Project 3: Quantum Algorithm as a PDE Solver for Computational Fluid Dynamics (CFD)

Task

Solve the **1-D Burgers' Equation with Shock Tube**:

$$\frac{\partial u}{\partial t} + \frac{u\partial u}{\partial x} = \frac{\nu\partial^2 u}{\partial x}$$

Domain: $x \in$ [0,1]
IC: Riemann step $u(x, 0) = 1$ for $x \leq 0.5$, 0 otherwise
BC (Dirichlet): $u(0, t) = u_L$, $u(L, t) = u_R$ for all $t > 0$

Instruction:

This open challenge tasks participants with designing and prototyping resource-lean quantum-enhanced PDE solvers based on either **Quantum Tensor-Network (QTN)** or **Hydrodynamic Shrödinger Equation (HSE)**; **hybrid QTN-HSE** approaches are also welcome.

Algorithm Design

Introduction: Description of the framework

For this challenge, I choose to use **Quantum Tensor-Network (QTN)**, by compressing the velocity field $u(x, t)$ into **Matrix-Product States (MPSs)** [1]. This protocol draws inspiration from the work done by Peddinti et al.[2] where this compression is followed by the evolution of MPSs with divergence-free projectors. Matrix product states (MPSs) are a natural choice for efficient representation of 1D quantum low energy states of physically realistic systems. Also known as tensor trains, these represent 2^N-dimensional quantum states as a set of 2N matrices whose maximal size (also called bond dimension) depends on the amount of entanglement[2]. Thanks to that capacity, low-entangled states of 1D systems can be exponentially compressed by MPSs; hence providing state-of-the-art framework for simulating complex quantum dynamics.

1. **First step: Definition of parameters**

   I start by defining some parameters such as the number N of qubits to represent states in MPS framework (N=8), the number n of grid points, taken to be a power of 2 (n=2^N). We also define the viscosity $\nu = 0.005$, the length L=1, the time evolution T=0.2, the number of time steps $n\_steps = 260$ and the time step itself dt $= \frac{T}{n\_steps}$.

2. **Second step: Discretization of space domain**

   I discretize the space domain by taking a uniform distribution of n points between 0 and L and then, I define the grid spacing step dx.

3. **Third step: Integration of IC and BC**

   I initiate a vector u satisfying the Initial Condition (IC) $u = u(x, 0) = 1$ for $x \leq 0.5$ and 0 otherwise, of the PDE, and I constraint it to also satisfy the Dirichlet Boundary Conditions (BCs) $u(0, t) = u_L = 1$, $u(L, t) = u_R = 0$ for all $t > 0$.

4. **Fourth step: MPS encoding of initial solution**

   After creating an array "solution" to store the numerical solution of the PDE at each time step, I encode the initial vector u into a MPS denoted by "mps".

5. **Fifth step: Mapping of the PDE.**

   In order to act on the MPS constructed previously which now simulates the solution of the PDE, I need to define Matrix Product Operators that, in MPS framework, represent the differential operators contained in the PDE.

   To do that, I define two functions "create_D1_matrix" and "create_D2_matrix" that approximate the spatial derivatives of the solution of the PDE, leveraging finite differences schemes.

   After that, I convert the two matrices created into MPOs via specific commands from quimb package.

   Remark: Since they act on many-body quantum system, the two MPOs, denoted by "D1_mpo" and "D2_mpo" constructed can then be seen as "gates" in the MPS protocol.

6. **Sixth step: Time evolution.**

   This is the tricky part. Despite that, I have been able to convert the differential operators into MPOs, the nonlinear term $\frac{u\partial u}{\partial x}$ contained in the equation prevents me to perform MPO-MPS operations directly. For time evolution, I choose an Euler explicit scheme following this protocol for each iteration:

   - First, I convert back the MPS "mps" and the MPO "D1_mpo" respectively into a dense vector and a simple matrix to compute the numerical nonlinear term classically
   - Next, I also convert back the "D2_mpo" into a simple matrix to compute the viscuous term (rhs of the PDE) and then I sum both terms. this represent the rate of change of the solution with respect to time
   - Finally, I convert the corresponding sum vector into a MPS that we evolve under Euler explicit scheme.

7. **Seventh step: Solution recovery.**

   At each time step, I update the MPS "mps", covert it back into a dense vector that I store for plotting and repeat the previous protocol on the "mps" obtained on the previous iteration. At the end of this, I plot the numerical velocity u(x,t) against x, after every 40 steps of time.

Resource estimates:

I am running the code locally on my computer with characteristics:

- Architecture: x86_64
- CPU op-mode(s): 32-bit, 64-bit
- Total logical CPUs (CPU(s)): 4
- Model name: Intel(R) Core(TM) i5 CPU M 560 @ 2.67GHz
- Core(s) per socket: 2

- Socket(s): 1
- Frequency boost: enabled
- CPU max MHz: 2667.0000
- CPU min MHz: 1199.0000
- RAM: 4Gi

With all these settings, the code takes about 12min for execution. It takes less time for less number of qubits (N), but I started observing promising results from N=8.

References

[1] Jacob C. Bridgeman and Christopher T. Chubb. Hand-waving and interpretive dance: An introductory course on tensor networks. Journal of Physics A: Mathematical and Theoretical, 50(22):223001, 2017.

[2] Raghavendra Dheeraj Peddinti, Stefano Pisoni, Alessandro Marini, Philippe Lott, Henrique Argentieri, Egor Tiunov, and Leandro Aolita. Quantum-inspired framework for computational fluid dynamics. Communications Physics, 7:135, 2024.