Preliminary Lattice Boltzmann Method Simulation using Intel Quantum SDK

Shraddha Mahesh Thanki, Tejas Shinde[1]

*Corresponding author: shraddha.thanki@stud.th-deg.de

Keywords:
Lattice Boltzmann Method (LBM)
Quantum Lattice Boltzmann Method (QLBM)
Advection-Diffusion Simulation
Computational Fluid Dynamics (CFD)
Quantum Simulation
State Vector Extraction

Abstract In this report, the Quantum Lattice Boltzmann Method (LBM) is applied to examine the novel intersection of quantum computing and computational fluid dynamics (CFD). This study, a collaboration between Intel Labs and Deggendorf Institute of Technology, utilizes the Intel Quantum Software Development Kit (SDK) to tackle some of the most pressing challenges in computational physics, such as tricky transport phenomena like heat and mass transfer. In this study, we implement the Quantum Lattice Boltzmann Method, as derived by Ljubomir Budinski, using Intel's Quantum SDK. The focus is on a simple 1-dimensional advection-diffusion equation to demonstrate the efficacy and potential of the Quantum LBM algorithm in a quantum computing environment. This report not only showcases the successful application of quantum algorithms in solving complex fluid dynamics problems but also paves the way for future advancements in the field, leveraging the unique capabilities of quantum computing.

 Submitted: 31 December 2023 as the Report.

1. Objectives of Simulation

This objective involves a comprehensive exploration and review of various quantum algorithms relevant to fluid dynamics, with a primary focus on those related to the Lattice Boltzmann method.

This objective aims to apply the knowledge gained from the quantum algorithms study to develop a practical solution. Specifically, it involves the implementation of a basic fluid dynamics problem using the Lattice Boltzmann method.

This objective involves the utilization of the Intel Quantum Software Development Kit (SDK) as a key tool for implementing the Quantum Lattice Boltzmann method. It includes learning and leveraging the features and capabilities of the SDK for quantum computing.

2. The Intel Quantum SDK

The Intel Quantum SDK is a toolkit designed for programming quantum computers. It bridges the gap between classical and quantum computing, allowing them to work together (see Figure 1). This hybrid approach is key because it lets us apply quantum computing power to specific parts of a problem while using conventional computing for the rest.

Included with the SDK is a full-state simulator, which acts like a quantum computer emulator on our regular computers. This is incredibly useful for testing our quantum programs and making sure they're ready before we run them on actual quantum hardware.

Figure 1 shows the process flow of the Intel Quantum SDK, highlighting how it integrates with classical computing. This flowchart details each stage of simulation and development, from writing the code to executing it on the simulator, and eventually on a quantum processor. Understanding this process is crucial as it allows us to optimize our quantum solutions and prepare them for real-world applications. [1]

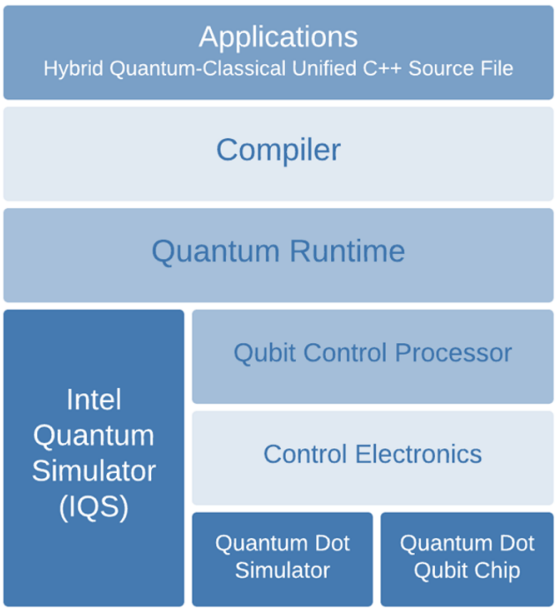

FIGURE 1. The Intel Quantum SDK

3. Classical Lattice Boltzmann Method

3.1 Introduction

Lattice Boltzmann Method is a dynamic method that simulates the macroscopic behavior of fluids by using a simple mesoscopic model. It inherited the main principles of Lattice Gas Automaton (LGA) and made improvements. From the lattice gas automaton, it is possible to derive the macroscopic Navier-Stokes and the Advection diffusion equations.

3.2 Derivation of the Lattice Boltzmann Method

The Boltzmann equation talks about the evolution of the probability distribution function. The Boltzmann transport equation is derived using conservation principles.

$$\frac{\partial f}{\partial t} + \frac{\partial f}{\partial x_i}\left(\frac{dx_i}{dt}\right) + \frac{\partial f}{\partial \xi_i}\left(\frac{d\xi_i}{dt}\right) = \Omega(f) \quad (3.2.1)$$

$\frac{d\xi_i}{dt} = \xi_i$ - microscopic particle velocity (3.2.2)

$\frac{F_i}{\rho} = \frac{d\xi_i}{dt}$ - Body force per unit mass (3.2.3)

F_i - Body force per unit volume (3.2.4)

LBM is a mesoscopic technique that makes use of microscopic and macroscopic properties.

So, this $\frac{d\xi_i}{dt} = \xi_i$ is the microscopic particle velocity.

The probability distribution function is connected to macroscopic variables density ρ.

3.3 Classical 1D Advection diffusion Lattice Boltzmann Method

The advection-diffusion equation is given as

$$\frac{\partial C}{\partial t} + \nabla \cdot (Cu) = \nabla \cdot (D \nabla C)$$

The discretized lattice Boltzmann equation in velocity and physical space and time using the Bhatnagar Gross Krook operator.

$$f_i(x + e_i\Delta t, t + \Delta t) = f_i(x,t) - \frac{\Delta t}{\tau}\left[f_i(x,t) - f_i^{\text{eq}}(x,t)\right]$$

The equilibrium distribution function for the advection-diffusion is given as

$$f_i^{\text{eq}}(\vec{x},t) = w_i \rho(x,t)\left(1 + \frac{e_i \cdot \vec{u}}{c_s^2}\right)$$

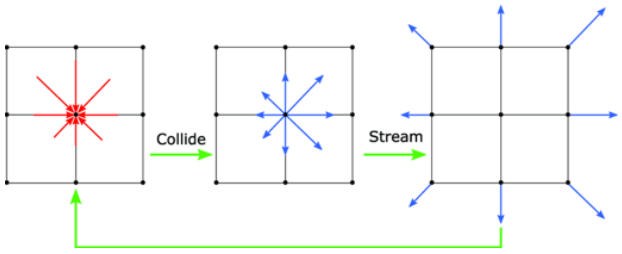

FIGURE 2. time step using LBM [2]

Collision step post-collision distribution function
$f_i(x,t) = f_i(x,t)(1 - \Delta t/\tau) + f_i^{eq}(x,t)\Delta t/\tau$

For simplicity we take $\tau = 1$ and $\Delta t = 1$

$$f_i(x,t) = f_i^{eq}(x,t)$$

The streaming step is given as

$$f_i(x + e_i\Delta t, t + \Delta t) = f_i(x,t)$$

At last, we calculate the concentration

$$\rho(x,t) = \sum_i f_i(x,t)$$

4. Problem Definition Motion of 1D Gaussian Hill

The Gauss-like distribution was used.

- Initial Conditions
- Periodic Boundary Condition
- No of Lattice Sites equals 32
- The initial Concentration
- C(5, 0) = C(7, 0) = 0.5 and C(6, 0) = 1.0

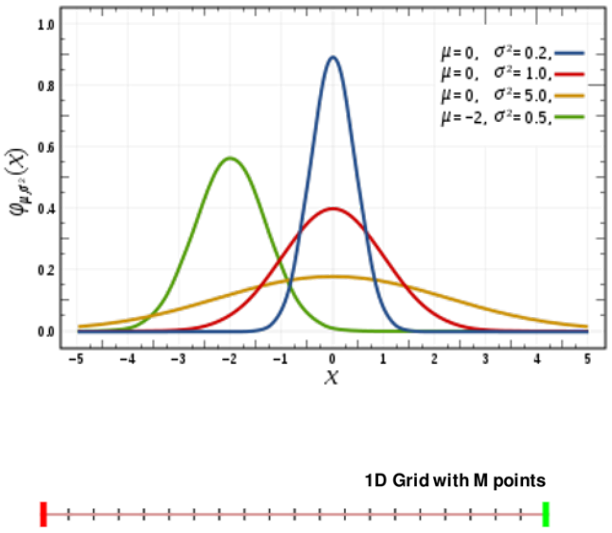

FIGURE 3. 1D Gaussian Hill

5. Quantum Lattice Boltzmann Method and its Implementation

Working with amplitudes encode amplitude and extract amplitudes Five major sections as per the given diagram [4]

1. Encoding
2. Collision
3. Propagation
4. Macroscopic Calculation
5. Extracting Concentration

5.1 Basic Flow

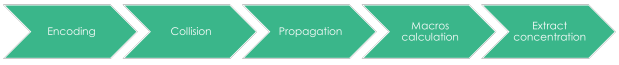

FIGURE 4. Basic Flow

5.2 Ideal Time Stepping

Figure [5] shows the steps for time stepping in our quantum simulations. Starting with a set number of iterations, we first establish an initial vector at the beginning of the simulation, which is then normalized and processed through Qiskit and exported as OpenQASM. This is translated to C++ and fed into the Intel Quantum SDK to run the Quantum Lattice Boltzmann Method (QLBM) for the advection-diffusion equation. After each run, the state vector is updated for the next iteration until the simulation ends when the set number of iterations is reached.

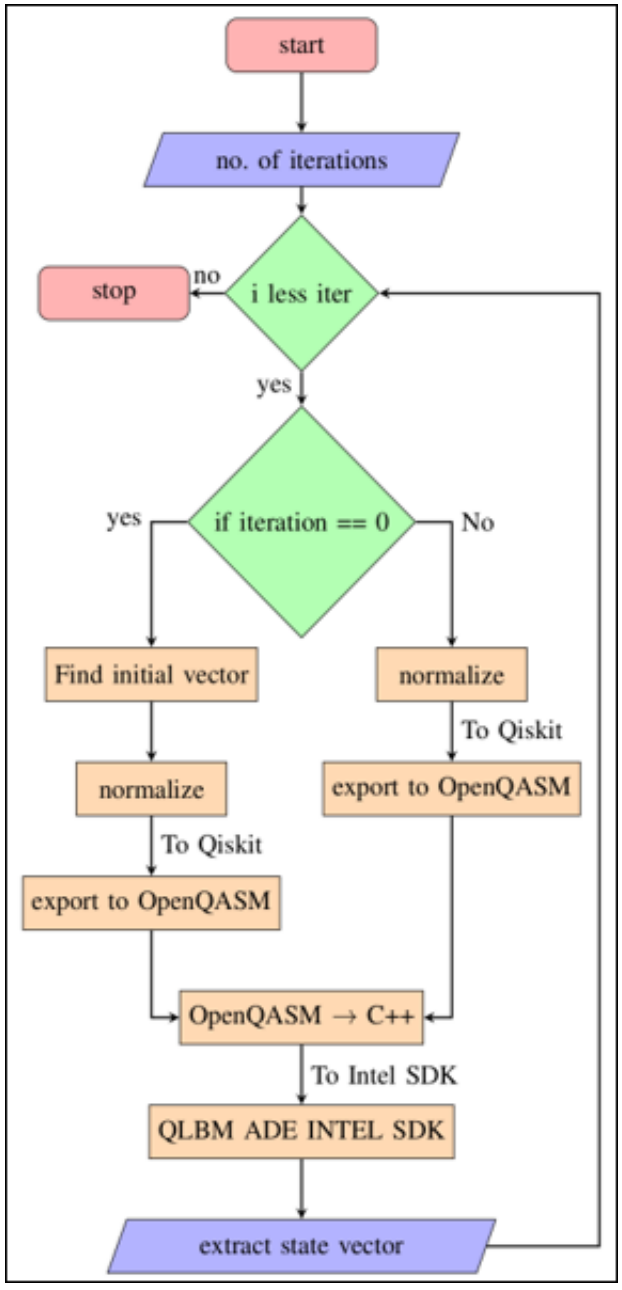

FIGURE 5. Workflow of the QLBM Advection-Diffusion Equation

6. Encoding and Initial state

Initial distribution of the concentration variable C(x, 0) in the form of a vector C = [C1,0 , . . . , C1,31 , C2,0 , . . . , C1,31]T

To encode the vector into quantum states we require the quantum register, q.
The qubit register consists of only 1 qubit and is used in the controlled operation in the collision step as shown. Let's us say,
qn = number of qubits in quantum register q
M = number of Lattice Sites
qn = log2 (2M)
In this case, M = 32, qn = 6

The initial concentrations C(5, 0) = C(7, 0) = 0.5 and C(6, 0) = 1.0

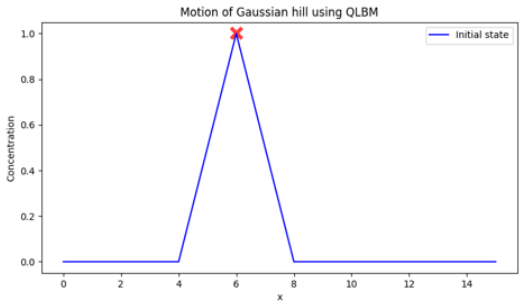

FIGURE 6. Motion of Gaussian hill using QLBM

we get a triangle-like shape as the initial condition.

6.1 Normalizing and Exporting to OpenQASM

In the normalization step, we first calculate the magnitude of a vector using the Euclidean norm, which is the square root of the sum of the squares of the vector's components. This is a crucial step in preparing for quantum simulations, as it ensures that the quantum state has a magnitude of one, meeting the requirement for a valid state in quantum mechanics.

$$\|\vec{V}\| = \sqrt{\vec{V} \cdot \vec{V}} = \sqrt{V_1^2 + \cdots + V_n^2}$$

Once we have the magnitude, we then normalize the vector by dividing each component by this magnitude, resulting in a unit vector. Normalization is essential because quantum computing operates in a normalized state space, and all quantum states must have a norm of one.

$$\hat{V} = \frac{\vec{V}}{\|\vec{V}\|} = \left(\frac{V_1}{\|\vec{V}\|}, \frac{V_2}{\|\vec{V}\|}, \ldots, \frac{V_n}{\|\vec{V}\|}\right)$$

After normalization, we export the quantum state to OpenQASM format. OpenQASM is an intermediate representation that allows for the description of quantum circuits in a hardware-agnostic manner. This step is vital for translating our high-level quantum algorithm into a form that can be executed on a quantum simulator or quantum processor, ensuring that the quantum states are represented accurately in the subsequent quantum computations.

6.2 Conversion from OpenQASM to C++

After normalizing our quantum state, the next step is to translate the quantum circuit from the OpenQASM format to C++. OpenQASM, which stands for Open Quantum Assembly Language, provides a way to express quantum circuits in a hardware-agnostic syntax. It is particularly useful for defining the sequence of quantum gates and measurements that make up our quantum algorithm.

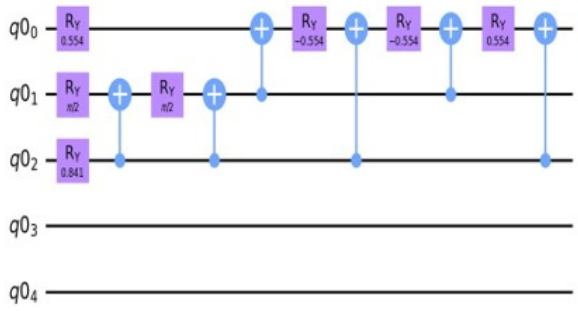

FIGURE 7. Quantum Circuit Implementation for State Preparation in QLBM Simulation

The circuit shown in the diagram [7] represents the quantum logic gates applied to the qubits. For instance, 'Ry' denotes a rotation around the y-axis of the Bloch sphere by a given angle, and the '+' symbols represent controlled-NOT (CNOT) gates that entangle qubits, a fundamental operation in quantum computing.

To execute this circuit on a classical computer or control actual quantum hardware, we convert it into C++, which involves translating these quantum operations into a set of instructions that can be understood by the Intel Quantum SDK. This conversion is essential for integrating quantum algorithms into classical simulation environments or for preparing them for execution on quantum hardware. The Intel SDK provides the necessary tools and interfaces to simulate the behavior of this quantum circuit on classical hardware, allowing us to test and refine our quantum algorithms within a familiar programming context.

7. Collision

In the collision step of the Lattice Boltzmann Method, the distribution functions are updated to simulate particle interactions. This is achieved by considering the pre-collisional states and transforming them through collision operators. The operators $\hat{A}_1$ and $\hat{A}_2$ are used to calculate the post-collisional states, defined by the equations:

$$\hat{A}_1 = A + i(I - A^2),$$
$$\hat{A}_2 = A - i(I - A^2),$$

where A represents the amplitude of the pre-collision state and I is the identity matrix. The overall state $\bar{A}$ is then reconstructed as the average of these two states:

$$\bar{A} = \frac{1}{2}(\hat{A}_1 + \hat{A}_2).$$

This calculation ensures that the quantum states adhere to the unitary property required for physical feasibility in quantum computations. The equilibrium distribution functions f_i^{eq} are then updated using the formula:

$$f_i^{\text{eq}}(\vec{x}, t) = w_i C(\vec{x}, t) \left(1 + \frac{\vec{e}_i \cdot \vec{u}}{c_s^2}\right),$$

where w_i are the weights, $C(\vec{x}, t)$ is a scalar function, $\vec{e}_i$ are the lattice velocity vectors, and $\vec{u}$ is the macroscopic velocity field with a speed of sound c_s. In the specific case where $\vec{u}$ is given as 0.2 and c_s is 1, the equilibrium functions for directions 1 and 2 are computed as:

$$f_1^{\text{eq}} = 0.6C(\vec{x}, t),$$
$$f_2^{\text{eq}} = 0.4C(\vec{x}, t),$$

assuming the respective directional velocities $e_1 = 1$ and $e_2 = -1$ with weights $w_1, w_2 = 0.5$. The quantum circuit representation of these operations involves applying unitary transformations $U_{\hat{A}_1}$ and $U_{\hat{A}_2}$ to the quantum bits, which correspond to the diagonal elements of the matrices $\hat{A}_1$ and $\hat{A}_2$, respectively. Such transformations are crucial for simulating the evolution of the quantum state corresponding to the fluid particles in the LBM simulation.

8. Propagation

After the collision phase, where the particle distribution functions are updated to simulate interactions at each lattice node, we proceed to the propagation step. During this phase, the updated distribution functions 'travel' to neighboring lattice nodes. This movement is guided by the discrete velocity vectors that define the direction and magnitude of particle motion in the lattice.

Propagation is a key step in the LBM algorithm as it simulates the advection process in fluid flow. The success of this step is critical for ensuring the correct macroscopic behavior of the simulated fluid. The process is mathematically represented by shifting the post-collision distribution functions along their respective lattice directions.

8.1 Quantum Kernel for Shifts

In our implementation, two distinct distribution functions, f_1 and f_2, represent particle populations moving in opposite directions. The distribution f_1 is associated with a rightward movement at the speed of $e_1 = 1$, corresponding to a shift to the right in the lattice. Conversely, f_2 moves left with a speed of $e_2 = -1$, indicating a leftward shift. These movements are encoded quantum mechanically through a series of quantum gates that enact the shift operations on our quantum state.

The accompanying quantum circuit diagrams depict the gate sequences required for these shifts. For example, the rightward shift for f_1 is achieved by the application of a quantum gate that increases the index of the position state of each particle, while the leftward shift for f_2 is implemented by a gate that decreases this index. This bidirectional propagation is crucial for the accurate simulation of particle dynamics, as it ensures that the distribution functions accurately reflect the movement of particles in both directions within the simulated fluid.

By leveraging quantum computing's parallelism, we can simultaneously propagate all particle positions, a task that would be significantly more resource-intensive on classical computers. This efficient propagation is key to advancing the simulation to the next step, where macroscopic variables such as density and velocity are calculated from the updated quantum state.

8.2 Decomposing multi-qubit CNOT

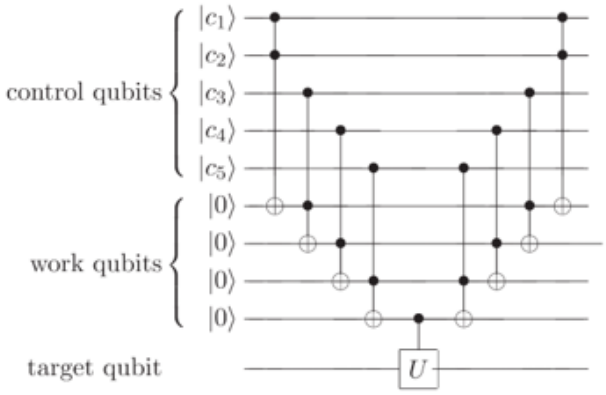

FIGURE 8. Decomposing multi-qubit CNOT

The quantum circuit depicted in Figure [8] is an essential component of the propagation step in our simulation. It demonstrates a sophisticated quantum gate, the multi-controlled unitary (MCU), which is pivotal for executing conditional operations on a single target qubit based on the state of several control qubits.

In this circuit, the control qubits, labeled $|c_1\rangle$ to $|c_5\rangle$, are responsible for activating the unitary operation U on the target qubit. The operation U is only applied if all control qubits are in the $|1\rangle$ state. To accommodate the limitations of current quantum hardware, which primarily supports two-qubit gates, work qubits (ancilla) are introduced. These work qubits, initialized in the $|0\rangle$ state, act as temporary storage to facilitate the decomposition of the MCU gate into a series of Toffoli (controlled-controlled-NOT) gates.

This decomposition process is a clever workaround that maintains the functionality of an MCU gate while only utilizing the two-qubit operations that quantum processors can natively execute. As such, the circuit provides a practical method for simulating complex, conditional dynamics within a quantum computing framework, a task central to the accurate representation of the fluid dynamics in our Lattice Boltzmann Method (LBM) simulation.

9. Macros Calculation

The macroscopic variables calculation is a critical phase in the quantum Lattice Boltzmann simulation where we derive observable quantities, such as fluid density and velocity, from the microstates of the system. This step translates the quantum information, represented by the state vector of the system, into classical information that characterizes the macroscopic behavior of the fluid.

Upon completing the encoding, collision, and propagation steps within the quantum circuit, we employ a series of quantum operations to compute the macroscopic variables. The circuit utilizes Hadamard gates to generate superposition states, facilitating the parallel processing inherent in quantum computation. Controlled operations follow, manipulating the state vector amplitudes based on the encoded data.

A specialized unitary operation is then applied, which is designed to perform the complex arithmetic involved in extracting macroscopic variables from the quantum state. This unitary operation takes into account the peculiarities of the Lattice Boltzmann model and the specific properties we aim to measure.

10. Extracting Concentration

In the final stage of our quantum Lattice Boltzmann simulation, we turn our attention to extracting the concentration of the fluid. This process begins with the application of the Euclidean norm to our quantum state vectors, allowing us to compute the magnitude of these vectors, which is indicative of the probability amplitude associated with each state. The Euclidean norm is defined mathematically as:

$$\|\vec{V}\| = \sqrt{\vec{V} \cdot \vec{V}} = \sqrt{V_1^2 + \cdots + V_n^2}$$

where $\vec{v}$ is a vector representing the state of the system, and $v_1, \ldots, v_n$ are the components of $\vec{v}$. In the context of our simulation, these components correspond to the probabilities of finding the system in specific states associated with different concentrations.

Utilizing high-performance computing resources, we perform these calculations for large sets of quantum states, ensuring that we can handle the computational load efficiently. The results are then processed to plot the concentration of the fluid across the simulated domain. Plotting these values provides us with a visual representation of the fluid's behavior, allowing us to analyze patterns such as flow direction, turbulence, and boundary layer phenomena.

Result

1D Gaussian hill

The plot effectively captures the temporal evolution of a Gaussian concentration profile, as modeled by the Quantum Lattice Boltzmann Method (QLBM). The initial state, indicated by the blue line, shows a pronounced peak, signifying a high concentration at the center of the domain. As the simulation progresses through each time stepsteps 1, 2, 3, and finally step 15the peak conspicuously diminishes and flattens out, illustrating the dispersion of the substance.

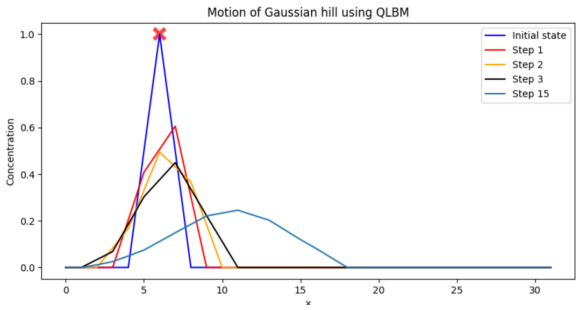

FIGURE 9. Result

Figure [9] demonstrates the diffusion phenomenon: the concentration diminishes from its peak and spreads towards the edges of the domain. The color-coded lines trace the decline in peak concentration over successive time steps, offering a clear visualization of the diffusive process as captured by the

QLBM. This spread from a concentrated source to a dispersed state is characteristic of Gaussian diffusion, validating the QLBM's effectiveness in simulating such a fundamental physical process within the quantum computing framework.

Conclusion

In conclusion, our research has conducted a thorough examination of both classical and quantum approaches to the Lattice Boltzmann Method, highlighting their parallels in modeling fluid dynamics. We have successfully reconstructed a novel quantum algorithm that solves the advection-diffusion equation using the Intel Quantum SDK.

The project centered on a one-dimensional Gaussian hill advection-diffusion problem, which was meticulously described and subsequently validated within the Intel Quantum SDK framework. Our implementation of the ideal time stepping has been critical in simulating the diffusion process, reflecting the practical application of quantum computing to solve real-world problems.

Acknowledgement

I want to express my sincere gratitude to **Tejas Shinde**. His hard work and unwavering commitment have played a crucial role in the success of this project. The impressive results we've achieved are a direct reflection of his thorough research and his use of the Intel Quantum SDK to apply the Quantum Lattice Boltzmann Method.

I would also like to extend my deepest appreciation to our esteemed mentors, **Prof. Dr. Helena Liebelt** and **Prof. Dr. Rui Li**, for their expert guidance and unwavering support. Their insights and encouragement were invaluable throughout the development of this project.

I am particularly grateful to **Dr. Kevin Rasch** from Intel Labs for the opportunity to work with the Intel Quantum SDK. His generous assistance and in-depth knowledge of the SDK have been pivotal to our project's success.

A special word of appreciation goes to **Team CFD meets Quantum**. The support provided by the team has been invaluable, and the collective wisdom has undoubtedly enriched the outcomes of this research.

References

[1] Khalate, Pradnya *et al.*: *An llvm-based c++ compiler toolchain for variational hybrid quantum-classical algorithms and quantum accelerators*, 2022. https://arxiv.org/pdf/2202.11142.pdf, visited on 2023-04-15.

[2] *Graphical illustration of a time step using lbm*. https://www.researchgate.net/figure/Graphical-illustration-of-a-time-step-using-LBM_fig9_311449012, visited on 2023-04-15.

[3] Nielsen, Michael A. and Isaac L. Chuang: *Quantum Computation and Quantum Information*. Cambridge University Press, 2010, ISBN 978-1-107-00217-3.

[4] Budinski, Ljubomir: *Quantum algorithm for the advection-diffusion equation simulated with the lattice boltzmann method*. Quantum Information Processing, 2021.

[5] Krüger, Timm *et al.*: *The Lattice Boltzmann Method - Principles and Practice*. Springer Nature, Year of Publication, ISBN 978-3-319-44647-9.

[6] Cross, Andrew W. *et al.*: *Open quantum assembly language*, 2017.

[7] contributors, Qiskit: *Qiskit: An open-source framework for quantum computing*, 2023.

[8] Kay, Alastair: *Quantikz*, 2019. https://royalholloway.figshare.com/articles/Quantikz/7000520.

[9] Childs, Andrew M.: *Universal computation by quantum walk*. Phys. Rev. Lett., 102(18):180501, 2009. https://link.aps.org/doi/10.1103/PhysRevLett.102.180501.