\neq j=1}^{N}\sum_{r=0}^{R-1}\sum_{s=0}^{R-1} b_{rs}^{ij}\, q_r^i q_s^j. \quad (3.7)$$

The coefficients a_r^i are called the *weights* and the coefficients b_{rs}^{ij} are the interaction *strengths*. Note that the algorithm requires a connectivity of K_{NR} for arbitrary matrices.

Let us first calculate the product $x^i x^j$ in (3.4). From (3.5) and (3.6), we find

$$x^i x^j = \left(2\sum_{r=0}^{R-1} 2^{-r} q_r^i - 1\right)\left(2\sum_{r'=0}^{R-1} 2^{-r'} q_{r'}^i - 1\right)$$

$$= 4\sum_{rr'} 2^{-(r+r')} q_r^i q_{r'}^j - 4\sum_{r} 2^{-r} q_r^i + 1 \quad (3.8)$$

$$= 4\sum_{r\neq r'} 2^{-(r+r')} q_r^i q_{r'}^j + 4\sum_{r} 2^{-2r} q_r^i - 4\sum_{r} 2^{-r} q_r^i + 1 \quad (3.9)$$

where we have used the idempotency condition $(q_r^i)^2 = q_r^i$ in the second term of (3.9). While the second form is one used by the code, it is more convenient algebraically to use the first form. Substituting (3.8) into the first term in (3.4) gives

$$H_1 \equiv \sum_{ijk} M_{ki}M_{kj}\, x_i x_j \quad (3.10)$$

$$= \sum_{ijk} M_{ki}M_{kj}\left\{4\sum_{rr'} 2^{-(r+r')} q_r^i q_{r'}^j - 4\sum_{r} 2^{-r} q_r^i + 1\right\} \quad (3.11)$$

$$= 4\sum_{ir}\sum_{js}\sum_{k} 2^{-r-s} M_{ki}M_{kj}\, q_r^i q_s^j - 4\sum_{ir}\sum_{k} 2^{-r} M_{ki}M_{ki}\, q_r^i$$

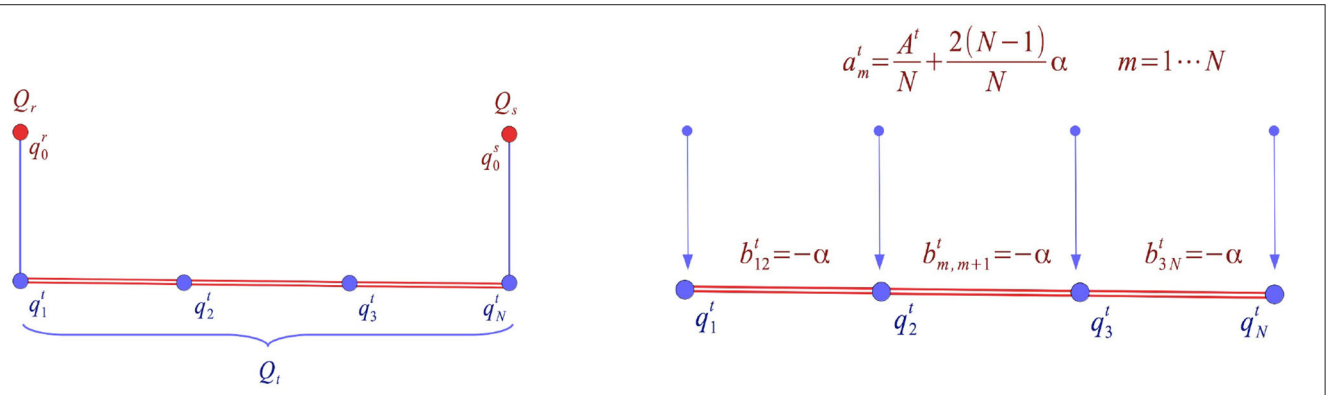

FIGURE 6 | Generalization of **Figure 5** to a chain of *N* linear qubits. The right panel illustrates the chain coupling parameters used to create strict correlations of the physical qubits within the chain.

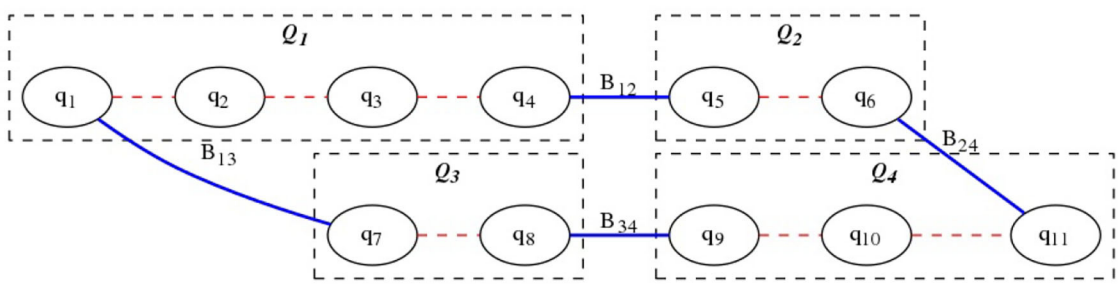

FIGURE 8 | A possible mapping of the logical qubits in **Figure 7** onto the physical device. Each logical qubit is modeled by a linear chain of strictly correlated qubits.

$$+ \sum_{ijk} M_{ki} M_{kj}. \tag{3.12}$$

The second term in (3.4) can be expressed as

$$H_2 \equiv -2 \sum_{ij} Y_j M_{ji}\, x_i = -2 \sum_{ij} Y_j M_{ji} \left(2 \sum_r 2^{-r} q_r^i - 1 \right) \tag{3.13}$$

$$= -4 \sum_{ij} \sum_r 2^{-r} M_{ji} Y_j\, q_r^i + 2 \sum_{ij} Y_j M_{ji}. \tag{3.14}$$

Adding H_1 and H_2 gives

$$H = 4 \sum_{ir} \sum_{js} \sum_k 2^{-r-s} M_{ki} M_{kj}\, q_r^i q_s^j - 4 \sum_{ir} \sum_k 2^{-r} M_{ki} M_{ki}\, q_r^i \tag{3.15}$$

$$- 4 \sum_{ij} \sum_r 2^{-r} M_{ji} Y_j\, q_r^i + 2 \sum_{ij} Y_j M_{ji} + \sum_{ijk} M_{ki} M_{kj}. \tag{3.16}$$

The QUBO coefficients for logical qubits are therefore

$$a_r^i = 4 \cdot 2^{-r} \sum_k M_{ki} \left\{ 2^{-r} M_{ki} - \left(Y_k + \sum_j M_{kj} \right) \right\} \tag{3.17}$$

$$b_{rs}^{ij} = 4 \cdot 2^{-(r+s)} \sum_k M_{ki}\, M_{kj}. \tag{3.18}$$

In the programming interface, the coefficients are addressed with a 1-dimensional linear index, while the parameters in 3.17 and 3.18 are defined in terms of the 2-dimensional indices i and r, where $i = 0, 1, \cdots, N-1$ and $r = 0, 1, \cdots, R-1$. Now, we define a 1-1 mapping between these indices and the linear index $\ell = 0, 1, \cdots, N \cdot R - 1$. This is just an ordinary linear indexing for 2-dimensional matrix elements, so we choose the usual row-major linear index mapping,

$$\ell(i, r) = i \cdot R + r \tag{3.19}$$

$$M_\ell = M_{ir}. \tag{3.20}$$

The inverse mapping gives the row and column indices as below,

$$i_\ell = \lfloor \ell / R \rfloor \tag{3.21}$$

$$r_\ell = \ell \bmod R \tag{3.22}$$

where $\lfloor n \rfloor$ is the greatest integer less than or equal to n. The expression "$\ell \bmod R$" is ℓ modulo R. This is a 1-1, invertible mapping between each pair of values of i and r in the matrix index space to every value of ℓ in the linear qubits index space. We can simply replace sums over all index pairs i, r by a single sum over ℓ, provided we also rewrite any isolated indices in i and r as functions of ℓ via their inverse mapping.

We may summarize this observation in the following formal identity. Given some arbitrary quantity, A, that depends functionally upon the tuple (i, r), and possibly upon the individual indices i and r, it is trivial to verify that

$$A[(i, r), i, r] = \sum_{\ell=0}^{N \cdot R - 1} A[\ell, i_\ell, r_\ell]\, \delta_{i, i_\ell} \delta_{r r_\ell} \tag{3.23}$$

where ℓ, i_ℓ, and r_ℓ are related as in Equations (3.19)–(3.22). This identity is useful for formal derivations. For example, we may use it to quickly derive the binary expansion of x_i in terms of logical qubits. Inserting (3.23) into (3.6) gives,

$$x_i = 2 \left(\sum_{r=0}^{R-1} 2^{-r} \sum_{\ell=0}^{N \cdot R - 1} q_\ell\, \delta_{i\, i_\ell} \delta_{r\, r_\ell} \right) - 1$$

$$= 2 \sum_{\ell=0}^{N \cdot R - 1} 2^{-r_\ell}\, q_\ell\, \delta_{i, i_\ell} - 1. \tag{3.24}$$

Clearly, x_i has non-zero contributions only for those indices corresponding to $i = i_\ell = \lfloor \ell / R \rfloor$, that is, only from those qubits within a row in the q_r^i array. Also, those contributions are summed along that row, i.e.,, over $r_\ell = \ell \bmod R$. This equation will be used to reconstruct the floating-point solution, $\mathbf{x}$, from the components q_ℓ of the binary solution returned from the D-Wave annealing runs. The weights and strengths now become

$$a_\ell = 4 \cdot 2^{-r_\ell} \sum_k M_{k\, i_\ell} \left\{ 2^{-r_\ell} M_{k\, i_\ell} - (Y_k + \sum_j M_{kj}) \right\} \tag{3.25}$$

$$b_{\ell m} = 4 \cdot 2^{-(r_\ell + r_{\ell'})} \sum_k M_{k\, i_\ell}\, M_{k\, i_m}. \tag{3.26}$$

For a 2×2 matrix to 4-bit accuracy, we need K_8 ($4 \times 2 = 8$), and to 8-bit accuracy we need K_{16} ($8 \times 2 = 16$). We have inverted matrices up to 3×3 to 4-bit accuracy, which requires K_{12} ($3 \times 4 = 12$). For an $N \times N$ matrix with R bits of resolutions, we must construct linear embeddings of K_{RN}. We could generalize this procedure for complex matrices.

4. CALCULATIONS

4.1. Implementation

The methods above were implemented using D-Wave's Python SAPI interface and tested on a large number of floating-point calculations. Initially, we performed floating-point division on simple test problems with a small resolution. Early on, we discovered that larger graph embeddings tended to produce noisier results. To better understand what was happening we started with a K_8 graph embedding to represent two floating-point numbers with only four bits of resolution. Since the D-Wave's dynamic range is limited to about a factor of 10 in the scale of the QUBO parameters, we determined that we could expect no more than 3–4 bits of resolution from any one calculation in any event. However, our binary offset representation (3.5) implies that we should expect no more than 3 bits of resolution in any single run. Indeed, using the K_8 embedding, we were able to get

exact solutions from the annealer for any division problems that had answers that were multiples of 0.25 between −1.0 and 1.0. Problems in this range that had solutions that were *not* exact multiples of 0.25 resulted in approximate solutions, effectively "rounded" to the nearest of ±0.25 or ±0.75. At this point we implemented an iterative scheme that uses the current error, or residual, as a new input, keeping track of the accumulated floating-point solution.

The iteration method has been implemented and tested for floating-point division, but we have not yet implemented iteration for matrix inversion. That can be done by using the previous residual (error) as the new inhomogeneous term in the matrix equation. We plan to implement an iterative method in the matrix inversion code eventually. However, we already have good preliminary results on matrix inversion that suggests that this should work reasonably well, at least for well-conditioned matrices. Currently, we are able to solve 2×2 and 3×3 linear equations involving floating-point numbers up to a resolution of 4 bits, and having well-conditioned matrices, *exactly* for input vectors with elements defined on $[-1, 1]$ and that are multiples of 0.25. Using an example matrix that is poorly-conditioned, we find that it is generally not possible to get the right answer without first doing some sort of preconditioning to the matrix. But, more importantly, we were able to obtain some insight about why ill-conditioned matrices can be difficult to solve as QUBO problems on a quantum annealer, which gives some hints about how to ameliorate the problem. We will discuss these results, and the effects of ill-conditioning on the QUBO matrix inverse problem below.

4.1.1. Note on Solution Normalization and Iteration

Allowing both the division and linear equation QUBO solvers to work for arbitrary floating-point numbers, and to allow for iterative techniques, requires normalizing the ratio of the current dividend and the divisor, or the residual and matrix, to a value in a range between −1 and 1. For the division problems, we wanted to avoid "dividing in order to divide," so we normalized each ratio using the difference between the binary exponents of ⌊divisor⌋ and ⌊dividend⌋. These can be found just by using order comparisons, with no explicit divisions. Adding 1 to this yields an "offset"—the largest binary exponent of the ratio—to within a factor of 2 (±1 in the offset), which is sufficient for scaling our QUBO parameters as needed. The fact that our QUBO solutions are always returned in binary representation provides a simple way to bound the solution into a range solvable with the annealer by simply shifting the binary representation of the current dividend by a few bits (using the current offset), which is why we refer to the solution exponent as an "offset." In this way, the solution can easily be guaranteed to be in the correct range without having to perform any divisions in Python. The "offset" is accumulated and used to construct the current approximation to the floating-point solution on each iteration. The iteration process continues until the error of the approximate solution is less than some tolerance specified by the user.

4.2. Results for Division

First, we present some examples for division without iteration. We used a K_4 graph embedding for expanding the unknown x up to a resolution of four bits. However, using the binary offset representation we can only get a true precisions of three bits. We solved the simple division problem,

$$x = \frac{y}{m}. \tag{4.1}$$

4.2.1. Basic Division Solver

Table 3 gives an extensive list of tested exact solutions returned by the floating-point annealing algorithm on the D-Wave machine using the K_4 graph embedding with an effective binary resolution of 3, corresponding to the multiples of 0.25 in the range $[-1, 1]$. The "Ground State" column lists the raw binary vector solutions, corresponding to the expansion in Equation (2.5). It is easy to check from Equations (2.5) and (2.4) that these give the floating-point solutions found in the corresponding "D-Wave Solution" column. In all of these cases, values of $\alpha \geq 0.5$ yielded the solution exactly; however, α is set to 20.0 here because that gives a better approximate solution for the inexact divisions, and faster convergence for the iterated divisions. It does not change the solutions for the exact cases.

Table 4 lists some illustrative division problems on $[-1, 1]$ that do not have solutions which are multiples of ±0.25, and they are therefore not solved exactly by the quantum annealing algorithm to 3 bits of resolution. Note that the energies are different for the ground states because the Hamiltonians are somewhat different for these problems. The "rounding"

TABLE 3 | Exact quantum annealed division problems to 3-bit resolution.

y	m	x, Exact	x, D-Wave	Ground state	Energy	α
DIVISION PROBLEMS WITH EXACT D-WAVE SOLUTIONS						
1.00	1.0	1.00	1.00	[1,0,0,0]	−2.0	20.0
0.50	0.5	1.00	1.00	[1,0,0,0]	−2.0	20.0
1.00	−1.0	−1.00	−1.00	[0,0,0,0]	0.0	20.0
−1.00	1.0	−1.00	−1.00	[0,0,0,0]	0.0	20.0
0.50	−0.5	−1.00	−1.00	[0,0,0,0]	0.0	20.0
−0.50	0.5	−1.00	−1.00	[0,0,0,0]	0.0	20.0
0.75	1.0	0.75	0.75	[0,1,1,1]	−1.53125	20.0
−0.75	1.0	−0.75	−0.75	[0,0,0,1]	−0.03125	20.0
0.75	−1.0	−0.75	−0.75	[0,0,0,1]	−0.03125	20.0
0.50	1.0	0.50	0.50	[0,1,1,0]	−1.125	20.0
−0.50	1.0	−0.50	−0.50	[0,0,1,0]	−0.125	20.0
0.50	−1.0	−0.50	−0.50	[0,0,1,0]	−0.125	20.0
0.25	1.0	0.25	0.25	[0,1,0,1]	−0.78125	20.0
−0.25	1.0	−0.25	−0.25	[0,0,1,1]	−0.28125	20.0
0.25	−1.0	−0.25	−0.25	[0,0,1,1]	−0.28125	20.0
0.25	0.5	0.50	0.50	[0,1,1,0]	−1.125	20.0
−0.25	0.5	−0.50	−0.50	[0,0,1,0]	−0.125	20.0
0.25	−0.5	−0.50	−0.50	[0,0,1,0]	−0.125	20.0
0.00	± 1.00	0.00	0.00	[0,1,0,0]	−0.5	20.0
0.00	± 0.75	0.00	0.00	[0,1,0,0]	−0.5	20.0
0.00	± 0.50	0.00	0.00	[0,1,0,0]	−0.5	20.0
0.00	± 0.25	0.00	0.00	[0,1,0,0]	−0.5	20.0

TABLE 4 | "Rounded" quantum annealed division solutions to 3-bit resolution.

y	m	x, Exact	x, D-Wave	Ground state	Energy	α
DIVISION PROBLEMS WITH APPROXIMATE D-WAVE SOLUTIONS						
0.90	1.0	0.90	1.00	[1,0,0,0]	−1.8	20.0
−0.90	1.0	−0.90	−1.00	[0,0,0,0]	0.0	20.0
0.80	1.0	0.80	0.75	[0,1,0,0]	−1.6875	20.0
−0.80	1.0	−0.80	−0.75	[0,0,0,1]	−0.01875	20.0
0.70	1.0	0.70	0.75	[0,1,0,0]	−1.44375	20.0
−0.70	1.0	−0.70	−0.75	[0,0,0,1]	−0.04374	20.0
0.60	1.0	0.60	0.50	[0,1,1,0]	−1.275	20.0
−0.60	1.0	−0.60	−0.50	[0,0,1,0]	−0.075	20.0
0.40	1.0	0.40	0.50	[0,1,1,0]	−0.975	20.0
−0.40	1.0	−0.40	−0.50	[0,0,1,0]	−0.175	20.0
0.30	1.0	0.30	0.25	[0,1,0,1]	−0.84375	20.0
−0.30	1.0	−0.30	−0.25	[0,0,1,1]	−0.24375	20.0
0.20	1.0	0.20	0.25	[0,1,0,1]	−0.71875	20.0
−0.20	1.0	−0.20	−0.25	[0,0,1,1]	−0.31875	20.0
0.10	1.0	0.10	0.00	[0,1,0,0]	−0.6	20.0
−0.10	1.0	−0.10	0.00	[0,0,1,1]	−0.4	20.0
0.30	0.9	$0.\bar{3}$	0.25	[0,1,0,1]	−0.88542	20.0
−0.30	0.9	$-0.\bar{3}$	−0.25	[0,0,1,1]	−0.21875	20.0
1.0	7.0	$0.14\bar{2}875$	0.25	[0,1,0,1]	−0.64732	20.0
−1.0	7.0	$-0.14\bar{2}875$	−0.25	[0,0,1,1]	−0.36161	20.0

TABLE 5 | Iterated quantum annealed division problems to resolution 1.0×10^{-6}.

y	m	x, Exact	x, D-Wave	α	Iterations
ITERATED DIVISION PROBLEMS ON THE D-WAVE ANNEALER					
0.25	1.0	0.25	0.25	20.0	1
−0.25	1.0	−0.25	−0.25	20.0	1
0.50	1.0	0.50	0.50	20.0	1
−0.50	1.0	−0.50	−0.50	20.0	1
0.75	1.0	0.75	0.75	20.0	1
−0.75	1.0	−0.75	−0.75	20.0	1
0.80	1.0	0.80	0.799999	20.0	5
−0.80	1.0	−0.80	−0.799999	20.0	5
0.70	1.0	0.70	0.700000	20.0	5
−0.70	1.0	−0.70	−0.700000	20.0	5
0.10	1.0	0.10	0.999999	20.0	5
−0.10	1.0	−0.10	−0.999999	20.0	5
0.30	0.9	$0.\bar{3}$	0.333333	20.0	10
−0.30	0.9	$-0.\bar{3}$	−0.333333	20.0	10
1.0	7.0	$0.14\bar{2}875$	0.1248751	20.0	7
−1.0	7.0	$-0.14\bar{2}875$	−0.1248751	20.0	7

here occurs naturally in the quantum annealing algorithm as the annealer settles into the lowest energy ground state that approximates the solution. The last four problems are "challenge" problems for the iterated division solver.

4.2.2. Iterated Division Solver

Table 5 lists a few example division problems returned from the iterated quantum annealing solver. These are problems selected from both **Tables 3**, **4** to illustrate the nature of the solutions returned for both cases. These problems were iterated to an error tolerance of 1.0×10^{-6}. The four "challenge" problems from **Table 4** can now be solved with the iterative method. The ground state is no longer given since the solution is generally the concatenation of multiple binary vectors for every iteration. Instead, the number of iterations is listed in the last column. Note that some of the energies are the same for the solutions of different problems. We have also left out an "Energy" column since it was only calculated for the partial solution from the last iteration.

4.3. Results for Matrix Equations

Note that we have occasionally been somewhat loose in calling this "matrix inversion" since we are technically solving the linear equation, rather than directly inverting the matrices. However, for the problems considered here, we may simply obtain the solutions to the equations using trivial orthonormal eigenvectors, such as $(1, 0)$ and $(0, 1)$, in which case the inverse of the matrix will just be the matrix having those solutions as columns.

The linear equation algorithm was implemented and used to solve several 2×2 and 3×3 matrices on the D-Wave quantum annealer. Floating-point numbers are represented using the same offset binary representation as was used for the division problems. There are thus 4 qubits per floating-point number. As in the previous section, this gives an effective resolution of 3 bits for floating-point numbers defined on $[-1, 1]$. In this case, however, we employed the normalization technique discussed in the division iteration to allow solutions with positive and negative floating-point numbers with larger magnitudes than 1. But, in these matrix problems we still use solution values with relatively small magnitudes and within an order of magnitude of each other for all solution vector elements. All of the cases shown here are matrix equations with exact solutions, in which case the values of the solution vector elements are multiples of 0.25. We are therefore optimistic that the iterative solver for the matrix inversion could be implemented fairly quickly.

In general, every qubit representing part of a floating-point number may be coupled to every other qubit representing part of the same number. In turn, every logical qubit may be connected to every other logical qubit, which implies that every qubit in the logical qubit representation of the problem, may be coupled to every other logical qubit in the problem. Therefore, the linear solution algorithm is implemented using a K_8 graph embedding to solve 2×2 matrix equations, having a 2 dimensional solution vector with 4 qubits per element, and using a K_{12} graph embedding to solve 3×3 matrix equations, having a 3 dimensional solution vector with 4 qubits per element.

Most of these solutions involve well-conditioned matrices; however, one does not generally find a feasible solution when using an ill-conditioned matrix. This is illustrated in two cases, one with a of a 2×2 matrix another with a 3×3 matrix. We were able to obtain the correct solutions by preconditioning these matrices before converting to QUBO form, however the 3×3 matrix, still had a nearly degenerate ground state and required a

very large chaining penalty α to get the correct solution. This is analyzed and discussed in detail below.

4.3.1. Simple Analytic Problem

Recalling equation (3.1), we shall obtain solutions $\mathbf{x}$ of the following matrix equation,

$$\boldsymbol{M} \cdot \mathbf{x} = \mathbf{Y}, \tag{4.2}$$

using values of $\boldsymbol{M}$ and $\mathbf{Y}$ listed in Matrix Test Problems. Here we present the first two tests as an example. Consider the following matrix:

$$\boldsymbol{M} = \begin{pmatrix} \frac{1}{2} & \frac{3}{2} \\ \frac{3}{2} & \frac{1}{2} \end{pmatrix}. \tag{4.3}$$

We can solve Equation (3.1) for $\boldsymbol{M}$, with the following two Y vectors:

$$\mathbf{Y_1} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \ \mathbf{Y_2} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{4.4}$$

The exact solutions are

$$\mathbf{x_1} = \begin{pmatrix} -\frac{1}{4} \\ \frac{3}{4} \end{pmatrix}, \ \mathbf{x_2} = \begin{pmatrix} \frac{3}{4} \\ -\frac{1}{4} \end{pmatrix}. \tag{4.5}$$

We may obtain M^{-1} simply as

$$\boldsymbol{M^{-1}} = \begin{pmatrix} -\frac{1}{4} & \frac{3}{4} \\ \frac{3}{4} & -\frac{1}{4} \end{pmatrix} \tag{4.6}$$

In the next section we summarize all of the solutions obtained by the DWave for all of our test problems.

4.3.2. QUBO Solution Results

The solutions for the 2×2 linear solves are presented in **Table 6**. Notice that all of the test problems are presented with $\alpha = 20.0$ except for the last two. This was done to illustrate the effect of preconditioning for the ill-conditioned case. However, for this example, the difference disappeared above $\alpha = 2.0$, and both began to give incorrect answers below $\alpha = 1.5$. This is in contrast to the 3×3 matrix solution cases, which are evidently more sensitive to condition number than the 2×2 tests.

The 3×3 matrix solutions are presented in **Table 7**. Note that we have not included the 12 digit binary ground states here because they take up too much room in the table and are not particularly illuminating. Problems 2(f) and 2(g) are the ill-conditioned matrix test and its preconditioned equivalent. For $\alpha = 20.0$ both versions of the poorly-conditioned problem gave only D-Wave solutions with broken chains. One only begins to get solutions with unbroken chains at a value of α above 1000, but those solutions are generally wrong and basically random until one gets to a very high α. We discuss this in greater detail in the following section.

TABLE 6 | 2×2 Matrix equation solutions to 3-bit resolution.

Test	Exact solution	D-wave solution	Ground state	Energy	α
DIVISION PROBLEMS WITH APPROXIMATE D-WAVE SOLUTIONS					
1(*a*)	(−0.25, 0.75)	(−0.25, 0.75)	[0,0,1,1,0,1,1,1]	−2.167	20.0
1(*b*)	(0.75, −0.25)	(0.75, −0.25)	[0,1,1,1,0,0,1,1]	−2.167	20.0
1(*c*)	(1.00, 1.00)	(1.00, 1.00)	[1,0,0,0,1,0,0,0]	−0.444	20.0
1(*d*)	(−1.00, 1.00)	(−1.00, 1.00)	[0,0,0,0,1,0,0,0]	−1.889	20.0
1(*e*)	(1.00, −1.00)	(1.00, −1.00)	[1,0,0,0,0,0,0,0]	−1.650	20.0
1(*f*)	(1.00, 0.00)	(1.00, 0.00)	[0,0,0,0,0,1,0,0]	−2.125	20.0
1(*g*)	(0.25, −0.50)	(0.25, −0.50)	[0,1,0,1,0,0,1,0]	−0.925	20.0
1(*h*)	(0.25, 0.25)	(0.25, 0.25)	[0,1,0,1,0,1,0,1]	−2.03125	20.0
1(*i*)	(2.00, 1.00)	(2.00, 1.00)	[1,1,0,0,1,0,0,0]	−2.450126	20.0
1(*j*)	(2.00, 1.00)	(2.00, 1.00)	[1,1,0,0,1,0,0,0]	−2.532545	20.0
1(*i*)	(2.00, 1.00)	(2.50, 0.75)	[1,1,1,0,0,1,1,1]	−2.887689	1.5
1(*j*)	(2.00, 1.00)	(2.00, 1.00)	[1,1,0,0,1,0,0,0]	−2.951557	1.75

TABLE 7 | 3×3 matrix equation solutions to 3-bit resolution.

Test	Exact solution	D-wave solution	Energy	α
DIVISION PROBLEMS WITH APPROXIMATE D-WAVE SOLUTIONS				
2(*a*)	(0.25, −0.5, 1.0)	(0.25, −0.5, 1.0)	−15.5625	20.0
2(*b*)	(0.25, −0.5, 0.0)	(0.25, −0.5, 0.0)	−12.5625	20.0
2(*c*)	(0.25, 0.0, −0.5)	(0.25, 0.0, −0.5)	−13.5	20.0
2(*d*)	(1.0, 0.25, −0.5)	(1.0, 0.25, −0.5)	−15.6875	20.0
2(*e*)	(0.0, 0.25, −0.5)	(0.0, 0.25, −0.5)	−12.75	20.0
2(*f*)	(0.0, 0.25, −0.75)	broken chains	N/A	20.0
2(*g*)	(0.0, 0.25, −0.75)	broken chains	N/A	20.0
2(*f*)	(0.0, 0.25, −0.75)	(1.75, 1.25, 0.75)	−58.188	2200.0
2(*g*)	(0.0, 0.25, −0.75)	(0.0, 0.25, −0.75)	−557.437	2200.0

4.4. Discussion

The algorithms described here generally worked quite well for these small test cases, with the exception of the ill-conditioned 3×3 matrix. The ill-conditioned cases clearly demonstrate not only the limitations of quantum annealing applied solving linear equations, but the limitations of quantum annealing in general. Consider the two ill-conditioned tests presented here. When translated to a QUBO problem, the Hamiltonian spectra for these tests contain many energy eigenvalues very close to the ground state energy. When these are embedded within a larger graph of physical qubits they result in a very nearly degenerate ground state, typically with thousands of states having energies within the energy uncertainty of the ground state over the annealing time, τ, given by

$$\Delta E = \frac{\hbar}{\tau}. \tag{4.7}$$

Consider a set of excited states with energy, E_n for $n > 0$, with $n = 0$, corresponding to the ground state with energy denoted by E_0, and with E_n ordered by energy. The quantum annealer near

the ground state evolves adiabatically whenever

$$E_1 - E_0 \gg \frac{\hbar}{\tau} \tag{4.8}$$

This is the adiabatic condition for quantum time evolution [6]. However, when this condition is badly violated, which can occur dynamically since the instantaneous energies (eigenvalues of H) are time dependent, the time evolution for the system near the ground state deviates significantly from adiabatic behavior, resulting in a highly mixed superposition of those eigenstates states close in energy to the ground state. Now, the energy spectra corresponding to poorly conditioned matrices have a large number of eigenstates sufficiently near the ground state to strongly violate the adiabaticity condition. Furthermore, these states, in general, will have no correlation to the solution encodings for any particular problem (e.g., the offset binary floating-point representation). For example, they are not, in general, related in any meaningful way to Hamming distance. Therefore, these problems effectively cannot resolve the true ground state and tend to give nearly random lowest energy "solutions" when the final state is measured on any individual annealing run. Since there are so many of these states for ill-conditioned problems, a very large number of "reads" (annealing runs) may be have to be specified to sufficient sample the solution space to find the true ground state. Thus, we determined that the condition number of a matrix has a strong effect on the ability to solve a linear equation using a quantum annealer, as it influences the shape of the energy surface near the ground state.

The preconditioning method we used was very simple and was probably too crude to be practical for arbitrary matrices. However, the intent here was simply to test the effects of preconditioning on the quantum annealing solutions. We have been studying this issue and believe it may be possible to precondition such problems to solve them more efficiently on a quantum annealing machine. We suspect that a related preconditioning method may be applicable to more general QUBO problems suffering from similar spectral density pathologies in order to better separate the ground state energy, thereby allowing more practical solution on a quantum annealer. This is work in progress. We plan to further develop and test those ideas in the future.

MATRIX TEST PROBLEMS

The problems we solved to test our quantum annealing algorithm to solve equation (3.1) are listed below. Note that, although the QUBO B_{ij} matrix is symmetric by construction, the matrix M need not be symmetric.

1. Test Problems with 2×2 Matrices

Test 1(a):

$$\boldsymbol{M} = \begin{pmatrix} 0.5 & 1.5 \\ 1.5 & 0.5 \end{pmatrix}, \ \mathbf{Y} = \begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} -0.25 \\ 0.75 \end{pmatrix}$$

Test 1(b):

$$\boldsymbol{M} = \begin{pmatrix} 0.5 & 1.5 \\ 1.5 & 0.5 \end{pmatrix}, \ \mathbf{Y} = \begin{pmatrix} 0.0 \\ 1.0 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} 0.75 \\ -0.25 \end{pmatrix}$$

Test 1(c):

$$\boldsymbol{M} = \begin{pmatrix} 2.0 & -1.0 \\ -0.5 & 0.5 \end{pmatrix}, \ \mathbf{Y} = \begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}$$

Test 1(d):

$$\boldsymbol{M} = \begin{pmatrix} 1.0 & 2.0 \\ 0.5 & 0.5 \end{pmatrix}, \ \mathbf{Y} = \begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} -1.0 \\ 1.0 \end{pmatrix}$$

Test 1(e):

$$\boldsymbol{M} = \begin{pmatrix} 3.0 & 2.0 \\ 2.0 & 1.0 \end{pmatrix}, \ \mathbf{Y} = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} 1.0 \\ -1.0 \end{pmatrix}$$

Test 1(f):

$$\boldsymbol{M} = \begin{pmatrix} 1.0 & 0.5 \\ 1.0 & -0.5 \end{pmatrix}, \ \mathbf{Y} = \begin{pmatrix} 1.0 \\ 1.0 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix}$$

Test 1(g):

$$\boldsymbol{M} = \begin{pmatrix} 0.0 & -2.0 \\ -2.0 & -1.5 \end{pmatrix}, \ \mathbf{Y} = \begin{pmatrix} 1.0 \\ 0.25 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} 0.25 \\ -0.5 \end{pmatrix}$$

Test 1(h):

$$\boldsymbol{M} = \begin{pmatrix} 0.0 & -2.0 \\ -2.0 & -1.5 \end{pmatrix}, \ \mathbf{Y} = \begin{pmatrix} -0.5 \\ -0.875 \end{pmatrix}, \ \mathbf{x} = \begin{pmatrix} 0.25 \\ 0.25 \end{pmatrix}$$

Test 1(i): Ill-conditioned problem with $\kappa \approx 25$

$$\boldsymbol{M} = \begin{pmatrix} 1.0 & 2.0 \\ 2.0 & 3.999 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 4.0 \\ 7.999 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 2.0 \\ 1.0 \end{pmatrix}$$

Test 1(j): Pre-conditioned version of 1(i) with $\kappa = 5.0$

$$\boldsymbol{M} = \begin{pmatrix} 1.80026 & 1.6019 \\ 1.6019 & 4.19974 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 5.2007 \\ 7.40013 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 2.0 \\ 1.0 \end{pmatrix}$$

2. Matrix Problems with 3×3 Matrices

Test 2(a):

$$\boldsymbol{M} = \begin{pmatrix} 0.0 & -2.0 & 0.0 \\ -2.0 & 1.5 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 1.0 \\ 0.25 \\ 1.0 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 0.25 \\ -0.5 \\ 1.0 \end{pmatrix}$$

Test 2(b):

$$\boldsymbol{M} = \begin{pmatrix} 0.0 & -2.0 & 0.0 \\ -2.0 & 1.5 & 0.0 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 1.0 \\ 0.25 \\ 0.0 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 0.25 \\ -0.5 \\ 0.0 \end{pmatrix}$$

Test 2(c):

$$\boldsymbol{M} = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -2.0 \\ 0.0 & -2.0 & -1.5 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 1.0 \\ 0.0 \\ 0.25 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 0.25 \\ 0.0 \\ -0.5 \end{pmatrix}$$

Test 2(d):

$$\boldsymbol{M} = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -2.0 \\ 0.0 & -2.0 & -1.5 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 1.0 \\ 1.0 \\ 0.25 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 1.0 \\ 0.25 \\ -0.5 \end{pmatrix}$$

Test 2(e):

$$\boldsymbol{M} = \begin{pmatrix} 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -2.0 \\ 0.0 & -2.0 & -1.5 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 0.0 \\ 1.0 \\ 0.25 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 0.0 \\ 0.25 \\ -0.5 \end{pmatrix}$$

Test 2(f): Ill-conditioned problem with $\kappa \approx 78$

$$\boldsymbol{M} = \begin{pmatrix} -4.0 & 6.0 & 1.0 \\ 8.0 & -11.0 & -2.0 \\ -3.0 & 4.0 & 1.0 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 0.75 \\ -1.25 \\ 0.25 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 0.0 \\ 0.25 \\ -0.75 \end{pmatrix}$$

Test 2(g): Pre-conditioned version 2(g) with $\kappa \approx 1$

$$\boldsymbol{M} = \begin{pmatrix} 6.1795 & 11.8207 & 2.0583 \\ 15.673 & -7.56717 & -3.8520 \\ -5.6457 & 7.96872 & 15.9418 \end{pmatrix}, \mathbf{Y} = \begin{pmatrix} 1.4114 \\ 0.9972 \\ 9.9643 \end{pmatrix}, \mathbf{x} = \begin{pmatrix} 0.0 \\ 0.25 \\ -0.75 \end{pmatrix}$$

AUTHOR CONTRIBUTIONS

All authors listed have made a substantial, direct and intellectual contribution to the work, and approved it for publication.

FUNDING

We received funding for this work from the *ASC Beyond Moore's Law Project* at LANL.

ACKNOWLEDGMENTS

We would like to thank Andrew Sornborger, Patrick Coles, Rolando Somma, and Yiğit Subaşı for a number of useful conversations.

REFERENCES

1. Harrow A, Hassidim A, Lloyd S. Quantum algorithm for linear systems of equations. *Phys Rev Lett.* (2009) 103:150502. doi: 10.1103/PhysRevLett.103.150502
2. Barz S, Kassal I, Ringbauer M, Ole Lipp Y, Dakic B, Aspuru-Guzik A, et al. Solving systems of linear equations on a quantum computer. *Sci. Rep.* (2014) 4:115. doi: 10.1038/srep06115
3. Cai XD, Weedbrook C, Su ZE, Chen MC, Gu M, Zhu MJ, et al. Experimental quantum computing to solve systems of linear equations. *arXiv:1302.4310v2* (2013). doi: 10.1103/PhysRevLett.110.230501
4. Farhi E, Goldstone J, Gutmann S, Sipser M. Quantum computation by adiabatic evolution. *arXiv:000110* (2000).
5. Ising E. Beitrag zur Theorie des Ferromagnetismus *Z Phys.* (1925) 31:253–18.
6. Albert M. *Chapter XVII: Quantum Mechanics.* New York, NY: Dover Publications (1999).

Conflict of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.