Solving Sparse Linear Systems on Large Unstructured Grids with Graph Neural Networks: Application to solve the Poisson's equation in Hall-Effect Thrusters simulations

Gabriel Vigot[1], Bénédicte Cuenot[1], Olivier Vermorel[1]

[1]Centre Européen de Recherche et de Formation Avancée en Calcul Scientifique
42 av. Gaspard Coriolis, Toulouse, 31100, France
vigot@cerfacs.fr, cuenot@cerfacs.fr, vermorel@cerfacs.fr

Abstract. The following work presents a new method to solve Poisson's equation and, more generally, sparse linear systems using graph neural networks. We propose a supervised approach to solve the discretized representation of Poisson's equation at every time step of a simulation. This new method will be applied to plasma physics simulations for Hall-Effect Thruster's modeling, where the electric potential gradient must be computed to get the electric field necessary to model the plasma's behavior. Solving Poisson's equation using classical iterative methods represents a major part of the computational costs in this setting. This is even more critical for unstructured meshes, increasing the problem's complexity. To accelerate the computational process, we propose a graph neural network to give an initial guess of Poisson's equation solution. The new method introduced in this article has been designed to handle any meshing structure, including structured and unstructured grids and sparse linear systems. Once trained, the neural network would be used inside a numerical simulation in inference to give an initial guess of the solution for each simulation time step for all right-hand sides of the linear system and all previous time step solutions. In most industrial cases, Hall-Effect thrusters' modeling requires a large unstructured mesh that one single processor cannot hold regarding memory capacity. We then propose a partitioning strategy to tackle the challenge of solving linear systems on large unstructured grids when they cannot be on a single processor.

Keywords: Plasma Physics · Partial differential equations · Graph Neural Networks · Sparse Linear systems · Partitioning

1 Introduction

Sparse linear systems may arise when discretizing Partial Differential Equations (PDEs) for numerical simulations. For elliptic problems like Laplace, Poisson, or Helmholtz equations, these problems are even more important when they need to be solved on a large unstructured mesh. Besides, the mesh itself could

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

be partitioned into multiple subgraphs and distributed in parallel over multiple processors to compute the solution of the discretized PDEs. The linear system, when ill-conditioned, becomes harder to solve with a low convergence rate. Decades of research have been led to reduce the computational cost of solving a sparse linear system with efficiency with preconditioning techniques (Chen *et al.* [9]), or multi-level approach (Saad *et al.* [8]). Since the introduction of Physics Informed Neural Networks by Raissi *et al.* [7], multiple efforts have been made to help different kinds of solvers, find the solution to elliptic problems, applied to unstructured data that could be modeled as graph neural networks (Pfaff *et al.* [6], Sanchez-Gonzalez, Godwin *et al.* [13]). Other groups have focused more of their attention on solving sparse linear systems using neural network operators (Jiang *et al.* [2]), whether with a supervised approach or an unsupervised approach (Stanziola *et al.* [18]). While some other groups dedicated their research to finding the right preconditioning matrix to help the iterative solvers converge faster (Li *et al.* [17], Sappl *et al.* [14], or Luz *et al.* [1]).
We propose a model to find the update $\Delta\Phi^t$ of the solution $\Phi^t = \Phi^{t-1} + \Delta\Phi^t$ for each time step t of a simulation. In the end, we would like to demonstrate the ability of a neural network to solve elliptic PDEs discretized as sparse linear systems $\mathbf{A}\,\Phi^t = \mathrm{b}^t$ where $\mathbf{A}$ represents the Laplacian operator and b^t the right-hand side of Poisson's equation at a given time step t. In the context of industrial applications, we extend this work to a large unstructured mesh where the memory size of the graph is too large to fit onto a single graphics processing unit (GPU). The final goal is to have a graph neural network capable of predicting an initial guess Φ^t for each time step t of a simulation based on the solution of the previous time step Φ^{t-1} and the right-hand side of the linear system b^t. The neural neural should also be able to understand the dynamics of the data represented in the graph, even if it is partitioned.
The following will first present the model and the neural network architecture. Then, two applications will be shown in the context of plasma simulations for Hall-effect Thruster design: the first one is a 2D simulation with triangular mesh, and the second one is a plasma discharge in 3D with irregular tetrahedral mesh, and the whole partitioned in multiple subgraphs.

2 Hybrid Model

The main objective of the method is to conciliate the solving speed of a linear system and the precision that the model has to reach at the end of each time step of a simulation. A particular focus is given to solving Poisson's equation, which is one fundamental step in plasma physics modeling. The expectations are that the neural network will be able to find a good approximation of the solution update for each time step of a simulation. During its training phase, the neural network is trained using a solution of Poisson's equation at a given simulation time step. Ultimately, the neural network should provide an initial guess of Poisson's equation that is close to the physical solutions given in its training phase.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

The computational cost would decrease because a neural network's inference phase is less computationally expensive than a traditional iterative solver.

2.1 Poisson's equation

Several discretization methods exist in numerical simulation to represent Poisson's equation on an unstructured grid. In this present work, Poisson's equation will be computed using the finite volume method. We describe our Poisson's problem on a given discretized domain $\mathring{\Omega}$ delimited by Dirichlet boundary conditions $\partial\Omega_{\mathrm{D}}$:

$$\begin{cases} \nabla^2\Phi & = -\dfrac{q}{\epsilon_0}(n_i - n_e) \quad \text{on } \mathring{\Omega} \\ \Phi & = \Phi_{\mathrm{D}} \qquad\qquad\quad \text{on } \partial\Omega_{\mathrm{D}} \end{cases} \tag{1}$$

where $-q(n_i - n_e)$ represents the right-hand side of our Poisson's equation with q the electric charge of an electron, ϵ_0 the vacuum permittivity, n_i the charge density of ions constituting the plasma, and n_e the electron charge density. The numerical scheme based on the AVBP solver created at CERFACS [12] defines the discretization of Poisson's equation with the Green-Ostrogradski theorem where for a nodal volume V_i of a node i belonging to the computational domain $E(i)$ for each cell τ:

$$\int_{V_i} \nabla^2\Phi dV = \int_{\partial V_i} \nabla\Phi \cdot \mathbf{n} dS = \sum_{\tau \in E(i)} \int_{\partial V_i \cap \tau} \nabla\Phi \cdot \mathbf{n}\, dS, \tag{2}$$

where $\mathbf{n}\, dS$ represents the orthogonal vector to the surface of the cell τ. So that it is possible to build the discretized Laplacian operator over an unstructured mesh with primal cell volume V_i, which gives us the following linear system to solve:

$$\mathbf{A}\, \Phi^{\mathrm{t}} = \mathrm{b}^{\mathrm{t}}, \tag{3}$$

where for t the designated time step of a simulation, $\mathbf{A}$ is the discretized Laplacian operator, Φ^{t} the solution of the linear system at time step t and b^{t} the right-hand side of the linear system at time step t since Φ and b will evolve over time.

2.2 Graph Convolution Network model

We use a non-structured approach to harness a graph convolutional network that leverages node information stored on large unstructured meshes affiliated with the physical problem. By learning the node features on the graph, the neural network should have a global understanding of the nature of the data represented on the graph. We use the SAGEConv operator proposed by Hamilton *et al.* [19]. It consists of sampling the nodes of a graph, aggregating the features of the sampled node to a certain level of the node's local neighborhood, and repeating the process until the network has a complete representation of the node features

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

of the graph.
SAGEConv is a mean aggregator which could be summarized with the following equation where for a designated graph convolutional layer k:

$$\mathbf{h}_v^k \leftarrow \sigma(\mathbf{W} \,.\, \text{MEAN}(\{\mathbf{h}_v^{k-1}\}) \cup \{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}) \tag{4}$$

where

- $\mathbf{h}_v^k$: represents the new node representation,
- v: the concatenation of the current node propagation,
- $\{\mathbf{h}_v^{k-1}\}$: the current node information propagation,
- $\{\mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}(v)\}$ means the information, aggregation from its immediate neighborhood,
- u the selected neighborhood nodes,
- $\mathbf{W}$ is the weight matrix of the trainable parameters,
- σ the nonlinear activation.

This strategy makes it possible to pass through a large graph and learn its dynamics in terms of node feature information.

2.3 Neural network architecture

As noted in section 2.2, the SAGEConv model requires several layers to fully increase the capacity of the neural network to analyze globally the node features, mostly in the case where the graph is very large. As the neural network's main architecture, we suggest stacking several SAGEConv convolution layers. Each layer will be associated with PReLU as a nonlinear activation layer since this type of activation layer has a trainable parameter to adapt the output of the convolution layer [20]. Finally, the activation layer will be followed by a LayerNorm to stabilize the neural network's learning process [21]. The chosen neural network architecture has five layers with a linear operator at the end of it. Each layer's structure comprises a sequential operator of the SAGEConv model, a PReLU activation layer, and a LayerNorm. The neural network was optimized using the Adam optimizer [22] with $\beta_1 = 0.95$ and $\beta_2 = 0.90$ with all the trainable parameters initialized by default with the Glorot Normal distribution [23]. The learning rate is fixed at 1×10^{-3}. All training methods are written using PyTorch [3] and PyTorch Geometric [5] to model graph neural networks and monitor their training. Every training was conducted using the Distributed Data-Parallel paradigm from PyTorch [4] and ran on 4 NVIDIA A30 GPUs for the 2D case and 4 NVIDIA V100 for the 3D case.

2.4 Hybridization method

The benefit of using a neural network inside a numerical simulation is that it decreases the computation time necessary for the discretized Poisson problem. In plasma physics simulation where we have to solve Poisson's equation at each

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

time step of a simulation, the solution at the current time step Φ^t could be found incrementally such as:

$$\Phi^{\mathrm{t}} = \Phi^{\mathrm{t}-1} + \Delta\Phi^{\mathrm{t}}, \tag{5}$$

where $\Delta\Phi^{\mathrm{t}}$ represents the solution update for the current time step, in the correction of what we had in the previous simulation time step. At first, the neural network is built using the same connectivity as the unstructured mesh from the numerical simulation. Then, the neural network is employed to find the solution update $\Delta\Phi^{\mathrm{t}}$ so that:

$$\Phi^{\mathrm{t}} = \Phi^{\mathrm{t}-1} + f_\theta(\Phi^{\mathrm{t}-1}, \mathrm{b}^{\mathrm{t}}), \tag{6}$$

where $f_\theta(\Phi^{\mathrm{t}-1}, \mathrm{b}^{\mathrm{t}})$ represents the output of the neural network, with hyperparameters θ. The output will update the current time step solution $\Delta\Phi^{\mathrm{t}}$. As explained in equation (6), the neural network will have as input the solution of the previous time step $\Phi^{\mathrm{t}-1}$, the current right-hand side of the discretized PDE b^{t}, source term as defined in equation (3). The supervised approach uses the neural network to provide an initial guess for each simulation time step. The initial guess is then refined by an iterative solver, which is supposed to perform a few iterations to converge.

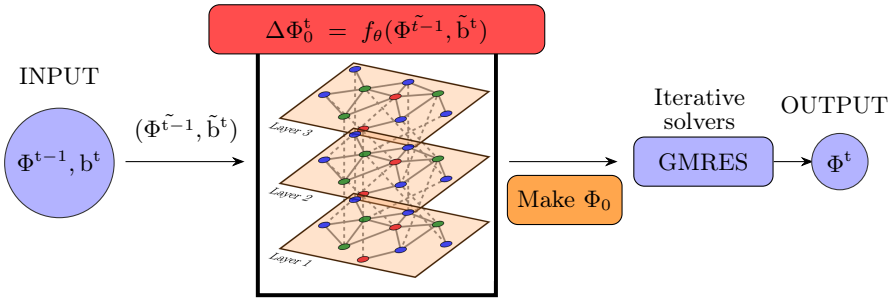


Fig. 1: Inference sketch for solving linear system $\mathbf{A}\,\Phi^{\mathrm{t}} = \mathrm{b}^{\mathrm{t}}$ throughout a simulation

As demonstrated in figure 1, we take inputs from the solution of the previous time step $\Phi^{\mathrm{t}-1}$, and the right-hand side b^{t} of the linear system. We normalize b^{t} and $\Phi^{\mathrm{t}-1}$ with their respective L_2 norm giving $\tilde{\Phi}^{\mathrm{t}-1}$ and $\tilde{\mathrm{b}}^{\mathrm{t}}$. Inside a simulation, the neural network will provide an update $\Delta\Phi_0^{\mathrm{t}}$ that will give an initial guess $x_0 = \Phi^{\mathrm{t}-1} + \Delta\Phi_0^{\mathrm{t}}$ for an iterative solver such as a GMRES solver [8]. The iterative solver in that method is here to ensure good accuracy for the next step to avoid the divergence of the simulation throughout time iterations.

2.5 Training and partitioning algorithm

As a training method, we propose a supervised approach where the neural network should make a prediction close to a solution of reference obtained with an

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

iterative solver for each simulation time step. The supervised learning objective will be the root-mean-squared error (RMSE) between the output of the neural network and the update of reference for the designated time step:

$$\mathcal{L}_\theta = \sqrt{\sum_{j=1}^{n} \frac{(f^t_{\theta,\mathrm{j}} - \Delta\Phi^t_{\mathrm{j,ref}})^2}{n}}, \tag{7}$$

where j corresponds to the node index of the update vector. The update of reference $\Delta\Phi^{\mathrm{t}}_{\mathrm{j,ref}} = \Phi^{\mathrm{t}}_{\mathrm{ref}} - \Phi^{\mathrm{t-1}}_{\mathrm{ref}}$ is also expressed with the same node index as the prediction of the neural network for all solution node n of the unstructured mesh. In the major cases of industrial numerical simulations for plasma physics, it is necessary to study a discretized domain that largely exceeds the memory capacity of one single processor. This issue remains consistent for large graphs where the training phase and the feature node information linked to the graph structure exceeds the memory capacity of one single GPU.
In 2019, Chiang *et al.* [24] suggested partitioning the graph into several clusters and making predictions for each cluster separately. Using the library METIS [25], the clustering is computed by subdividing the initial graph $\bar{G}$ into k groups of nodes $G_1, G_2, ..., G_k$, where $G_k := \{\mathcal{V}_i, \mathcal{E}_i\}\ \forall i \in [\![1, k]\!]$. $\mathcal{E}_i$ represents the edge connectivity of subgraph G_i. The partitioning is realized without overlapping. Hence, the neural network should have only one representation of a node on a designated subgraph. Once the graph is partitioned, the training could be organized as follows:

1. For each time step we partition the input data $\Phi^{\mathrm{t-1}}$ and b^{t} and distribute each partitioned data to their respective subgraph,
2. We do a forward passing for each subgraph independently from one another without communication between them,
3. We concatenate each subgraph output into a single vector and reorder this vector the same way as the global node ordering before the partitioning as for $\Phi^{\mathrm{t-1}}$ or b^{t},
4. This final vector will be used to compute the RMSE loss for each time step of a simulation,
5. We repeat the process for each time step of a simulation.

The process is repeated for each training iteration, giving the neural network an overview of the whole dynamics of the simulation. Despite the missing communication between the subgraph of the problem, we expect, with the backpropagation, a global understanding of the problem. Each subgraph's output vector has gradient values, which will be concatenated for backpropagation. Even though the forward pass of the neural network is done locally for each subgraph, the update of the neural network weights is done globally by backpropagating the final vector that regroups all the gradients of each subgraph output. In the end, the optimization process will be global, and the neural network must be able to learn the global patterns of the main graph before its partitioning.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

3 Test cases and training datasets

The present work has a special application to electric space propulsion. Hence, the datasets that constitute the training process for our method are based on the specific application field.

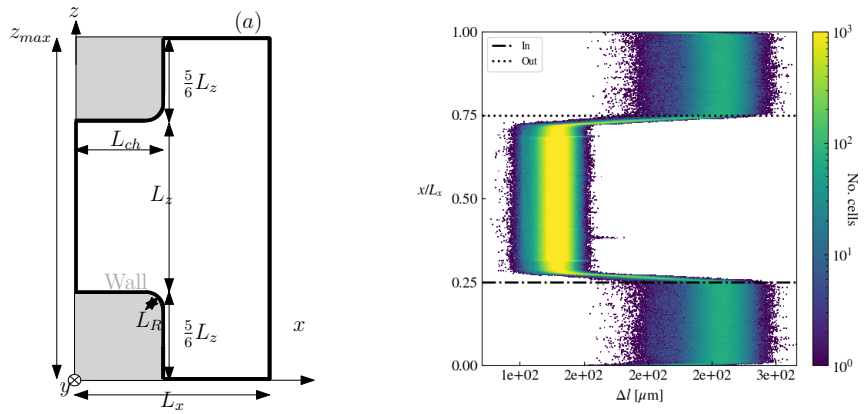
3.1 2D MTSI in a radial-azimuthal (r, θ) plan

As a first example, we propose a 2D Particle-In-Cell (PIC) simulation from Petronio *et al.* [26] where we study the interaction of two opposite flows called Modified-Two-Stream-Instability (MTSI), meaning two species of charged particles going in opposite directions: one constituted of ions and the other, of electrons.
In the context of this simulation, the domain is discretized with regular squares, each split in half into triangles. A static electromagnetic field $\mathbf{E}\times\mathbf{B}$ is imposed on the virtual axis of the simulation (axial direction for $\mathbf{E}_x$ and azimuthal direction $\mathbf{B}_y$) so that the plasma instability will be correctly presented. This simulation aims to demonstrate the feasibility of solving Poisson's equation for a regular grid enough to fit in the memory of one single GPU (33,153 nodes). In this simulation, when a particle leaves on the right side of the domain, it will be reinjected on the left side, which is assimilated as a periodic boundary condition. The remaining boundaries marked in green are considered Dirichlet boundary conditions where the electric field is forced to zero.

3.2 3D PIC simulation for Electron Drift Instability (EDI)

The second case is the study led by Villafana *et al.* [27], where 3D Particle-In-Cell simulation was made to describe the plasma instabilities due to the electrons which are very difficult to track inside a Hall-Thruster. A 3D unstructured mesh comprising 2,739,491 nodes was set to capture this physical phenomenon. The mesh is a section of the Hall Thruster channel in the azimuthal direction. Periodicity is added for the faces belonging to the azimuthal direction to guarantee an electric equilibrium of the plasma. The mesh itself is fully unstructured with a refined region inside the chamber of a Hall-Thruster. On this complex large geometry with different cell sizes, using a graph neural network becomes advantageous when capturing the dynamics of the physical fields without interpolating the latter. Due to the size of the graph, this graph will be split into 100 partitions to ensure every partition will fit in the memory capacity of one GPU for the training sequence.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41



(a) 2D radial-axial $(z - x)$ view of the 3D geometry

(b) Radial distribution of the cell size normalized on the radial length L_z

Fig. 2: 3D global view of 3D PIC simulation

3.3 Datasets overview

In figure (2), the simulations constitute the datasets for the neural network's training. Throughout the simulation of the 2D case, every linear system solution is computed using a GMRES solver from PETSc [10] with stopping criteria of 1×10^{-8} for the residual norm value without preconditioner. For the 3D case, an iterative solver MAPHYS [11] was used to obtain the solution with a similar threshold convergence.

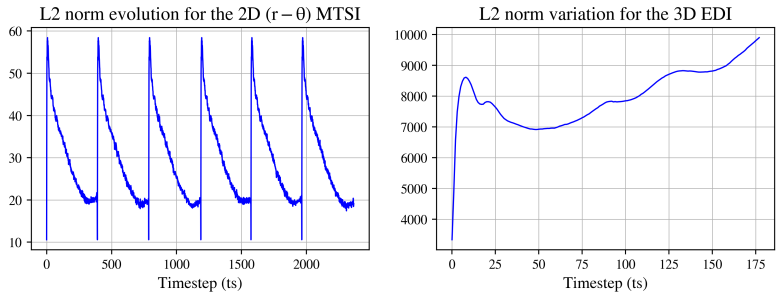


Fig. 3: Variation of L_2 norm $||\mathbf{A}\,\Phi_{\text{ref}}^{\text{t-delay}} - \text{b}^{\text{t}}||_2/||\text{b}^{\text{t}}||_2$ for MTSI 2D simulation (with delay $= 1$) and the 3D EDI simulation on the right (with delay $= 5,000$)

In the 2D case, the solutions that compose the dataset of our neural network are saved every 100 time steps. For every 100 time steps, we store the current solution, the solution of the previous time step, and the right-hand side of the

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

current time step. Overall when computing the mean L_2 norm of the whole dataset $||\mathbf{A}\Phi^t - b^t||_2$, the L_2 norm of the simulation turns approximately around 30. In the 3D case, the solutions to save are more computationally expensive to store. The solutions are then stored every 5,000-time step without saving the solution of the previous time step for the current time step solution we want to store. Consequently, the mean L_2 norm of the 3D case is higher than the 2D case, with approximately 7,900.

4 Numerical Experiments

In this section, we report the results of our neural networks in training and inference. For both cases, we use the simulations reduced in PyTorch format datasets presented in section 3.3. We split each simulation into training and validation sets. The validation represents the last 5% of the simulation itself, where we check the learning evolution of the model.

4.1 Training and inference for the 2D case

We present a synthesis of our training and inference results and the training and validation set for both cases. For the training process, we plot a map of neural network predictions along the time steps of the simulation and compare them to the solution of reference, which has a precision error of 1×10^{-8}. The timeline results for the 300^{th} epochs of the training process are shown in figure (6a) in the appendix (6).

Table 1: Mean and standard deviation of the RMSE of the training set during the training phase, the validation set during inference, compared to the reference

Training	**Validation**	**Reference**
$\approx 0.07 \pm 3 \times 10^{-4}$	$\approx 0.10 \pm 3 \times 10^{-4}$	$\approx 0.19 \pm 0.04$

From figure (6a), the predictions made by the neural network during the training process show a prediction with an absolute error that does not exceed 1.0. As we can notice in table 1, the training phase shows an RMSE that does not exceed the mean level of 0.10. With a standard deviation lesser than 0.001, the neural network approximates the solution update well for all simulation time steps for the training and validation phase. The RMSE of reference refers to the RMSE between the previous time step and the current time step solution, meaning the error of the initial guess before the neural network correction giving $\Delta\Phi_0^t$ as mentioned in section 2.4.

The validation set in inference presented in figure (4) below seems to follow the same trend as for the training set during the training phase. Indeed, supervised learning is expected to reach a certain level of accuracy for both training

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

and validation sets since the difference b^t and Φ^t have the same profile for both sets. If the solution of reference is an acceptable target for the user with a known precision error, then optimizing the neural network with a supervised approach is recommended.

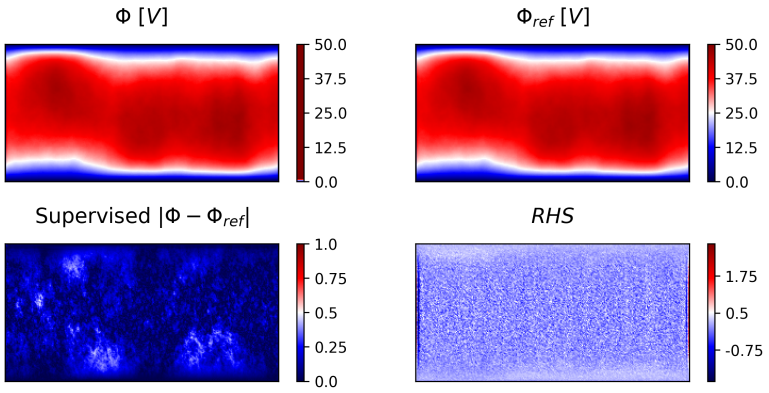


Fig. 4: Comparison of the solution updated by the neural network Φ, the reference Φ_{ref}, the absolute difference between them $|\Phi - \Phi_{\text{ref}}|$ and the right-hand side of the Poisson's equation at time step $t_s = 3.55\ \mu s$ for the validation set

4.2 Training and inference for the 3D case

Contrary to section 4.1, the inputs given to the neural network for the current time step are not the solution from the previous time step but from the 5,000$^{\text{th}}$ before the current one. Thus, the learning process would become harder, explaining the difference in resolution between the RMSE of the 2D case's RMSE and the 3D case. As in section 4.1, we present a synthesis of our results for training and inference, training and validation set combined. For the training process, we plot in the appendix (6) on the figure (6b) a map of neural network predictions along the time steps of the simulation and compare them to the solution of reference, which has a precision error of 1×10^{-12}.

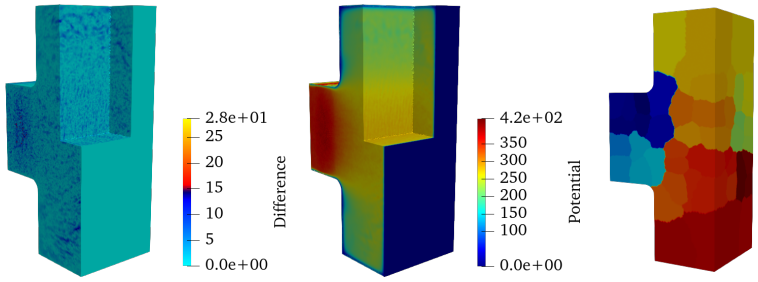
Table 2: 3D Case: Mean and standard deviation of the RMSE for all the training set during the training, the validation set during inference, compared to the reference

Training	**Validation**	**Reference**
$\approx 3.49 \pm 8 \times 10^{-3}$	$\approx 4.92 \pm 0.02$	$\approx 7.02 \pm 0.59$

Overall, since the predictions made by the neural network are based on the previous time step with a delay of 5,000-time steps and the current right-hand

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

side b^t, it becomes more difficult for the neural network to capture the update of the current time step properly. This could be explained by the plasma being stabilized progressively during the simulation, giving the same trend for all time steps at this phase. But it is not at the beginning of the simulation where we have a high variation of the Poisson's equation solution every time step. Consequently, the training process shows a prediction with an absolute error of 100 V amplitude at the training dataset's beginning. It ends with an absolute error of approximately 20 V of amplitude. As we can notice in table 2, the training phase shows an RMSE with a mean level of 3.49, significantly higher than the results given in table 1. But with a standard deviation lesser than 0.05, the neural network can provide for all time steps a good approximation of the solution update without a high dispersion degree of the solution update's precision for all time steps. The standard deviation of RMSE for the reference dataset is significantly higher. This could be explained by the heterogeneity of the dataset, which results in a higher dispersion of solution accuracy throughout the simulation.



(a) $|\Phi - \Phi_{\text{ref}}|$ $[V]$ (b) Φ $[V]$ (c) Map of partitions

Fig. 5: Absolute difference between the solution updated by the neural network and the solution of reference (a) $|\Phi - \Phi_{\text{ref}}|$, the solution updated by the neural network (b) Φ, at time step $t_s = 4.25$ μs for the validation set, and the partition map of the graph using METIS [25]

For the validation set, with the partitioning as for the training set, the supervised approach remains closer to the reference solution. Even with a partitioned process, the prediction does not seem to present the same discontinuities as the partitions presented in figure (5c). As shown in figure (5a), the amplitude of the difference does not exceed 15 V for a solution of reference where the amplitude exceeds 400 V despite that the neural network has an input of Φ^{t-5000}. Nonetheless, for training and inference, the neural network seems to capture the global pattern of the solution even though the forward passing is realized locally for each subgraph.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

5 Conclusion & perspectives

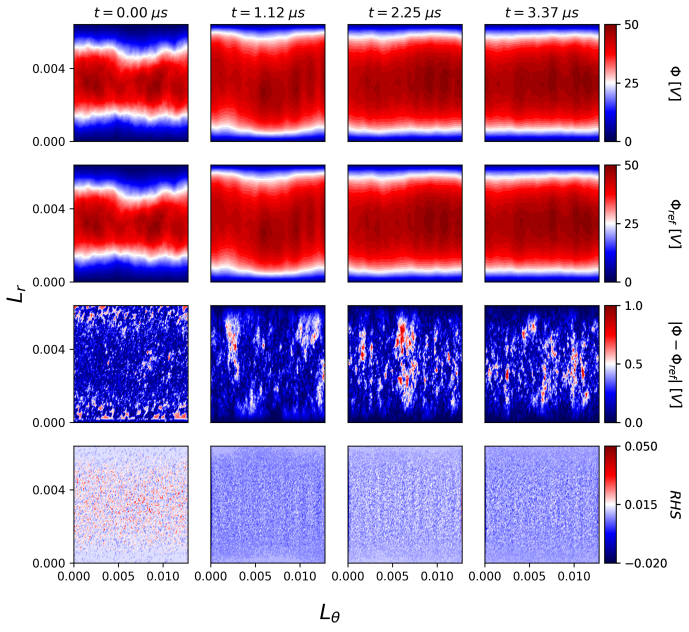
In this work, we proposed a method capable of giving an initial guess for each time step of a simulation to speed up the solving process of a linear system. This method has been applied to plasma physics problems to obtain the electric potential from Poisson's equation. The method proposed in this study is also applied to large graph problems concerning unstructured mesh, which cannot be treated with one GPU processor. The partitioning process explained in this article proposed an approach to tackle this challenge. By summing all the subgraphs' outputs in one single vector indexed in global graph node ordering, we proceed with a global backpropagation of all the subgraphs' outputs to let the neural network learn about the problem globally. In the end, the neural network should be able to make an initial guess of our current linear system for large graph problems, which could be an asset for industrial applications, specifically in our case to Hall-effect thrusters where the size of the unstructured mesh is often very large.
As we intended to prove that the neural network can predict the solution for multiple b^t for one single geometry, meaning only a unique $\mathbf{A}$ matrix, we expect to generalize the process by training a neural network on a geometry then testing the model on a different geometry to observe the variations of the predictions.
Until now, we have proven the possibility of using a neural network to give a good initial guess for each simulation time step. To validate the method, we also need to show the ability of this new method to accelerate the convergence of iterative solvers. The present work is to prove the viability of using a neural network as an accelerator for an iterative solver like GMRES. With this hybrid implementation, we expect the hybrid method to be faster and have fewer iterations to converge. Future results will be presented at the conference with a fully coupled neural network with an iterative solver. In the end, once the neural network is trained, the latter will be used in inference inside a numerical simulation to provide a good approximation of the solution that helps the iterative solver to converge faster.

Acknowledgments. This research has been realized under the supervision of Bénédicte Cuenot and Olivier Vermorel (senior researchers at CERFACS). Gabriel Vigot acknowledges the support of Luciano Drozda (senior researcher at CERFACS) in conceiving and validating this present article for the machine learning part. Gabriel Vigot also acknowledges the support of Luc Giraud (senior researcher at INRIA) in conceiving and validating this present article for the part concerning linear algebra.

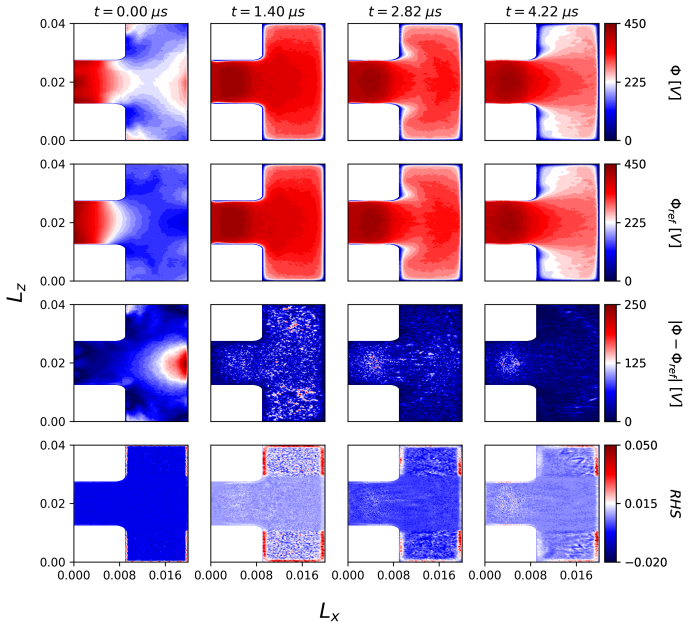
Disclosure of Interests. The present authors have no competing interests with the participants of this article. Gabriel Vigot acknowledges the financial support from Safran Spacecraft Propulsion, under the supervision of Benjamin Laurent, and the French Space Agency (CNES: Centre National d'Études Spatiales), under the supervision of Ulysse Weller, under the EPIC convention. This work is introduced within the CHEOPS project.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

6 Appendix



(a) 2D radial-azimuthal (L_r, L_θ) map at epoch 300



(b) 3D map in the centered axial-radial (L_x, L_z) plane at epoch 300

Fig. 6: (a) 2D map along the simulation time line t in microseconds. From the first to the fourth row is the update Φ, the solution of reference Φ_{ref}, the absolute difference $|\Phi - \Phi_{\text{ref}}|$, and RHS the right-hand side of the Poisson's equation.
(b) Neural network predictions Φ with the same variable order as in fig (6a)

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

References

1. Luz, I., Galun, M., Maron, H., Basri, R., and Yavneh, I. Learning algebraic multigrid using graph neural networks. In: International Conference on Machine Learning, pp. 6489–6499. PMLR, 2020.
2. Jiang, Z., Jiang, J., Yao, Q. et al. A neural network-based PDE solving algorithm with high precision. Sci Rep 13, 4479 (2023), (https://doi.org/10.1038/s41598-023-31236-0)
3. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Advances in Neural Information Processing Systems 32, pp. 8024– 8035. Curran Associates, Inc., 2019.
4. S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, S. Chintala, PyTorch distributed: experiences on accelerating data parallel training, In: VLDB Endowment, 2020, vol. 13, n. 12, (https://doi.org/10.14778/3415478.3415530)
5. Fey, M. and Lenssen, J. E. Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
6. Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. Learning mesh-based simulation with graph networks. In the International Conference on Learning Representations, 2020.
7. Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics Informed Neural Networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational physics, 378:686–707, 2019, (https://doi.org/10.1016/j.jcp.2018.10.045)
8. Saad, Y. and Zhang, J. Enhanced multi-level block ilu preconditioning strategies for general sparse linear systems. Journal of Computational and Applied Mathematics, 130(1):99–118, 2001. ISSN 0377-0427, (https://doi.org/10.1016/S0377-0427(99)00388-X).
9. Chen, J., Schäfer, F., Huang, J., and Desbrun, M. Multiscale Cholesky preconditioning for ill-conditioned problems. ACM Trans. Graph., 40(4), jul 2021c. ISSN 0730- 0301. (https://doi.org/10.1145/3450626.3459851)
10. S. Balay, W. D. Gropp, L. Curfman McInnes, B. F. Smith, Efficient Management of Parallelism in Object Oriented Numerical Software Libraries, In: Modern Software Tools in Scientific Computing, p. 163–202 (1997), (https://doi.org/10.1007/978-1-4612-1986-6_8)
11. E. Agullo, L. Giraud, A. Guermouche, J. Roman, Parallel hierarchical hybrid linear solvers for emerging computing platforms, Comptes Rendus Mécanique, Volume 339, Issues 2–3, (2011), Pages 96-103, ISSN 1631-0721, (https://doi.org/https://doi.org/10.1016/j.crme.2010.11.005)
12. N. Gourdain, Prediction of the unsteady turbulent flow in an axial compressor stage. Computers & Fluids, (2015), (https://doi.org/10.1016/j.compfluid.2014.09.052).
13. Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In International Conference on Machine Learning, pp. 8459–8468. PMLR, 2020.

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41

14. Sappl, J., Seiler, L., Harders, M., and Rauch, W. Deep learning of preconditioners for conjugate gradient solvers in urban water related problems, 2019. https://arxiv.org/abs/1906.06925.
15. Schäfer, F., Katzfuss, M., and Owhadi, H. Sparse Cholesky factorization by Kullback–Leibler minimization. SIAM Journal on Scientific Computing, 43(3):A2019–A2046, 2021, (https://doi.org/10.1137/20M1336254)
16. A. Kopaničáková and G. E. Karniadakis, DeepOnet Based Preconditioning Strategies For Solving Parametric Linear Systems of Equations, (2024), 2401.02016, arXiv
17. Y. Li, P. Y. Chen, T. Du, W. Matusik, Learning Preconditioner for Conjugate Gradient PDE Solvers, In International Conference on Machine Learning Computer Science, Mathematics, Engineering (2023), (https://proceedings.mlr.press/v202/li23e.html)
18. A. Stanziola, S. R. Arridge, B. T. Cox, B. E. Treeby, A Helmholtz equation solver using unsupervised learning: Application to transcranial ultrasound, Journal of Computational Physics, Volume 441, (2021), (https://doi.org/10.1016/j.jcp.2021.110430)
19. W. L. Hamilton, R. Ying, J. Leskovec: Inductive representation learning on Large graphs. In: 31st International Conference on Neural Information Processing Systems (NIPS) (2017),
20. K. He, X. Zhang, S. Ren, J. Sun: Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification, International Conference on Computer Vision (2015), (https://doi.org/10.1109/ICCV.2015.123)
21. J. L. Ba, J. R. Kiros, G.E. Hinton, Layer Normalization, In: Neural Information Processing System (2016)
22. D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).
23. X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, In: Proceedings of the thirteenth international conference on artificial intelligence and statistics, JMLR Workshop and Conference Proceedings, 2010, pp. 249–256.
24. W. L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, C. J. Hsieh, Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks, In: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data MiningJuly (2019), Pages 257–266, (https://doi.org/10.1145/3292500.3330925)
25. G. Karypis, V. Kumar. A fast and high-quality multilevel scheme for partitioning irregular graphs. SIAMJ. Sci. Comput. 20,1, (1998), 359–392, (https://doi.org/10.1137/S1064827595287997)
26. W. Villafana, F. Petronio, A. C. Denig, M. J. Jimenez, D. Eremin, L. Garrigues, F. Taccogna, A. Alvarez-Laguna, J. P. Boeuf, A. Bourdon, P. Chabert, T. Charoy, B. Cuenot, K. Hara, F. Pechereau, A. Smolyakov, D. Sydorenko, A. Tavant and O. Vermorel, 2D radial-azimuthal particle-in-cell benchmark for E×B discharges, In Plasma Sources Science and Technology, Volume 30, Number 7, (2021), (https://doi.org/10.1088/1361-6595/ac0a4a)
27. W. Villafana, G. Fubiani, L. Garrigues, G. Vigot, B. Cuenot, O. Vermorel, 3D Particle-In-Cell modeling of anomalous transport driven by the Electron Drift Instability in Hall Thrusters, In: 37^th International Electric Propulsion Conference, MIT, (2022)

ICCS Camera Ready Version 2024
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-63759-9_41