Computing a Sparse Approximate Inverse on Quantum Annealing Machines

Sanjay Suresh\textsuperscript{1} and Krishnan Suresh\textsuperscript{2*}

\textsuperscript{1}Computer Science, University of Wisconsin, Madison, 1210 W. Dayton Street, Madison, 53706, WI, USA.

\textsuperscript{2*}Mechanical Engineering, University of Wisconsin, Madison, 1513 University Avenue, Madison, 53706, WI, USA.

\textsuperscript{*}Corresponding author(s). E-mail(s): ksuresh@wisc.edu; Contributing authors: ssuresh27@wisc.edu;

Abstract

Many engineering problems involve solving large linear systems of equations. Conjugate gradient (CG) is one of the most popular iterative methods for solving such systems. However, CG typically requires a good preconditioner to speed up convergence. One such preconditioner is the sparse approximate inverse (SPAI).

In this paper, we explore the computation of an SPAI on quantum annealing machines by solving a series of quadratic unconstrained binary optimization (QUBO) problems. Numerical experiments are conducted using both well-conditioned and poorly-conditioned linear systems arising from a 2D finite difference formulation of the Poisson problem.

Keywords: QUBO; Linear system of equations; Quantum annealing; Conjugate gradient; Pre-conditioner; sparse approximate inverse; D-WAVE; Quantum computing

1 Introduction

Many engineering problems result in linear systems of equations [1] of the form:

\[ Ku = f \] (1)

where \( K \) is a symmetric, positive-definite, sparse \( N \times N \) matrix, \( u \) is the unknown field (for example, the temperature field), and \( f \) is the applied force (for example, the
heat-flux). Solving such linear systems for large $N$ is a computationally intensive task [2], and can be time-consuming on classical computers. Quantum computers have been proposed as an alternate since they can potentially accelerate the computation; see [3, 4] for recent reviews on the potential role of quantum computers in engineering.

In particular, the Harrow-Hassidim-Lloyd (HHL) algorithm is a landmark strategy for solving linear systems of equations on quantum-gate computers. In theory, it offers an exponential speed-up over classical algorithms [5], and it has been further improved recently [6–8]. However, due to the accumulation of errors in current noisy intermediate-scale quantum (NISQ) computers [9], the HHL algorithm and its variants are limited, in practice, to very small ($N < 5$) systems. Other strategies such as Grover’s algorithm [10] and quantum approximate optimization algorithm (QAOA) [11] have been proposed to solve linear systems on quantum gate computers, but they suffer from the same limitation. Hybrid solvers such as the variational quantum linear solver [12] have also been proposed to mitigate some of these challenges.

In parallel, quantum annealing machines, such as the D-wave systems with several thousand qubits [13], have also been proposed for solving linear systems since they are less susceptible to noise [14, 15]. The basic principle is to pose the solution of a linear system of equations as a minimization problem and then convert this into a series of quadratic unconstrained binary optimization (QUBO) problems. For example, O’Malley and Vesselinov used a least-squares formulation and a finite-precision qubit representation to pose QUBO problems [16]. Borle and Lomonaco carried out a theoretical analysis of this approach [17]. Park et. al. showed how QUBO problems can be simplified using matrix congruence [18], while its application in solving 1D Poisson problems was demonstrated in [19]. If the matrix is positive-definite, which is often the case in engineering problems, it is much more efficient to use a potential-energy formulation, rather than the least-squares formulation, to pose QUBO problems. Using the potential-energy formulation, Srivastava et. al. described a box algorithm to solve QUBO problems arising from finite element analysis of one-dimensional differential equations [20].

However, despite these advances, only small-size ($N < 100$) problems have been demonstrated on current quantum annealing machines, while most engineering problems of practical interest result in much larger-size ($N > 10^5$) matrices. To address this gap, we propose here an alternate paradigm where quantum annealing machines are used, not to directly solve such large linear systems, but to compute sparse preconditioners (SPAI). Once again, the strategy is to rely on the potential energy formulation to pose a series of quadratic unconstrained binary optimization (QUBO) problems. However, the sparsity can be fully exploited to significantly reduce the size of the QUBO problems, making it more amenable to quantum computing. The computed SPAI can then be used as a preconditioner to rapidly solve large systems of equations using the conjugate gradient (CG) method on classical computers. The efficacy of this approach is demonstrated by solving both well-conditioned and poorly-conditioned linear systems of equations arising from finite difference formulation of 2D Poisson problems. Furthermore, by exploiting the well-structured nature of the finite-difference formulation, we show how one can compute the SPAI in constant time, independent of the size ($N$) of the linear system.
2 Proposed Methodology

2.1 Poisson Problem

To provide a context to this paper, we consider solving a Poisson problem in 2D, governed by [21]:

\[ k \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = -f \]  

(2)

where \( u \) is the unknown (temperature) field, \( f \) is the (heat) source, and \( k \) is the underlying material property (example: conductivity). We will assume that the field takes a value of zero on the boundary.

A classic approach for solving Eq. 2 is the finite difference method [21], where the geometry is sampled by a uniform grid, over which the field \( u \) is to be determined. Then, the partial derivatives are approximated as follows [21]:

\[ \frac{\partial^2 u}{\partial x^2}_{m,n} \approx \frac{u_{m+1,n} - 2u_{m,n} + u_{m-1,n}}{\Delta x^2} \]  

(3)

\[ \frac{\partial^2 u}{\partial y^2}_{m,n} \approx \frac{u_{m,n+1} - 2u_{m,n} + u_{m,n-1}}{\Delta y^2} \]  

(4)

Substituting the above approximations in Eq. 2, and with \( \Delta x = \Delta y = h \), results in:

\[ 4ku_{m,n} - k \left( u_{m-1,n} + u_{m+1,n} + u_{m,n-1} + u_{m,n+1} \right) = f_{m,n}h^2 \]  

(5)

When applied at all grid points, and after eliminating the rows and columns corresponding to the boundary nodes, we arrive at the linear system in Eq. (1). As one can observe, the resulting \( K \) matrix is symmetric. Further, for this problem, the sparsity is 5, i.e., \( K \) has at most 5 entries in any row or column. Once the rows and columns corresponding to the boundary nodes are eliminated, and the resulting \( K \) matrix can be shown to be positive-definite [21].

While the finite difference method applies to arbitrary domains, we will focus on rectangular domains for simplicity; see Fig 1. Since the field is assumed to take a value of zero on the boundary, we will only solve for the field in the interior, i.e., \( N \) denotes the number of interior nodes.

![Fig. 1: Finite-difference grid over a rectangle with zero Dirichlet boundary conditions.](image)
Finally, if the material property \((k)\) is not a constant over the domain, Eq. 5 can be generalized to

\[
4k_{m,n}u_{m,n} = (k_{m,n}^{m-1,n}u_{m-1,n} + k_{m,n}^{m+1,n}u_{m+1,n} + ...) = f_{m,n}h^2
\]

where \(k_{m,n}\) is the average \(k\) over the (4 or less) elements surrounding the node \((m,n)\), while \(k_{m,n}^{m-1,n}\) is the average \(k\) over the (2 or 1) elements adjacent to the edge joining node \((m,n)\) and \((m-1,n)\). The resulting \(K\) matrix is still symmetric, sparse, and positive definite, but can become poorly conditioned depending on the material distribution (see Section 3).

2.2 Classic Linear Solvers

The two main strategies for solving Eq.(1) are: direct and iterative. Direct solvers usually rely on LU-decomposition [22], and generally require a significant amount of memory for large systems. Iterative solvers require less memory but converge to a solution gradually. The rate of convergence of iterative methods depends on several factors including the condition number of \(K\), sparsity, etc [23]. One of the most popular iterative methods is conjugate gradient whose run-time complexity is given by [24, 25]:

\[
CG \sim O(Ns\sqrt{\kappa}\log \frac{1}{\epsilon_{cg}})
\]

where \(N\) is the dimension of \(K\), \(s\) is the sparsity \((s = 5\) for the problem described above), \(\kappa\) is the condition number, and \(\epsilon_{cg}\) is the desired residual error. Thus, for poorly conditioned systems (i.e. when \(\kappa\) is large), CG does not perform well, and preconditioners are essential. Several preconditioners are widely used today [26–28]; these include Jacobi, incomplete Cholesky, sparse approximate inverse, etc. In this paper, we will rely on the sparse approximate inverse of \(K\), and we propose a simple algorithm to compute this preconditioner on quantum annealing computers using a QUBO formulation.

2.3 Computing a Sparse Approximate Inverse

Note that \(K^{-1}\), the exact inverse of \(K\), can be computed by solving

\[
Ky_j = e_j, j = 0,1...N - 1
\]

where \(e_j\) is the unit vector corresponding to the \(j^{th}\) dimension. In general, \(K^{-1}\) will be dense. Our objective is to compute a sparse approximate inverse (SPAI) \(M\). There are various \(a\ priori\) and adaptive techniques for forcing sparsity on \(M\); see [26, 27]. We will use a well-known \(a\ priori\) technique where the sparsity pattern of \(K\) is imposed on \(M\) [27]. In other words, to compute \(M\), we once again solve

\[
Km_j = e_j, j = 0,1...N - 1
\]
but with the constraint that $m_j$ must have the same sparsity pattern as the $j^{th}$ column of $K$.

To compute $m_j$, let the row-sparsity index of the $j^{th}$ column of $K$ be $s$, i.e., $s[i]$ stores the $i^{th}$ non-zero row of $K[:,j]$, $0 \leq i \leq s - 1$, and $s \ll N$. We can rearrange $m_j$ as

$$m_j = \begin{bmatrix} \hat{m}_j \\ 0 \end{bmatrix}$$

(10)

where $\hat{m}_j$ is of length $s$. We can also rearrange $K$ using the same reordering as:

$$K = \begin{bmatrix} A_j & B_j \\ B_j^T & C_j \end{bmatrix}$$

(11)

where $A_j$ is $s \times s$, and $C_j$ is $(N - s) \times (N - s)$, and rearrange $e_j$ as

$$e_j = \begin{bmatrix} \hat{e}_j \\ 0 \end{bmatrix}$$

(12)

Thus Eq. 9 reduces to:

$$\begin{bmatrix} A_j & B_j \\ B_j^T & C_j \end{bmatrix} \begin{bmatrix} \hat{m}_j \\ 0 \end{bmatrix} = \begin{bmatrix} \hat{e}_j \\ 0 \end{bmatrix}$$

(13)

Discarding the last $(N - s)$ equations, we arrive at:

$$A_j \hat{m}_j = \hat{e}_j$$

(14)

where $A_j$ is a small $s \times s$ matrix constructed from the sparsity pattern of the $j^{th}$ column. Eq. 14 must be posed and solved for each column $j$ of $K$. However, we show in Section 2.5 that we only need to solve for a few columns by exploiting the structured nature of the finite difference formulation.

2.4 QUBO Formulation

Observe that solving Eq. (14) is equivalent to minimizing the potential energy (we have dropped the subscript $j$ to avoid clutter):

$$\min_{\hat{m}} \Pi = \frac{1}{2} \hat{m}^T \hat{A} \hat{m} - \hat{m}^T \hat{e}$$

(15)

This is a quadratic unconstrained minimization problem involving real variables $\hat{m}$. To solve this on a quantum annealing machine, we represent each real component $\hat{m}_i$ using qubit variables. A well known strategy is the radix representation, also referred to here as the box representation [16, 29]:

$$\hat{m} = c + L(-2q_1 + q_2)$$

(16)

where $c$ and $L$ are real value parameters that we will iteratively improve, while $q_1$ and $q_2$ are qubit vectors of length $s$ each, i.e., a total of $2s$ qubits is used to capture $\hat{m}$. 
Since $\hat{m}$ is linear in $q_1$ and $q_2$, substituting Eq. (16) into Eq. (15) will lead to a quadratic unconstrained binary optimization (QUBO) problem:

$$\min_{q=(q_1, q_2)} \Pi = \frac{1}{2} q^T Q q + q^T d$$

(17)

Further, since the qubit variables can only take the binary values 0 or 1, the linear term can be absorbed into the quadratic term [30], resulting in the standard form:

$$\min_{q=(q_1, q_2)} \Pi = \frac{1}{2} q^T Q q$$

(18)

where $Q$ is symmetric (but not positive definite).

The overall strategy therefore is as follows: For the first column of $K$, the parameters $c$ and $L$ are initialized to $c = 0$ and $L = 0$. The resulting QUBO problem in Eq. 18 is solved on a quantum annealing machine. Then $c$ and $L$ in Eq. 16 are updated via the sparse box algorithm, discussed in Section 2.6. The process is repeated until convergence is reached (in typically 25 ∼ 35 iterations; see Section 3). Then, the next column of $K$ is processed until the entire $M$ matrix is constructed.

### 2.5 Node Mapping for a Structured Grid

For a generic $K$ matrix, one must explicitly process each of the $N$ columns. However, for the structured finite difference grid, one can significantly reduce the computation. Observe that each column of $K$ corresponds to a unique node in the grid. Now consider a typical node highlighted using a square box in Fig. 2. The $5 \times 5$ matrix $A$ in Eq. 15 corresponding to this node (column), is entirely determined by the rows and columns of $K$ associated with this node and the 4 neighboring nodes (highlighted using circles in Fig. 2). From Eq. 6, we observe that the diagonal entries of $A$ depend only on the material property ($k_{m,n}$) associated with the square node, while the non-diagonal entries depend only on the material property (example: $k_{m-1,n}$) associated with the edge joining the square node and a circle node.

![Fig. 2: A node and its 4 neighboring nodes.](image)

Since this pattern repeats over the entire grid, the matrix $A$ associated with the two square nodes in Fig. 3, for example, are identical. Consequently, the solution vectors $\hat{m}$ for the two nodes are identical.
With exceptions made at the corner and edge nodes, one can conclude that, for a single material domain, it is sufficient to compute 9 independent columns of $M$, independent of the size $N$. A typical set of these 9 independent nodes is illustrated in Fig. 4. This can be further reduced to 3 independent nodes (one corner, one edge, and one interior node) with appropriate transformations. Exploiting this node mapping, one can compute the SPAI matrix $M$ in constant time, independent of $N$. To the best of our knowledge, this has not been exploited previously to compute SPAI in classical, or quantum settings.

If the domain is composed of two materials as illustrated in Fig. 5, then one can show that only 21 independent columns of $M$ need to be computed, independent of $N$. This can be further reduced to 10 independent columns through appropriate transformations.
2.6 Proposed Algorithms

To summarize, the proposed algorithm to compute $M$ is described in Alg. 1 where

1. The sparsity pattern of $K$ is first copied over to $M$.
2. Then, for each column of $K$, if that column (node) is mapped to a previously computed column (node), we copy the previously computed solution.
3. Else we call the *sparse box algorithm* (see below), and the computed solution is pushed to $M$.
4. Finally, we force symmetry on $M$ to address possible numerical errors.

### Algorithm 1: Sparse Approximate Inverse

```python
1: procedure QUBOSPARSEAPPROXIMATEINVERSE(K)
2:   M ← K  \(\triangleright\) M has same sparsity as K
3:   N ← \text{dim}K  \(\triangleright\) number of rows in K
4:   for j ← 0 to N − 1 do
5:     if nodeMap[j] < j then  \(\triangleright\) If node is mapped to another node
6:       M[:, j] ← M[:, nodeMap[j]]  \(\triangleright\) copy solution
7:     else
8:       \(\hat{m}\) ← SparseBoxAlg(K, j)  \(\triangleright\) solve via QUBO
9:       s ← K[:, j]  \(\triangleright\) non-zero rows of column j
10:      M[s, j] ← \(\hat{m}\)  \(\triangleright\) copy solution
11:   end if
12: end for
13: M ← (M + M^T)/2  \(\triangleright\) impose symmetry
14: end procedure
```

The above algorithm uses a sparse box algorithm, which is a generalization of the box algorithm proposed in [20]. The original box algorithm solves small dense linear systems using a QUBO formulation (see [20] for details). It is modified here to solve the sparse problem in Eq. 14. A few observations regarding the proposed sparse box algorithm (see Alg. 2) are:

1. We have chosen an initial box size of $L = 1$. This is an arbitrary choice; the algorithm is robust for any reasonable value [20]; see numerical experiments. Choosing a large initial value for $L$ will increase the number of contraction steps, while a small initial value will increase the number of translation steps.
2. A total of $2s$ qubits are created, and the initial potential energy is zero.
3. In the main iteration, the QUBO problem is constructed using the software packages pyQUBO [30].
4. The QUBO problem can be solved in exactly or through (3) quantum annealing; see numerical experiments.
5. If the computed potential energy is less than the current minimum, then we have found a better solution. In this case, we move the box center $c$ to the new computed
solution (translation) to improve the accuracy. Otherwise, the current solution is optimal, and we shrink the box size $L$ (contraction) to improve the precision.

6. For termination, using a very small value, say, $\epsilon_{\text{box}} \approx 10^{-14}$, is not desirable since (1) it will increase the computation cost, and (2) as the box size becomes small, the potential energy function $\Pi$ becomes relatively flat, and the computed potential energies from various qubit configurations will be numerically equal; this will result in the algorithm choosing the wrong step (i.e. translate versus contract). Further, $\epsilon_{\text{box}} \approx 10^{-14}$ is not required since we are only solving for an approximate inverse. A typical choice is $\epsilon_{\text{box}} \approx 10^{-6}$; see numerical experiments.

Algorithm 2: Sparse Box Algorithm

1: procedure SparseBoxAlg($K$, $j$)
2:   $s \leftarrow K[:, j]$ \hspace{1em} $\triangleright$ non-zero rows of column $j$
3:   $s \leftarrow \text{dim}(s)$ \hspace{1em} $\triangleright$ sparsity of column $j$
4:   $i \leftarrow (s[i] == j)$ \hspace{1em} $\triangleright$ find the entry for column $j$
5:   $c \leftarrow 0$ \hspace{1em} $\triangleright$ center of length $n$
6:   $L \leftarrow 1$ \hspace{1em} $\triangleright$ initialize box size
7:   $q_1, q_2 \leftarrow \text{Qubits}(s)$ \hspace{1em} $\triangleright$ create qubit arrays of length $s$
8:   $iter = 0$
9:   $\Pi_{\text{min}} \leftarrow 0$ \hspace{1em} $\triangleright$ energy initialization
10:  repeat \hspace{1em} $\triangleright$ until convergence
11:    $\hat{m} \leftarrow c + L(-2q_1 + q_2)$ \hspace{1em} $\triangleright$ symbolic expression
12:    $\Pi \leftarrow \frac{1}{2}\hat{m}Ks, s|\hat{m} - \hat{m}[i]$ \hspace{1em} $\triangleright$ construct QUBO
13:    $\Pi^*, q^*_1, q^*_2 \leftarrow \text{minimize}(H)$ \hspace{1em} $\triangleright$ solve QUBO
14:    if $\Pi^* < \Pi_{\text{min}}$ then
15:       $c \leftarrow c + L(-2q^*_1 + q^*_2)$ \hspace{1em} $\triangleright$ translation
16:       $\Pi_{\text{min}} \leftarrow \Pi^*$ \hspace{1em} $\triangleright$ new minimum
17:    else
18:       $L \leftarrow L/2$ \hspace{1em} $\triangleright$ contraction
19:    end if
20: until $(L < \epsilon_{\text{box}})$ or $(iter > \text{iter}_{\text{max}})$ \hspace{1em} $\triangleright$ termination
21: end procedure \hspace{1em} $\triangleright$ Output: solution $c$

2.7 D-WAVE Embedding

We now consider mapping the QUBO problems generated by Alg. 2 to the D-Wave Advantage quantum annealing machine. The D-wave Advantage is equipped with 5000+ qubits, embedded in a Pegasus architecture. Each QUBO problem involves at most 10 logical qubits with the connectivity graph illustrated in Fig. 6a. Using the default embedding, these logical qubits are mapped to 18 physical qubits illustrated in Fig. 6b. The default chain strength was found to be sufficient for all numerical experiments.
3 Numerical Experiments

In the following experiments, we consider a \((g_x + 2) \times (g_y + 2)\) rectangular finite-difference grid as shown earlier in Fig. 1. We assume that the field on the boundary is zero, i.e. we only solve for the interior \(g_x \times g_y\) grid. The default values, unless otherwise noted, for all the experiments are:

- The grid size is \(g_x = 401\) and \(g_y = 301\).
- The rectangle is composed of a single material with \(k = 1\).
- The box-tolerance in Algorithm 2 is set to \(\epsilon_{\text{box}} = 10^{-6}\).
- The box length in Algorithm 2 is initialized to \(L = 1\).
- The maximum box iterations in Algorithm 2 is set to 100.
- The conjugate gradient residual tolerance is set to \(10^{-10}\).

In the experiments, Q-PCG refers to the standard preconditioned conjugate gradient method where the proposed SPAI preconditioner is used. In each of the following experiments, we graph the residual error against the number of iterations for both Q-PCG and CG. The implementation is in Python, and uses pyQUBO [30] to create the QUBO model, and pyAMG [31] to construct the \(K\) matrix.

3.1 Performance Evaluation

In the first experiment, we compare the convergence of Q-PCG and CG using the default values listed above; this results in \(N = 120,701\) (size of the \(K\) matrix). The convergence plots using D-WAVE’s dimod exact QUBO solver are illustrated in Fig. 7a. Exactly the same Q-PCG convergence plot was obtained when using the D-WAVE quantum annealing solver. The SPAI does not significantly improve the convergence of CG here since \(K\) is relatively well-conditioned. The computed Poisson field is illustrated in Fig. 7b.
Fig. 7: (a) Convergence plot for $g_x = 401$, $g_y = 301$. (b) Computed Poisson field.

The two solvers are compared in Table 1.

<table>
<thead>
<tr>
<th>Task</th>
<th>Milliseconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access</td>
<td>25</td>
</tr>
<tr>
<td>Programming</td>
<td>15</td>
</tr>
<tr>
<td>Sampling</td>
<td>11</td>
</tr>
<tr>
<td>Readout</td>
<td>7</td>
</tr>
<tr>
<td>Post-process</td>
<td>2</td>
</tr>
<tr>
<td>Anneal</td>
<td>2</td>
</tr>
<tr>
<td>Delay</td>
<td>2</td>
</tr>
<tr>
<td>Access overhead</td>
<td>1.6</td>
</tr>
</tbody>
</table>

Table 2: QPU timing per QUBO solve.

Table 1: Comparing the two solvers.

<table>
<thead>
<tr>
<th></th>
<th>Exact</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total QUBO solves</td>
<td>294</td>
<td>294</td>
</tr>
<tr>
<td>Avg. time per QUBO solve</td>
<td>0.9 milliseconds</td>
<td>26 milliseconds</td>
</tr>
<tr>
<td>Time to compute $M$</td>
<td>0.5 seconds</td>
<td>209 seconds</td>
</tr>
</tbody>
</table>

Additional details on the QPU timing for a single QUBO solve (using the default 100 samples) are provided in Table 2. As one can observe there is significant overhead in D-WAVE QPU allocation, programming, access and post-process.

3.2 Multiple Materials

The real advantage of the SPAI becomes evident when we consider two materials with the $k_1$ on the left and $k_2$ on the right (see Fig. 5). For the default grid size of $g_x = 401$ and $g_y = 301$, we observe in Fig. 8a that Q-PCG provides a speed-up of 3.3 when $k_1 = 1$ and $k_2 = 10$, and a speed-up of 11 when $k_1 = 1$ and $k_2 = 100$. Such
multi-material problems are fairly common in engineering, especially during topology optimization [32, 33].

![Fig. 8: Convergence plots for two-material domain: (a) \(k_1 = 1, k_2 = 10\). (b) \(k_1 = 1, k_2 = 100\).]

### 3.3 Impact of box tolerance

We now study the impact of the box tolerance on the convergence of Q-PCG. For a single material problem, with default values, Table 3 summarizes the Q-PCG iterations and total box iterations using the exact and quantum annealing solvers (recall from Fig. 7a that regular CG converges in about 900 iterations). As the box tolerance is varied between \(10^{-8}\) and \(10^{-1}\), one can observe in Table 3 that the number of Q-PCG iterations remains around 459, while the number of box iterations decreases, as expected. However, Q-PCG did not converge to the desired tolerance, i.e., the SPAI matrix is not effective when the box tolerance is too coarse (\(10^{-1}\)).

<table>
<thead>
<tr>
<th>(\epsilon_{\text{box}})</th>
<th>Exact</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>(10^{-8})</td>
<td>458 (401)</td>
<td>459 (401)</td>
</tr>
<tr>
<td>(10^{-6})</td>
<td>459 (294)</td>
<td>459 (294)</td>
</tr>
<tr>
<td>(10^{-4})</td>
<td>459 (202)</td>
<td>459 (202)</td>
</tr>
<tr>
<td>(10^{-2})</td>
<td>464 (96)</td>
<td>473 (96)</td>
</tr>
<tr>
<td>(10^{-1})</td>
<td>- (50)</td>
<td>- (50)</td>
</tr>
</tbody>
</table>

### 3.4 Impact of box length

Next we study the impact of the box length on the convergence of Q-PCG. For a single material problem, with default values, Table 4 summarizes the Q-PCG iterations and total box iterations using exact and quantum annealing solvers. As the box length is varied between \(10^{-2}\) and \(10^{4}\), one can observe in Table 4 that the number of Q-PCG iterations remains at 459, while the number of box iterations is minimum when \(L = 1\). A star (*) indicates that the desired box tolerance was not achieved, and the
box algorithm exited when the maximum iteration was reached. However, even in this case, Q-PCG converged.

**Table 4**: Impact of box length on Q-PCG iterations and (total box iterations).

<table>
<thead>
<tr>
<th>$L$</th>
<th>Exact</th>
<th>Quantum</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^1$</td>
<td>458 (437)</td>
<td>459 (437)</td>
</tr>
<tr>
<td>$10^2$</td>
<td>458 (390)</td>
<td>459 (365)</td>
</tr>
<tr>
<td>$10^3$</td>
<td>458 (315)</td>
<td>458 (315)</td>
</tr>
<tr>
<td>$10^0$</td>
<td>458 (294)</td>
<td>458 (294)</td>
</tr>
<tr>
<td>$10^{-1}$</td>
<td>458 (333)</td>
<td>458 (333)</td>
</tr>
<tr>
<td>$10^{-2}$</td>
<td>691 (431*)</td>
<td>691 (431*)</td>
</tr>
</tbody>
</table>

**4 Conclusions**

A hybrid classical-quantum strategy for solving large linear systems of equations was proposed. The strategy relied on computing a sparse approximate preconditioner (SPAI) on a quantum annealing machine and using this preconditioner, along with an iterative solver, on a classical machine. Its effectiveness was demonstrated on large ($N > 100,000$) ill-conditioned matrices arising from a finite-difference formulation of the Poisson problem.

There are many directions for continued research. (1) While we demonstrated the method using quantum annealing machines, it can be extended to quantum gate computers since QUBO problems can be solved (approximately) on such machines via quantum approximate optimization algorithms [34, 35]. (2) The finite difference formulation of the Poisson problem resulted in matrices with small sparsity ($s = 5$). Other formulations and other field problems would lead to a larger $s$. For example, 3D structured-grid finite element analysis of elasticity problems would result in $s = 27 \times 3 = 81$. The proposed strategy and algorithm, in theory, extends to such scenarios. However, the performance of SPAI and efficient embedding of the QUBO problems need to be investigated. (3) We selected a simple a priori sparsity pattern for the preconditioner $M$; adaptive patterns and their impact on Q-PCG need to be explored. (4) One can potentially exploit parallel annealing [36] for computing $M$. (5) We limited the box representation to 2 qubits per real variable (see Eq. 16); extending to multiple qubits will accelerate convergence but will require a larger number of qubits.

**Compliance with ethical standards**

The authors declare that they have no conflict of interest.

**Replication of Results**

The Python code pertinent to this paper is available at https://github.com/UW-ERSL/SPAI.
Acknowledgments

We would like to thank the Graduate School of UW-Madison for the the Vilas Associate grant.

References


