Neural Domain Decomposition for Variable Coefficient Poisson Solvers

Sebastian Barschkis*
University of California, Irvine
Irvine, CA, USA
sbarschk@uci.edu

Zitong Li*
University of California, Irvine
Irvine, CA, USA
zitongl5@uci.edu

Hengjie Wang
Modular Inc
USA
hengjiew@uci.edu

Aparna Chandramowlishwaran
University of California, Irvine
Irvine, CA, USA
amowli@uci.edu

ABSTRACT

The computational bottleneck in many fluid simulations arises from solving the variable coefficient Poisson equation. To tackle this challenge, we propose a novel neural domain decomposition algorithm to accelerate its solution. Our approach hinges on two key ideas: first, using neural PDE solvers to approximate the solutions within subdomains, and second, ensuring continuity across subdomain boundaries by solving a Schur complement system derived from the cell-centered discretized Poisson equation. A distinct advantage of our approach lies in generating a large dataset consisting only of small-scale problems to train the subdomain solver. This trained model can subsequently be applied to problems with large and complex geometries. Moreover, by batching the independent subdomain solves, we achieve high GPU utilization with neural solvers compared to state-of-the-art numerical methods. In contrast to neural domain decomposition algorithms that rely on Schwarz overlapping methods, our optimization-based approach, coupled with neural PDE solvers, improves accuracy and performance.

KEYWORDS

Poisson equation, variable coefficient, neural PDE solver

1 MOTIVATION

The Poisson equation solving step is the bottleneck of many fluid flow simulations. MFiX's [4] fluid solver, for example, spends more than 80% of its time on solving the Poisson equation. Methods that speed up this process and that generalize to a wide range of Poisson equation configurations, i.e. methods that can solve Poisson equations with varying boundary conditions and variable coefficients, could improve the efficiency of these numerical flow simulators.

Predicting solutions of partial differential equations (PDEs) with NNs is generally faster than solving PDEs numerically. NNs are therefore considered a promising tool to solve PDEs efficiently. While their training is usually compute intensive it only needs to be done once. Trained models can predict solutions subject to many input boundary conditions (BC) – not just those that were shown to the network during training.

Among NNs, convolutional neural networks (CNNs) are particularly suited for solving PDEs since they, similarly to finite differences (FD), capture the correlation of adjacent cells. In contrast to fully-connected NNs which would require very wide layers to map the right hand side (RHS) and the variable coefficient to the solution, CNNs generalize well to large domains.

2 BACKGROUND

2.1 The Variable Coefficient Poisson Equation

We consider the variable coefficient 2D Poisson equation

$$\begin{aligned} \nabla \cdot (a \nabla p(\boldsymbol{x})) &= f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \\ p(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega \end{aligned} \tag{1}$$

with Dirichlet BC. For our numerical ground truth data, we solve the equation by discretizing it with FD and solving the corresponding linear system

$$A(a)p = b(a, f)$$

with a preconditioned conjugate gradient (PCG) solver. For a constant coefficient, the system becomes $Ap = b$ where A only depends on size of the domain.

2.2 The Superposition Principle

Since the Laplace operator is linear, we can decompose the solution to our original Poisson equation into the sum of the same Poisson equation with 0 as BC and a Laplace equation with the BC from the original problem. Formally we have:

$$p(\boldsymbol{x}) = p^p(\boldsymbol{x}) + p^l(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega$$

where p^l is the solution to the Laplace equation:

$$\begin{aligned} \nabla^2 p^l(\boldsymbol{x}) &= 0, \qquad \boldsymbol{x} \in \Omega \\ p^l(\boldsymbol{x}) &= g(\boldsymbol{x}), \quad \boldsymbol{x} \in \partial\Omega \end{aligned}$$

and p^p is the solution to the Poisson equation with 0-BC:

$$\begin{aligned} \nabla^2 p^p(\boldsymbol{x}) &= f(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \\ p^p(\boldsymbol{x}) &= 0, \qquad \boldsymbol{x} \in \partial\Omega \end{aligned}$$

*Both authors contributed equally to this research.

SC23, November 12–17, 2023, Denver, CO, USA
2023. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

3 NEURAL PDE SOLVER

The neural PDE solver for the inhomogeneous variable coefficient Poisson equation is based on the observation that its solution can be derived from the sum of the Poisson equation with zero-BC and the Laplace equation.

3.1 Architecture

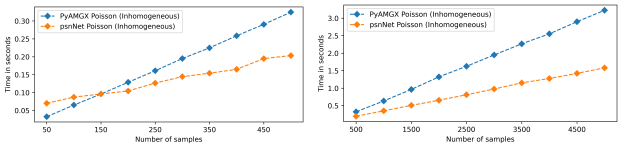
The architecture for all of our Poisson / Laplace equation models is based on the original U-Net [5] with skip connections. To prevent overfitting and improve convergence we optimize this architecture by using batch-normalization layers after each convolution and 'LeakyRelu' activations in contracting and expanding paths.

4 PRELIMINARY RESULTS

Solving Poisson with a NN has two advantages: (1) Speedup over numerical solvers when solving many samples, and (2) improved accuracy when breaking up the inhomogeneous Poisson equation into separate models for the Laplace and Poisson zero-BC equations.

4.1 Performance

The NN performs better than the numerical solver [3] when solving lots of samples. While the latter computes solutions iteratively, the neural PDE solver predicts n at once – resulting in a speedup.



(a) 50 to 500 samples (b) 500 to 5,000 samples

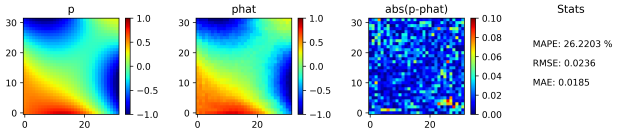
Figure 1: Time taken to solve (`pyamgx.solve()`) and predict (`tf.predict()`) n (32x32) inhomogeneous Poisson instances using one Nvidia V100 GPU (excluding GPU data upload / download times). With a large n the neural solver becomes more performant than the numerical method (Fig. 1b).

4.2 Accuracy

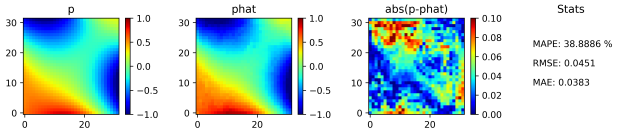
When predicting the solution $\hat{p}$ of the inhomogeneous Poisson equation given previously unseen g, a, and f, more accurate results can be achieved when predicting the solution to the Poisson zero-BC and Laplace equations using separate models and then summing their predictions (Figure 2). While test inputs g, a, and f are unseen, they follow a smooth distribution and have a value range that matches the training data.

5 DOMAIN DECOMPOSITION METHOD

To solve the large domain in parallel, we leverage the superposition principle introduced earlier. The Poisson equation with 0-BC can be solved by the neural network model introduced in previous section. To solve the Laplace equation, we need the solution to the original Poisson equation on the interface between subdomains. In our FD discretization scheme, the unknown elements that we solve for are on the cell center while the boundary conditions, including the



(a) Sum of predictions from Poisson zero-BC and Laplace models



(b) Predictions from inhomogeneous Poisson equation model

Figure 2: Comparison of ground truth p, predictions $\hat{p}$ and errors. More accurate results can be achieved when separating the solution to the inhomogeneous Poisson equation into the sum of Poissson zero-BC and Laplace equations (a). The model that has learned to predict the solution to the inhomogeneous Poisson equation directly (b) yields a higher MAE.

ones lying in between subdomains, are on the cell interface. For any point g on the interface, we can form the following linear system of equations:

$$2g = p_0 + p_1 \tag{2}$$

$$p_0 = p_0^l + p_0^p = \boldsymbol{c}_0^T \boldsymbol{g}_0 + p_0^p \tag{3}$$

$$p_1 = p_1^l + p_1^p = \boldsymbol{c}_1^T \boldsymbol{g}_1 + p_1^p \tag{4}$$

where p_0 and p_1 are the unknowns at the centers of the cells adjacent to g. Note here $p_i^l = \boldsymbol{c}_i^T \boldsymbol{g}_i$ derives from the fact that in FD, the solution to the Lapalce equation at any point is the linear combination of the boundary condition of the subdomain. Using this system of equation require us to know the $\boldsymbol{c}$. Currently we compute $\boldsymbol{C}$ through computing the inverse of the discretization matrix of each subdomain. Having computed $\boldsymbol{c}$, we solve this linear system through AMG. The resulting boundary conditions can then be used to solve the Laplace equation on each subdomain. The pseudo-code of this algorithm is outlined as the following: Taking the inverse

1: **function** Poisson-DDM($\bar{\mathbf{A}}$, $\bar{\mathbf{F}}$, $\boldsymbol{g}$)
2: $\mathbf{P}^p$ = UNet($\bar{\mathbf{A}}$, $\bar{\mathbf{F}}$)
3: Find C such that C$g = p^l$
4: Assemble and solve linear system using C to solve for $\boldsymbol{p^l}$ on the subdomain interface
5: $\mathbf{P}^l$ = UNet($\bar{\boldsymbol{a}}$, $\boldsymbol{g}$)
6: $\mathbf{P} = \mathbf{P}^l + \mathbf{P}^p$
7: **end function**

of the discretization matrix is expensive and numerically unstable, thus we are currently investigating potential ways to replace this with a neural network without sacrificing accuracy.

REFERENCES

[1] Ekhi Ajuria Illarramendi, Michael Bauerheim, Neil Ashton, Corentin Lapeyre, and Bénédicte Cuenot. 2023. Performance Study of Convolutional Neural Network Architectures for 3D Incompressible Flow Simulations. In *Proceedings of the Platform*

for Advanced Scientific Computing Conference (Davos, Switzerland) *(PASC '23)*. Association for Computing Machinery, New York, NY, USA, Article 17, 11 pages. https://doi.org/10.1145/3592979.3593416

[2] Ismayil Ismayilov, Javid Baydamirli, Doğan Sağbili, Mohamed Wahib, and Didem Unat. 2023. Multi-GPU Communication Schemes for Iterative Solvers: When CPUs Are Not in Charge. In *Proceedings of the 37th International Conference on Supercomputing* (Orlando, FL, USA) *(ICS '23)*. Association for Computing Machinery, New York, NY, USA, 192–202. https://doi.org/10.1145/3577193.3593713

[3] Nvidia. 2023. Algebraic Multigrid Solver (AmgX) Library, https://github.com/NVIDIA/AMGX, (2023-08-01).

[4] U.S. Department of Energy. 2023. MFIX-Exa, https://amrex-codes.github.io/MFIX-Exa/, (2023-08-01).

[5] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer, 234–241.

[6] Hengjie Wang, Robert Planas, Aparna Chandramowlishwaran, and Ramin Bostanabad. 2022. Mosaic flows: A transferable deep learning framework for solving PDEs on unseen domains. *Computer Methods in Applied Mechanics and Engineering* 389 (2022), 114424.

[7] Ali Girayhan Özbay, Arash Hamzehloo, Sylvain Laizet, Panagiotis Tzirakis, Georgios Rizos, and Björn Schuller. 2021. Poisson CNN: Convolutional neural networks for the solution of the Poisson equation on a Cartesian mesh. *Data-Centric Engineering* 2 (2021). https://doi.org/10.1017/dce.2021.7