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

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

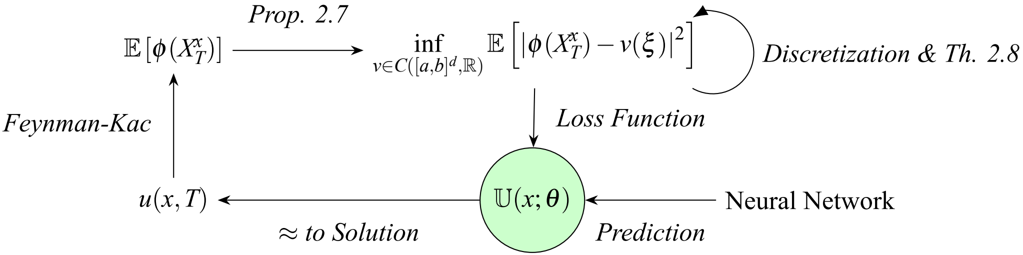


Figure: Deep Neural Network (DNN) Methodology of Solving Kolmogorov PDEs [**FIRST**]

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

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

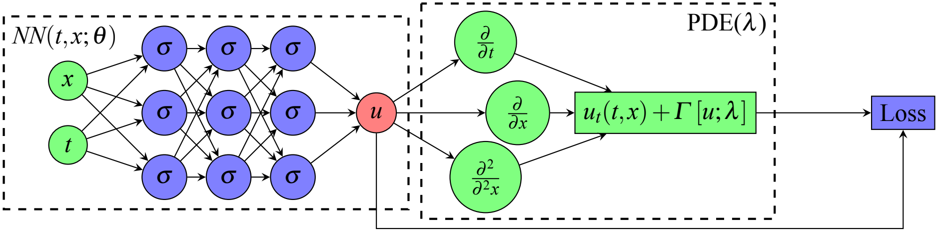


Figure: PINN, with 3 fully connected hidden layers

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

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

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

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

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

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

Thermal Conduction Systems
Heat Equation 3D

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.

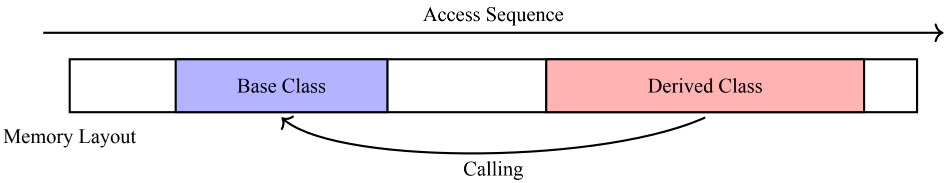


Figure: Derived Class calling members in Base class, timing is not predictable.

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.

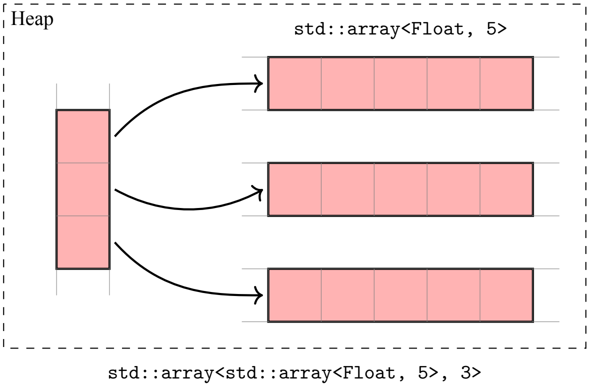


Figure: Using nested `std::array<T, N>` to store 2D array data.

- Separate into two detail and user interface objects adhering RAII rules.

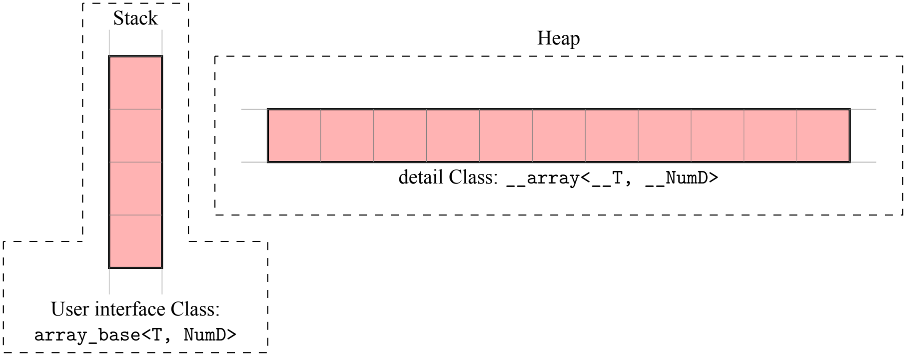


Figure: The solution of N-dimension Matrix, using detail Class, user interface Class

N-dimension Matrix
Solution

- Separate into two detail and user interface objects adhering RAII rules.
- An external small `__multi_array_shape` object defines the routines for accessing the elements.

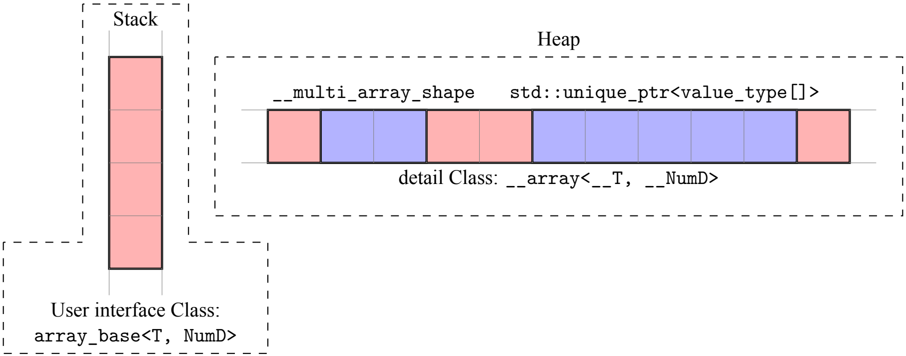


Figure: The solution of N-dimension Matrix, using detail Class, user interface Class and a shape management structure.

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

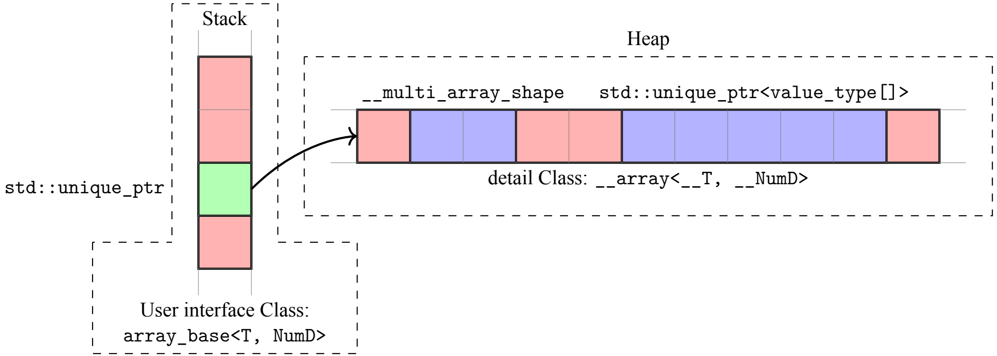


Figure: The solution of N-dimension Matrix, using detail Class, user interface Class and a shape management structure.

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.

Parallelization of N-dimension Arrays
MPI Environment

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.

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

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.

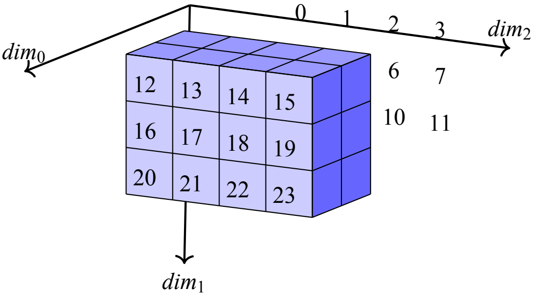


Figure: The Schematic representation of the MPI Cartesian topology scheme of 24 processors on 3 dimension space, which has a $2 \times 3 \times 4$ grid of processes

2. Using `MPI_Type_create_subarray` for creating Ghost MPI datatype for communication.

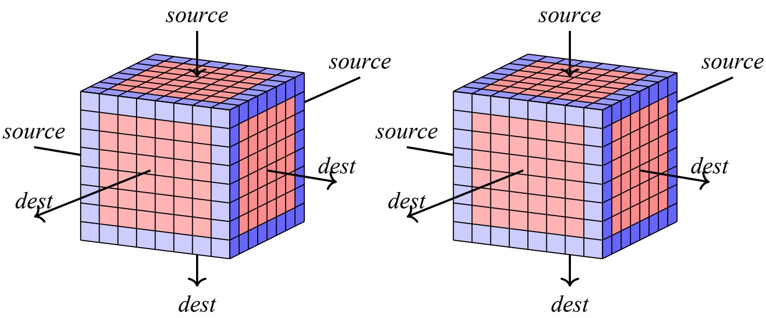


Figure: Representation of a 3D MPI Communication Scheme of 2 among 24 processes between 3 dimension sub-arrays.

3. Cartesian array has members Topology class and N-dimension array class, to ensure they are closely located on memory.
4. The external functions provide gather-based I/O and MPI I/O for handling different scenarios.
    - gather-based function is designed for small scale problem,
      it collects all data included with the boundaries to the root process, and use the I/O of N-dimension array.
    - MPI I/O function is designed for handling large scale problem,
      it only collects the internal bulk without boundary.

Parallelization of N-dimension Arrays
MPI Topology - Solution

5. User interface class unified features for both topology and array classes.
6. Separate into two detail and user interface objects adhering RAII rules.

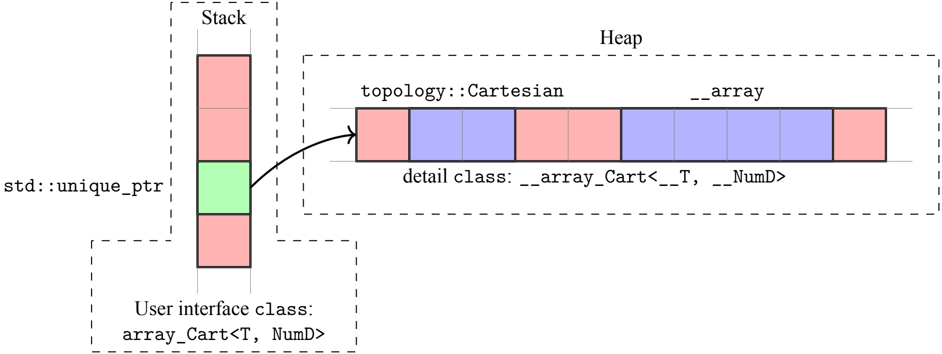


Figure: The solution of distributed N-dimension Array, using detail `class`, user interface `class` and a Cartesian topology `class`.

Outline

Outline

4. Implementations
   - PDE Solvers
     - Boundary/Initial Conditions
     - Pure MPI
     - Master-only, no overlapping
     - Master-only, with overlapping
   - PINN Model

N-dimension Boundary and Initial Conditions

Challanges

1. Mathematical function needs be discretized and distributed as well.
2. Initial and Dirichlet boundary conditions only applies once.
3. Von Neumann boundary condition participates the evolving process in FDM.
4. Need to access the data.

N-dimension Boundary and Initial Conditions

Solution

1. Use lambda function to construct classes.
2. Creating external classes for each conditions as the friend classes of PDE solver classes.
3. Set Bool vectors help to determine the status of conditions and type of boundary conditions.

1. PDE solver of Heat Equations in different dimension space have similar parameters and features, type-field solution lead to code redundancy.
2. Applying MPI communications between local arrays, avoiding overhead.
3. Three type of strategies:
   1. Pure MPI parallel
   2. Master-only, no overlapping hybrid parallel.
   3. Master-only, communication/computation (comm./comp.) overlapping hybrid parallel.

Outline

4 Implementations

- PDE Solvers
  - Boundary/Initial Conditions
  - Pure MPI
  - Master-only, no overlapping
  - Master-only, with overlapping
- PINN Model

`virtual` function is resolved at run-time, and only lose up to about 25% efficiency in terms of the function call mechanism,

1. Create an abstract Heat base class of Heat Equation, and overriding functions in derived class on every dimension.

2. MPI communications are implemented in blocking and non-blocking ways using `MPI_Sendrecv` and `MPI_Isend`/`MPI_Irecv`.

3. Three type of strategies
    1. Pure MPI parallel

3. Three type of strategies
    1. Master-only, no overlapping hybrid parallel.

3. Three type of strategies
   1. Master-only, comm./comp. overlapping hybrid parallel.

For neural network implementations, there are some issues in practice:
`Python` has many easy-use libraries such as `Pytorch`, `Tensorflow` and `Caffe`.

1. Interpret language Python is significant slower than compile language C/C++
2. Worse resource management than C/C++
3. Lower security in parallel program.

For getting higher performance and better safety, I chose to use

1. Pytorch C++ API: Libtorch.

to reproduce the Python version of PINN.

Outline

Single Node Scaling Tests

2D Heat Equation
Single Precision
Problem Size: $512^2, 1024^2, 2048^2, 4096^2, 8192^2, 16384^2, 32768^2$
Non-unified Memory Access (NUMA) node Topology:

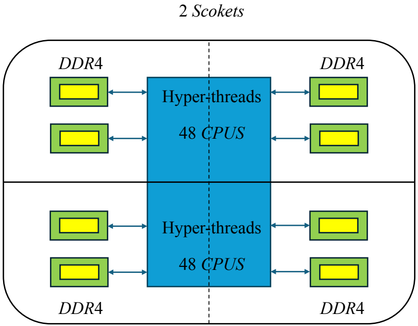
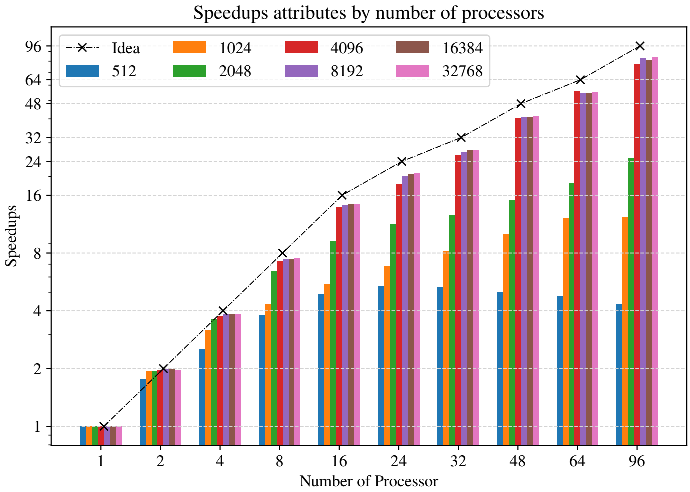


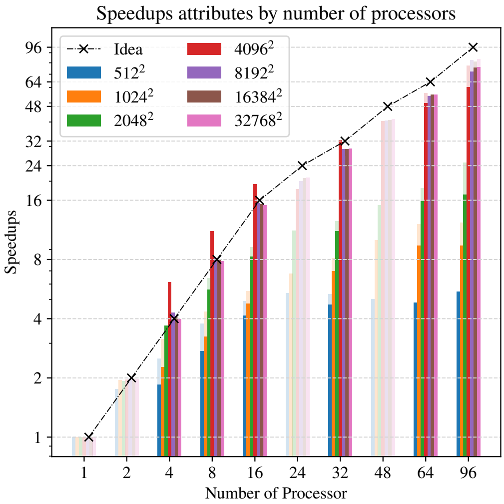
Figure: NUMA topology of single node on Cluster

Single Node Scaling Tests

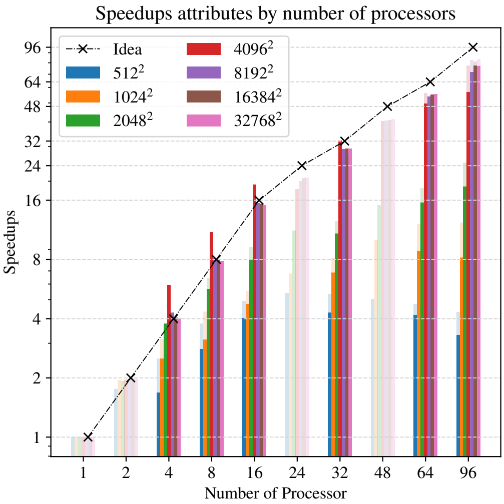
Strong Scaling - pure MPI



Single Node Scaling Tests
Strong Scaling - MPI/OpenMP Hybrid



(a) No overlapping comm./comp.



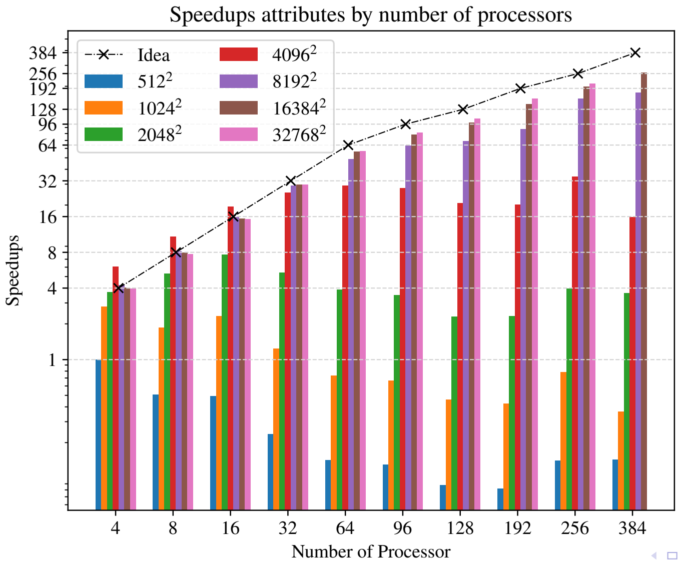
(b) With overlapping comm./comp.

Single Node Scaling Tests

Weak Scaling

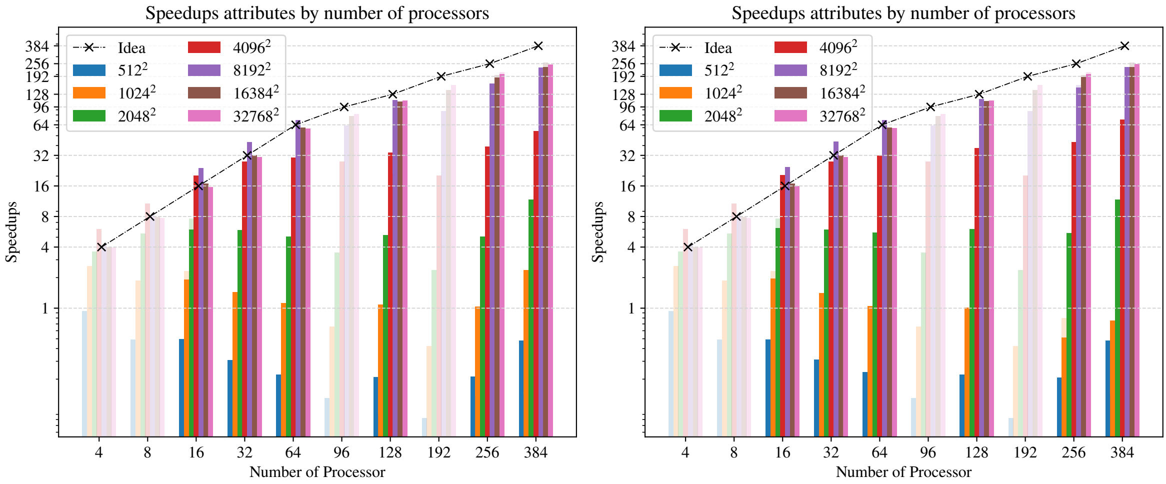
Strategy	Size	Number of CPUs 4	16	64	$f_p(\%)$
Pure MPI	512^2	4.006	12.497	47.849	75.1
No Overlap		2.876	11.206	42.754	67.0
With Overlap		3.173	10.818	42.282	66.2
Pure MPI	1024^2	3.838	9.304	33.707	53.2
No Overlap		3.947	12.995	33.447	54.1
With Overlap		4.024	12.932	33.361	54.0
Pure MPI	2048^2	2.376	8.245	31.203	49.0
No Overlap		3.874	8.972	31.510	49.8
With Overlap		3.740	8.989	31.430	49.7
Pure MPI	4096^2	3.543	8.245	31.203	77.5
No Overlap		3.953	13.799	49.515	78.0
With Overlap		3.948	13.800	49.989	78.7

Multi-node Scaling Tests
Strong Scaling - pure MPI



Single Node Scaling Tests

Strong Scaling - MPI/OpenMP Hybrid



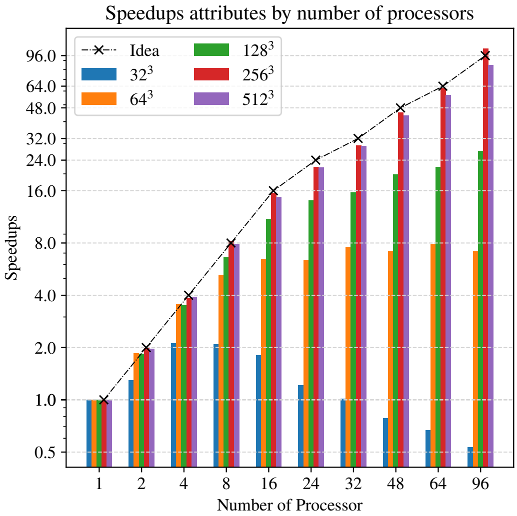
(a) No overlapping comm./comp.

(b) With overlapping comm./comp.

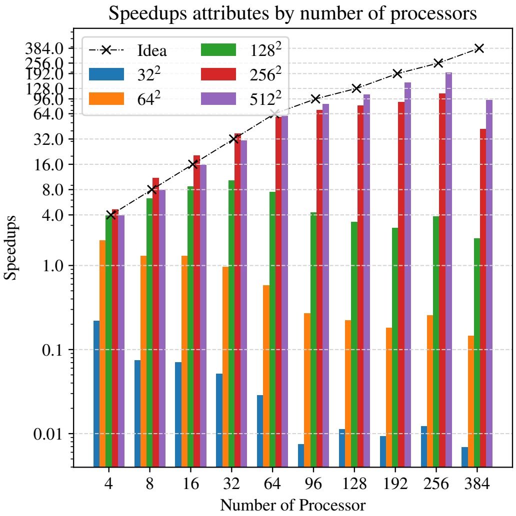
Single Node Scaling Tests
Weak Scaling

Strategy	Size	Number of CPUs 4	16	64	256	$f_p(\%)$
Pure MPI		3.300	10.379	26.000	124.375	48.2
No Overlap	512^2	-	8.094	25.931	127.648	49.3
With Overlap		-	8.386	27.126	116.951	45.5
Pure MPI		3.848	13.037	30.143	-	49.3
No Overlap	1024^2	-	13.702	44.040	-	69.8
With Overlap		-	13.834	44.090	-	69.9
Pure MPI		3.787	9.172	32.013	-	50.6
No Overlap	2048^2	-	13.936	34.368	-	55.7
With Overlap		-	14.274	34.414	-	55.9
Pure MPI		3.884	13.859	51.041	-	80.2
No Overlap	4096^2	-	15.315	53.157	-	83.8
With Overlap		-	15.294	53.401	-	84.2

3D Space Heat Equation
Strong Scaling - pure MPI



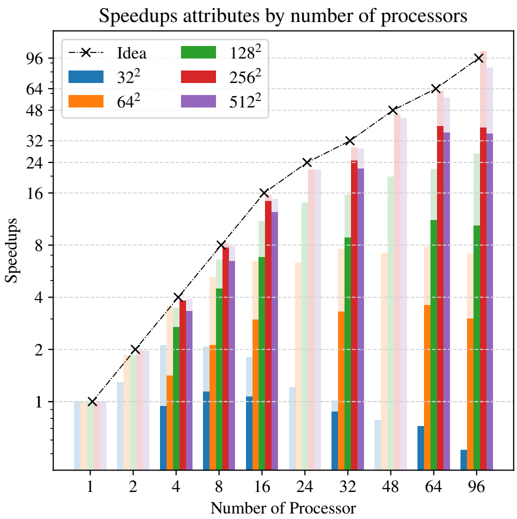
(a) Single node



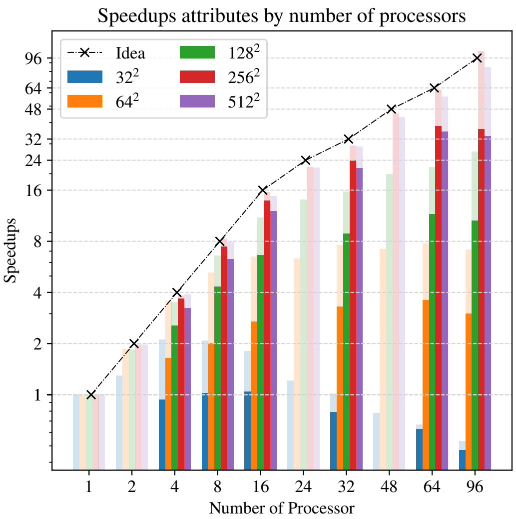
(b) Four nodes

3D Space Heat Equation

Strong Scaling - MPI/OpenMP Hybrid, Single Node



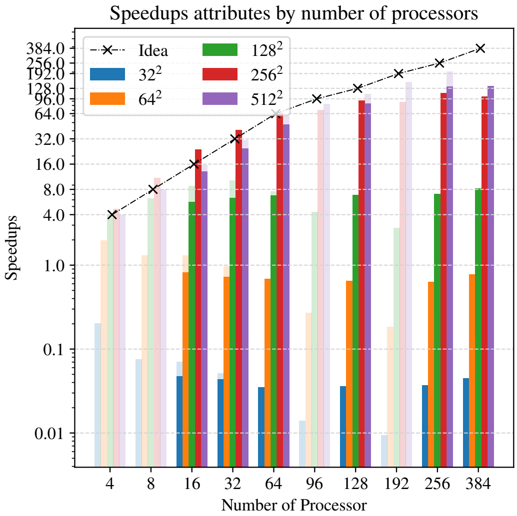
(a) Non-overlapping



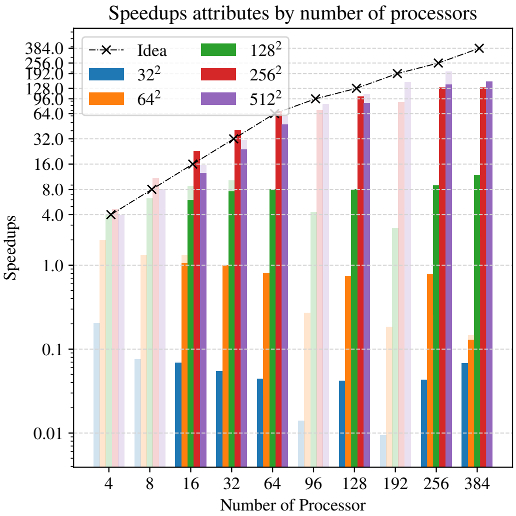
(b) With overlapping

3D Space Heat Equation

Strong Scaling - MPI/OpenMP Hybrid, Four Nodes



(a) Non-overlapping



(b) With overlapping

Strategy	**Size**	**Number of CPUs** 8	64	$\mathbf{64}_{\textbf{Multi-node}}$	$f_p(\%)$	$f_p(\%)_{\textbf{Multi-node}}$
Pure MPI	32^3	5.243	26.405	9.078	41.6	14.2
No Overlap		2.124	13.386	8.094	21.0	12.7
With Overlap		2.014	13.884	9.586	21.8	39.7
Pure MPI	64^3	7.937	32.034	30.106	50.8	47.1
No Overlap		5.393	20.007	33.460	31.8	52.3
With Overlap		5.200	19.558	35.254	31.1	31.6
Pure MPI	128^3	3.513	24.239	25.428	38.0	39.7
No Overlap		3.303	15.235	35.254	24.1	55.1
With Overlap		3.187	15.107	20.096	23.9	31.4

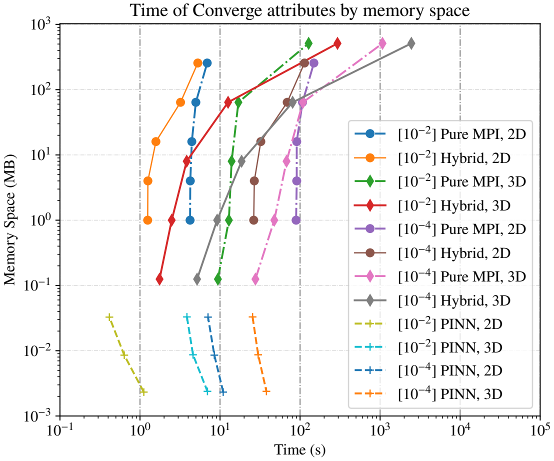
Table: Mean Square Error of Results

Method	**Heat 2D**	**Heat 3D**
FDTD	1.7287	953.84
PINN	13.37	24.896

Tolerance: 10^{-4}

PINN v.s. FDTD

Memory Usage v.s. Time of Convergence



Figure

Outline

For 2D and 3D Thermal Conduction PDE systems
I implemented the FDTD methods in fined region with three parallel models:

- Pure MPI model
    1. had best performance on single compute node overall.
    2. speedup ratio quickly drop on multi-node.
- MPI/OpenMP Hybrid models
    1. had lower performance in general.
    2. can make more use of the advantage of L3 cache.
    3. superliner speedup in certain scenario.

For 2D and 3D Thermal Conduction PDE systems
I implemented PINN models and trained on generated datasets, comparing with FDTD, the PINN

1. had identical or higher accuracy.
2. had less memory usage.
3. took shorter time to get the predictions(results).
4. had more flexibility on producing the results.

Outline

Further Research Directions

- Resource Management
- Workload Management
- MPI/CUDA Hybrid parallel of FDTD/PINN
- Other PDE Systems

Closing Remarks

Thank you for your attention!

Any questions?