

# Meta-Programming and Hybrid Parallel Strategies for Solving PDEs: An FDM and PINN Comparison<sup>1 2</sup>

## Seminar Presentation III

LI YIHAI

Student ID: 23345919

Supervision: Michael Peardon

September 21, 2024



Trinity College Dublin  
Coláiste na Tríonóide, Baile Átha Cliath

---

<sup>1</sup>full docs: <https://liyihai.com/html/index.html>

<sup>2</sup>repository: <https://github.com/liyihai-official/Final-Project>

## 1 Introduction

- Related Work
- Challenges & Objectives

## 2 Problem Setups

- General Form
- Thermal Conduction Systems

## 3 Methodology

- N-Dimension Matrix
- Parallelization of N-dimension Arrays

## 4 Implementations

- PDE Solvers
- PINN Model

## 5 Experiments

- On Single Node
- On Multi-node
- On Different Dimensions
- With PINN on accuracy

## 6 Discussion

## 7 Further Research Directions

## 1 Introduction

- Related Work
- Challenges & Objectives

## 2 Problem Setups

- General Form
- Thermal Conduction Systems

## 3 Methodology

- N-Dimension Matrix
- Parallelization of N-dimension Arrays

## 4 Implementations

- PDE Solvers
- PINN Model

## 5 Experiments

- On Single Node
- On Multi-node
- On Different Dimensions
- With PINN on accuracy

## 6 Discussion

## 7 Further Research Directions

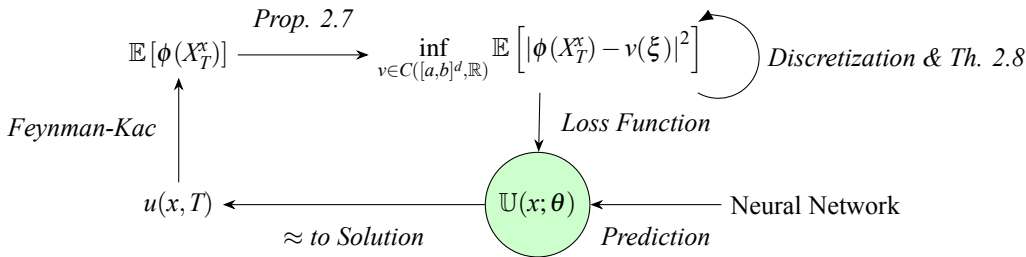
# Introduction

## Recap - Deep Neural Network

### Kolmogorov PDEs

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

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



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

# Introduction

## Recap - Physics Informed Neural Network

### General Form of PDEs

- $u(t, x)$  denotes with the target function,  $x \in \mathbb{R}^d$ .
- $\Gamma[\cdot; \lambda]$  is a non-linear operator parameterized by  $\lambda$ .

$$u_t(t, x) + \Gamma[u; \lambda] = 0 \quad (2)$$

Define  $f(t, x)$  to be given by  $f(t, x) = u_t(t, x) + \Gamma[u; \lambda]$  (3)

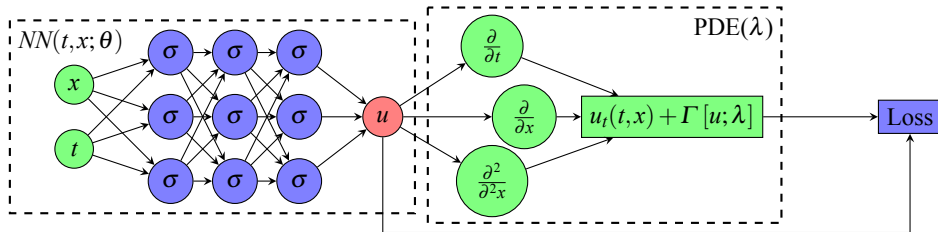


Figure: PINN, with 3 fully connected hidden layers

### 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.

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.



## 1 Introduction

- Related Work
- Challenges & Objectives

## 2 Problem Setups

- General Form
- Thermal Conduction Systems

## 3 Methodology

- N-Dimension Matrix
- Parallelization of N-dimension Arrays

## 4 Implementations

- PDE Solvers
- PINN Model

## 5 Experiments

- On Single Node
- On Multi-node
- On Different Dimensions
- With PINN on accuracy

## 6 Discussion

## 7 Further Research Directions

# General Form of problem

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

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

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

## Boundary Conditions

The Dirichlet and Von Neumann boundary conditions are formed as

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

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

# Thermal Conduction Systems

## Heat Equation 2D

The function

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

is the solution of 2D Heat Equation 7 below

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

# Thermal Conduction Systems

## Heat Equation 3D

The function

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

is the solution of 3D Heat Equation 9 below

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

## 1 Introduction

- Related Work
- Challenges & Objectives

## 2 Problem Setups

- General Form
- Thermal Conduction Systems

## 3 Methodology

- N-Dimension Matrix
- Parallelization of N-dimension Arrays

## 4 Implementations

- PDE Solvers
- PINN Model

## 5 Experiments

- On Single Node
- On Multi-node
- On Different Dimensions
- With PINN on accuracy

## 6 Discussion

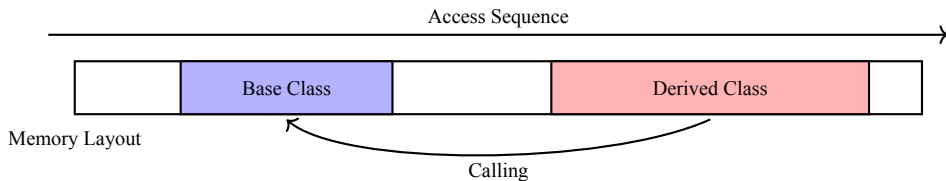
## 7 Further Research Directions

# N-dimension Matrix

## Challenges

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.

- ❶ Nesting multi-dimension array has non-contiguous memory layout.
- ❷ Derived class needs more time to access members in base class.
- ❸ Poor cache utilization leads to poor performance.
- ❹ MPI type create requires contiguous memory layout.

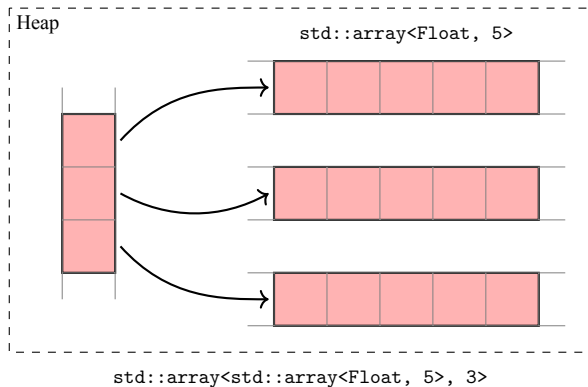
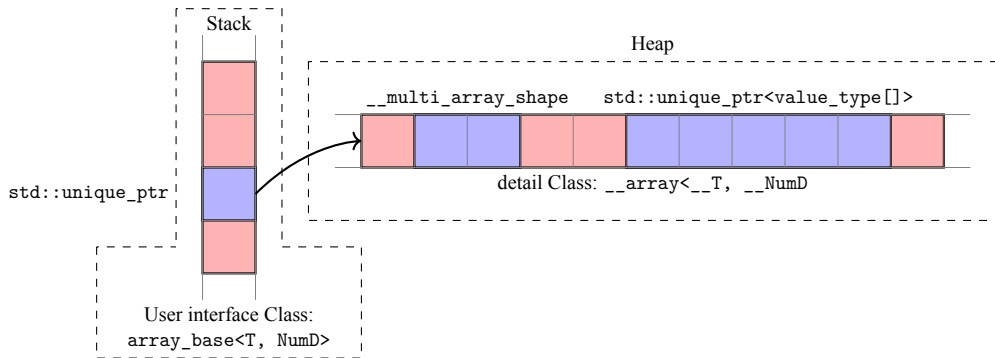


Figure: Using nested `std::array<T, N>` to store 2D array data.

# 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.



# 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.

# Parallelization of N-dimension Arrays

## MPI Topology - Challenges

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.
- ② Using `MPI_Type_create_subarray` for creating Ghost MPI datatype for communication.
- 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.
- ⑤ User interface class unified features for both topology and array classes.
- ⑥ Separate into two detail and user interface objects adhering RAII rules.

## 1 Introduction

- Related Work
- Challenges & Objectives

## 2 Problem Setups

- General Form
- Thermal Conduction Systems

## 3 Methodology

- N-Dimension Matrix
- Parallelization of N-dimension Arrays

## 4 Implementations

- PDE Solvers
- PINN Model

## 5 Experiments

- On Single Node
- On Multi-node
- On Different Dimensions
- With PINN on accuracy

## 6 Discussion

## 7 Further Research Directions

# N-dimension Boundary and Initial Conditions Challenges

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

# 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.

# N-dimension PDE solvers

## Challenges

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

# N-dimension PDE solvers

## Solutions

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.



# N-dimension PDE solvers

## Solutions

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

# N-dimension PDE solvers

## Solutions

- ③ Three type of strategies
  - ① Pure MPI parallel

# N-dimension PDE solvers

## Solutions

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

# N-dimension PDE solvers

## Solutions

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

# PINN Model

## Challenges

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.

# PINN Model

## Solution

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

- ① Pytorch C++ API: Libtorch.

to reproduce the Python version of PINN.

## 1 Introduction

- Related Work
- Challenges & Objectives

## 2 Problem Setups

- General Form
- Thermal Conduction Systems

## 3 Methodology

- N-Dimension Matrix
- Parallelization of N-dimension Arrays

## 4 Implementations

- PDE Solvers
- PINN Model

## 5 Experiments

- On Single Node
- On Multi-node
- On Different Dimensions
- With PINN on accuracy

## 6 Discussion

## 7 Further Research Directions

# 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:

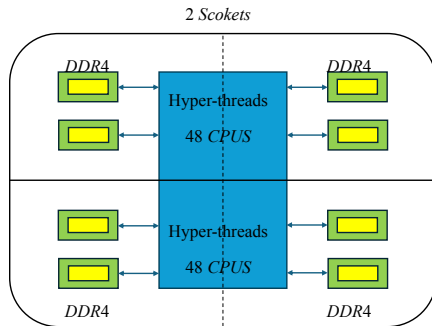
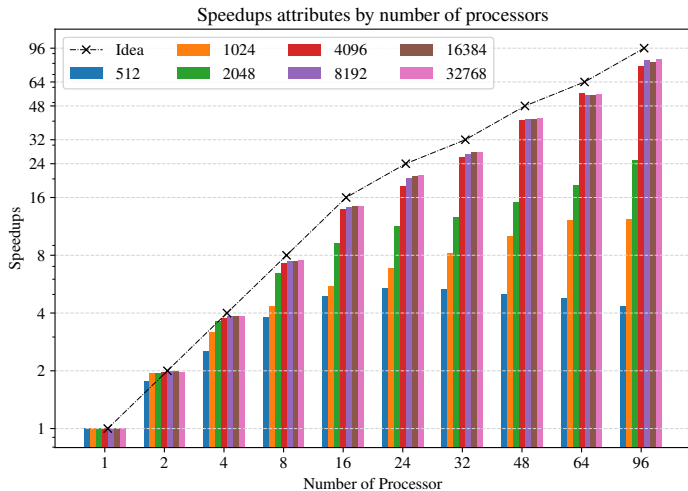


Figure: NUMA topology of single node on Cluster



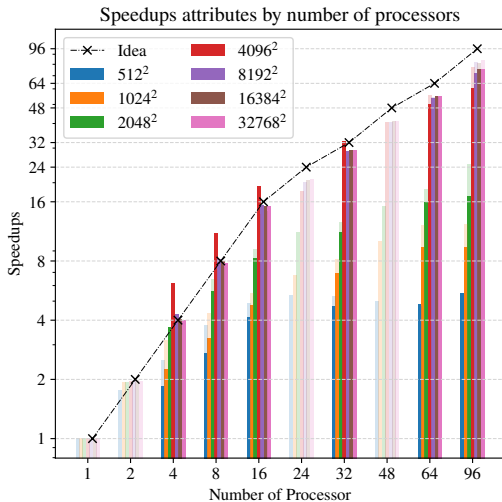
# Single Node Scaling Tests

## Strong Scaling - pure MPI

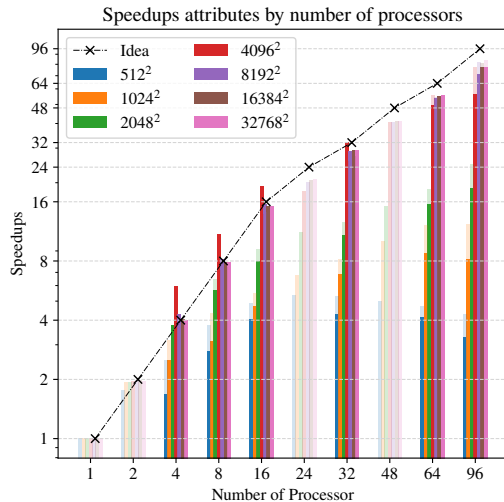


# Single Node Scaling Tests

## Strong Scaling - MPI/OpenMP Hybrid



(a) No overlapping comm./comp.



(b) With overlapping comm./comp.

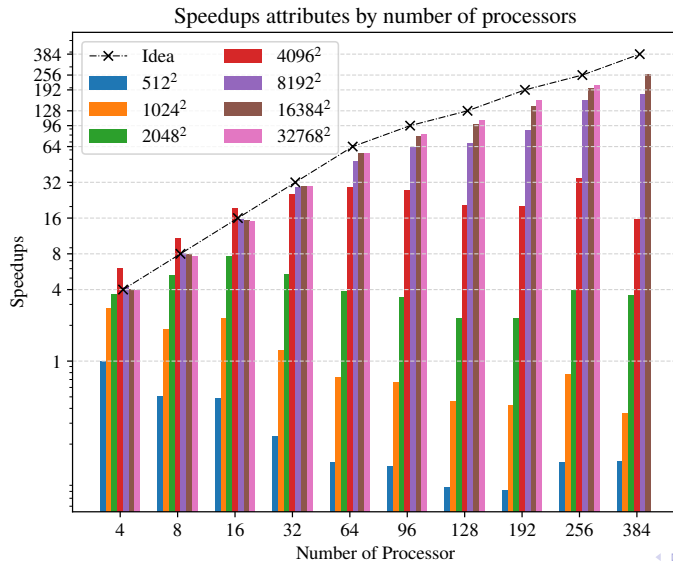
# Single Node Scaling Tests

## Weak Scaling

Strategy	Size	Number of CPUs			$f_p(\%)$
		4	16	64	
Pure MPI	512 <sup>2</sup>	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 <sup>2</sup>	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 <sup>2</sup>	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 <sup>2</sup>	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

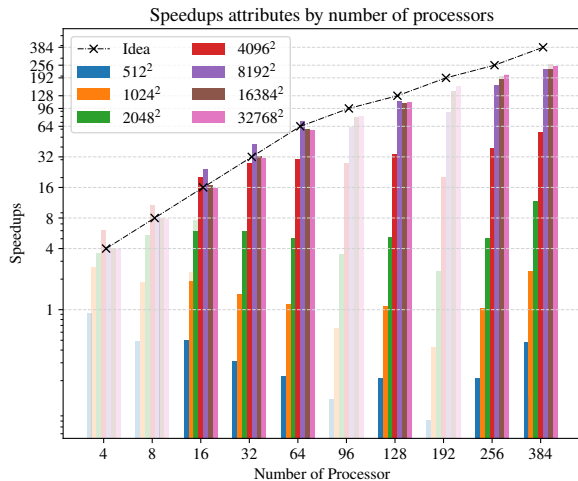
# Multi-node Scaling Tests

## Strong Scaling - pure MPI

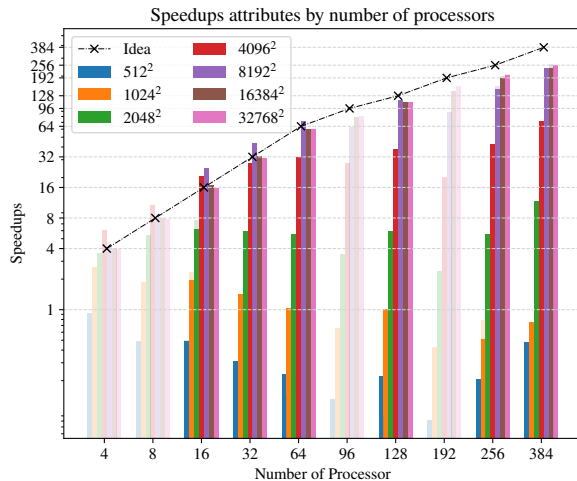


# Single Node Scaling Tests

## Strong Scaling - MPI/OpenMP Hybrid



(a) No overlapping comm./comp.



(b) With overlapping comm./comp.

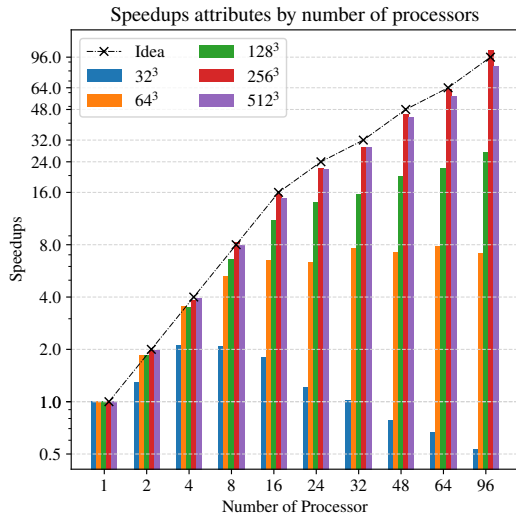
# Single Node Scaling Tests

## Weak Scaling

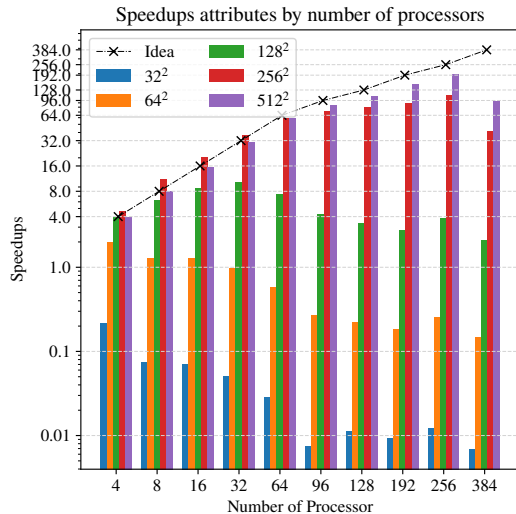
Strategy	Size	Number of CPUs				$f_p(\%)$
		4	16	64	256	
Pure MPI	512 <sup>2</sup>	3.300	10.379	26.000	124.375	48.2
No Overlap		-	8.094	25.931	127.648	49.3
With Overlap		-	8.386	27.126	116.951	45.5
Pure MPI	1024 <sup>2</sup>	3.848	13.037	30.143	-	49.3
No Overlap		-	13.702	44.040	-	69.8
With Overlap	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

# 3D Space Heat Equation

## Strong Scaling - pure MPI



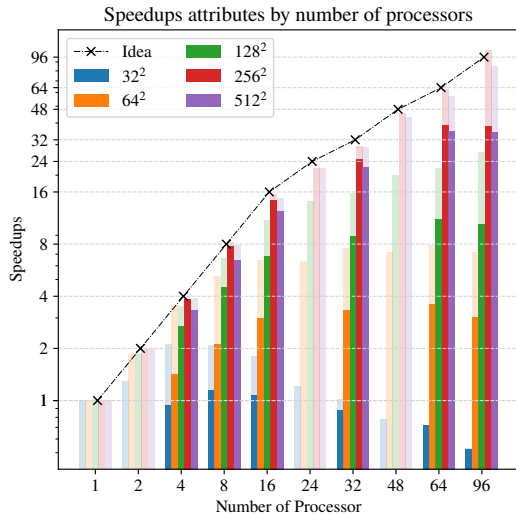
(a) Single node



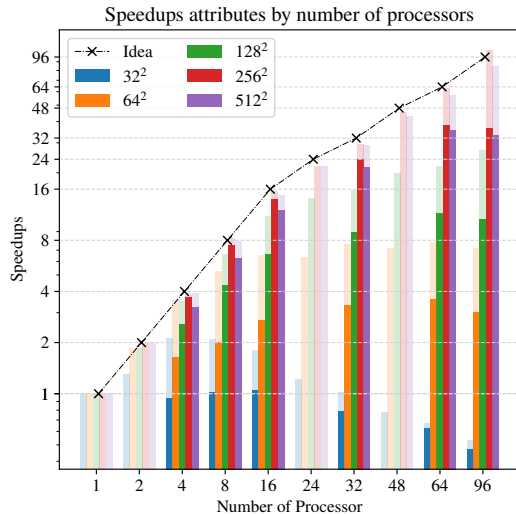
(b) Four nodes

# 3D Space Heat Equation

## Strong Scaling - MPI/OpenMP Hybrid, Single Node



(a) Non-overlapping

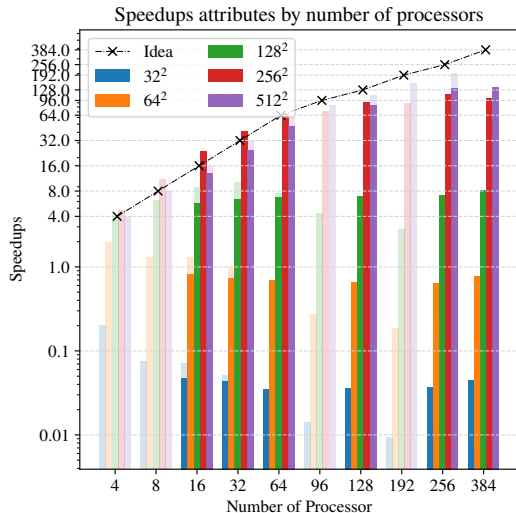


(b) With overlapping

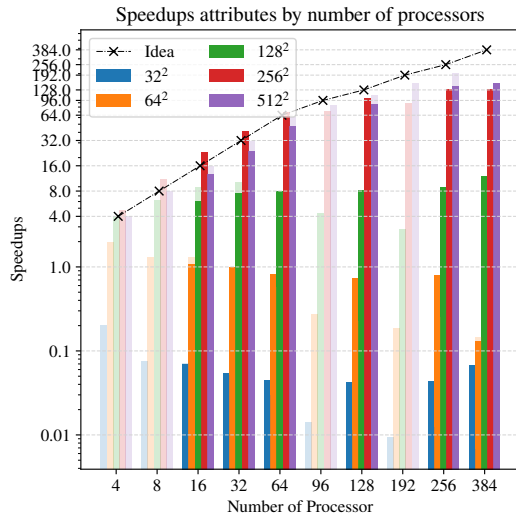


# 3D Space Heat Equation

## Strong Scaling - MPI/OpenMP Hybrid, Four Nodes



(a) Non-overlapping



(b) With overlapping

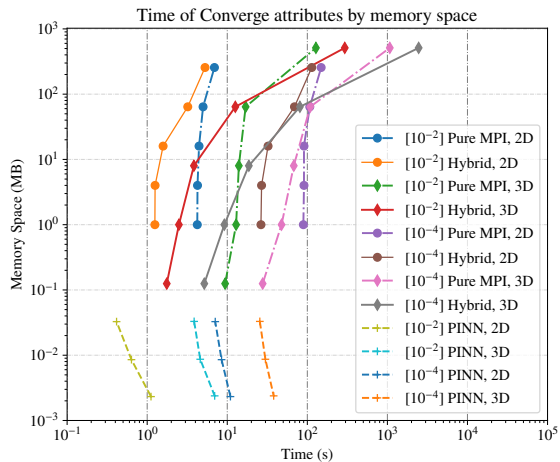
## 3D Space Heat Equation

### Weak Scaling

Strategy	Size	Number of CPUs			$f_p(\%)$	$f_p(\%)$ Multi-node
		8	64	64Multi-node		
Pure MPI	32 <sup>3</sup>	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 <sup>3</sup>	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 <sup>3</sup>	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

# PINN v.s. FDTD

## Memory Usage v.s. Time



Figure

# PINN v.s. FDTD

## Accuracy

## 1 Introduction

- Related Work
- Challenges & Objectives

## 2 Problem Setups

- General Form
- Thermal Conduction Systems

## 3 Methodology

- N-Dimension Matrix
- Parallelization of N-dimension Arrays

## 4 Implementations

- PDE Solvers
- PINN Model

## 5 Experiments

- On Single Node
- On Multi-node
- On Different Dimensions
- With PINN on accuracy

## 6 Discussion

## 7 Further Research Directions

# Conclusion

## 1 Introduction

- Related Work
- Challenges & Objectives

## 2 Problem Setups

- General Form
- Thermal Conduction Systems

## 3 Methodology

- N-Dimension Matrix
- Parallelization of N-dimension Arrays

## 4 Implementations

- PDE Solvers
- PINN Model

## 5 Experiments

- On Single Node
- On Multi-node
- On Different Dimensions
- With PINN on accuracy

## 6 Discussion

## 7 Further Research Directions





Thank you for your attention!

Any questions?