Parallel Physics-Informed Neural Networks via Domain Decomposition

Raj Shukla

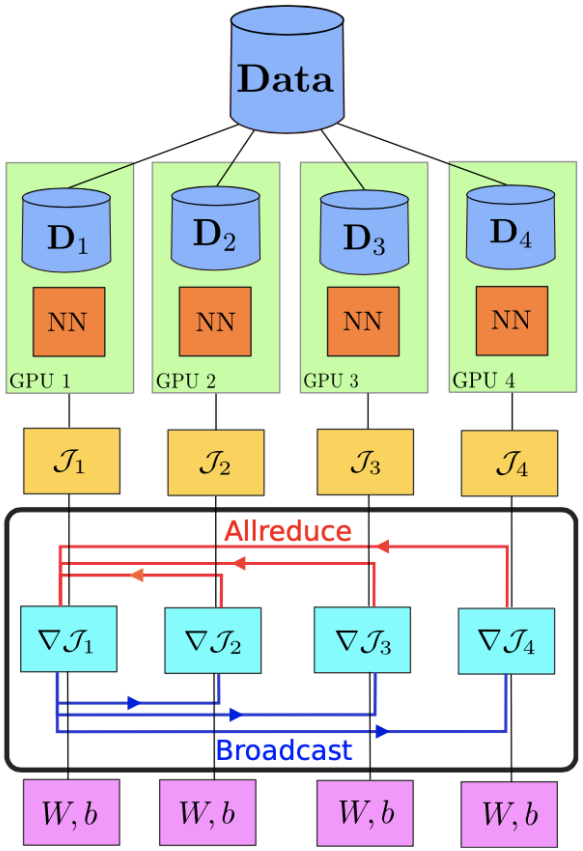
with A. Jagtap and G. Karniadakis

CRUNCH Group
Division of Applied Mathematics
Brown University

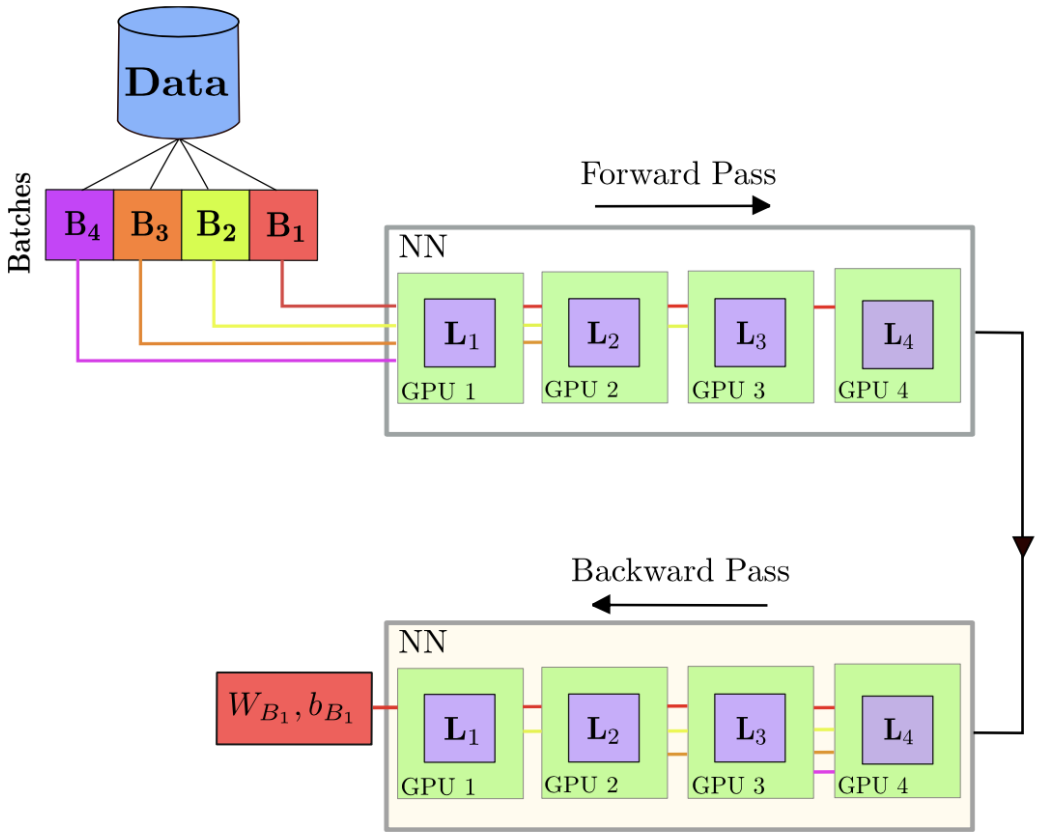
Pittsburgh Supercomputing Center

January 25, 2023

Motivation



Data parallel



Model parallel

Motivations

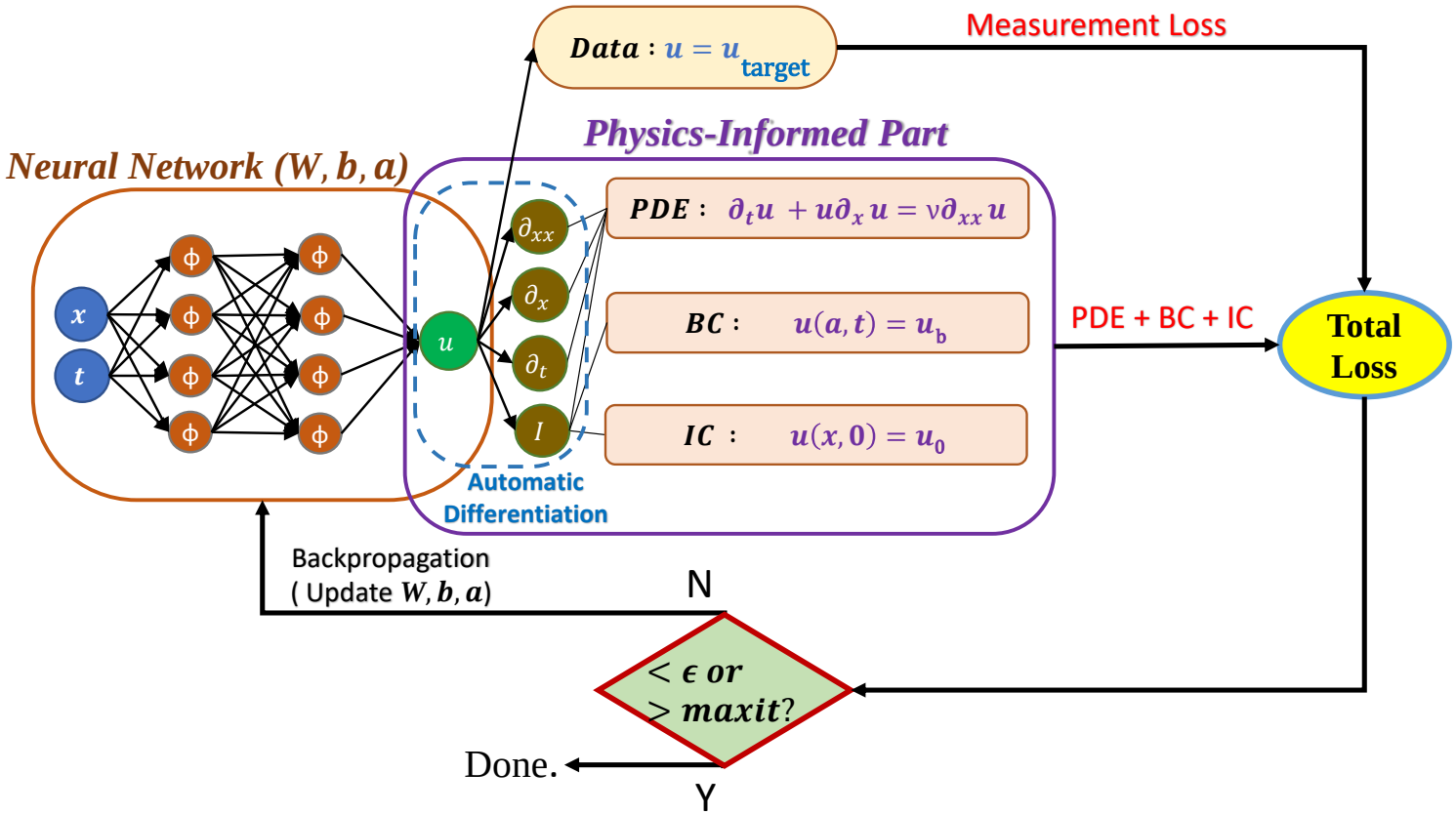
cPINN, XPINN: PINN + Domain Decomposition

cPINNs: A Jagtap, E Kharazmi, GE Karniadakis, CMAME 365 (2020) 113028
XPINNs: A Jagtap, GE Karniadakis,CiCP 28 (5), 2002-2041, 2020

Outline

1. cPINNs
2. XPINNs
3. Parallel Implementations
4. LES modeling: Ongoing project

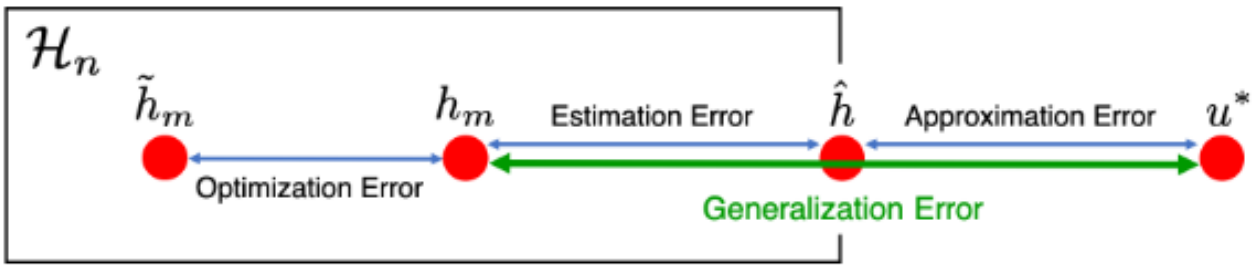
Physics-Informed Neural Networks: Recap



$$\mathcal{L}(\tilde{\Theta}) = \frac{1}{N_u}\sum_{i=1}^{N_u} |u^i_{\text{target}} - u_{\tilde{\Theta}}(x_i^u)|^2 + \frac{1}{N_f}\sum_{i=1}^{N_f} |\mathcal{F}_{\tilde{\Theta}}(x_i^f)|^2,$$

PINN Limitations:

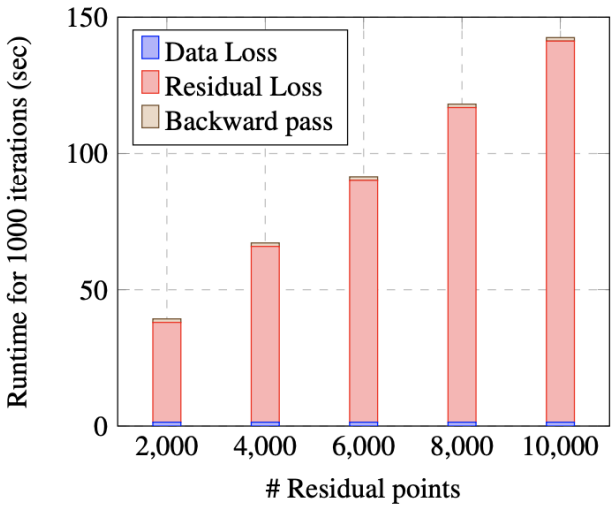
1. Large training time (Domain decomposition)
2. Due to high-dimensional non-convex optimization problem, the accuracy of the method suffers.

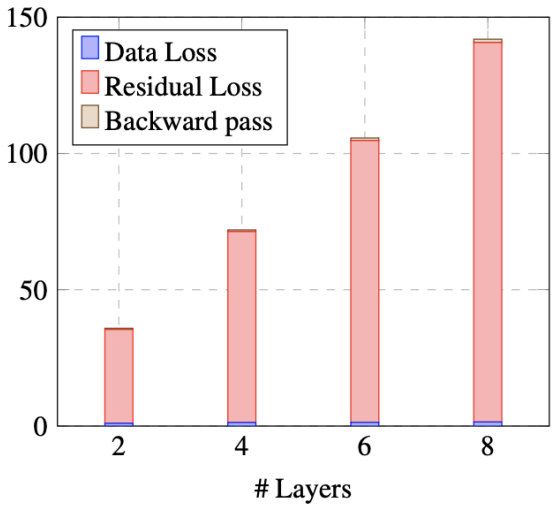


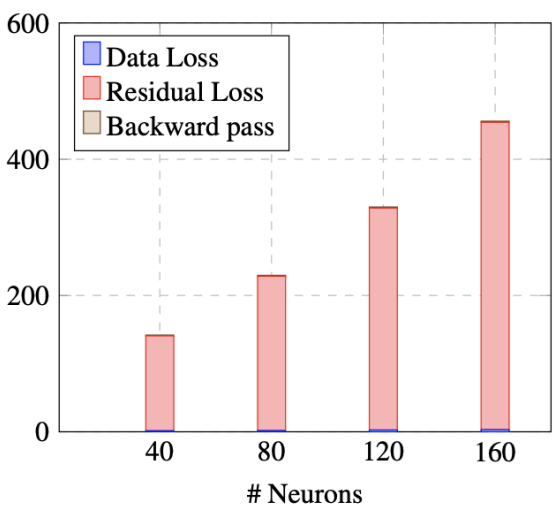
Physics-Informed Neural Networks: Profiling

Viscous Burger's equation:

$u_t + uu_x = \nu u_{xx},\ x \in \mathbb{R}, t > 0$ with IC $u(x, 0) = -\sin(\pi x)$ and BCs $u(t, 1) = u(t, -1) = 0$.

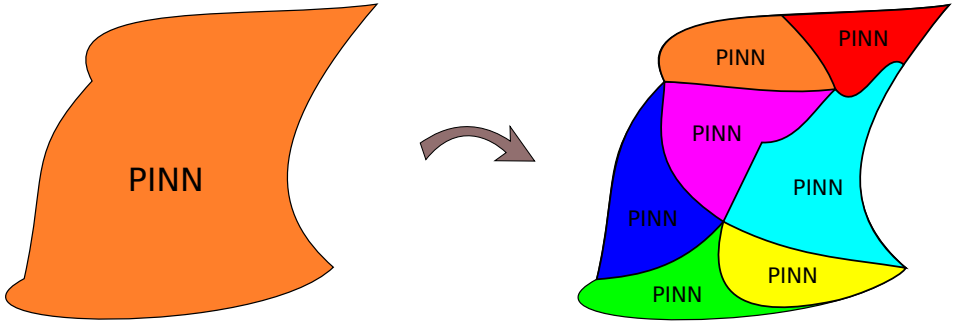


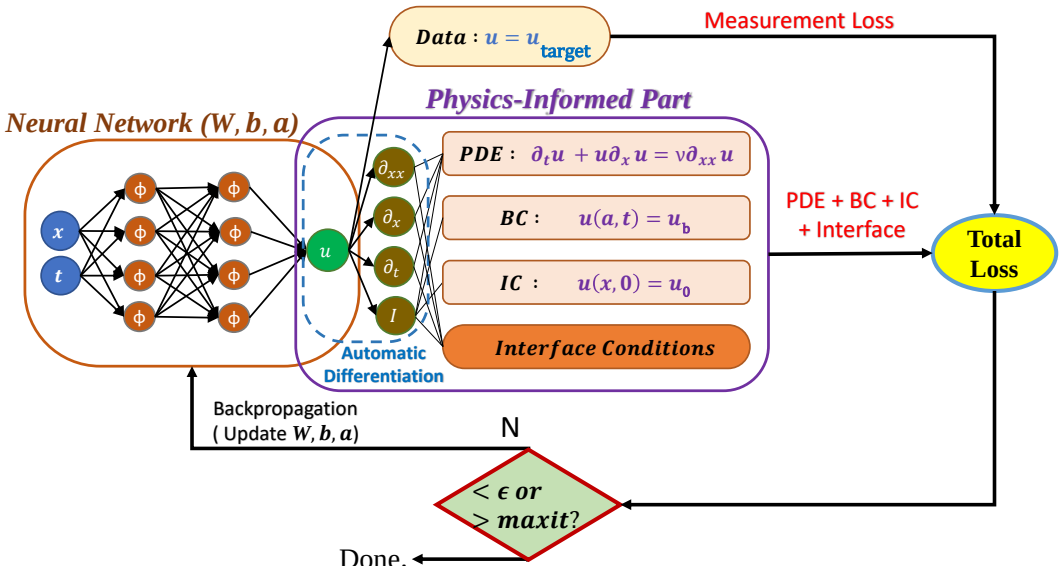




Reverse-mode AD: Graph Traversal: $\mathcal{O}(|E| + |V|)$

Domain Decomposition based PINNs





$$\mathcal{L}(\tilde{\Theta}) = \frac{1}{N_u}\sum_{i=1}^{N_u} |u^i_{\text{target}} - u_{\tilde{\Theta}}(x^u_i)|^2 + \frac{1}{N_f}\sum_{i=1}^{N_f} |\mathcal{F}_{\tilde{\Theta}}(x^f_i)|^2 \ + \text{Interface Loss}$$

Advantages

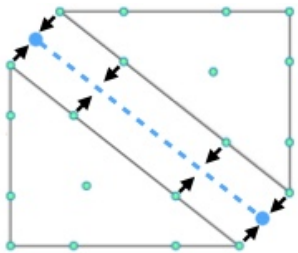
1. **Parallelization capacity** : The partial independence of individual PINNs in decomposed domains can be further employed to implement cPINN in a parallelized algorithm.
2. **Representation capacity** : Due to deployment of individual network in each sub-domain by the proposed cPINN method, the representation capacity of the network increases.
3. **Efficient hyper-parameter adjustment** : Based on prior (and sparse) knowledge of the solution regularity in each sub-domain, the hyper-parameter set of corresponding PINN is properly adjusted.
4. **Reduction of error propagation in the domain** : Individual networks in each sub-domain provide additional information about the solution using interface conditions, which results in reduction of error propagation in the neighbouring sub-domains as well as faster convergence.

Conservative PINNs (cPINNs) : Applications to conservation laws

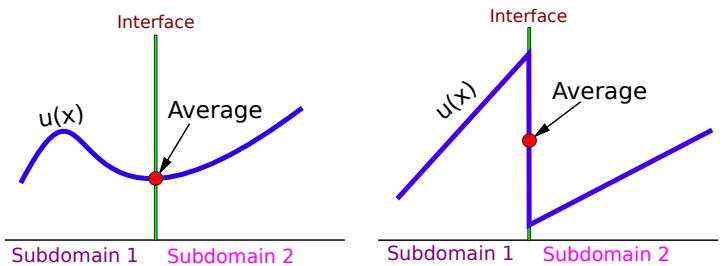
Conservation Laws:

$$u_t + \nabla \cdot (fu) = 0$$

$$\text{Flux continuity} = \frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| f_q(u(\mathbf{x}^i_{I_q})) \cdot \mathbf{n} - f_{q^+}(u(\mathbf{x}^i_{I_q})) \cdot \mathbf{n} \right|^2$$



$$\text{Avg.Solution continuity} = \frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| u_q(\mathbf{x}^i_{I_q}) - \left\{ \left\{ u(\mathbf{x}^i_{I_q}) \right\} \right\} \right|^2$$



What Next?

- DD strategy for every PDE (not necessarily the conservation laws).
- Also, it will be more efficient if we can do DD in space-time domain.

eXtended PINNs (XPINNs)

XPINNs: **Interface conditions** Avg.Solution continuity $=$

$$\frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| u_q(\mathbf{x}_{I_q}^i) - \left\{ \left\{ u(\mathbf{x}_{I_q}^i) \right\} \right\} \right|^2$$

Residual continuity $= \frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| \mathcal{R}_q(u(\mathbf{x}_{I_q}^i)) - \mathcal{R}_{q^+}(u(\mathbf{x}_{I_q}^i)) \right|^2$

$+$

Additional continuity conditions

eXtended PINNs (XPINNs)

XPINNs: **Interface conditions** Avg.Solution continuity $=$

$$\frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| u_q(\mathbf{x}_{I_q}^i) - \left\{ \left\{ u(\mathbf{x}_{I_q}^i) \right\} \right\} \right|^2$$

Residual continuity $= \frac{1}{N_{I_q}} \sum_{i=1}^{N_{I_q}} \left| \mathcal{R}_q(u(\mathbf{x}_{I_q}^i)) - \mathcal{R}_{q^+}(u(\mathbf{x}_{I_q}^i)) \right|^2$

$+$

Additional continuity conditions

Advantages

1. **Extension to any differential equation(s)**
2. **Generalized space-time domain decomposition**
3. **Simple interface conditions**

Parallel Implementation

cPINN Loss

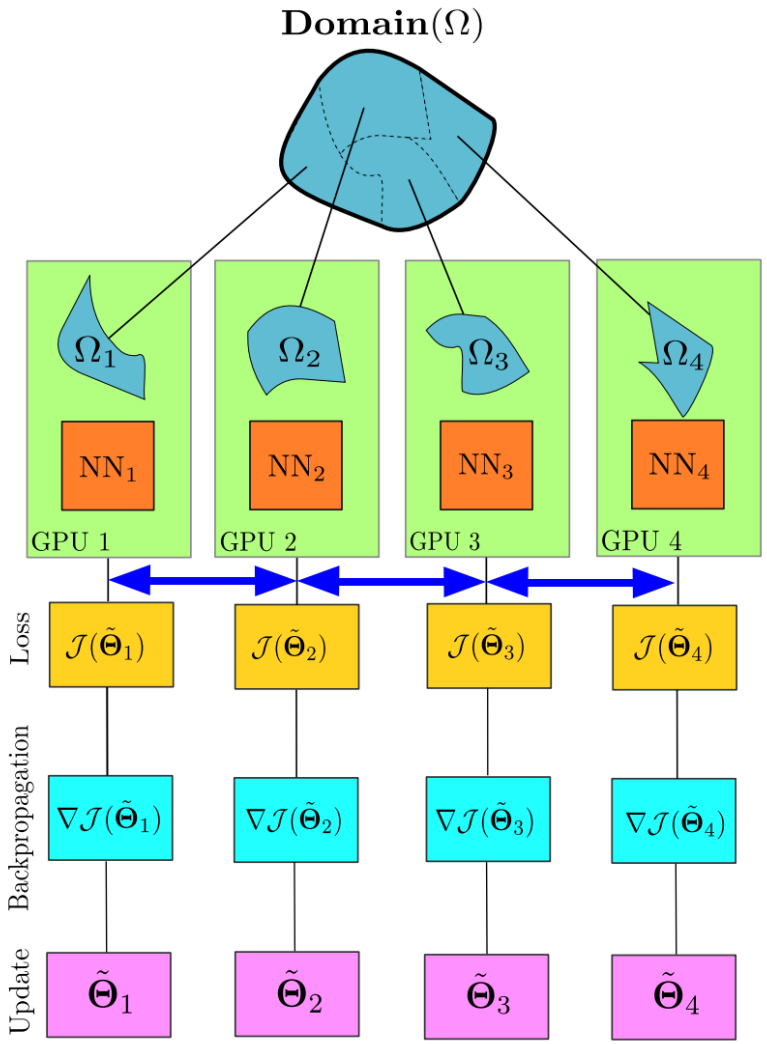
$$\mathcal{L} = \mathcal{L}_D + \mathcal{L}_F + \mathcal{L}_f + \mathcal{L}_c$$

XPINN Loss

$$\mathcal{L} = \mathcal{L}_D + \mathcal{L}_F + \mathcal{L}_{F_i} + \mathcal{L}_c$$

$\mathcal{L}_D$: Volume term - Concurrent evaluation
$\mathcal{L}_F$: Volume term - Concurrent evaluation
$\mathcal{L}_f$: Surface term - Communication bound
$\mathcal{L}_{F_i}$: Surface term - Communication bound
$\mathcal{L}_c$: Surface term - Communication bound

Parallel Implementation

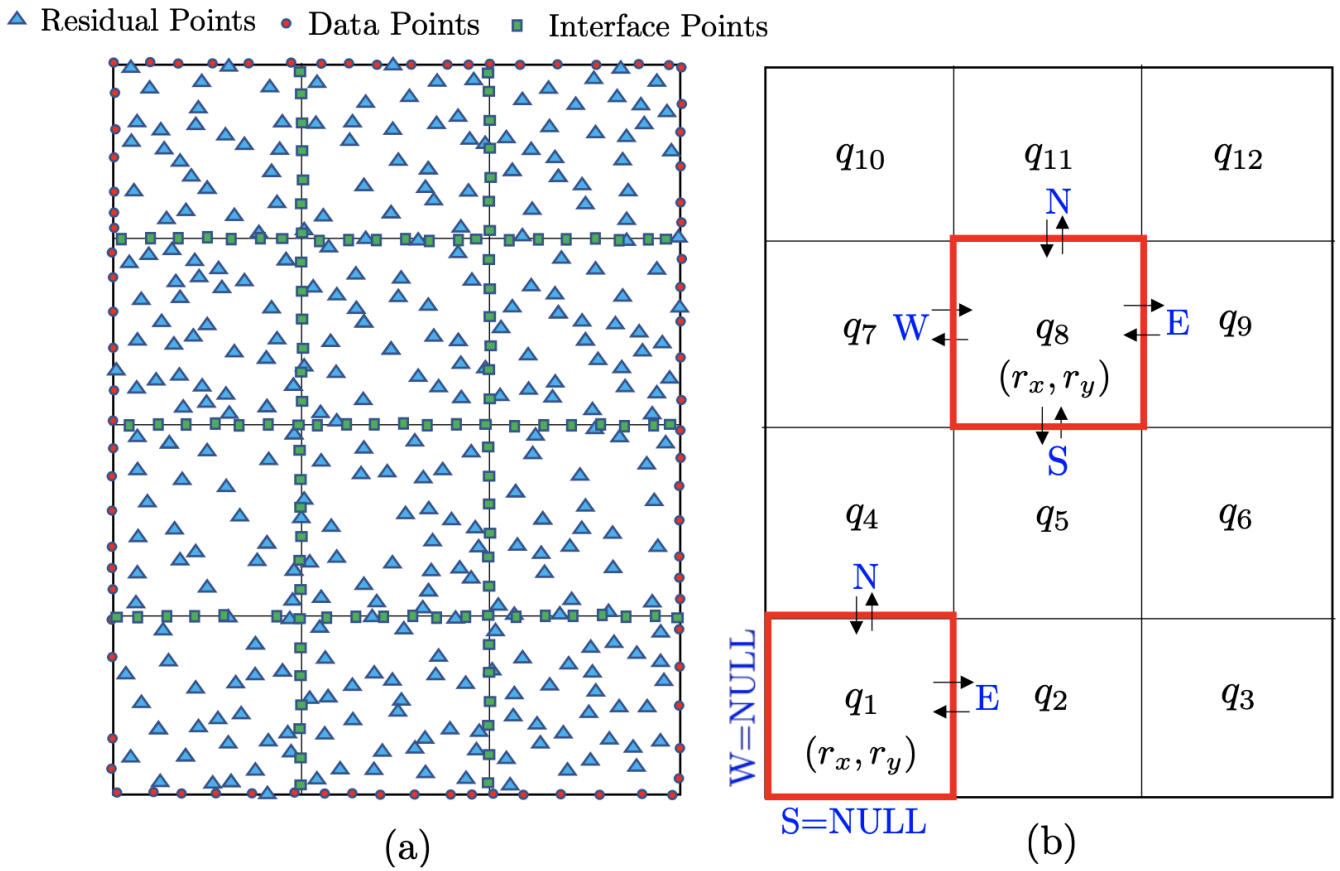


cPINN or XPINN parallel approach

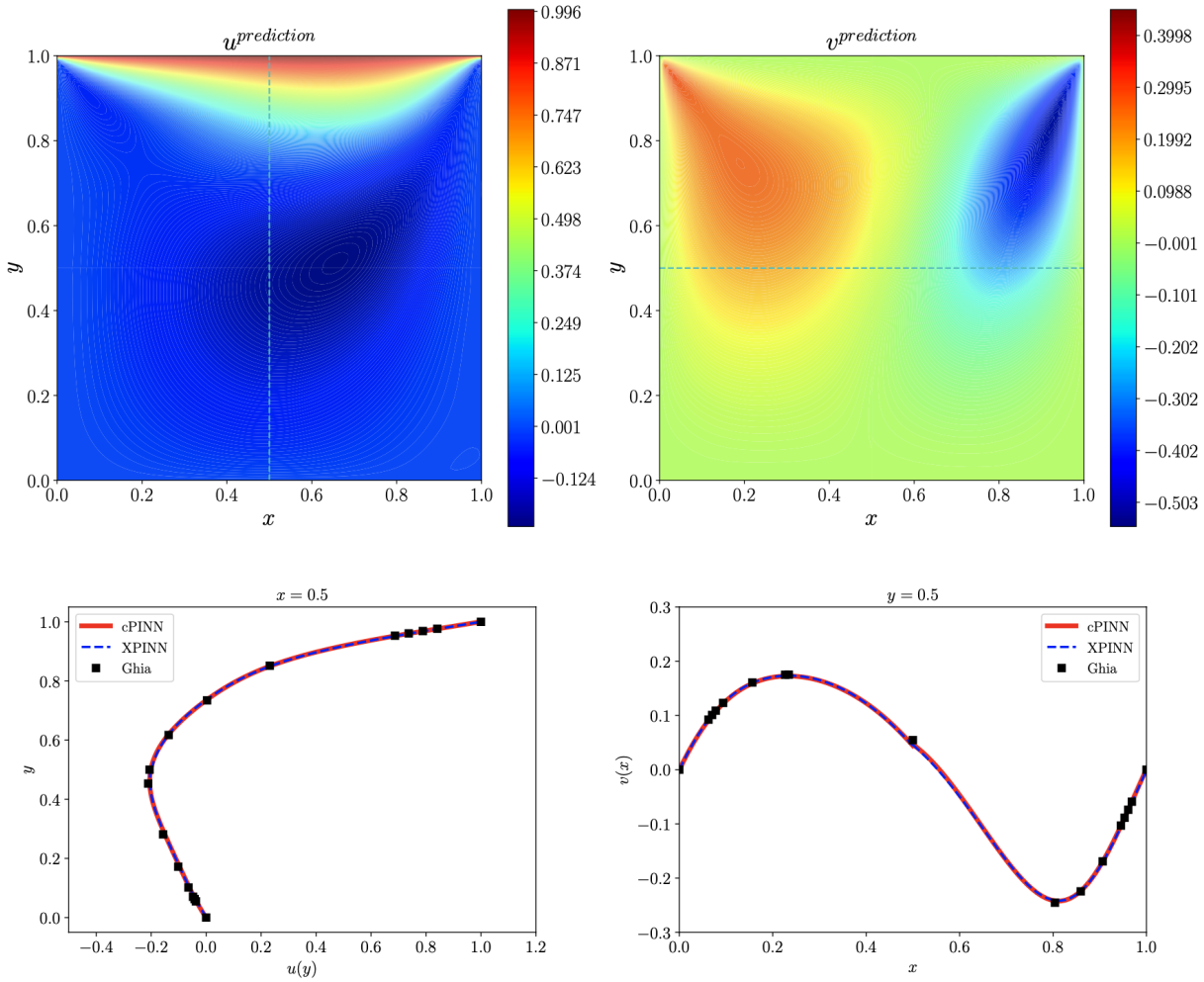
Domain Partitioning: 2D Incompressible Navier-Stokes equations

$$(\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2\mathbf{u}, \quad \text{in } \Omega$$

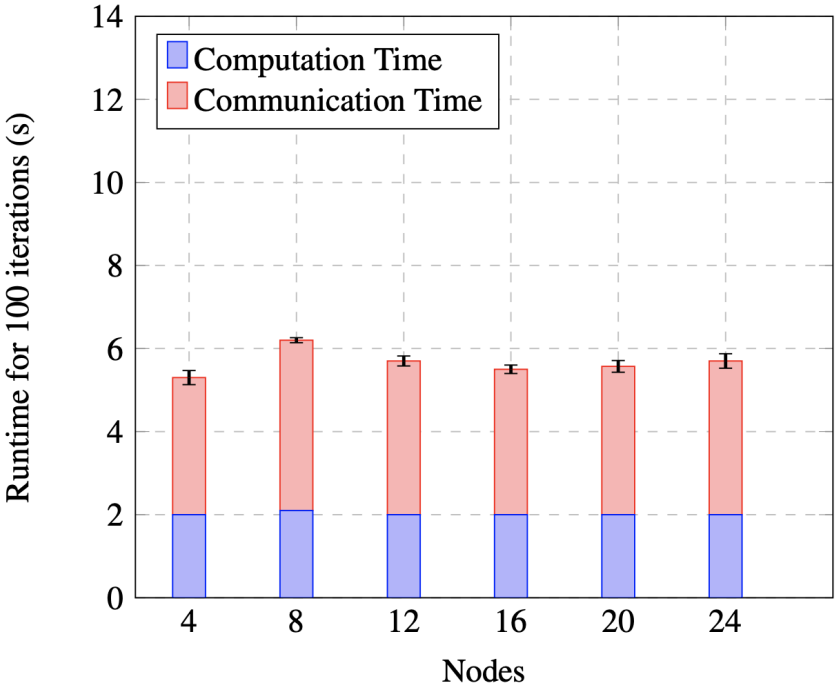
$$\nabla\cdot\mathbf{u} = 0, \quad \text{in } \Omega$$



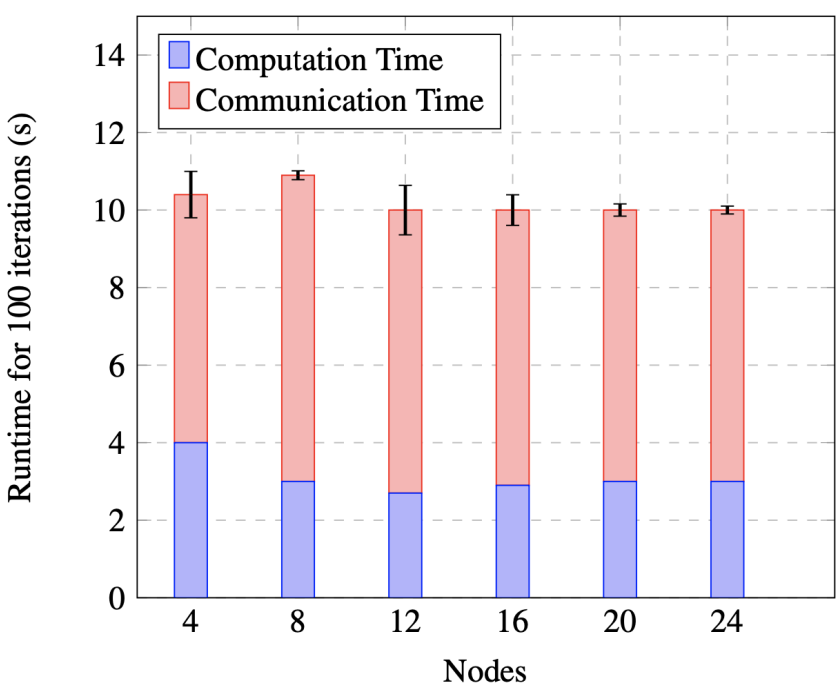
Accuracy of parallel cPINN and XPINN



Computation vs Communication Time: CPUs



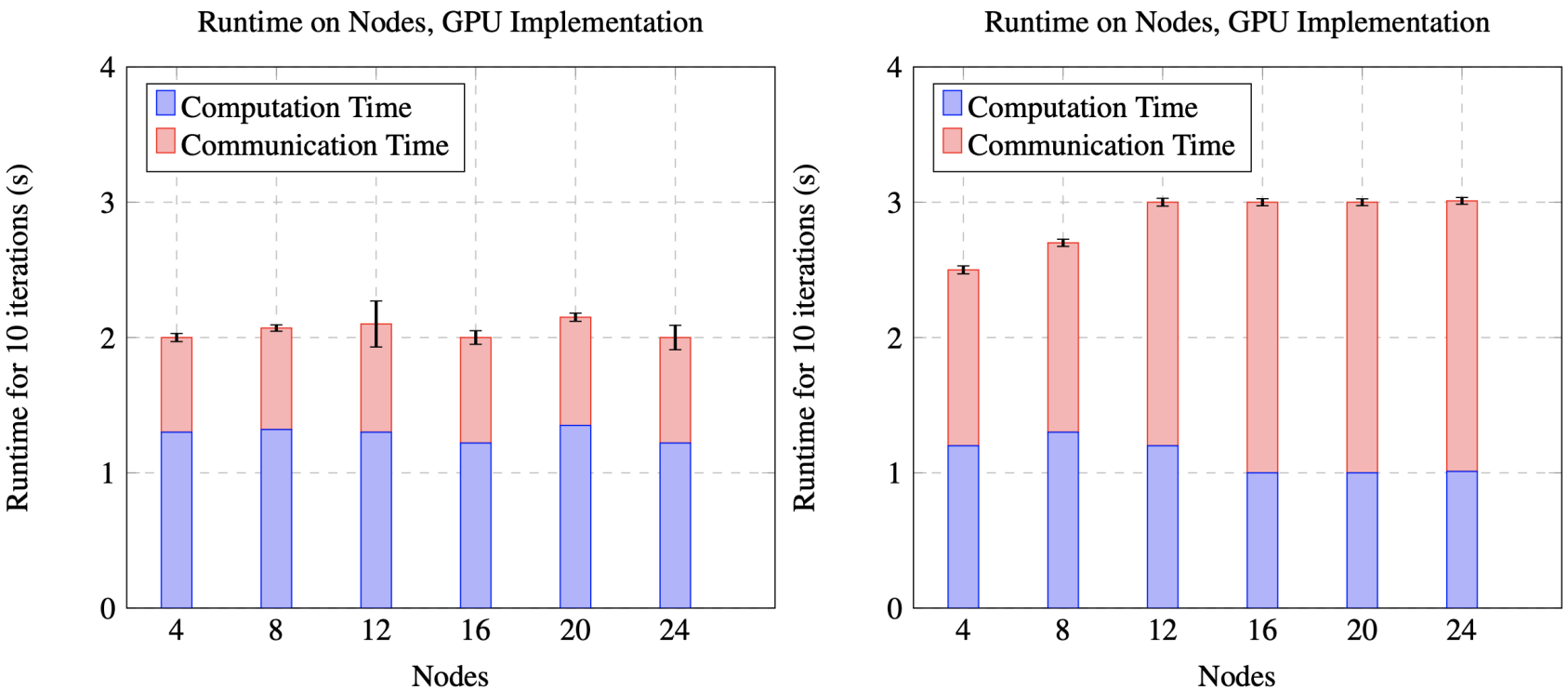
(a) Compute vs Communication for cPINN



(b) Compute vs Communication for xPINN

Figure 2: Computation and communication time for (a) cPINN and (b) xPINN with $N_f = 100$ and $N_{f_i} = 20$

Computation vs Communication Time: GPUs



(a) Compute vs Communication for cPINN

(b) Compute vs Communication for xPINN

Figure 3: Computation and communication time for (a) cPINN and (b) xPINN with $N_f = 4000$ and $N_{f_i} = 200$

Communication: XPINN > cPINN, Why?

NS Equation:

$$(\boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla p + \frac{1}{Re}\nabla^2 \boldsymbol{u}, \qquad \text{in } \Omega$$
$$\nabla \cdot \boldsymbol{u} = 0, \qquad \text{in } \Omega$$

cPINN Loss: : $F(\boldsymbol{u}) + \{\boldsymbol{u}^+ - \boldsymbol{u}^-\}$

Flux	X-dir	Y-dir
Div*,†	u	v
Mom. X	$u^2 + p - \frac{1}{Re}\frac{\partial u}{\partial x}$	$uv - \frac{1}{Re}\frac{\partial u}{\partial y}$
Mom Y	$uv - \frac{1}{Re}\frac{\partial v}{\partial x}$	$v^2 + p - \frac{1}{Re}\frac{\partial v}{\partial y}$

XPINN Loss: : $\mathcal{F}(\boldsymbol{u}) + \{\boldsymbol{u}^+ - \boldsymbol{u}^-\}$

Weak Scaling: cPINN and XPINN

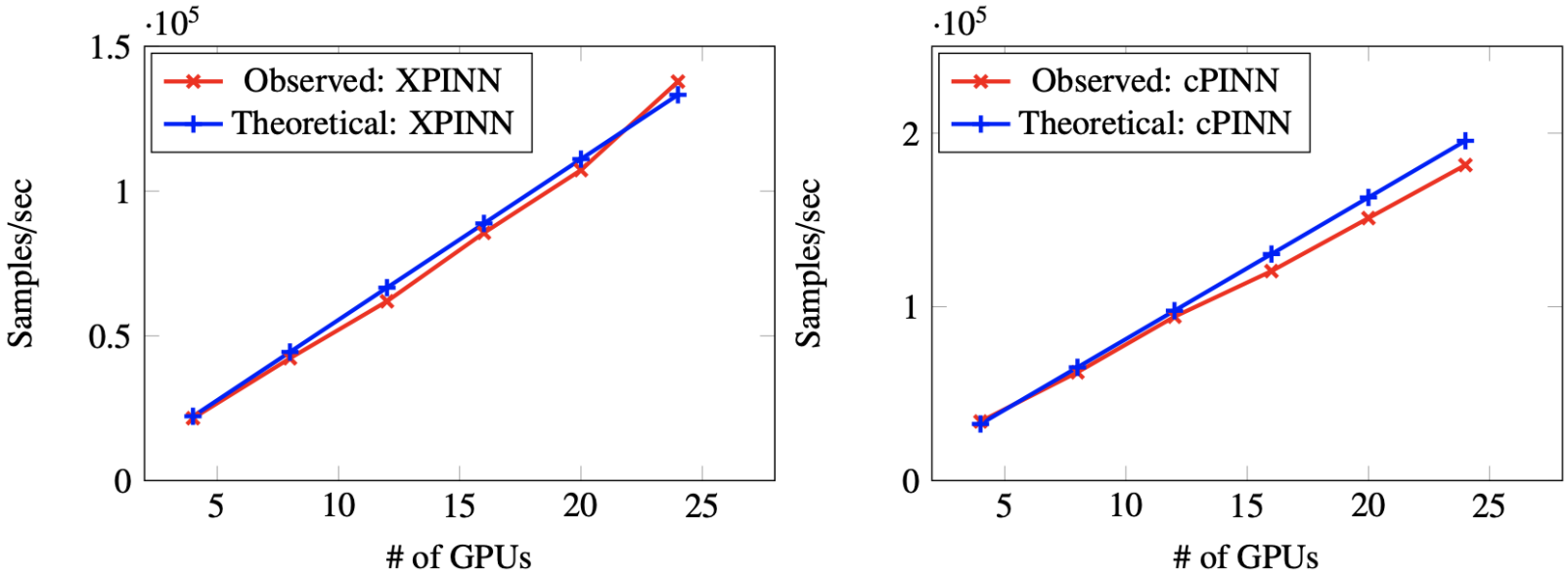
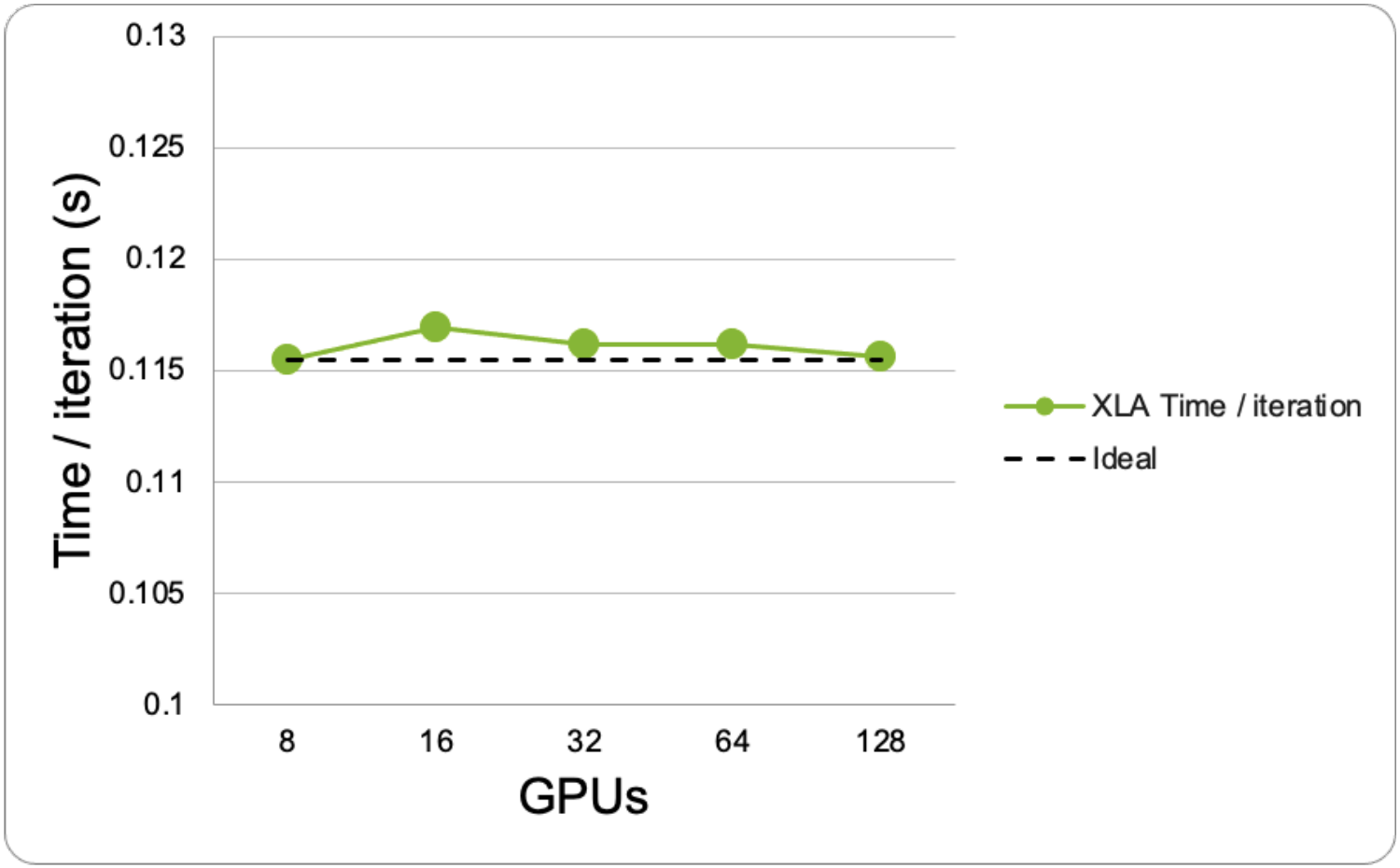


Figure 5: Weak scaling for the cPINN and the XPINN methods.

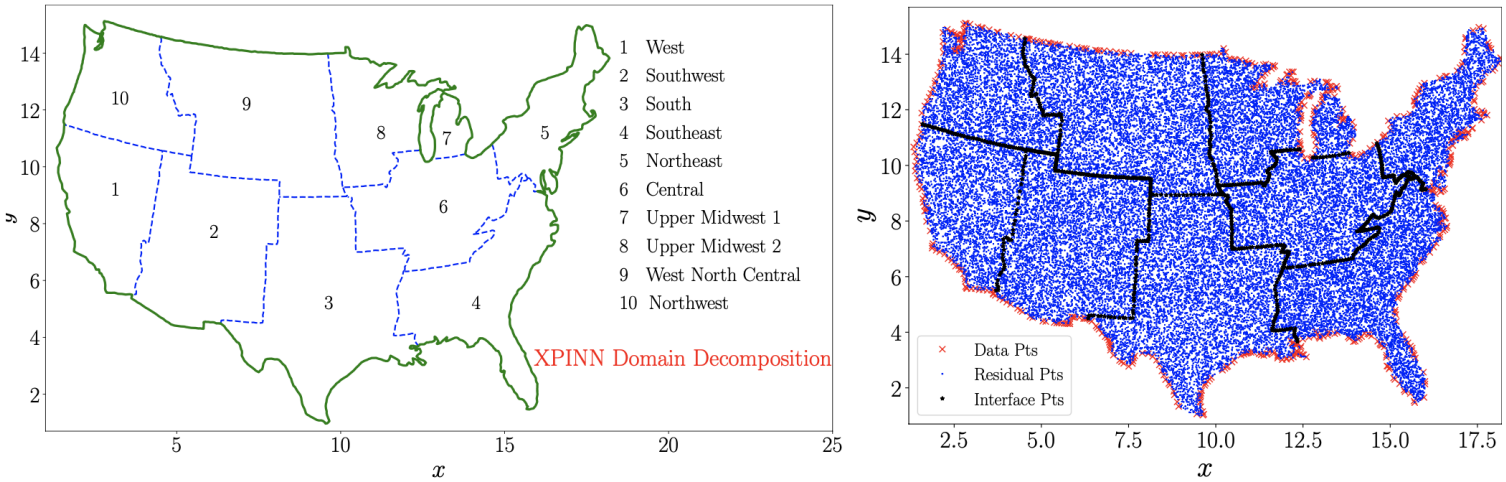
Weak Scaling: More GPUs NVIDIA-ANL Hackathon



Inverse problem: Steady state heat conduction with variable conductivity

$$\partial_x(K(x,y)T_x) + \partial_y(K(x,y)T_y) = f(x,y)$$

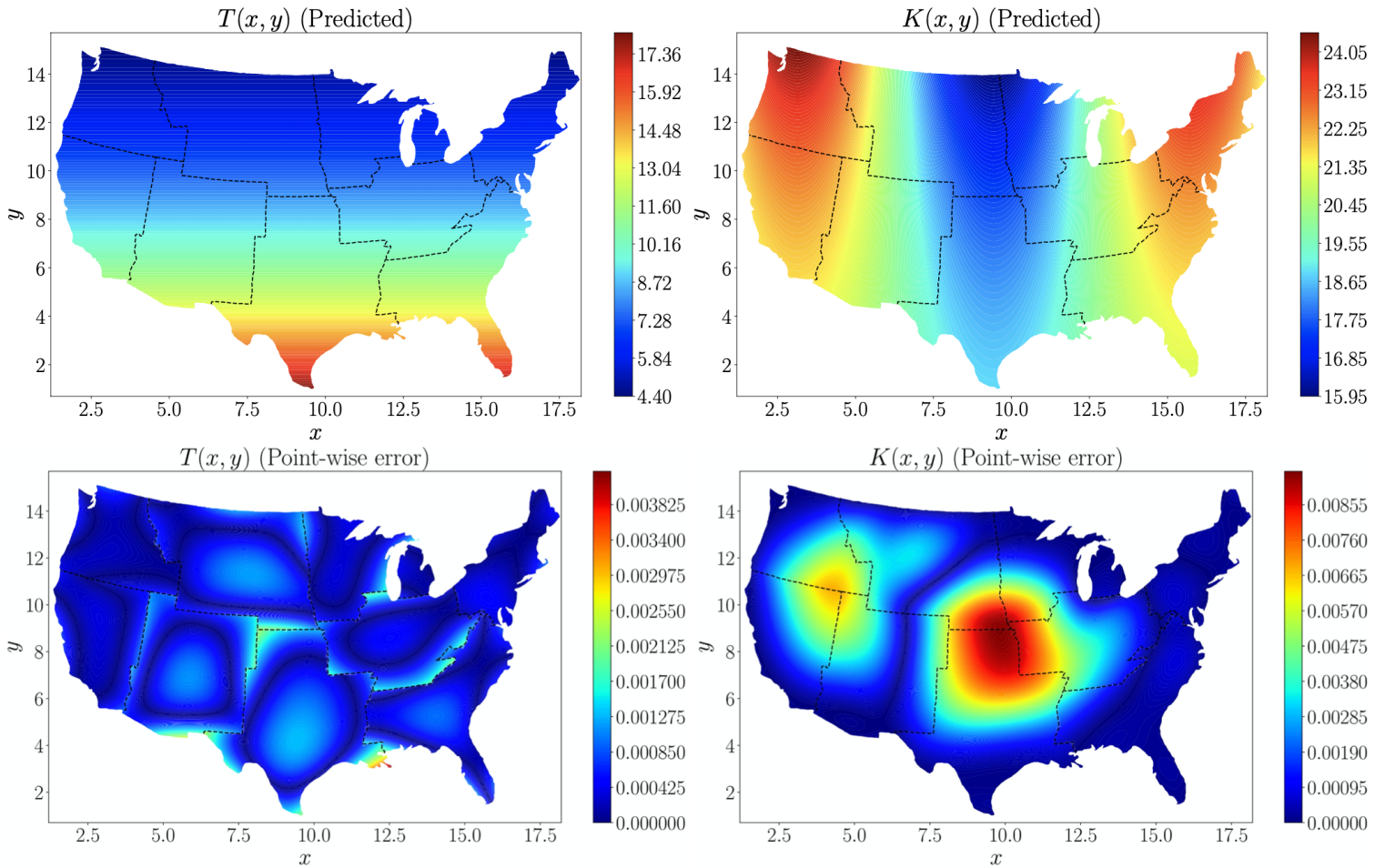
Domain partitioning



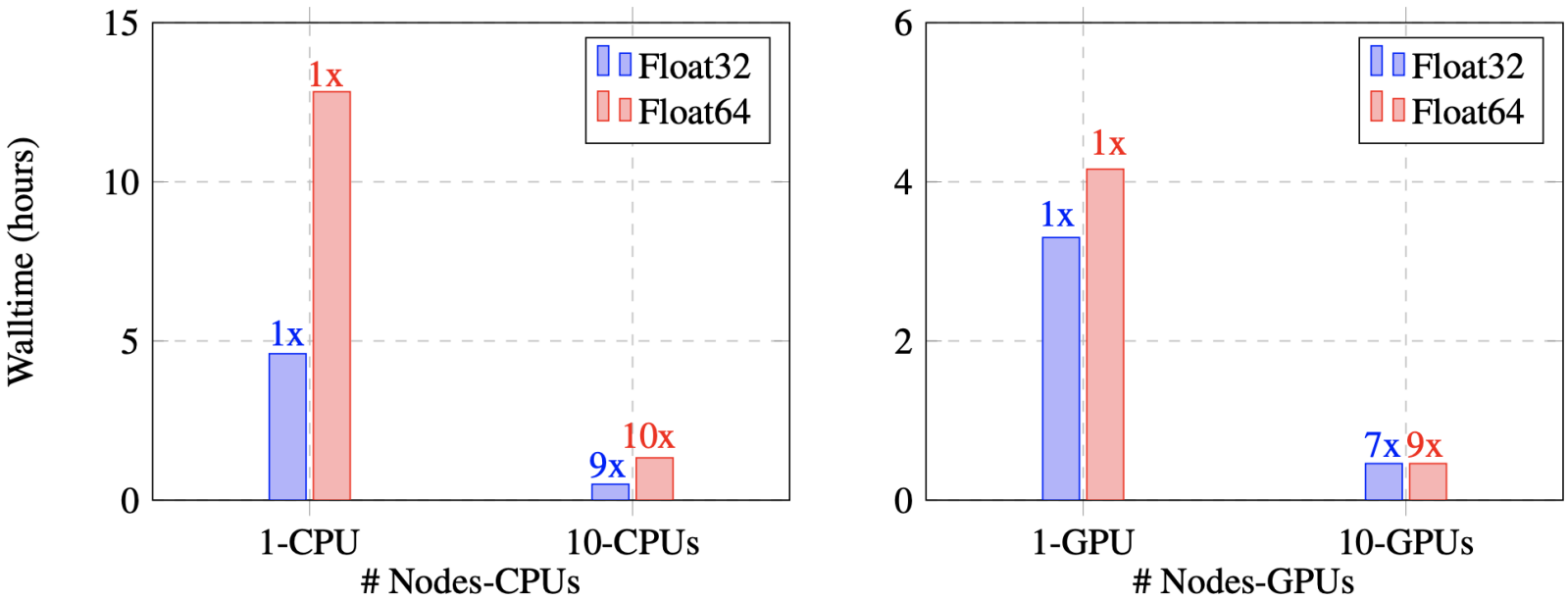
Hyperparameters

Subdomain number	1	2	3	4	5	6	7	8	9	10
# Residual points	3000	4000	5000	4000	3000	4000	800	3000	5000	4000
Adaptive Activation function	tanh	sin	cos	tanh	sin	cos	tanh	sin	cos	tanh

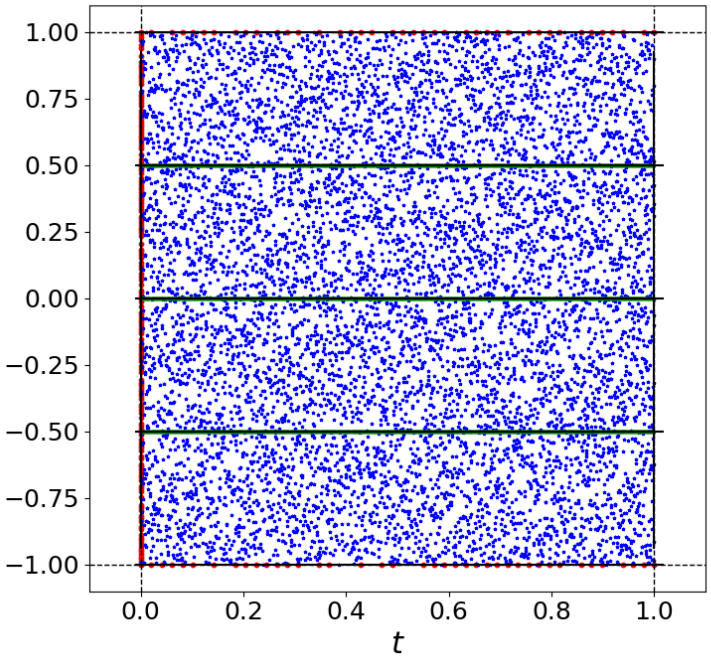
Temperature and Conductivity (T, K)

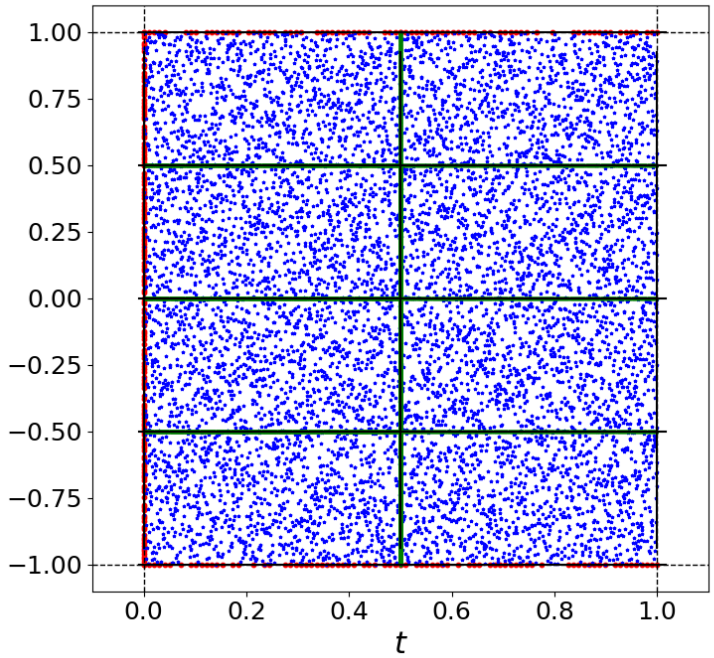


Scaling


Walltime (hours)
15
10
5
0
1x
1x
10x
9x
Float32
Float64
1-CPU
10-CPUs
Nodes-CPUs
6
4
2
0
1x
1x
7x
9x
Float32
Float64
1-GPU
10-GPUs
Nodes-GPUs

Space-Time partitioning





cPINN vs XPINN partition			
# $x-$ partitions	# t partitions	cPINN time per iter. (s)	XPINN time per iter. (s)
4	1	0.14	-
4	2	-	0.060

LES Modeling- Ongoing project

Filtred Navier-Stokes equations

$$\frac{\partial \widetilde{\boldsymbol{u}}}{\partial t} + \widetilde{\boldsymbol{u}} \cdot (\nabla \widetilde{\boldsymbol{u}}) = -\nabla \widetilde{\boldsymbol{p}} + \nu \nabla^2 \widetilde{\boldsymbol{u}} - \nabla \cdot \widetilde{\boldsymbol{\tau}} + F$$
$$\nabla \cdot \widetilde{\boldsymbol{u}} = 0,$$

where $\widetilde{\tau}$ is subgrid stress and computed using Smagorinsky model with van Driest damping, which reads

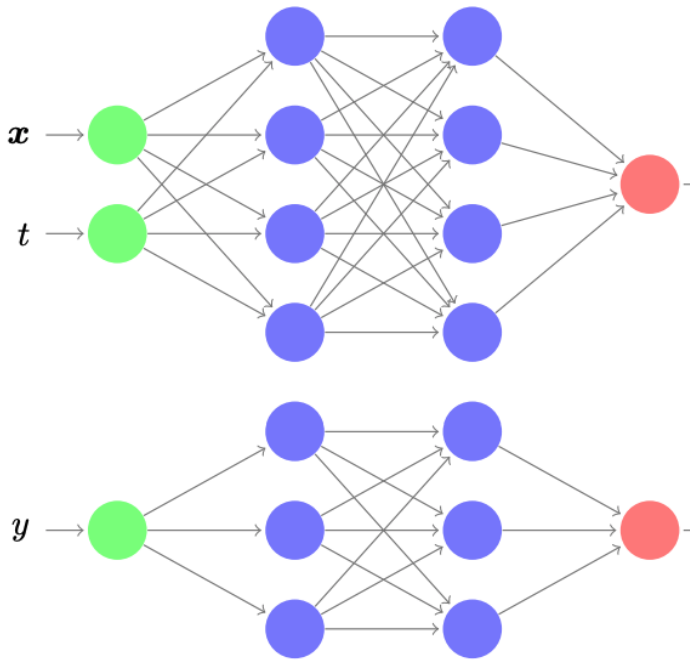
$$\tau_{ij} = -2(c_s(y)\Delta)^2 \sqrt{\widetilde{S_{kl}}\widetilde{S_{kl}}}\widetilde{S_{ij}},$$

with Driest-damped Smagorinsky constant
$c_s(y) = c_0(1 - \exp(-y/A))$

The corrected subgrid stress term

$$\tau_{ij} = -2(c_s(y)\Delta)^2(1 + \delta_c)\sqrt{\widetilde{S_{kl}}\widetilde{S_{kl}}}\widetilde{S_{ij}},$$

Inverse PINN for $c_s(y)$



$$\tilde{\boldsymbol{u}}(\boldsymbol{x},t)$$
$$p(\boldsymbol{x},t)$$
$$\delta_c(y)$$

$$L_{data} = |\tilde{\boldsymbol{u}}(\boldsymbol{x},t) - \boldsymbol{u}^*|^2$$

$$L_{phys} = \left|\frac{\partial \tilde{\boldsymbol{u}}}{\partial t} + \tilde{\boldsymbol{u}}\cdot\boldsymbol{\nabla}\tilde{\boldsymbol{u}} + \boldsymbol{\nabla}\tilde{p} + P_x\hat{x} - \nu\nabla^2\tilde{\boldsymbol{u}} + \boldsymbol{\nabla}\cdot\boldsymbol{\tau}\right|^2 + |\boldsymbol{\nabla}\cdot\tilde{\boldsymbol{u}}|^2$$

$$\boldsymbol{u}^* = \widetilde{\boldsymbol{u}^{DNS}}$$

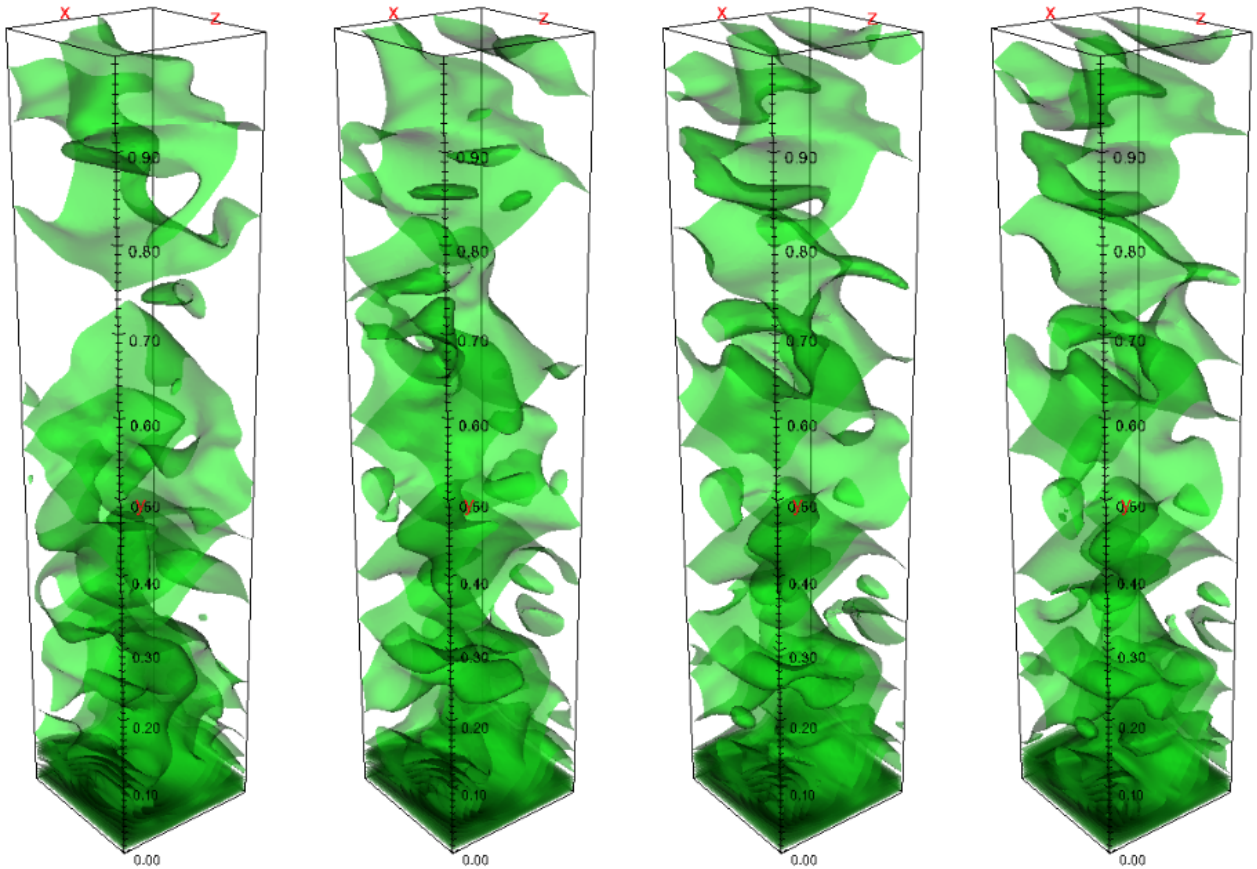
$$\tau_{ij} = -2(c_S(y)\Delta)^2(1+\delta_c)\sqrt{2\tilde{S}_{kl}\tilde{S}_{kl}}\tilde{S}_{ij}$$

$$\tilde{S}_{ij} = \frac{1}{2}\left(\frac{\partial \tilde{u}_i}{\partial x_j} + \frac{\partial \tilde{u}_j}{\partial x_i}\right)$$

$$L = \lambda L_{data} + L_{phys}$$

Turbulence Data- JHTDB

- Domain Size $= 210\delta_\nu \times 1000\delta_\nu \times 210_\nu$
- $(N_x, N_y, N_z) = 17 \times 256 \times 34$
- Viscous Length $= 1.0006 \times 10^{-3}$
- NT $= 64 \rightarrow 1000$



A-priori testing

- The SGS dissipation rate

$$\Pi = -\tau_{ij}\tilde{S}_{ij}.$$

- An approximate value for c_S can be obtained by studying the ratio between the exact dissipation rate given as

$$\Pi^e = -\tau^e_{ij}\tilde{S}_{ij},$$

and the modeled dissipation rate, which is

$$\Pi^S = -\tau^S_{ij}(c_S)\tilde{S}_{ij}.$$

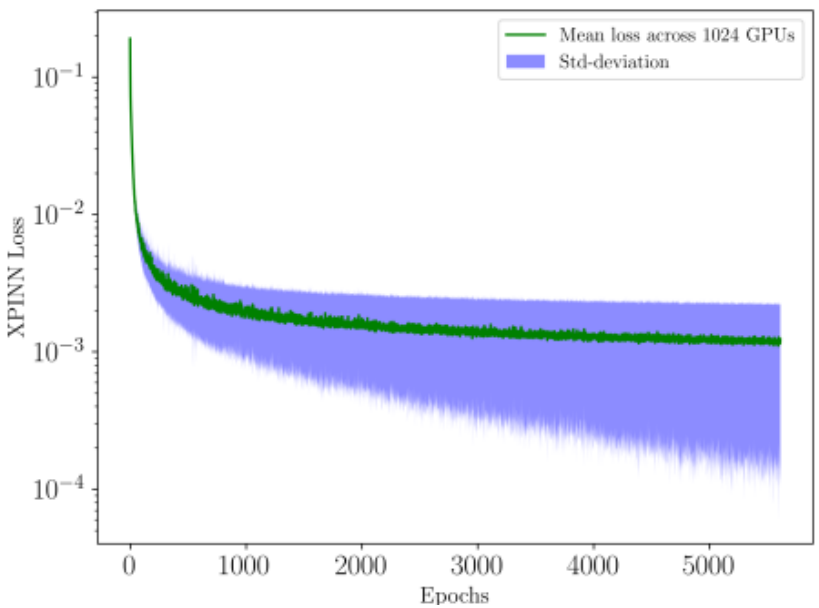
- After rearranging the different terms, we get

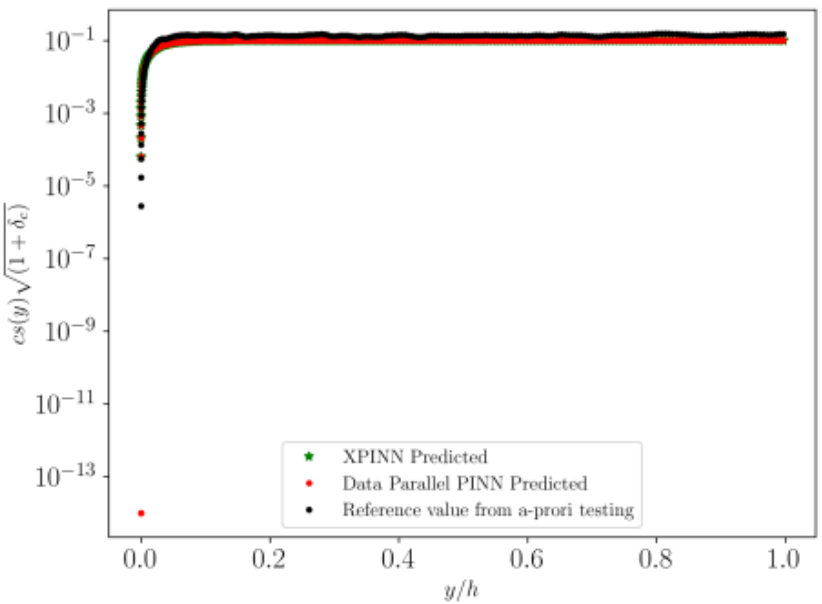
$$c_s(y) = \sqrt{\frac{\langle \tau^e_{ij}\tilde{S}_{ij}\rangle_{x,z,t}}{\langle \tau^S_{ij}(c_S = 1)\tilde{S}_{ij}\rangle_{x,z,t}}},$$

where the dissipation rates were averaged horizontally and temporally.

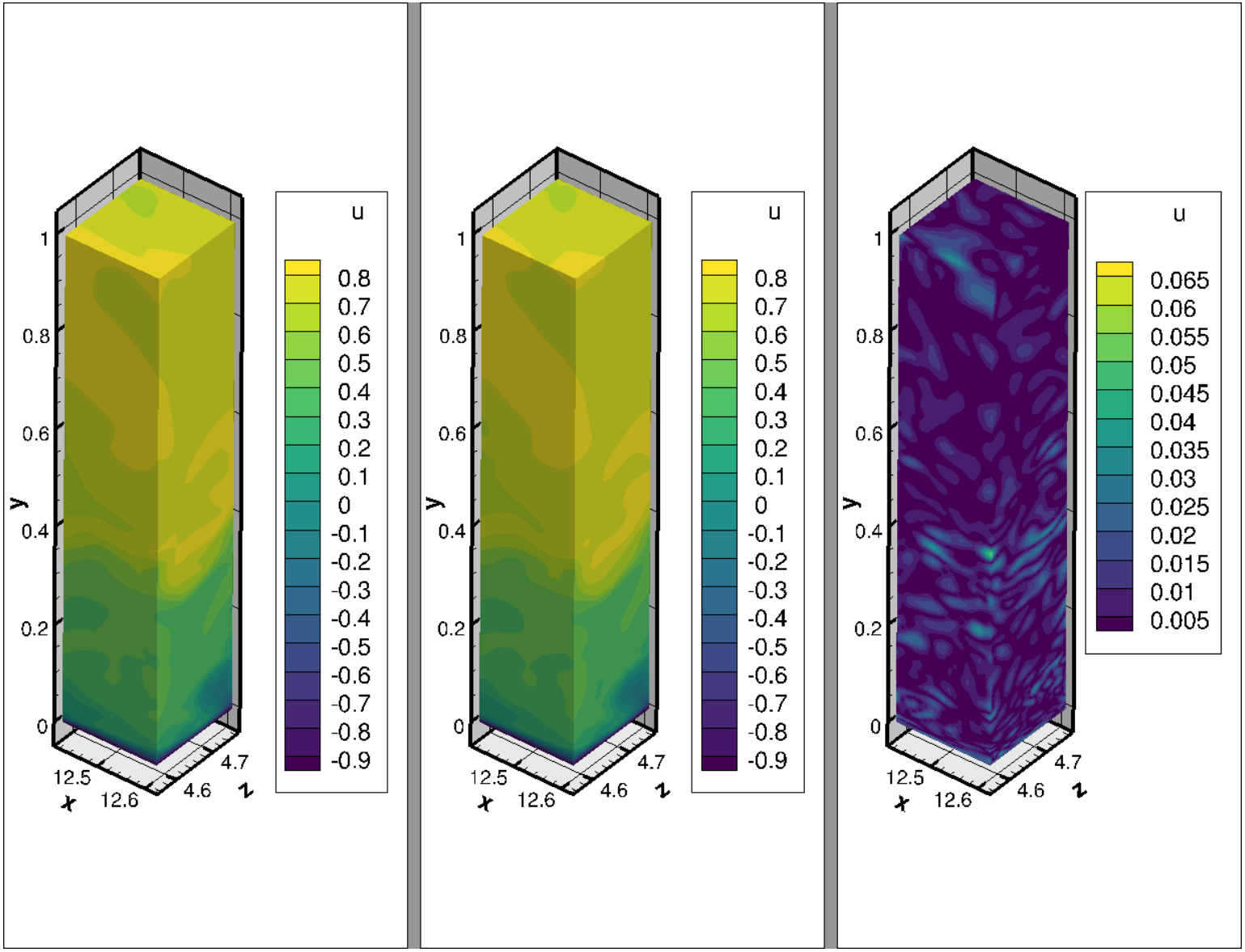
Result $c_s(y)$

- 1024 Cubes (Snapshot), 1024 GPUs (Polaris: A100) 1 Node: 4 GPUs
- Tested with 2 architectures: Data Parallel and XPINN.



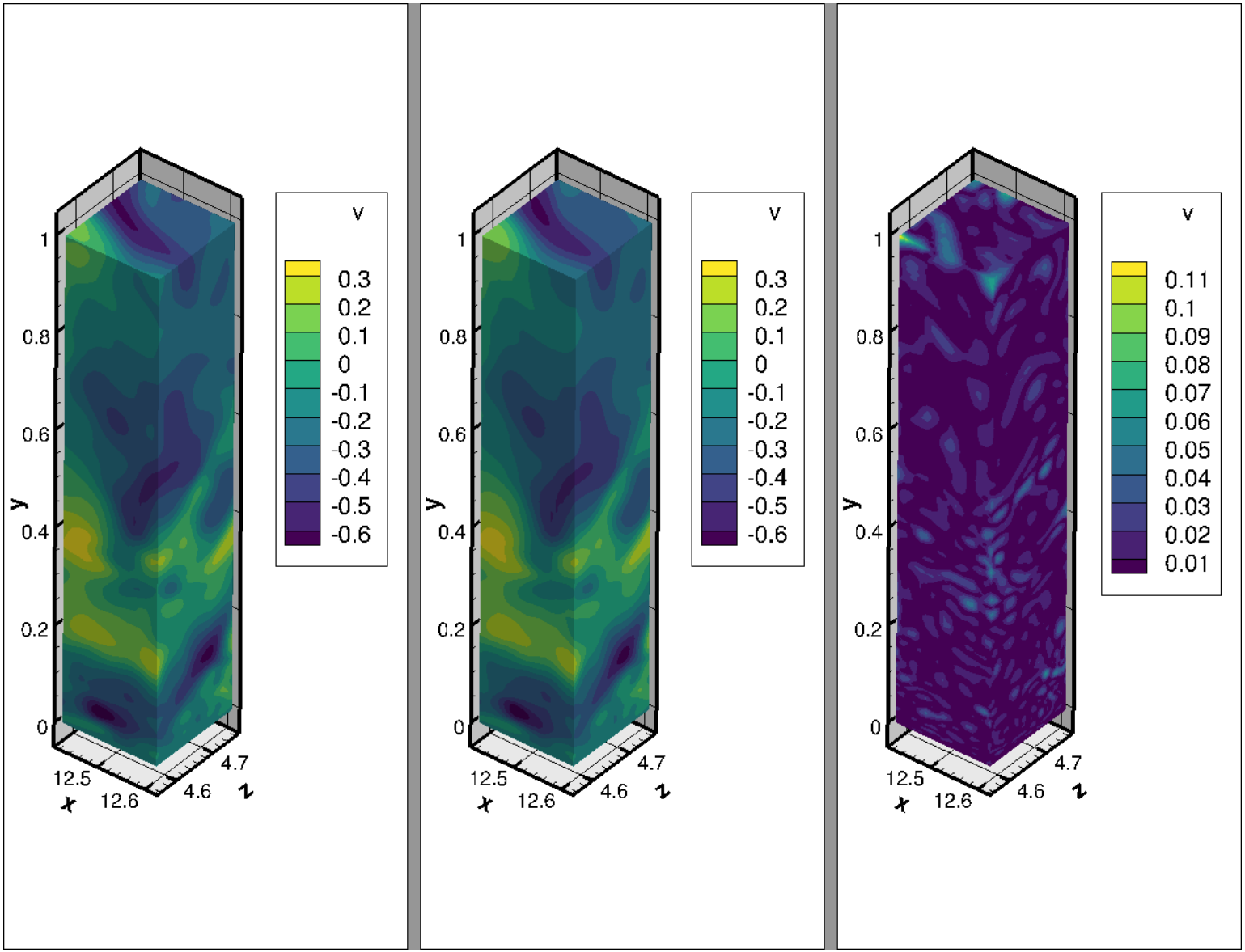


Fields and prediction error: u



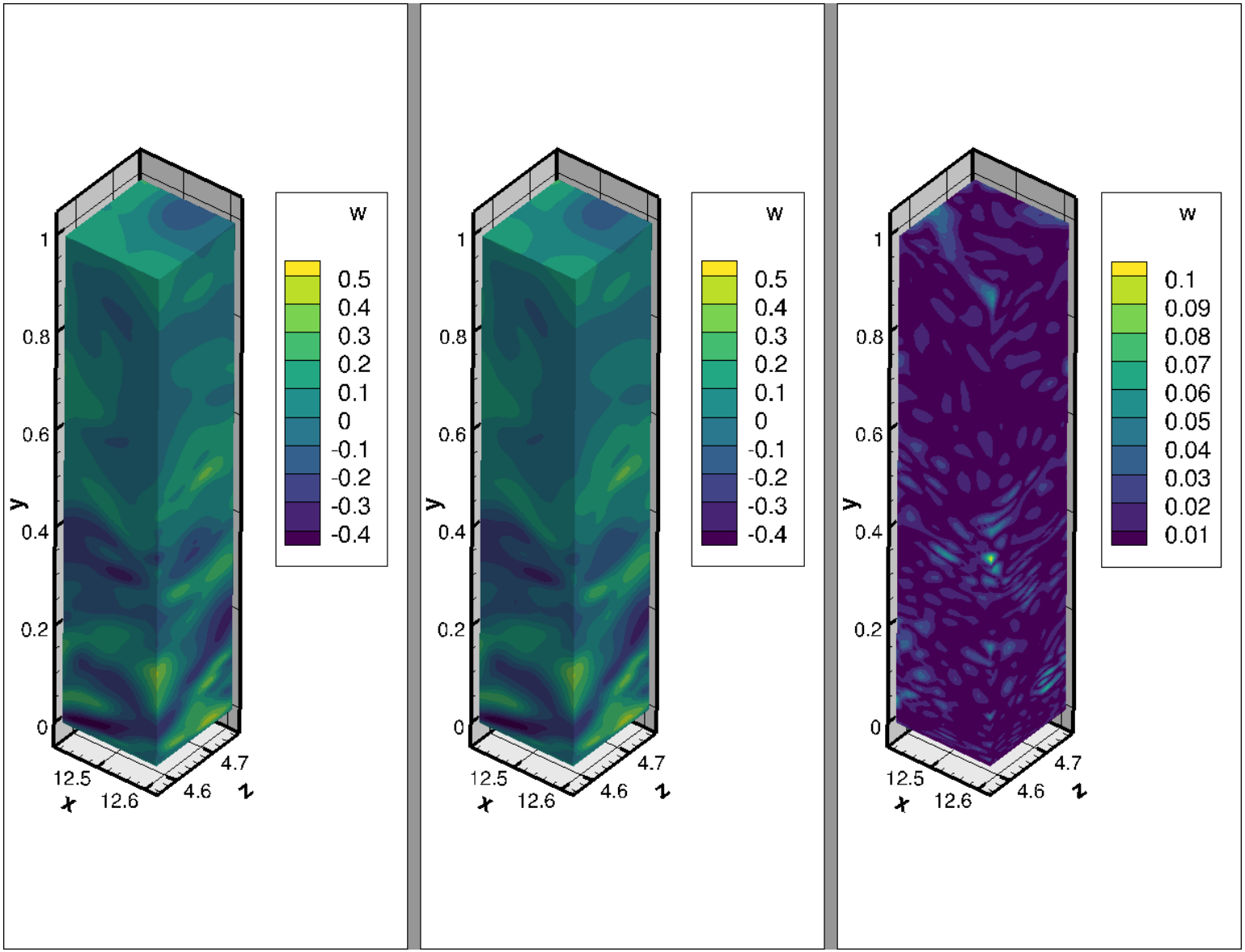
Actual Predicted Point-wise error

Fields and prediction error: v



Actual | Predicted | Point-wise error

Fields and prediction error: w

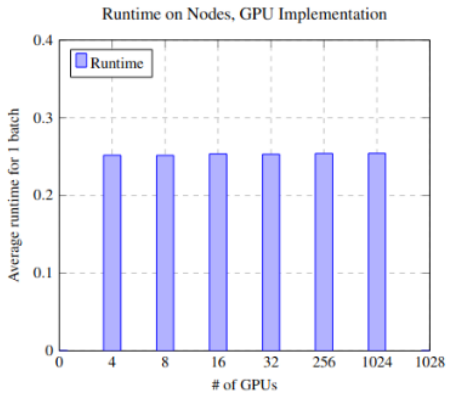


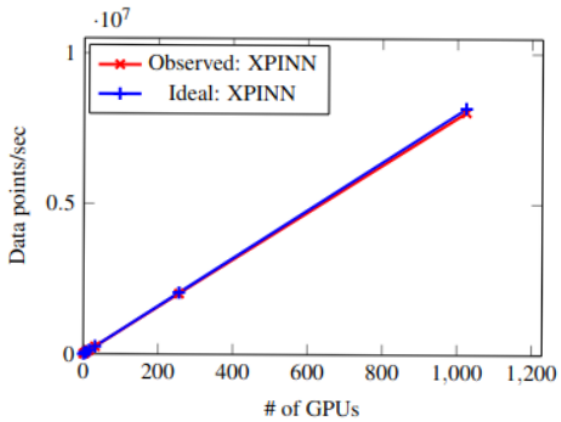
Actual | Predicted | Point-wise error

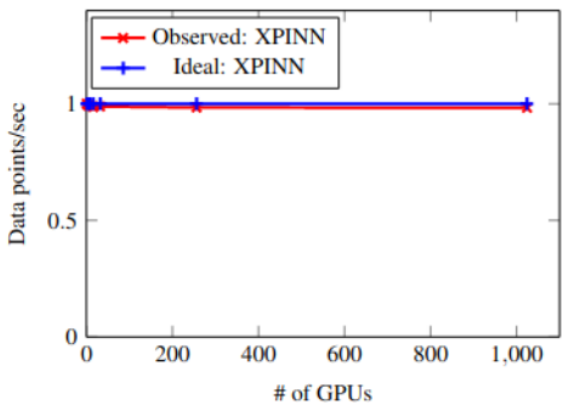
Different architectures vs error

Methods	Rel. L_2 error in u	Rel. L_2 error in v	Rel. L_2 error in w
XPINN 64GPU-64 Time Steps	0.99%	2.6%	2.4%
Data parallel PINN 64GPU-64 Time Steps	6.8%	13.02%	15.44%
XPINN 128GPU-1024 Time Steps	5.87%	11.91%	18.44%
Data parallel PINN 128GPU-1024 Time Steps	3.60%	11.99%	13.23%
XPINN 256GPU-1024 Time Steps	8.02%	5.45%	13.79%
XPINN 1024GPU-1024 Time Steps	1.53%	3.77%	6.10%

Scaling for LES







Conclusions

- cPINN is only applicable for conservation laws. However, application of XPINN is independent of the nature of DEs.
- cPINN is more efficient than XPINN if decomposition is performed in space only.
- For transient problems, the communication overhead in XPINN (compared to cPINN) due to spatial decomposition will be compensated by partitioning the domain along the time axis as well.
- Weak scaling is achieved for $\boldsymbol{x} \in \{CPUs, GPUs\}$.
- Implementation of XPINN for LES modeling acheives very good scaling.

Acknowledgment

- OSD/AFOSR MURI
- DARPA CompMods
- DOE PhILMs
- Oak Ridge National Laboratory (ORNL)
- ALLC, ANL
- CCV, Brown University

Thank You!