Conversion of HHL Algorithm to QCFD Model

Samer Rahmeh
sam@samrahmeh.com [samgr55]

Platform: Dynex Marketplace
JOB POSTING ID: #54

Document Overview

This document provides my steps and methodology on converting the Harrow-Hassidim-Lloyd (HHL) algorithm, typically used for solving linear systems on quantum computers, into a Quadratic Unconstrained Binary Optimization (QUBO) model termed as "QCFD" to be computed on DYNEX Neuromorphic Network. This adaptation allows the use of classical and quantum-inspired solvers (a.k.a Simulated Annealing Sampler) and DYNEX Network users for finding solutions.

1 Introduction

1.1 HHL Algorithm Overview

The Harrow-Hassidim-Lloyd (HHL) algorithm is a quantum algorithm introduced by Aram Harrow, Avinatan Hassidim, and Seth Lloyd in 2009. It is designed to solve systems of linear equations, specifically of the form $Ax = b$, where A is a known matrix and b is a known vector. The HHL algorithm is significant in the realm of quantum computing as it offers an exponential speedup over the best-known classical algorithms for certain types of problems.

The HHL algorithm is particularly useful in fields requiring the solution of large systems of linear equations, which is common in computational fluid dynamics (CFD). CFD involves the use of computers to solve the mathematical equations governing fluid flow and is widely used in engineering, physics, and other disciplines for simulation and analysis of fluid dynamics.

The general steps of the HHL algorithm include:

1. Prepare the quantum state $|b\rangle$ corresponding to the vector b.
2. Apply the quantum phase estimation algorithm to the unitary operator e^{iAt}, encoding the eigenvalues of A into a set of qubits.
3. Perform controlled rotations on an ancillary qubit based on the encoded eigenvalues, effectively inverting the eigenvalues.
4. Uncompute the phase estimation to decode the solution into the final state $|x\rangle$ which is proportional to the solution $A^{-1}b$.
5. Measure the ancillary qubit to ensure it is in the correct state, effectively post-selecting the valid solutions.

The HHL algorithm's complexity and runtime depend on the sparsity and condition number of the matrix A, with the best speedups achieved for sparse and well-conditioned matrices. The HHL algorithm is promising for accelerating simulations in CFD, where solving large, sparse linear systems is common.

In CFD, the HHL algorithm can be particularly impactful in simulations involving complex geometries and boundary conditions, where the computational burden of solving linear systems is a significant bottleneck. By leveraging the quantum parallelism and speedups offered by the HHL algorithm, researchers and engineers can potentially achieve more accurate simulations in less time compared to classical methods.

The typical representation of the HHL algorithm involves the following equations:

$$\begin{aligned}
A|x\rangle &= b \quad \text{(The linear system)} \\
|b\rangle &= \sum_i \beta_i |u_i\rangle \quad \text{(Decomposition of } b \text{ into eigenvectors of } A\text{)} \\
|x\rangle &\propto A^{-1}|b\rangle \quad \text{(The solution state)}
\end{aligned}$$

However, it is important to note that while the HHL algorithm offers theoretical speedups, practical implementation on current quantum hardware may face challenges due to errors, qubit limitations, and the complexity of quantum state preparation and measurement. Ongoing research in quantum algorithms and hardware continues to advance the potential for practical applications of the HHL algorithm and its variants in fields like CFD and beyond.

2 Problem Formulation

2.1 Linear System Representation

Linear systems form the backbone of numerous scientific and engineering disciplines. A linear system is typically represented in the form of $Ax = b$, where A is a matrix representing the coefficients of the system, x is the vector of unknowns, and b is the result vector. These systems can vary in size, complexity, and the properties of the matrix A, such as sparsity and condition number, which directly affect the approach and efficiency of solution methods.

In the realm of classical computing, various numerical methods are employed to solve linear systems, including direct methods like Gaussian elimination and iterative methods like Jacobi, Gauss-Seidel, and conjugate gradient algorithms. In the quantum domain, algorithms like the HHL algorithm have been proposed to offer exponential speedup for certain types of linear systems.

Qiskit's NumPyLinearSolver: As a representative of classical solution methods, Qiskit's NumPyLinearSolver is an efficient solver for small to medium-sized linear systems. It provides a direct interface to leverage classical computational resources to solve linear systems accurately. This solver is often used as a benchmark or comparison point for quantum linear system algorithms.

Conversion to QUBO Approach: Moving from classical or quantum linear solvers to optimization-based methods involves representing the linear system as an optimization problem. The QUBO format is particularly suitable for optimization problems and is defined as minimizing a quadratic function of binary variables, expressed as $x^T Qx$. To convert a linear system into a QUBO problem, one generally minimizes the residual $||Ax - b||^2$. This conversion process involves discretizing the continuous variables of the linear system into binary variables and then formulating the objective function in the quadratic binary form.

The Dynex platform, equipped with memristor simulators, provides a powerful environment for solving these QUBO problems. Memristors, or memory resistors, are non-volatile resistive devices that can retain a state of resistance even after the power is turned off. They are particularly effective in simulating and solving optimization problems due to their ability to compactly represent and process binary variables. By leveraging memristor simulators, the Dynex platform offers an efficient way to solve QUBO problems derived from linear systems, making it a suitable choice for tasks requiring rapid and accurate optimization solutions.

In practice, the conversion from a linear system to a QUBO model and solving it via Dynex's memristor simulators can significantly enhance the performance and applicability of solving linear systems, especially in scenarios where traditional methods are infeasible or less efficient. This approach aligns well with modern computational needs in areas ranging from material science to financial modeling, where large-scale optimization is frequently required.

By utilizing the conversion to a QUBO approach and the advanced simulation capabilities of the Dynex platform, developers and researchers can tackle complex linear systems more effectively, harnessing the power of optimization and advanced computational technologies.

2.2 Objective Function

The objective function in the context of solving linear systems is a measure of the solution's accuracy or quality. For a linear system represented as $Ax = b$, where A is a known matrix, x is the vector of unknowns, and b is the result vector, the most common objective function is the residual sum of squares, represented as $||Ax - b||^2$. This objective function quantifies the difference between the left-hand side and the right-hand side of the linear system equation, aiming to minimize this difference to find the best solution.

Formulation of the Objective Function: The residual sum of squares can be expanded as follows:

$$||Ax - b||^2 = (Ax - b)^T(Ax - b) = x^T A^T Ax - 2b^T Ax + b^T b$$

Here, $x^T A^T Ax$ represents the quadratic terms, $-2b^T Ax$ represents the linear terms with respect to x, and $b^T b$ is a constant term. The goal is to minimize this objective function to find the vector x that makes Ax as close to b as possible.

Technical Rationale and Equations: The decision to use $||Ax - b||^2$ as the objective function is driven by its suitability for least squares problems, a common approach in regression and fitting tasks. The least squares method is particularly effective because it provides a single solution in cases where the system might be over-determined (more equations than unknowns) or under-determined (more unknowns than equations), and it tends to minimize the impact of errors or noise in the data.

By converting this objective function into a QUBO format, it allows leveraging advanced computational techniques, including quantum computing, to solve the linear system. The process of reaching this formulation involves a careful balance between maintaining the integrity of the original problem and adapting it to the constraints and advantages of binary optimization methods.

This conversion process is a key step in translating classical linear algebra problems into a format suitable for the next generation of computation technologies, opening up new avenues for solving complex problems more efficiently.

3 Discretization and Encoding

3.1 Precision and Range

The precision and range of variables in a solution vector x are critical in the conversion of linear systems to the QUBO format. Precision refers to how finely a variable can be represented, typically dictated by the number of binary variables used in its encoding. Range, on the other hand, refers to the span of values that a variable can represent. In the context of solving linear systems via QUBO, higher precision allows for a more accurate representation of continuous variables, while the range ensures that the solution space covers all possible solutions.

Choosing the right balance between precision and range is crucial. Higher precision typically requires more binary variables, increasing the size and complexity of the QUBO problem. However, insufficient precision can lead to inaccurate solutions. The range must be sufficiently large to encompass all possible values of the original variables but not so large as to unnecessarily increase the problem's complexity.

3.2 Binary Encoding

Binary encoding is the process of representing continuous variables from the linear system as binary variables for the QUBO model. Each continuous variable x_i from the original problem is represented by a series of binary variables. This process typically involves deciding the number of binary variables (or bits) to represent each original variable, which directly influences the precision.

The binary encoding can be represented as:

$$x_i \approx \sum_{k=0}^{m-1} b_{ik} \cdot 2^k$$

Where b_{ik} are the binary variables and m is the total number of binary variables used for each original variable x_i. This encoding translates the original problem into a binary domain, where it can be solved using optimization techniques suitable for QUBO problems.

4 QUBO Formulation

4.1 Objective Function Transformation

The transformation of the objective function $||Ax - b||^2$ into a quadratic binary form $x^T Qx$ is central to formulating the QUBO problem. This transformation involves expressing the quadratic and linear terms of the original objective function in terms of binary variables derived from the binary encoding of the original variables. The objective function in the QUBO format aims to find binary variable values that minimize the quadratic form, which corresponds to the solution of the original linear system.

The transformation leverages the binary encoding of the original variables and reformulates the objective function into a form that is compatible with binary optimization techniques. This involves significant algebraic manipulation to ensure that all quadratic and linear terms are correctly represented in terms of binary variables.

4.2 Constructing the QUBO Matrix

Constructing the QUBO matrix involves translating all terms of the reformulated objective function into a matrix format. The QUBO matrix Q is a square matrix where each element represents the coefficient of the interaction between two binary variables. The diagonal elements represent the linear coefficients of the binary variables, while the off-diagonal elements represent the quadratic interactions.

The construction of the QUBO matrix is a critical step as it directly affects the solution quality and efficiency of solving the QUBO problem. The matrix must accurately represent all aspects of the original problem, including the relationships between variables and the objective function's structure.

4.3 Decoding Each Original Variable

Once the solution to the QUBO problem is obtained in the form of binary variables, it is essential to decode these back into the original variable space to interpret the solution in the context of the original problem. The decoding process involves reversing the binary to integer conversion used during the encoding phase.

Binary to Integer Conversion: For a given original variable x_i, assume it was encoded using m binary variables b_{ik}. The value of x_i can be reconstructed from its binary representation using the following formula:

$$x_i \approx \sum_{k=0}^{m-1} b_{ik} \cdot 2^k$$

Decoding Procedure: The decoding procedure involves iterating over each set of binary variables associated with the original variables and converting them back using the binary to integer conversion formula. This process ensures that the solutions derived from the QUBO model are interpretable and can be validated against the original linear system. Accurate decoding is crucial for verifying the quality and applicability of the QUBO solution to the original problem.

By implementing this decoding procedure, the binary solution obtained from the QUBO model can be effectively translated back into a meaningful solution for the original linear system, allowing for validation and application of the results.

5 Implementation in QCFD Class

The QCFD (Quantum Computational Fluid Dynamics) class is a Python implementation designed to convert traditional linear systems into a QUBO model suitable for optimization techniques, specifically for fluid dynamics problems but applicable to a wider range of linear systems. The class provides methods for encoding the problem, solving it using various backends, and decoding the solution. The implementation was carried out using Python 3.9.17, with Qiskit 0.32.0 for potential quantum enhancements, dimod 0.12.14 for handling discrete models, and Dynex 0.1.11 for Neuromorphic Super Computing (DYNEX Blockchain).

5.1 Class Overview

The QCFD class serves as a framework for converting linear systems described by $Ax = b$ into their equivalent QUBO formulations. It encapsulates the entire process from encoding the linear system, solving the QUBO problem using different computational backends, to decoding the binary solution back to the original problem space. The class aims to provide a seamless interface for users to leverage both classical and quantum computational resources to solve linear systems more efficiently.

5.2 Lin2QUBO Method

The `Lin2QUBO` method is a central component of the QCFD class. It converts the given linear system into a QUBO problem by discretizing the continuous variables into binary variables and formulating the quadratic binary objective function. This method meticulously constructs the QUBO matrix Q representing the objective function $x^T Qx$, ensuring that it accurately reflects the original linear system's characteristics and constraints.

5.3 Compute Method

The `compute` method is the primary interface for solving the QUBO problem. It takes a matrix and vector representing the linear system, along with a parameter specifying the computational backend ('local' for classical simulated annealing or 'dynex' for using the Dynex platform). The method first invokes `Lin2QUBO` to transform the linear system into a QUBO matrix and then solves it using the specified backend. The choice of backend allows for flexibility in computational resources and methods, adapting to the needs and availability of different solving techniques.

5.4 Decoding Solution

After obtaining the binary solution from the QUBO solver, the `DecodeSol` method decodes this binary representation back into the original variable space. This process involves reversing the binary encoding and reconstructing the continuous variables from their binary counterparts. The decoding is a crucial step as it translates the optimization solution back into a form that is directly interpretable and applicable to the original linear system.

By providing these functionalities, the QCFD class allows users to tackle linear systems using advanced optimization techniques, harnessing both classical and quantum computational resources. The implementation in Python, coupled with the use of specific versions of Qiskit, dimod, and Dynex, ensures a robust and versatile framework for solving a wide range of linear systems more efficiently.

6 Solution Methods

This section explores the various solution methods employed by the QCFD class to solve QUBO problems derived from linear systems. These methods leverage both local simulation and advanced computational resources provided by the Dynex Marketplace, offering a flexible and powerful approach to solving complex optimization problems.

6.1 Local Simulation

The `SimulatedAnnealingSampler` from the dimod library is a classical algorithm that mimics the process of annealing in metallurgy to solve optimization problems, including QUBO. This sampler iteratively updates the solution, attempting to minimize the energy (objective function) of the system. It's particularly useful for local simulations where quick and cost-effective solutions are needed.

In the QCFD class, the `SimulatedAnnealingSampler` is utilized as the default or local method for solving QUBO problems. Users can opt for this method when they prefer to run the computations locally or when they seek a more accessible or immediate solution. While it may not always provide the global minimum, especially for very complex or large problems, it's a robust and widely used method for a broad range of optimization tasks.

6.2 Dynex Marketplace

The Dynex Marketplace is an innovative platform that connects users to the Dynex Neuromorphic Network, a decentralized system that simulates memristors using an extensive array of over 250,000 GPUs. This network represents one of the most comprehensive deployments of neuromorphic computing, providing immense computational power and speed for solving complex problems.

Through the DynexSDK, users can access the Dynex Marketplace to submit QUBO problems derived from linear systems. The marketplace facilitates the connection to the Neuromorphic Network, allowing users to leverage its computational resources for optimizing and solving their problems. This access opens up new possibilities for handling larger and more complex QUBO problems that would be infeasible or less efficient to solve locally or with traditional methods.

The integration of the QCFD class with the Dynex Marketplace is particularly powerful for tasks requiring rapid and accurate optimization solutions. By leveraging the decentralized Neuromorphic computing system, users can achieve significant speedups and improved solution quality for a wide range of applications, from scientific research to industrial problem-solving.

In summary, the Dynex Marketplace represents a perfect option for users of the QCFD class to solve their QUBO problems, providing access to a vast and powerful computational ecosystem that stands at the forefront of neuromorphic and optimization technologies.

7 Validation and Testing

Validation and testing are crucial components of developing reliable and accurate computational models. For the QCFD class, which translates linear systems into QUBO problems and solves them using various computational backends, ensuring the fidelity and accuracy of the solutions is paramount. This section outlines the strategies employed for testing and validating the QCFD model and how to interpret and validate the results.

7.1 Testing Strategy

The testing strategy for the QCFD model involves several key components:

1. **Benchmarking Against Known Solutions**: The QCFD model's solutions are compared against known solutions of the linear systems. This might involve comparing against analytical solutions where available or against solutions computed by well-established classical methods.

2. **Fidelity Measurements**: Using Qiskit's `state_fidelity` method, the fidelity of the quantum solutions obtained via the HHL algorithm or the QUBO solutions obtained via the QCFD model is compared against the classical solutions. Fidelity serves as a measure of the closeness between the quantum state produced by the algorithm and the ideal state corresponding to the true solution.

3. **Comparative Analysis**: The results are compared against documented benchmarks, specifically referencing the paper "An Introduction to Algorithms in Quantum Computation of Fluid Dynamics." The comparison focuses on fidelity scores reported in Section 10.2.2 "Results" of the paper, where the **HHL Quantum solution's** fidelity with the **Classical Solution** was noted as **96.8017**.

7.2 Results Interpretation

The results from the QCFD model are interpreted in the context of the original problem's requirements and constraints. The fidelity score is a key metric, providing a quantitative measure of the solution's quality. For instance, in the QCFD.ipynb output cell 11, the fidelity result of the **DYNEX QCFD** versus **Classical Solution** is reported as **98.5251**, indicating a high degree of precision.

When interpreting results, it is important to consider:

- **Fidelity Scores**: Higher fidelity scores indicate a solution closer to the ideal or true solution. Scores close to 1 or 100% are indicative of high accuracy.

- **Contextual Factors**: The nature of the problem, the precision of encoding, and the computational resources used can all impact the solution quality and fidelity scores. These factors should be taken into account when evaluating the results.

- **Comparative Benchmarks**: Comparing the QCFD model's results with those obtained from classical methods or other quantum algorithms provides a baseline for assessing improvements and the relative performance of the model.

Validation and testing not only ensure the reliability of the QCFD model but also help in identifying areas for improvement and optimization in future iterations. By rigorously comparing the model's solutions with known benchmarks and measuring fidelity, developers and users can have confidence in the model's capabilities and the solutions it provides.

8 Conclusion

This document has detailed the journey of converting the Harrow-Hassidim-Lloyd (HHL) algorithm, traditionally used in quantum computing for solving linear systems, into a Quadratic Unconstrained Binary Optimization (QUBO) model through the QCFD class. This conversion facilitates the use of various computational backends, including classical simulated annealing and the advanced neuromorphic computing resources provided by Dynex. The QCFD model represents a significant advancement in solving linear systems more efficiently and effectively.

From my experience as a Machine Learning Engineer and Computational Designer, the QCFD model has the potential to revolutionize several industries by significantly enhancing the efficiency and accessibility of computational fluid dynamics (CFD) simulations. Industries such as aviation, automotive, and architecture engineering construction (AEC) stand to benefit immensely from the reduced computational time and resources offered by QCFD when integrated with Dynex's Decentralized Neuromorphic Super Computing Platform.

Potential Applications:

- **Aviation**: Faster and more efficient design and testing of aircraft, leading to improved performance and safety.
- **Automotive**: Enhanced aerodynamics and thermal management simulations for vehicles, contributing to better fuel efficiency and design.
- **Architecture Engineering Construction (AEC)**: Advanced simulations for building and structural design, promoting sustainability, safety, and innovation.

Future Work: Looking ahead, the QCFD model will continue to evolve, incorporating feedback and advancements in both quantum computing and optimization techniques. Continuous collaboration with industry and academic partners will be crucial in refining the model and exploring new applications. As computational power and algorithms advance, the QCFD's impact is expected to grow, driving innovation and efficiency across multiple sectors.

In conclusion, the QCFD model, coupled with Dynex's computational platform, represents a powerful tool for industries requiring rapid and accurate CFD simulations. Its potential to reduce computational time and resource requirements marks a significant step forward in the broader adoption and application of advanced CFD solutions.

References

1. Bharadwaj, S. S., & Sreenivasan, K. R. *An Introduction to Algorithms in Quantum Computation of Fluid Dynamics*. Department of Mechanical and Aerospace Engineering, New York University.
2. Farhi, E., Goldstone, J., & Gutmann, S. (2014). A Quantum Approximate Optimization Algorithm. *arXiv preprint arXiv:1411.4028*.
3. Qiskit: An Open-source Framework for Quantum Computing. `https://qiskit.org/`
4. DynexSDK Documentation. (2023). Retrieved from `https://docs.dynexcoin.org/`
5. Python Software Foundation. `https://www.python.org/`
6. D-Wave Systems Inc. (2021). Dimod. Retrieved from `https://docs.ocean.dwavesys.com`