Meta-Programming and Hybrid Parallel Strategies for Solving PDEs: An FDM and PINN Comparison [1] [2]

Seminar Presentation III

LI YIHAI

Student ID: 23345919

Supervision: Michael Peardon

September 18, 2024

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath

[1] full docs: `https://liyihai.com/html/index.html`

[2] repository: `https://github.com/liyihai-official/Final-Project`

Outline

Outline

Introduction

Recap - Deep Neural Network

Kolmogorov PDEs

Solving $u(x,T)$, for $\mathbb{R}^1 \ni T > 0$, $x \in \mathbb{R}^d$, $t \in [0,T]$, $u(t,x) = u \in \mathbb{R}^1$, $\mu(x) \in \mathbb{R}^d$, $\sigma(x) \in \mathbb{R}^{d\times d}$,

$$u_t = \frac{1}{2}\operatorname{Trace}_{\mathbb{R}^d}\left[\sigma(x)\left[\sigma(x)\right]^*\operatorname{Hess}_x u\right] + \langle \mu(x), \nabla_x u\rangle_{\mathbb{R}^d} \tag{1}$$

$$\begin{tikzcd}
\mathbb{E}\left[\phi(X_T^x)\right] \arrow[rr, "\textit{Prop. 2.7}"] & & \inf\limits_{v\in C([a,b]^d,\mathbb{R})} \mathbb{E}\left[\left|\phi(X_T^x) - v(\xi)\right|^2\right] \arrow[loop right, "\textit{Discretization \& Th. 2.8}"] \arrow[d, "\textit{Loss Function}"] & \\
u(x,T) \arrow[u, "\textit{Feynman-Kac}"] & & \mathbb{U}(x;\theta) \arrow[ll, "\approx \textit{ to Solution}"'] & \text{Neural Network} \arrow[l, "\textit{Prediction}"']
\end{tikzcd}$$

Figure: Deep Neural Network (DNN) Methodology of Solving Kolmogorov PDEs [**FIRST**]

General Form of PDEs

– $u(t,x)$ denotes with the target function, $x \in \mathbb{R}^d$.

– $\Gamma\left[\,\cdot\,;\lambda\right]$ is a non-linear operator parameterized by λ.

$$u_t(t,x) + \Gamma\left[u;\lambda\right] = 0 \tag{2}$$

Define $f(t,x)$ to be given by

$$f(t,x) = u_t(t,x) + \Gamma\left[u;\lambda\right] \tag{3}$$

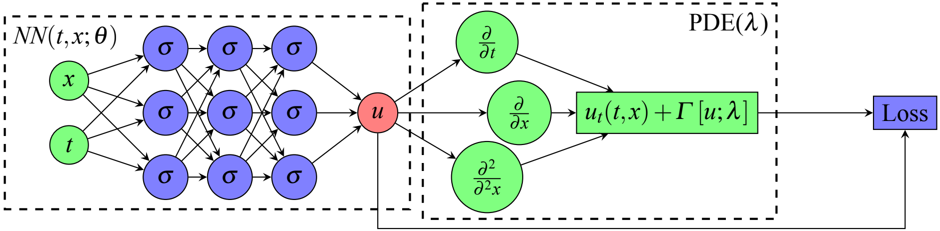

Figure: PINN, with 3 fully connected hidden layers

Comparing With Finite Difference Time Domain Method (FDTD)

1. Deep Neural Network [**FIRST**]
    - Gives lower quality approximations.
    - Takes longer time to train.
    - Possible to solve high dimension PDEs
2. Physics Informed Neural Network
    - Gives higher quality approximations.
    - Takes longer time to train.
    - Has more flexible way to get results.
    - Possible to solve high dimension PDEs

Objectives

This project focused on following objectives:

1. Implement FDTD and PINN in `C++/C`.
2. Implement FDTD hybrid parallel version using `MPI/OpenMP`.
3. Implement PINN GPU parallel version using `Libtorch/CUDA`.
4. Evaluate the efficiency and accuracy of FDTDs and PINNs.

Challanges

However, there were many obstacles including:

1. Portability.
2. Overlapping Communication and Computation.
3. Unnecessary intra-node communication
4. Communication Overhead.
5. Scalability Issues.
6. Memory Management.

Outline

General Form of problem

The PDE parametrized by number λ and an operator $\mathcal{N}[\cdot;\lambda]$, and assume the variable x is a 2D or 3D spatio-vector which is written in

$$\begin{cases} \dfrac{\partial u}{\partial t}(t,\vec{x}) + \mathcal{N}[u;\lambda] = 0 \\ u(0,\vec{x}) = \varphi(\vec{x}) \end{cases} \tag{4}$$

where φ is the initial condition, and $\vec{x} \in \Omega, t \in [0,+\infty)$.

Boundary Conditions

The Dirichlet and Von Neurmann boundary conditions are formed as

$$\begin{cases} u(t,\vec{x}) = g(t,\vec{x}) \\ \dfrac{\partial u}{\partial \vec{n}} = g(t,\vec{x}) \end{cases} \tag{5}$$

where $\vec{n}$ is the normal vector on $\overline{\Omega}$ the boundary of domain Ω.

The function

$$u(t,x,y) = x + y - xy, \ \forall \alpha \in \mathbb{R}^1 \tag{6}$$

is the solution of 2D Heat Equation 7 below

$$\begin{cases} \dfrac{\partial u}{\partial t} = \alpha \left(\dfrac{\partial u^2}{\partial^2 x} + \dfrac{\partial u^2}{\partial^2 y} \right) & (x,y) \in \Omega, \ t \in [0, +\infty) \\ u(0,x,y) = \varphi(x,y) = 0 & (x,y) \in \Omega \\ u(t,x,y) = g(x,y) = \begin{cases} y, \ x = 0, y \in (0,1) \\ 1, \ x = 1, y \in (0,1) \\ x, \ y = 0, x \in (0,1) \\ 1, \ y = 1, x \in (0,1) \end{cases} & t \in [0, +\infty) \end{cases} \tag{7}$$

The function

$$u(t,x,y,z) = x + y + z - 2xy - 2xz - 2yz + 4xyz, \ \forall \alpha \in \mathbb{R}^1 \tag{8}$$

is the solution of 3D Heat Equation 9 below

$$\begin{aligned}
&\frac{\partial u}{\partial t} = \alpha \left(\frac{\partial u^2}{\partial^2 x} + \frac{\partial u^2}{\partial^2 y} + \frac{\partial u^2}{\partial^2 z} \right) && (x,y,z) \in \Omega,\ t \in [0,+\infty) \\
&u(0,x,y,z) = \varphi(x,y,z) = 0 && (x,y,z) \in \Omega \\
&u(t,x,y,z) = g(x,y,z) = \begin{cases} y + z - 2yz, & x = 0, \\ 1 - y - z + 2yz, & x = 1, \\ x + z - 2xz, & y = 0, \\ 1 - x - z + 2xz, & y = 1, \\ x + y - 2xy, & z = 0, \\ 1 - x - y + 2xy, & z = 1 \end{cases} && t \in [0,+\infty)
\end{aligned} \tag{9}$$

Outline

STL provides containers `std::array` and `std::vector` for creating one-dimension array. There are two way for dealing with high-dimension data:

1. Nesting the one dimension arrays or vectors.
2. Hierarchy approach, designing derived classes of one-dimension array base class.

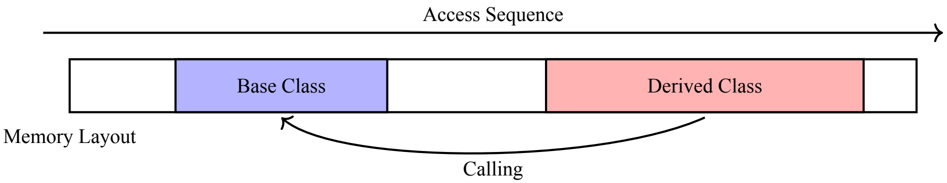

Figure: Derived Class calling members in Base class, timing is not predictable.

1. Nesting multi-dimension array has non-contiguous memory layout.
2. Derived class needs more time to access members in base class.
3. Poor cache utilization leads to poor performance.
4. MPI type create requires contiguous memory layout.

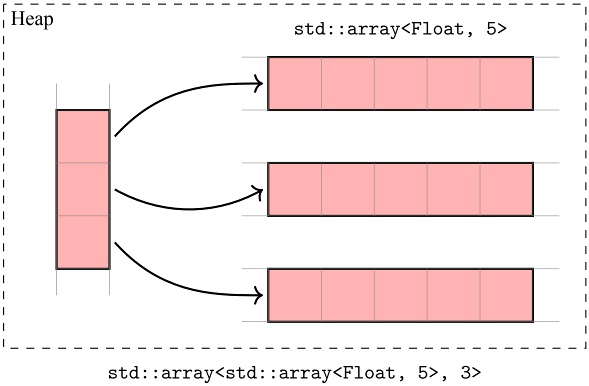

Figure: Using nested `std::array<T, N>` to store 2D array data.

N-dimension Matrix

Solution

- An external small `__multi_array_shape` object defines the routines for accessing the elements.
- Smart pointer, ensure memory's contiguous layout and safety.
- Separate into two detail and user interface objects adhering RAII rules.

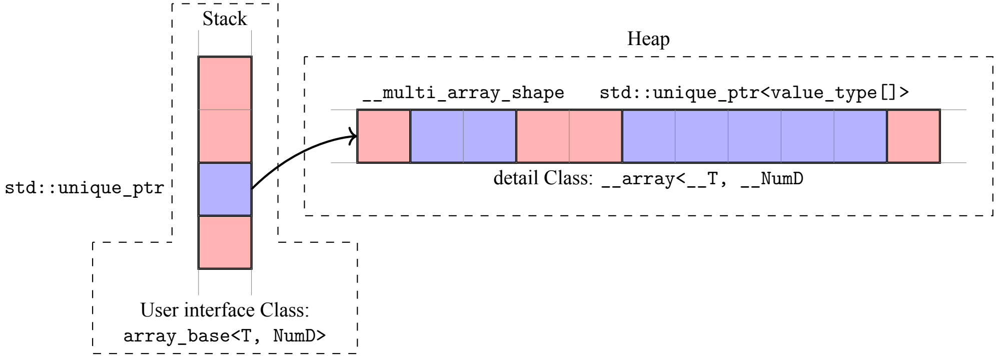

Figure: The solution of N-dimension Matrix, using detail Class, user interface Class and a shape management structure.

The hybrid PDE solver requires the MPI supports multi-threads on each processes.

- High-level libraries like Boost.MPI
  1. have better MPI resource management and other basic communication features.
  2. only provide limited useful features for latter PDE solvers.
  3. lead to lower performance than low-level OpenMPI.
- I developed an environment class for MPI
  1. better resource management than raw MPI.
  2. provides basic features exclusively for this project.

MPI Topology - Challanges

Distributed N-dimension arrays are created based on MPI N-dimension Cartesian topology.

1. Ghost communication is required in overlapping of MPI communication and local computation.
2. Parallel I/O is needed for debugging and storing results.
3. Topology information will be frequently used in PDE solver.

Parallelization of N-dimension Arrays

MPI Topology - Solution

1. An MPI Topology class defines the distribution details of N-dimension array and creating Ghost MPI datatype for communication.
2. Cartesian array has members Topology class and N-dimension array class, to ensure they are closely located on memory.
3. The external functions provide gather-based I/O and MPI I/O for handling different scenarios.
4. User interface class unified features for both topology and array classes.
5. Separate into two detail and user interface objects adhering RAII rules.

Outline

Outline

Outline

Thank you for your attention!

Any questions?