Version of Record: https://www.sciencedirect.com/science/article/pii/S0045782520304357
Manuscript_589ab6be842116a9e190e7f91f6c9b52

PPINN: Parareal Physics-Informed Neural Network for time-dependent PDEs

Xuhui Meng[1*], Zhen Li[2*], Dongkun Zhang[1] and George Em Karniadakis[1†]

[1] Division of Applied Mathematics, Brown University, Providence, RI 02912, USA

[2] Department of Mechanical Engineering, Clemson University, Clemson, SC 29634, USA

June 27, 2020

Abstract

Physics-informed neural networks (PINNs) encode physical conservation laws and prior physical knowledge into the neural networks, ensuring the correct physics is represented accurately while alleviating the need for supervised learning to a great degree [1]. While effective for relatively short-term time integration, when long time integration of the time-dependent PDEs is sought, the time-space domain may become arbitrarily large and hence training of the neural network may become prohibitively expensive. To this end, we develop a *parareal* physics-informed neural network (PPINN), hence decomposing a long-time problem into many independent short-time problems supervised by an inexpensive/fast coarse-grained (CG) solver. In particular, the serial CG solver is designed to provide approximate predictions of the solution at discrete times, while initiate many fine PINNs simultaneously to correct the solution iteratively. There is a two-fold benefit from training PINNs with small-data sets rather than working on a large-data set directly, i.e., training of individual PINNs with small-data is much faster, while training the fine PINNs can be readily parallelized. Consequently, compared to the original PINN approach, the proposed PPINN approach may achieve a significant speed-up for long-time integration of PDEs, assuming that the CG solver is fast and can provide reasonable predictions of the solution, hence aiding the PPINN solution to converge in just a few iterations. To investigate the PPINN performance on solving time-dependent PDEs, we first apply the PPINN to solve the Burgers equation, and subsequently we apply the PPINN to solve a two-dimensional nonlinear diffusion-reaction equation. Our results demonstrate that PPINNs converge in a few iterations with significant speed-ups proportional to the number of time-subdomains employed.

Keywords: deep neural network, machine learning, parallel-in-time, long-time integration, multi-scale, PINN

1 Introduction

At a cost of a relatively expensive computation in the training process, deep neural networks (DNNs) provide a powerful approach to explore hidden correlations in massive data, which in many cases are physically not possible with human manual review [2]. In the past decade, the large computational

*The first two authors contributed equally to this work.
†Corresponding Email: george_karniadakis@brown.edu

cost for training DNNs has been mitigated by a number of advances, including high-performance computers [3], graphics processing units (GPUs) [4], tensor processing units (TPUs) [5], and fast large-scale optimization schemes [6], i.e., the adaptive moment estimation (Adam) [7] and the adaptive gradient algorithm (AdaGrad) [8]. In many instances for modeling physical systems, physical invariants, e.g., momentum and energy conservation laws, can be built into the learning algorithms in the context of DNN and their variants [9–14]. This leads to a physics-informed neural network (PINN), where physical conservation laws and prior physical knowledge are encoded into the neural networks [1, 15]. Consequently, the PINN model relies partially on the data and partially on the physics described by partial differential equations (PDEs).

Different from traditional PDE solvers, although a PDE is encoded in the neural network, the PINN does not need to discretize the PDE or employ complicated numerical algorithms to solve the equations. Instead, PINNs take advantage of the *automatic differentiation* employed in the backward propagation to represent all differential operators in a PDE, and of the training of a neural network to perform a nonlinear mapping from input variables to output quantities by minimizing a loss function. Unlike numerical differentiation (e.g., finite difference method, FDM), automatic differentiation does not differentiate the data and hence it can tolerate noisy data [16]. In addition, the PINN model is a grid-free approach as no mesh is needed for solving the equations [16], which saves much effort for generating grids in the conventional PDE solvers. All the complexities of solving a physical problem are transferred into the optimization/training stage of the neural network. Consequently, a PINN is able to unify the formulation and generalize the procedure of solving physical problems governed by PDEs regardless of the structure of the equations, which is another attractive feature of the PINN. Figure 1 graphically describes the structure of the PINN approach [1], where the loss function of PINN contains a mismatch in the given data on the state variables or boundary condition (BC) and initial condition (IC), i.e., $\text{MSE}_{\{u,\text{BC,IC}\}} = N_u^{-1} \sum \|\mathbf{u}(x,t) - \mathbf{u}_\star\|$ with $\mathbf{u}_\star$ being the given data, combined with the residual of the PDE computed on a set of random points in the time-space domain, i.e., $\text{MSE}_R = N_R^{-1} \sum \|R(x,t)\|$. Then, the PINN can be trained by minimizing the total loss $\text{MSE} = \text{MSE}_{\{u,\text{BC,IC}\}} + \text{MSE}_R$. For solving forward problems, the first term represents a mismatch of the NN output $\mathbf{u}(x,t)$ from boundary and/or initial conditions, i.e., $\text{MSE}_{\text{BC,IC}}$. For solving inverse problems, the first term considers a mismatch of the NN output $\mathbf{u}(x,t)$ from additional data sets, i.e., MSE_u. In general, PINNs contain three steps to solve a physical problem involving PDEs:

Step 1. Define the PINN architecture.

Step 2. Define the loss function $\text{MSE} = \text{MSE}_{\{u,\text{BC,IC}\}} + \text{MSE}_R$.

Step 3. Train the PINN using an appropriate optimizer, i.e., Adam [7], AdaGrad [8], L-BFGS [1], etc.

Recently, PINNs and their variants have been successfully applied to solve both forward and inverse problems involving PDEs. Examples include the Navier-Stokes and the KdV equations [1], stochastic PDEs [17–20], and fractional PDEs [21].

In modeling problems involving long-time integration of PDEs, the large number of spatio-temporal degrees of freedom leads to a large size of data for training the PINN. This will require that PINNs solve long-time physical problems, which may be computationally prohibitive. The parallel-in-time (parareal) scheme, which decomposes the long-time domain and solves the subdomains in parallel, is widely used to accelerate the computations of the long-time PDE problems in conventional numerical methods [23–29]. Examples for the applications of the traditional parareal approaches include solving Navier-Stokes equations [30], modeling fluid-structure interactions [31], and kinetic systems [32]. However, the existing parareal methods may have difficulties in treating PDE problems when only partial observations on the boundary/initial conditions are available, wherein the PINN algorithm exploits both data and physical constraints and provides an efficient approach for dealing

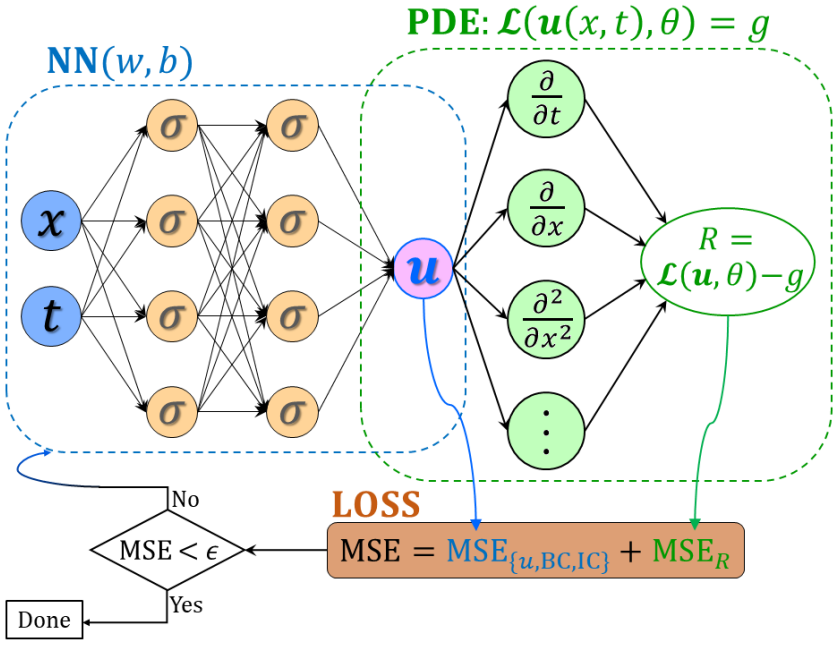

Figure 1: Schematic of a physics-informed neural network (PINN), where the loss function of PINN contains a mismatch in the given data on the state variables or boundary and initial conditions, i.e., $\text{MSE}_{\{u,\text{BC,IC}\}} = N_u^{-1} \sum \|\mathbf{u}(x,t) - \mathbf{u}_\star\|$, and the residual for the PDE on a set of random points in the time-space domain, i.e., $\text{MSE}_R = N_R^{-1} \sum \|R(x,t)\|$. The hyperparameters of PINN can be learned by minimizing the total loss $\text{MSE} = \text{MSE}_{\{u,\text{BC,IC}\}} + \text{MSE}_R$. σ represents the activation function. Specifically, the hyperbolic tangent function is employed as the activation function in the present study [1]. In addition, all the hyperparameters in the DNNs are initialized by the Xavier initialization method [22].

with these types of problems [1]. To this end, we propose a *parareal physics-informed neural network* (PPINN) to split one long-time problem into many independent short-time problems supervised by an inexpensive/fast coarse-grained (CG) solver, inspired by the existing parareal algorithms [23–25, 30, 32–34] and a more recent supervised parallel-in-time algorithm [35]. Because the computational cost of training a DNN quickly increases with the size of data set [36], this PPINN framework is able to maximize the benefit of high computational efficiency of training a neural network with small-data sets. More specifically, there is a two-fold benefit from training PINNs with small-data sets rather than working on a large-data set directly, i.e., training of individual PINNs with small-data is much faster, while training of fine PINNs can be readily parallelized on multiple GPU-CPU cores. Consequently, compared to the original PINN approach, the proposed PPINN approach may achieve a significant speed-up for solving long-time physical problems; this will be verified with benchmark tests for one-dimensional and two-dimensional nonlinear problems. This favorable speed-up will depend on a good supervisor that will be represented by a coarse-grained (CG) solver expected to provide reasonable accuracy.

The remainder of this paper is organized as follows. In Section 2 we describe the details of the PPINN framework and its implementation. In Section 3 we first present a pedagogical example to demonstrate accuracy and convergence for a one-dimensional time-dependent problem. Subsequently, we apply a PPINN to solve a two-dimensional nonlinear time-dependent problem, where we demonstrate the speed-up of the PPINN. Finally, we end the paper with a brief summary and discussion in Section 4.

2 Parareal PINN

2.1 Methodology

For a time-dependent problem involving long-time integration of PDEs for $t \in [0, T]$, instead of solving this problem directly in one single time domain, PPINN splits $[0, T]$ into N sub-domains with equal length $\Delta T = T/N$. Then, PPINN employs two propagators, i.e., a serial CG solver represented by $\mathcal{G}(\mathbf{u}_i^k)$ in Algorithm 1, and N fine PINNs computing in parallel represented by $\mathcal{F}(\mathbf{u}_i^k)$ in Algorithm 1. Here, $\mathbf{u}_i^k$ denotes the approximation to the exact solution at time t_i in the k-th iteration. Because the CG solver is serial in time and fine PINNs run in parallel, to have the optimal computational efficiency, we encode a simplified PDE (sPDE) into the CG solver as a prediction propagator while the true PDE is encoded in fine PINNs as the correction propagator. Using this prediction-correction strategy, we expect the PPINN solution to converge to the true solution after a few iterations.

The details for the PPINN approach are displayed in Algorithm 1 and Fig. 2, which are explained step by step in the following: Firstly, we simplify the PDE to be solved by the CG solver. For example, we can replace the nonlinear coefficient in the diffusion equation shown in Sec. 3.3 with a constant to remove the variable/multiscale coefficients. For instance, we can use a CG PINN as the fast CG solver but we can also explore standard fast finite difference solvers. Secondly, the CG PINN is employed to solve the sPDE serially for the entire time-domain to obtain an initial solution. Due to the fact that we can use less residual points as well as smaller neural networks to solve the sPDE rather than the original PDE, the computational cost in the CG PINN can be significantly reduced. Thirdly, we decompose the time-domain into N subdomains. We assume that $\mathbf{u}_i^k$ is known for $t_k \leq t_i \leq t_N$ (including $k = 0$, i.e., the initial iteration), which is employed as the initial condition to run the N fine PINNs in parallel. Once the fine solutions at all t_i are obtained, we can compute the discrepancy between the coarse and fine solution at t_i as shown in Step 3(b) in Algorithm 1. We then run the CG PINN serially to update the solution $\mathbf{u}$ for each interface between two adjacent subdomains, i.e., $\mathbf{u}_{i+1}^{k+1}$ (Prediction and Refinement in Algorithm 1). Interpolations using PINN/FDM are employed to map the coarse/fine solutions to the corresponding fine/coarse ones. Step 3 in Algorithm 1 is performed iteratively until the following criterion is satisfied

$$E = \frac{\sqrt{\sum_{i=0}^{N-1} \|\mathbf{u}_i^{k+1} - \mathbf{u}_i^k\|^2}}{\sqrt{\sum_{i=0}^{N-1} \|\mathbf{u}_i^{k+1}\|^2}} < E_{tol}, \tag{1}$$

where E_{tol} is a user-defined tolerance, which is set as 1% in the present study. In addition, E denotes the relative error of the predictions from the fine solvers at two adjacent iterations. Specifically, u^{k+1} represents the prediction from the fine solver at $(k+1)$th step, and u^k is the prediction from the fine solver at kth step. We note that the training of PINN terminates as the loss reaches a small value, which is of order 10^{-6} in our numerical examples. Consequently, the PINN solutions contain a stochasticity leading to a mean squared error in the order of 10^{-6}, which is a result of the stochastic nature of PINN, such as (1) the randomly initialized hyperparameters of the neural network, and (2) the optimizer, e.g, Adam [7]. It is necessary to define the convergence criterion by $E_{\text{tol}} = 1\%$ that is much larger than the error induced by the PINN training, so that the stochasticity in PINN solutions does not affect the convergence of the parareal iterations.

Algorithm 1: PPINN algorithm

1: Model reduction: Specify a simplified PDE for the fast CG solver to supervise the fine PINNs.
2: Initialization:
Solve the sPDE with the CG solver for the initial iteration: $\mathbf{u}_{i+1}^0 = \mathcal{G}(\mathbf{u}_i^0)$ for all $0 \le t_i \le t_N$.
3: Assume $\{\mathbf{u}_i^k\}$ is known for $t_k \le t_i \le t_N$ and $k \ge 0$:
Correction:
(a) Advance with fine PINNs in parallel: $\mathcal{F}(\mathbf{u}_i^k)$ for all $t_k \le t_i \le t_{N-1}$.
(b) Correction: $\delta_i^k = \mathcal{F}(\mathbf{u}_i^k) - \mathcal{G}(\mathbf{u}_i^k)$ for all $t_k \le t_i \le t_{N-1}$.
Prediction:
Advance with the fast CG solver in serial: $\mathcal{G}(\mathbf{u}_i^{k+1})$ for all $t_{k+1} \le t_i \le t_{N-1}$.
Refinement:
Combine the correction and prediction terms: $\mathbf{u}_{i+1}^{k+1} = \mathcal{G}(\mathbf{u}_i^{k+1}) + \mathcal{F}(\mathbf{u}_i^k) - \mathcal{G}(\mathbf{u}_i^k)$
4: Repeat *Step 3* to compute $\mathbf{u}_i^{k+2}$ for all $1 \le t_i \le t_N$ until a convergence criterion is met.

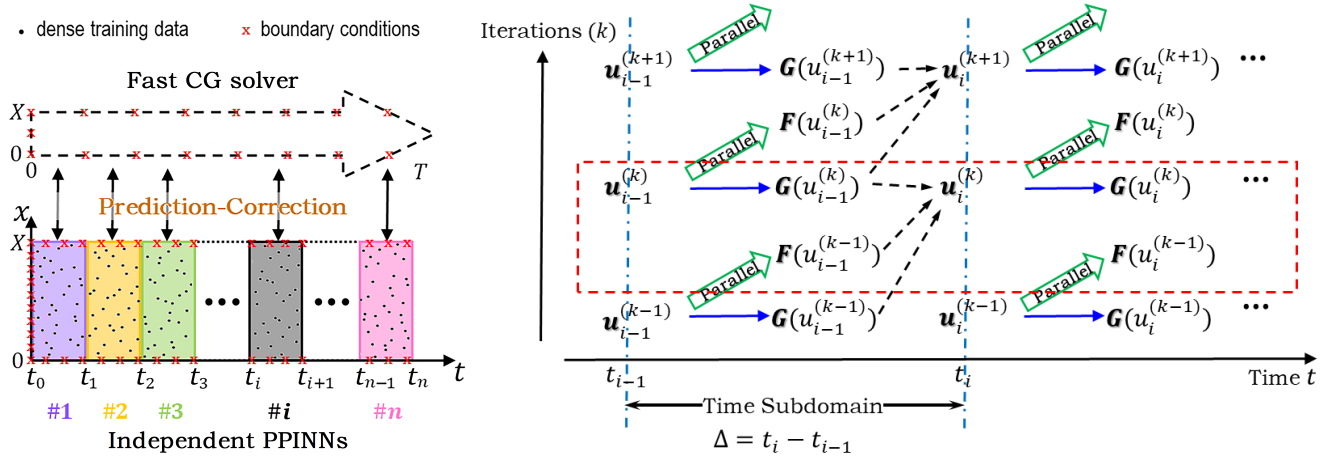

Figure 2: Overview of the parareal physics-informed neural network (PPINN) algorithm. Left: Schematic of the PPINN, where a long-time problem (PINN with full-sized data) is split into many independent short-time problems (PINN with small-sized data) guided by a fast coarse-grained (CG) solver. Right: A graphical representation of the parallel-in-time algorithm used in PPINN, in which a cheap serial CG solver is used to make an approximate prediction of the solution $\mathcal{G}(\mathbf{u}_i^k)$, while many fine PINNs are performed in parallel for getting $\mathcal{F}(\mathbf{u}_i^k)$ to correct the solution iteratively. Here, k represents the index of iteration, and i is the index of time subdomain.

2.2 Speed-up analysis

The walltime for the PPINN can be expressed as $T_{PPINN} = T_c^0 + \sum_{k=1}^{K}(T_c^k + T_f^k)$, where K is the total number of iterations, T_c^0 represents the walltime taken by the CG solver for the initialization, while T_c^k and T_f^k denote the walltimes taken by the coarse and fine propagators at k-th iteration, respectively. Let τ_c^k and τ_f^k be the walltimes used by CG solver and fine PINN for one subdomain at k-th iteration, T_c^k and T_f^k can be expressed as

$$T_c^k = N \cdot \tau_c^k, \quad T_f^k = \tau_f^k, \tag{2}$$

where N is the number of subdomains. Therefore, the walltime for the PPINN is

$$T_{PPINN} = T_c^0 + \sum_{k=1}^{K} \left(N \cdot \tau_c^k + \tau_f^k \right) = N \cdot \tau_c^0 + \sum_{k=1}^{K} \left(N \cdot \tau_c^k + \tau_f^k \right). \tag{3}$$

Furthermore, the walltime for the fine PINN to solve the same PDE in serial is expressed as $T_{PINN} = N \cdot T_f^1$. To this end, we can obtain the speed-up ratio of the PPINN as

$$S = \frac{T_{PINN}}{T_{PPINN}} = \frac{N \cdot T_f^1}{N \cdot \tau_c^0 + \sum_{k=1}^{K} \left(N \cdot \tau_c^k + \tau_f^k \right)}, \tag{4}$$

which can be rewritten as

$$S = \frac{N \cdot \tau_f^1}{N \cdot \tau_c^0 + \tau_f^1 + N \cdot K \cdot \tau_c^k + (K-1) \cdot \tau_f^k}. \tag{5}$$

For step $k = 1$, the hyperparameters in each fine PINN are set as random numbers using the Xavier initialization. However, the hyperparameters in the PINNs for step $k > 1$ are inherited from those at the previous step. Considering that the solution in each subdomain for two adjacent iterations ($k \geq 2$) does not change dramatically, the training process converges faster for $k \geq 2$ than that of $k = 1$ in each fine PINN, i.e., $\tau_f^k < \tau_f^1$. The training process of PPINN for $k \geq 2$ is similar as in transfer learning, which is a common technique to accelerate the training of DNNs [37,38]. Therefore, the lower bound for S can be expressed as

$$S_{min} = \frac{N \cdot \tau_f^1}{N \cdot \tau_c^0 + N \cdot K \cdot \tau_c^k + K \cdot \tau_f^1}. \tag{6}$$

This shows that S increases with N if $\tau_c^0 \ll \tau_f^1$, suggesting that the more subdomains we employ, the larger the speed-up ratio for the PPINN.

The finite-step convergence of the PPINN is guaranteed by the parareal framework [25, 32]. Assuming that the time domain is divided into N sub-domains, the parareal algorithm will definitely converge after N iterations in the worst case that the coarse-solver provides completely wrong predictions. As long as the coarse-solver can give reasonable predictions, the parareal solution will converge within N iterations. In our examples, the coarse-solvers provide valuable predictions and the tested cases converge in a few iterations. This convergence of course assumes that the optimization process of the DNN will not fail.

3 Results

We first show two simple examples for a deterministic and a stochastic ordinary differential equation (ODE) to explain the method in detail. We then present results for the Burgers equation and a two-dimensional diffusion-reaction equation.

3.1 Pedagogical examples

We first present two pedagogical examples, i.e., a deterministic and a stochastic ODE, to demonstrate the procedure of using PPINN to solve simple time-dependent problems; the main steps and key features of PPINN can thus be easily understood.

3.1.1 Deterministic ODE

The deterministic ODE considered here reads as

$$\frac{du}{dt} = a + \omega \cos(\omega t), \tag{7}$$

where the two parameters are $a = 1$ and $\omega = \pi/2$. Given an initial condition $u(0) = 0$, the exact solution of this ODE is $u(t) = t + \sin(\pi t/2)$. The length of time domain we are interested is $T = 10$.

In the PPINN approach, we decompose the time-domain $t \in [0, 10]$ into 10 subdomains, with length $\Delta t = 1$ for each subdomain. We use a simplified ODE (sODE) $du/dt = a$ with IC $u(0) = 0$ for the CG solver, and use the exact ODE $du/dt = a + \omega \cos(\omega t)$ with IC $u(0) = 0$ for the ten fine PINNs. In particular, we first use a CG PINN to act as the fast solver. Let $[N_I] + [N_H] \times D + [N_O]$ represent the architecture of a DNN, where N_I is the width of the input layer, N_H is the width of the hidden layer, D is the depth of the hidden layers and N_O is the width of the output layer. The CG PINN is constructed as a $[1] + [4] \times 2 + [1]$ DNN encoded with the sODE $du/dt = a$ and IC $u(0) = 0$, while each fine PINN is constructed as a $[1] + [16] \times 3 + [1]$ DNN encoded with the exact ODE $du/dt = a + \omega \cos(\omega t)$ and IC $u(0) = 0$.

In the first iteration, we train the CG PINN for 5,000 epochs using the Adam algorithm with a learning rate of $\alpha = 0.001$, and obtain a rough prediction of the solution $\mathcal{G}(u(t))|_{k=0}$, as shown in Fig. 3(a1). Let $t_i = i$ for $i = 0, 1, \cdots, 9$ be the starting time of each fine PINN in the chunk i+1. To compute the loss function of the CG PINN, 20 residual points uniformly distributed in each subdomain $[t_i, t_{i+1}]$ are used to compute the residual MSE_R. Subsequently, with the CG PINN solutions, the IC of each fine PINN is given by the predicted values $\mathcal{G}(u(t))|_{k=0}$ at $t = t_i$ except the first subdomain. Similarly

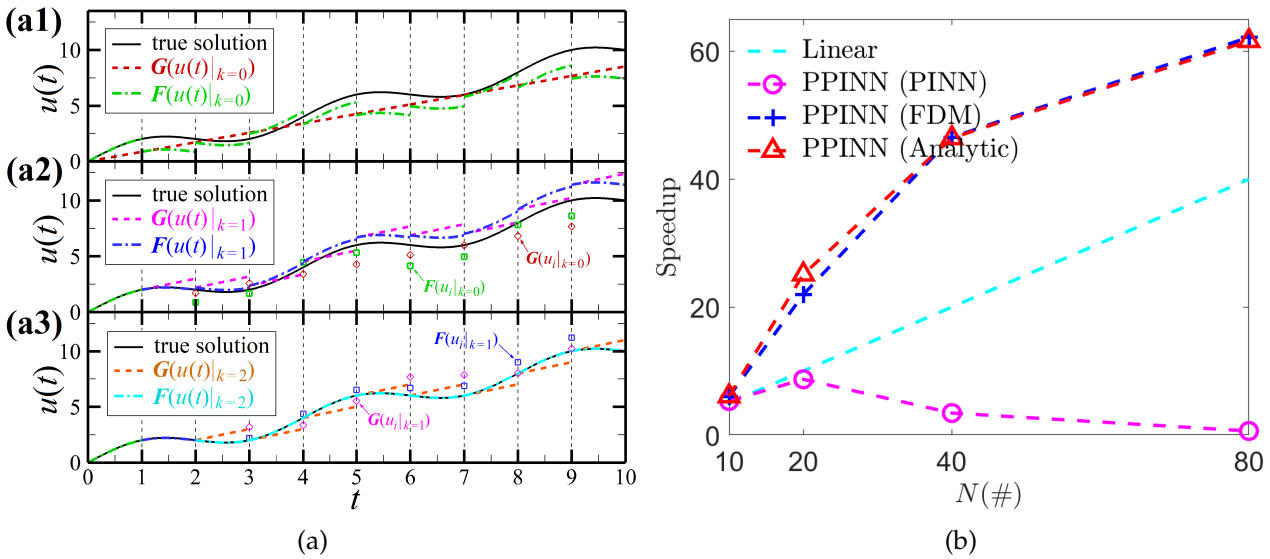

Figure 3: A pedagogical example to illustrate the procedure of using PPINN to solve a deterministic ODE. Here a PINN is also employed for the CG solver. (a) The solutions of CG PINN $\mathcal{G}(u(t)|_k)$ and fine PINNs $\mathcal{F}(u(t)|_k)$ in different iterations are plotted for (a1) $k = 0$, (a2) $k = 1$ and (a3) $k = 2$ to solve the ODE $du/dt = 1 + \pi/2 \cdot \cos(\pi t/2)$. (b) Speed-ups of the PPINN for different numbers of time-intervals with different coarse solvers, i.e. PINN (magenta dashed line with circle), FDM (blue dashed line with plus), and analytic solution (red dashed line with triangle). N denotes the number of the subdomains. The linear speed-up ratio is calculated as N/K. Cyan dashed line: $K = 2$.

for training the CG PINN, we train ten individual fine PINNs in parallel using the Adam algorithm with a learning rate of $\alpha = 0.001$. We accept the PINN solution if the loss function of each fine PINN $loss = \text{MSE}_R + \text{MSE}_{\text{IC}} < 10^{-6}$, wherein 101 residual points uniformly distributed in $[t_i, t_{i+1}]$ are used to compute the residual MSE_R. We can observe in Fig. 3(a1) that the fine PINN solutions correctly capture the shape of the exact solution, but their magnitudes significantly deviate from the exact solution because of the poor quality of the IC given by $\mathcal{G}(u(t))|_{k=0}$.

The deviation of PPINN solution from the exact solution is quantified by the l_2 relative error norm

$$l_2 = \frac{\sum_p (u_p^k - \hat{u}_p)^2}{\sum_p \hat{u}_p^2}, \tag{8}$$

where p denotes the index of residual points. The PPINN solution after the first iteration presents obvious deviations, as shown in Fig. 3(a1). In the second iteration, given the solutions of $\mathcal{G}(u_i)|_{k=0}$ and $\mathcal{F}(u_i)|_{k=0}$, we set the IC for CG PINN as G. Then, we train the CG PINN again for 5,000 epochs using a learning rate $\alpha = 0.001$ and obtain the solutions $\mathcal{G}(u(t))|_{k=1}$ for the chunks 2 to 10, as shown in Fig. 3(a2). The IC for fine PINN of the second chunk is given by $\tilde{u}(t = t_1) = F(u_1)_{k=1}$, and the ICs of fine PINNs of the chunks 3 to 10 are given by a combination of $\mathcal{G}(u(t))|_{k=0}$, $\mathcal{F}(u(t))|_{k=0}$ and $\mathcal{G}(u(t))|_{k=1}$ at $t = t_{i=2,3\ldots,9}$, i.e., $\tilde{u}(t = t_i) = \mathcal{G}(u_i)|_{k=1} - [\mathcal{F}(u_i)|_{k=0} - \mathcal{G}(u_i)|_{k=0}]$. With these ICs, we train the nine individual fine PINNs (chunks 2 to 10) in parallel using the Adam algorithm with a learning rate of $\alpha = 0.001$ until the loss function for each fine PINN drops below 10^{-6}. We find that the fine PINNs in the second iteration converge faster than in the first iteration, with an average epochs of 4836 (varies from 4500 to 5100) compared to 11524 (varies from 10000 to 13000) in the first iteration. The accelerated training process benefits from the results of training performed in the previous iteration. The PPINN solutions for each chunk after the second iteration are presented in Fig. 3(a2), showing a significant improvement of the solution with a relative error $l_2 = 9.97\%$. Using the same method, we perform a third iteration, and the PPINN solutions converge to the exact solution with a relative error $l_2 = 0.08\%$, as shown in Fig. 3(a3).

We proceed to investigate the computational efficiency of the PPINN. As mentioned in Sec. 2.2, we may obtain a speed-up ratio, which grows linearly with the number of subdomains if the coarse solver is efficient enough. We first test the speed-ups for the PPINN using a PINN as the coarse solver. Here we test four different subdomain decompositions, i.e. 10, 20, 40, and 80. For the coarse solver, we assign one PINN (CG PINN, [1] + [4] $\times$ 2 + [1]) for each subdomain. We keep the total number of residual points fixed, which are evenly distributed into each subdomain. Specifically, 1,000 randomly sampled residual points are drawn in the entire domain. The convergence criterion for all the four cases is given by Eq. (1) to ensure that the relative error between two solutions in adjacent iterations is less than 1%. We run all the CG PINNs serially using one CPU (Intel Xeon E5-2670). For the fine solver, we again employ one PINN (fine PINN) for each subdomain to solve Eq. (7), and each subdomain is assigned to one CPU (Intel Xeon E5-2670). The total number for the residual points is 400,000, which are randomly sampled in the entire time domain and will be uniformly distributed in each subdomain. Meanwhile, the architecture for the fine PINN in each subdomain is [1]+[20]$\times$2+[1] for the first two cases (i.e., 10 and 20 subdomains), which is then set as [1] + [10] $\times$ 2 + [1] for the last two cases (i.e., 40 and 80 subdomains). The speed-ups are displayed in Fig. 3(b) (magenta dashed line), where we observe that the speed-up first increases with N as $N \leq 20$, then it decreases with the increasing N. We further look into the details of the computational time for this case. As shown in Table 1, we found that the computational time taken by the CG PINNs increases with the number of the subdomains. In particular, more than 90 % of the computational time is taken by the coarse solver as $N \geq 40$, suggesting the inefficiency of the CG PINN. To obtain satisfactory speed-ups using

the PPINN, a more efficient coarse solver should be utilized.

Subdomains (#)	Iterations (#)	l_2	$\mathcal{N}_{CG}$ (#)	$T_{CG}(s)$	T_{total} (s)	S
1	-	1.55×10^{-4}	-	-	1,793.0	-
10	2	3.16×10^{-5}	100	45.3	337.7	5.3
20	2	3.74×10^{-5}	50	130.5	206.1	8.7
40	2	3.17×10^{-5}	25	469.1	523.2	3.4
80	2	3.42×10^{-5}	12	2,857.3	2,891.6	0.62

Table 1: Speed-ups for the PPINN using a PINN as coarse solver to solve Eq. (7). $\mathcal{N}_{CG}$ is the number of residual points used for the CG PINN in each subdomain, T_{CG} represents the computational time taken by the coarse solver, l_2 is the error defined in Eq. (8) after convergence, and T_{total} denotes the total computational time taken by the PPINN. Note that (1) the saturation of error at the 10^{-5} level is related to the optimization error, and (2) the number of residual points adopted for the case with 10 subdomains here is different from that used in Fig. 3(a), leading to different iterations in these two cases.

Considering that the simplified ODE can be solved analytically, we can thus directly use the analytic solution for the coarse solver which has no cost. In addition, all parameters in the fine PINN are kept the same as the previously used ones. For validation, we present the l_2 relative error between the predicted and analytic solutions after convergence in Table 2, which confirms the accuracy of the PPINN. In addition, the speed-ups for the four cases are displayed in Fig. 3(b) (red dashed line with triangle). It is interesting to find that the PPINN can achieve a superlinear speed-up here. We further present the computational time at each iteration in Table 2. We see that the computational time taken by the coarse solver is now negligible compared to the total computational time. Since the fine PINN converges faster after the first iteration, we can thus obtain a superlinear speed-up for the PPINN.

Instead of the analytic solution, we can also use other efficient numerical methods to serve as the coarse solver. For demonstration purpose, we then present the results using the finite difference method (FDM) as the coarse solver. The entire domain is discretized into 1,000 uniform elements, which are then uniformly distributed to each subdomain. Similarly, we also present the l_2 relative errors between the predicted and exact solutions in Table 2, which again confirms the accuracy of the PPINN. In addition, we can also obtain a superlinear speed-up which is quite similar to the case using the analytic solution due to the efficiency of the FDM (Fig. 3(b) and Table 2).

3.1.2 Stochastic ODE

In addition to the deterministic ODE, we can also apply this idea to solve stochastic ODEs. Here, we consider the following stochastic ODE

$$\frac{du}{dt} = \beta\left[-u + \beta \sin(\frac{\pi}{2}t) + \frac{\pi}{2}\cos(\frac{\pi}{2}t)\right], t \in [0, T], \tag{9}$$

where $T = 10$, $\beta = \beta_0 + \epsilon$, $\beta_0 = 0.1$, and ϵ is drawn from a normal distribution with zero mean and 0.05 standard deviation. In addition, the initial condition for Eq. (9) is $u(0) = 1$.

In the present PPINN, we employ a deterministic ODE for the coarse solver as

$$\frac{du}{dt} = -\beta_0 u, t \in [t_0, T]. \tag{10}$$

	Subdomains (#)	Iterations (#)	l_2	$\mathcal{N}_{CG}$ (#)	$T_{CG}(s)$	T_{total} (s)	S
PPINN (Analytic)	1	-	1.55×10^{-4}	-	-	1,793.0	-
	10	2	9.43×10^{-6}	-	< 0.05	295.8	6.1
	20	2	4.56×10^{-6}	-	< 0.05	71.4	25.1
	40	2	2.77×10^{-6}	-	< 0.05	38.6	46.4
	80	2	7.40×10^{-6}	-	< 0.05	29.0	61.6
PPINN (FDM)	1	-	1.55×10^{-4}	-	-	1,793.0	-
	10	2	1.19×10^{-5}	100	< 0.05	298.7	6.0
	20	2	5.50×10^{-6}	50	< 0.05	81.6	22.0
	40	2	5.49×10^{-6}	40	< 0.05	38.5	46.5
	80	2	7.42×10^{-6}	12	< 0.05	28.8	62.2

Table 2: Walltimes for using the PPINN with different coarse solvers to solve Eq. (7). $\mathcal{N}_{CG}$ is the number of elements used for the FDM in each subdomain, T_{CG} represents the computational time taken by the coarse solver, and T_{total} denotes the total computational time taken by the PPINN.

Given the initial condition $u(t_0) = u_0$, we can obtain the analytic solution for Eq. (10) as $u = u_0 \exp(-\beta_0(t - t_0))$. For the fine solver, we draw 100 samples for the β using the quasi-Monte Carlo method, which are then solved by the fine PINNs. Similarly, we utilize three different methods for the coarse solver, i.e. the PINN, FDM, and analytic solution.

For validation purposes, we first decompose the time-domain into 10 uniform subdomains. For the FDM, we discretize the whole domain into 1,000 uniform elements. For the fine PINNs, we employ 400,000 randomly sampled residual points for the entire time domain, which are uniformly distributed to all the subdomains. We employ one fine PINN for each subdomain to solve the ODE, which has an architecture of $[1] + [20] \times 2 + [1]$. Finally, the simplified ODE in each subdomain for the coarse solver is solved serially, while the exact ODE in each subdomain for the fine solver is solved in parallel. We illustrate the comparison between the predicted and exact solutions at two representative β, i.e. $\beta = 0.108$ and 0.090 in Fig. 4(a). We see that the predicted solutions converge to the exact ones after two iterations, which confirms the effectiveness of the PPINN for solving stochastic ODEs. The solutions from the PPINN with the other two different coarse solvers (i.e., the PINN and analytic solution) also agree well with the reference solutions, but they are not presented here. Furthermore, we also present the computational efficiency for the PPINN using four different numbers of subdomains, i.e. 10, 20, 40 and 80 in Table 3 as well as Fig. 4(b). Note that (1) the number of iterations differs for different β, which is thus not presented in Table 3, and (2) all fine solvers here converge in 2 to 4 iterations, we therefore present the linear speed-up ratios as $K = 2$ and 4 for reference in Fig. 4(b). It is clear that the PPINN can still achieve a superlinear speed-up if we use an efficient coarse solver such as FDM or analytic solution. The speed-up for the PPINN with the PINN as coarse solver is similar to the results in Sec. 3.1.1, i.e., the speed-up ratio first slightly increases with the number of subdomains as $N \leq 20$, then it decreases with the increasing N. Finally, the speed-ups for the PPINN with FDM coarse solver are almost the same as the PPINN with analytic solution coarse solver, which are similar as the results in Sec. 3.1.1 and will not be discussed in detail here.

In summary, the PPINN can work for both deterministic and stochastic ODEs. In addition, using the PINN as the coarse solver can guarantee the accuracy for solving the ODE, but the speed-up may decrease with the number of subdomains due to the inefficiency of the PINN. We can achieve both high accuracy and good speed-up if we select more efficient coarse solvers, such as an analytic solution, a finite difference method, and so on.

	Subdomains (#)	Iterations (#)	$\mathcal{N}_{CG}$ (#)	$T_{CG}(s)$	T_{total} (s)	S
PPINN (PINN)	1	-	-	-	14,239.0	-
	10	-	100	64.6	1,418.6	10.0
	20	-	50	478.3	1,195.1	11.9
	40	-	40	2165.1	2,466.8	5.8
	80	-	12	21,409.8	21,600.0	0.66
PPINN (Analytic)	1	-	-	-	14,239.0	-
	10	-	-	< 0.05	1,356.0	10.5
	20	-	-	< 0.05	726.5	19.6
	40	-	-	< 0.05	291.9	48.8
	80	-	-	< 0.05	181.9	78.3
PPINN (FDM)	1	-	-	-	14,239.0	-
	10	-	100	< 0.05	1,361.6	10.5
	20	-	50	< 0.05	659.6	21.6
	40	-	40	< 0.05	296.5	48.0
	80	-	12	< 0.05	186.0	76.6

Table 3: Walltimes for using the PPINN with different coarse solvers to solve Eq. (10). $\mathcal{N}_{CG}$ is the number of residual points used for the CG PINN in each subdomain, and represents the number of elements in each subdomain as the FDM is used, T_{CG} represents the computational time taken by the coarse solver, and T_{total} denotes the total computational time taken by the PPINN.

3.2 Burgers equation

We now consider the viscous Burgers equation

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \tag{11}$$

which is a mathematical model for viscous flow, or gas dynamics, with u denoting the speed of the fluid, ν the kinematic viscosity, x the spatial coordinate and t the time. Given an initial condition $u(x,0) = -\sin(\pi x)$ in a domain $x \in [-1,1]$, and the boundary condition $u(\pm 1, t) = 0$ for $t \in [0,1]$, the PDE we would like to solve is Eq. (11) with a viscosity of $\nu = 0.03/\pi$.

In the PPINN, the temporal domain $t \in [0,1]$ is decomposed into 10 uniform subdomains. Each subdomain has a time length $\Delta t = 0.1$. The simplified PDE for the CG PINN is also a Burgers equation, which uses the same initial and boundary conditions but with a larger viscosity $\nu_c = 0.05/\pi$. It is well known that the Burgers equation with a small viscosity will develop steep gradient in its solution as time evolves. The increased viscosity will lead to a smoother solution, which can be captured using much less computational cost. Here, we use the same NN for the CG and fine PINNs for each subdomain, i.e., 3 hidden layers with 20 neurons per layer. The learning rates are also set to be the same, i.e., 10^{-3}. Instead of using one optimizer in the last case, here we use two different optimizations, i.e., we first train the PINNs using the Adam optimizer (first-order convergence rate) until the loss is less than 10^{-3}, then we proceed to employ the L-BFGS-B method to further train the NNs. The L-BFGS is a quasi-Newtonian approach which has second-order convergence rate and can enhance the convergence of the training [1].

For the CG PINN in each subdomain, we use 300 randomly sampled residual points to compute the MSE_R, while 1,500 random residual points are employed in the fine PINN for each subdomain. In addition, 100 uniformly distributed points are employed to compute the MSE_{IC} is in each subdomain,

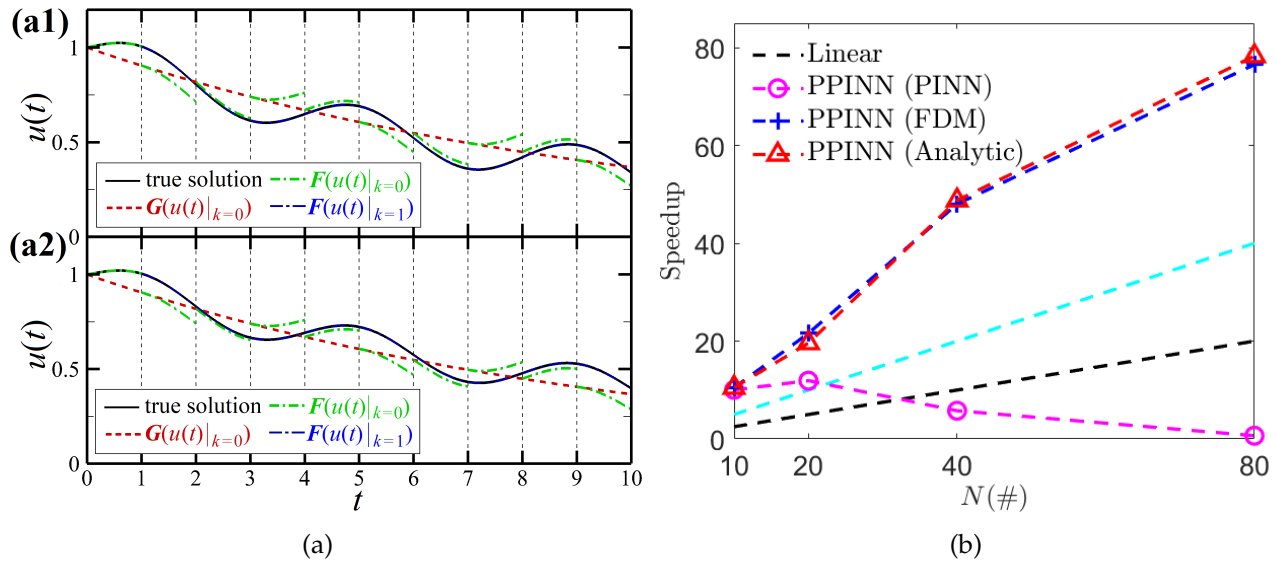

Figure 4: A pedagogical example for using the PPINN to solve stochastic ODE. (a) The solutions of CG solver $\mathcal{G}(u(t)|_k)$ and fine PINNs $\mathcal{F}(u(t)|_k)$ in different iterations are plotted for (a1) $\beta = 0.108$, and (a2) $\beta = 0.090$. The reference solution is $u(t) = \exp(-\beta t) + \beta \sin(\frac{\pi}{2}t)$ for each β. (b) Speed-ups of the PPINN for different numbers of time-intervals with different coarse solvers, i.e. the PINN (magenta dashed line with circle), the FDM (blue dashed line with plus), and analytic solution (red dashed line with triangle). The linear speed-up ratio is calculated as N/K. Black dashed line: $K = 4$, Cyan dashed line: $K = 2$. For the CG PINN, we use 1,000 randomly sampled residual points in the whole time domain, which are uniformly distributed to the subdomains. The architecture of the PINN for each subdomain is the same, i.e. $[1] + [10] \times 2 + [1]$. For the fine PINN, the architecture is $[1] + [20] \times 2 + [1]$ for the cases with 10 and 20 subdomains, and it is $[1] + [10] \times 2 + [1]$ for the cases with 40 and 80 subdomains.

and 10 randomly sampled points are used to compute the MSE_{BC} in both the CG and fine PINNs. For this particular case, it takes only 2 iterations to meet the convergence criterion, i.e., $E_{tol} = 1\%$. The distributions of the u at each iteration are plotted in Fig. 5. As shown in Fig. 5(a), the solution from the fine PINNs ($\mathcal{F}(u|_{k=0})$) is different from the exact one, but the discrepancy is observed to be small. It is also observed that the solution from the CG PINNs is smoother than that from the fine PINNs especially for solutions at large times, e.g., $t = 0.9$. In addition, the velocity at the interface between two adjacent subdomains is not continuous due to the inaccurate initial condition for each subdomain at the first iteration. At the second iteration (Fig. 5(b)), the solution from the Refinement step i.e. $u|_{k=2}$ shows little difference from the exact one, which confirms the effectiveness of the PPINN. Moreover, the discontinuity between two adjacent subdomains is significantly reduced. Finally, it is also interesting to find that the number of the training steps for each subdomain at the first iteration is from ten to hundred thousand, but they decrease to a few hundred at the second iteration. Similar results are also reported and analyzed in Sec. 3.1, which will not be presented here again.

To test the flexibility of the PPINN, we further employ two much larger viscosities in the coarse solver, i.e. $\nu_c = 0.09/\pi$ and $0.12/\pi$, which are 3× and 4× the exact viscosity, respectively. All the parameters (e.g., the architecture of the PINN, the number of residual points, etc.) in these two cases are kept the same as the case with $\nu_c = 0.05/\pi$. As shown in Table 4, it is interesting to find that the computational errors for these three cases are comparable, but the number of iterations increases with the viscosity employed in the coarse solver. Hence, in order to obtain an optimum strategy in

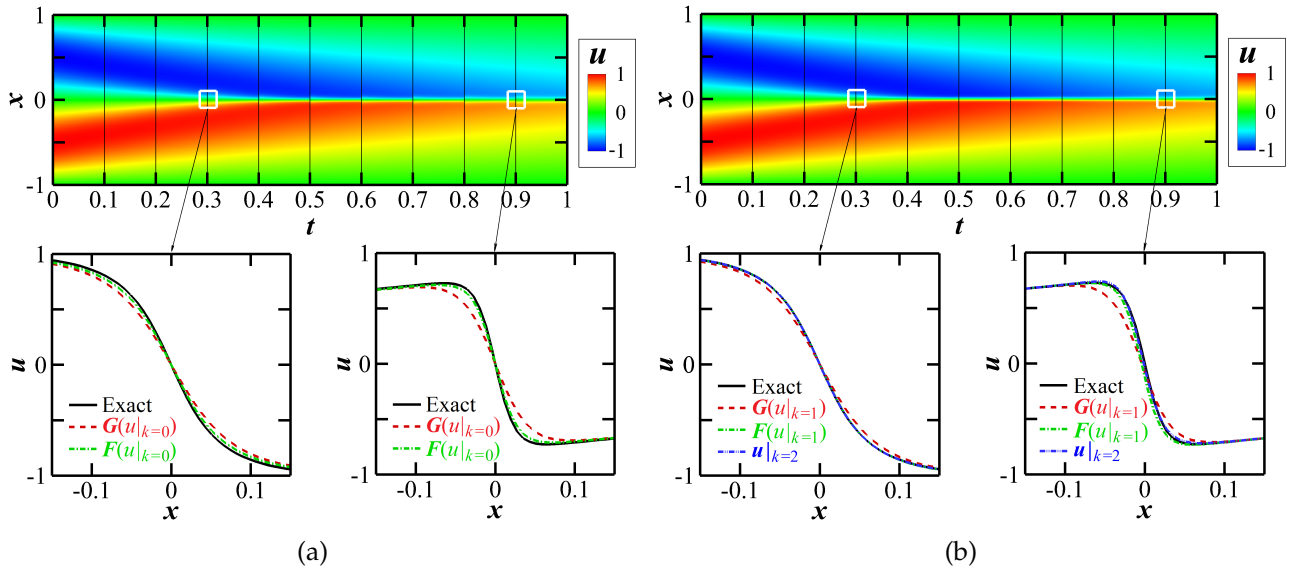

Figure 5: Example of using the PPINN for solving the Burgers equation. $\mathcal{G}(u(t)|_k)$ represents the rough prediction generated by the CG PINN in the $(k+1)$-th iteration, while $\mathcal{F}(u(t)|_k)$ is the solution corrected in parallel by the fine PINNs. (a) Predictions after the first iteration ($k = 0$) at $t = 0.3$ and 0.9), (b) Predictions after the second iteration ($k = 1$) at $t = 0.3$ and 0.9.

selecting the CG solver we have to consider the trade-off between the accuracy that the CG solver can obtain and the number of iterations required for convergence of the PPINN.

ν_c	$0.05/\pi$	$0.09/\pi$	$0.12/\pi$
Iterations (#)	2	3	4
l_2 (%)	0.13	0.19	0.22

Table 4: The PPINN for solving the Burgers equation with different viscosities in the coarse solver. ν_c represents the viscosity employed in the coarse solver.

3.3 Diffusion-reaction equation

We consider the following two-dimensional diffusion-reaction equation

$$\partial_t C = \nabla \cdot (D\nabla C) + 2A \sin(\frac{\pi x}{l}) \sin(\frac{\pi y}{l}),\ x,\ y \in [0, l],\ t \in [0, T], \tag{12}$$

$$C(x, y, t = 0) = 0, \tag{13}$$

$$C(x = 0, y = 0, t) = C(x = 0, y = l, t) = C(x = l, y = 0, t) = C(x = l, y = l, t) = 0, \tag{14}$$

where $l = 1$ is the length of the computational domain, C is the solute concentration, D is the diffusion coefficient, and A is a constant. Here D depends on C as follows:

$$D = D_0 \exp(RC), \tag{15}$$

where $D_0 = 1 \times 10^{-2}$.

We first set $T = 1$, $A = 0.5511$, and $R = 1$ to validate the proposed algorithm for the two-dimensional case. In the PPINN, we use the PINNs for both the coarse and fine solver. The time domain is divided into 10 uniform subdomains with $\Delta t = 0.1$ for each subdomain. For the coarse solver, the following simplified PDE $\partial_t C = D_0 \nabla^2 C + 2A \sin(\pi x/l) \sin(\pi y/l)$ is solved with the same initial and boundary conditions described in Eq. (12). The diffusion coefficient for the coarse solver is about 1.7 times smaller than the maximum D in the fine solver. The architecture of the NNs employed in both the coarse and fine solvers for each subdomain is kept the same, i.e., 2 hidden layers with 20 neurons per layer. The employed learning rate as well as the optimization method are the same as those applied in Sec. 3.2.

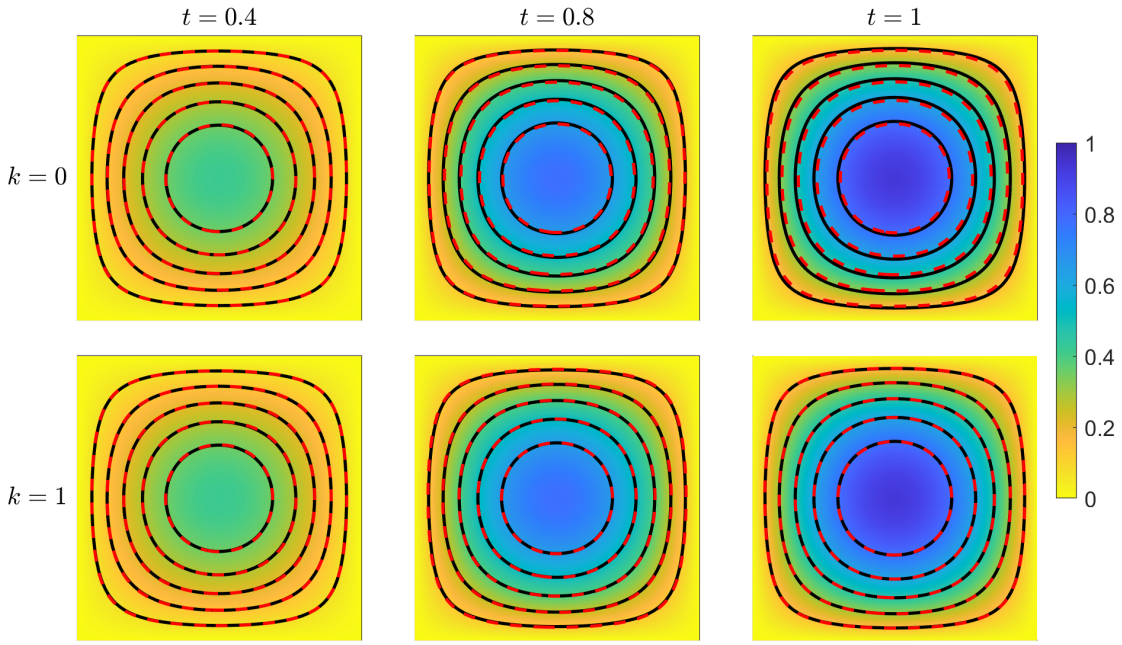

Figure 6: Example of using PPINN to solve the nonlinear diffusion-reaction equation. First row ($k = 0$): First iteration, Second row ($k = 1$): Second iteration. Black solid line: Reference solution, which is obtained from the lattice Boltzmann method using a uniform grid of $x \times y \times t = 200 \times 200 \times 100$ [39]. Red dashed line: Solution from the PPINN.

We employ 2,000 random residual points to compute the MSE_R in each subdomain for the coarse solver. We also employ 1,000 random points to compute the MSE_{BC} for each boundary, and 2,000 random points to compute the MSE_{IC}. For each fine PINN, we use 5,000 random residual points for MSE_R, while the numbers of training points for the boundary and initial conditions are kept the same as those used in the coarse solver. It takes two iterations to meet the convergence criterion, i.e. $E_{tol} = 1\%$. The PPINN solutions as well as the distribution of the diffusion coefficient for each iteration at three representative times (i.e., $t = 0.4$, 0.8, and 1) are displayed in Figs. 6 and 7, respectively. We see that the solution at $t = 0.4$ agrees well with the reference solution after the first iteration, while the discrepancy between the PPINN and reference solutions increases with the time, which can be observed for $t = 0.8$ and 1 in the first row of Fig. 6. This is reasonable because the error of the initial condition for each subdomain will increase with time. In addition, the solutions at the three representative times agree quite well with the reference solutions after the second iteration, as demonstrated in the second row in Fig. 6. All the above results again confirm the accuracy of the

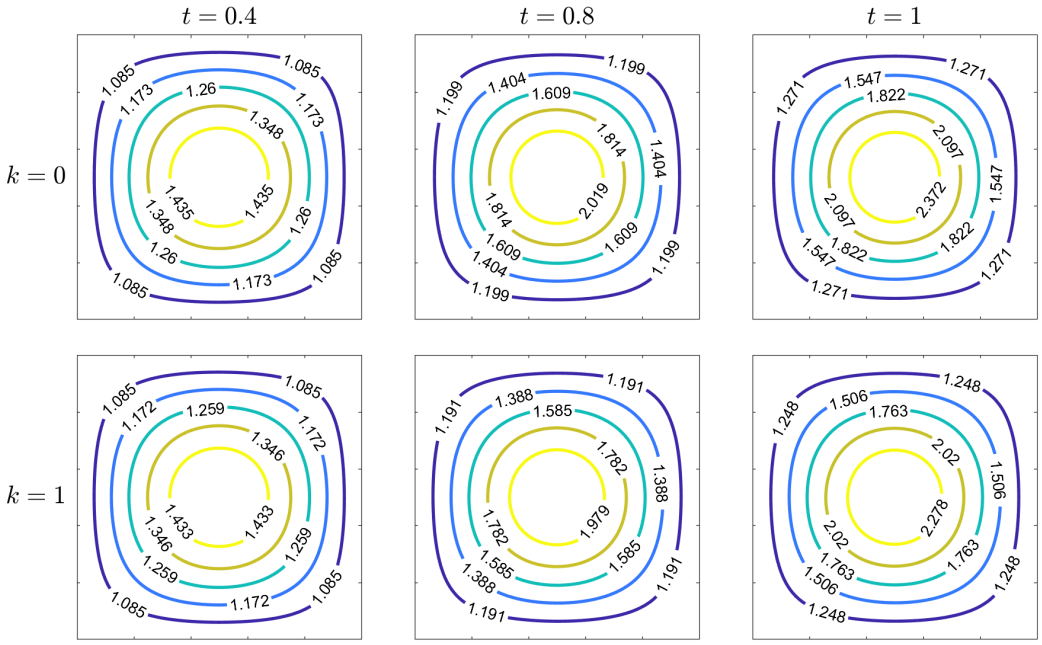

Figure 7: Contours of the normalized diffusion coefficient field (D/D_0). First row ($k = 0$): First iteration, Second row ($k = 1$): Second iteration.

present approach.

Subdomains (#)	NNs	MSE_R	MSE_{BC}	MSE_{IC}	Optimizer	Learning rate
1	$[20] \times 3$	200,000	$10,000 \times 4$	2,000	Adam, L-BFGS	10^{-3}
10	$[20] \times 2$	20,000	$1,000 \times 4$	2,000	Adam, L-BFGS	10^{-3}
20	$[16] \times 2$	10,000	500×4	2,000	Adam, L-BFGS	10^{-3}
40	$[16] \times 2$	5,000	250×4	2,000	Adam, L-BFGS	10^{-3}
50	$[16] \times 2$	400	200×4	2,000	Adam, L-BFGS	10^{-3}

Table 5: Parameters used in the PPINN for modeling the two-dimensional diffusion-reaction system. The NNs are first trained using the Adam optimizer with a learning rate 10^{-3} until the loss is less than 10^{-3}, then we utilize the L-BFGS-B to further train the NNs [1].

We further investigate the speed-up of the PPINN using the PINN as coarse solver. Here $T = 10$, $A = 0.1146$, and $R = 0.5$, and we use five different subdomains, i.e., 1, 10, 20, 40, and 50. The parallel code is run on multiple CPUs (Intel Xeon E5-2670). The total number of the residual points in the coarse solver is 20,000, which is uniformly divided into N subdomains. The number of training points for the boundary condition on each boundary is 10,000, which is also uniformly assigned to each subdomain. In addition, 2,000 randomly sampled points are employed for the initial condition. The architecture and other parameters (e.g., learning rate, optimizer, etc) for the CG PINN in each subdomain are the same as the fine PINN, which are displayed in Table 5. The speed-up ratios for the representative cases are shown in Table 6. We notice that the speed-up does not increase monotonically with the number of subdomains. On the contrary, the speed-up decreases with the number of subdomains. This result suggests that the cost for the CG PINN is not only related to the number of training data but is also affected by the number of hyperparameters. The

	Subdomains (#)	Iterations (#)	l_2 (%)	T_{total} (s)	S
PPINN (PINN)	1	-		27,627	-
	10	2	0.40	6,230	4.3
	20	3	0.70	7,620	3.6
	40	3	0.54	20,748	1.3
	50	3	0.62	> 27,627	-
PPINN (FDM)	1	-		27,627	-
	10	2	0.20	5,421	5.1
	20	2	0.48	2,472	11.2
	40	3	0.13	1,535	18.0
	50	3	0.32	1,620	17.1

Table 6: Speed-ups for using the PPINN with different coarse solvers to model the nonlinear diffusion-reaction system. T_{total} denotes the total computational time taken by the PPINN, and S is the speed-up ratio. Note that results from the lattice Boltzmann simulation with a grid size $x \times y \times t = 200 \times 200 \times 100$ serve as reference solution for the computation of l_2 since no exact solution is available here.

total hyperparameters in the CG PINNs increases with the number of subdomains, leading to the walltime for the PPINN to increase with the number of subdomains.

To validate the above hypothesis, we replace the PINN with the finite difference method [40] (FDM, grid size: $x \times y = 20 \times 20$, time step $\delta_t = 0.05$) for the coarse solver, which is much more efficient than the PINN. The walltime for the FDM in each subdomain is negligible compared to the fine PINN. As shown in Table 6, the speed-ups are now proportional to the number of subdomains as expected.

4 Summary and Discussion

Our overarching goal is to develop *data-driven* simulation capabilities that go beyond the traditional data assimilation methods by taking advantage of the recent advances in deep learning methods. Physics-informed neural networks (PINNs) [1] play exactly that role but they become computationally intractable for very long-time integration of time-dependent PDEs. Here, we address this issue for the first time by proposing a parallel-in-time PINN (PPINN). In this approach, two different PDEs are solved by the coarse-grained (CG) solver and the fine PINN, respectively. In particular, the CG solver can be any efficient numerical method since the CG solver solves a surrogate PDE, which can be viewed as either a simplified form of the exact PDE or it could be a reduced-order model or any other surrogate model, including an offline pre-trained PINN although this was not pursued here. The solutions of the CG solver are then serving as initial conditions for the fine PINN in each subdomain. The fine PINNs can then run in parallel independently based on the provided initial conditions, hence significantly reducing their training cost. In addition, it is worth mentioning that the walltime for the PINN in each subdomain at k-th ($k > 1$) iteration is negligible compared to the first iteration for the fine PINN, which leads to a superlinear speed-up of PPINNs assuming an efficient CG solver is employed.

The convergence of employing a PPINN algorithm is first validated for deterministic and stochastic ODE problems. In both cases, we employ a simplified ODE in the coarse solver to supervise

the fine PINNs. The results demonstrate that the PPINN can converge in a few iterations as we decompose the time-domain into 10 uniform subdomains. Furthermore, superlinear speed-ups are achieved for both cases. Two PDE problems, i.e., the one-dimensional Burgers equation and the two-dimensional diffusion-reaction system with nonlinear diffusion coefficient, are further tested to validate the present algorithm. For the 1D case, the simplified PDE is also a Burgers equation but with an increased viscosity, which yields much smoother solution and thus can be solved more efficiently. In the diffusion-reaction system, we use a constant diffusion coefficient instead of the nonlinear one in the CG solver, which is also much easier to solve compared to the exact PDE. The results showed that the fine PINNs can converge in only a few iterations based on the initial conditions provided by the CG solver in both cases. Finally, we also demonstrate that a superlinear speed-up can be obtained for the two-dimensional case if the CG solver is efficient.

We would like to point out another advantage of the present PPINN. In general, GPUs have very good computational efficiency in training DNNs with big-data sets. However, when one works on training a DNN with small-data sets, CPUs may have comparable performance to GPUs. The reason is that we need to transfer the dataset from the CPU memory to the GPU, and then transfer it back to the CPU memory, which can be even more time consuming than the computational process for small dataset [41]. For example, given a DNN with architecture of [2] + [20] $*$ 3 + [1], the wall time of training the PINN (1D Burgers equation) for 10,000 steps is about 100 seconds on an Intel Xeon CPU (E5-2643) and 172 seconds on a NVIDIA GPU (GTX Titan XP) for a training data set consisting of 5,000 points. Because the proposed PPINN approach is able to break a big-data set into multiple small-data sets for training, it enables us to train a PINN with big-data sets on CPU resources if one does not have access to sufficient GPU resources.

It is worth mentioning that the method using restriction/lifting type operators is generally used to map the coarse/fine solution to the corresponding fine/coarse states [42,43]. When the solutions in both coarse and fine solvers do not change dramatically in space, the integration and interpolation in space can provide results of comparable accuracy. In the numerical examples, we employed interpolations using PINN to map the coarse/fine solutions to the corresponding fine/coarse ones. However, the approach to separate/integrate out the fine-scale solutions for the coarse solver [35,42,43] could be a more general method to connect micro and macro solvers in multi-scale parareal problems.

The concept of PPINN could be readily extended to domain decomposition of physical problems with large spatial databases by drawing an analog of multigrid/multiresolution methods [44], where a cheap coarse-grained (CG) PINN should be constructed and be used to supervise and connect the PINN solutions of sub-domains iteratively. Moreover, in this prediction-correction framework, the CG PINN only provides a rough prediction of the solution, and the equations encoded into the CG PINN can be different from the equations encoded in the fine PINNs. Therefore, PPINN is able to tackle multi-fidelity modeling of inverse physical problems [22]. Both topics are interesting and should be investigated in future work.

Acknowledgements

This work was supported by the DOE PhILMs project DE-SC0019453 and the DOE-BER grant DE-SC0019434. This research was conducted using computational resources and services at the Center for Computation and Visualization, Brown University. The authors would like to thank Dr. Zhiping Mao, and Dr. Ansel L Blumers for helpful discussions.

References

[1] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.*, 378:686–707, 2019.

[2] N. O. Hodas and P. Stinis. Doing the impossible: Why neural networks can be trained at all. *Front. Psychol.*, 9(1185), 2018.

[3] T. Kurth, S. Treichler, J. Romero, M. Mudigonda, N. Luehr, E. Phillips, A. Mahesh, M. Matheson, J. Deslippe, M. Fatica, Prabhat, and M. Houston. Exascale deep learning for climate analytics. *arXiv preprint arXiv:1810.01993*, 2018.

[4] J. Lew, D. A. Shah, S. Pati, S. Cattell, M. Zhang, A. Sandhupatla, C. Ng, N. Goli, M. D. Sinclair, T. G. Rogers, and T. M. Aamodt. Analyzing machine learning workloads using a detailed GPU simulator. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 151–152, 2019.

[5] Y. You, Z. Zhang, C. Hsieh, J. Demmel, and K. Keutzer. Fast deep neural network training on distributed systems and cloud TPUs. *IEEE Transactions on Parallel and Distributed Systems*, pages 1–14, 2019.

[6] L. Bottou, F. Curtis, and J. Nocedal. Optimization methods for large-scale machine learning. *SIAM Rev.*, 60(2):223–311, 2018.

[7] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12(Jul):2121–2159, 2011.

[9] R. K. Tripathy and I. Bilionis. Deep UQ: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *J. Comput. Phys.*, 375:565–588, 2018.

[10] C. Michoski, M. Milosavljevic, T. Oliver, and D. Hatch. Solving irregular and data-enriched differential equations using deep neural networks. *arXiv preprint arXiv:1905.04351*, 2019.

[11] N. A. K. Doan, W. Polifke, and L. Magri. Physics-informed echo state networks for chaotic systems forecasting. In *ICCS 2019 - International Conference on Computational Science*, Faro, Portugal, 2019.

[12] M. Mattheakis, P. Protopapas, D Sondak, M. Di Giovanni, and E. Kaxiras. Physical symmetries embedded in neural networks. *arXiv preprint arXiv:1904.08991*, 2019.

[13] P. Stinis. Enforcing constraints for time series prediction in supervised, unsupervised and reinforcement learning. *arXiv preprint arXiv:1905.07501*, 2019.

[14] N. Winovich, K. Ramani, and G. Lin. ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *J. Comput. Phys.*, 2019.

[15] M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial-differential equations. *Comm. Numer. Meth. Eng.*, 10(3):195–201, 1994.

[16] L. Lu, X. Meng, Z. Mao, and G. E Karniadakis. DeepXDE: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

[17] L. Yang, D. Zhang, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.

[18] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *J. Comp. Phys.*, 397:108850, 2019.

[19] D. Zhang, L. Guo, and G. E. Karniadakis. Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. *arXiv preprint arXiv:1905.01205*, 2019.

[20] Y. Yang and P. Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *J. Comput. Phys.*, 394:136–152, 2019.

[21] G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *arXiv preprint arXiv:1811.08967*, 2018.

[22] X. Meng and G. E. Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *arXiv preprint arXiv:1903.00104*, 2019.

[23] G. Bal and Y. Maday. A parareal time discretization for non-linear PDEs with application to the pricing of an American put. In *Recent developments in domain decomposition methods*, pages 189–202. Springer, 2002.

[24] Y. Maday and G. Turinici. A parareal in time procedure for the control of partial differential equations. *CR. Math.*, 335(4):387–392, 2002.

[25] Y. Maday and G. Turinici. The parareal in time iterative solver: a further direction to parallel implementation. In *Domain decomposition methods in science and engineering*, pages 441–448. Springer, 2005.

[26] M. J. Gander. 50 years of time parallel time integration. In *Multiple shooting and time domain decomposition methods*, pages 69–113. Springer, 2015.

[27] B. W. Ong and J. B. Schroder. Applications of time parallelization. *http://mathgeek.us/research/papers/pint_review.pdf*, 2019.

[28] X. Du, M. Sarkis, C. E. Schaerer, and D. B. Szyld. Inexact and truncated parareal-in-time Krylov subspace methods for parabolic optimal control problems. *Electron. Numer. Ana.*, 40:36–57, 2013.

[29] A. Baudron, J. Lautard, Y. Maday, M. K. Riahi, and J. Salomon. Parareal in time 3D numerical solver for the LWR benchmark neutron diffusion transient model. *J. Comp. Phys.*, 279:67–79, 2014.

[30] P. F. Fischer, F. Hecht, and Y. Maday. A parareal in time semi-implicit approximation of the Navier-Stokes equations. In *Domain decomposition methods in science and engineering*, pages 433–440. Springer, 2005.

[31] C. Farhat and M. Chandesris. Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid–structure applications. *Int. J. Numer. Methods Fluids*, 58(9):1397–1434, 2003.

[32] Y. Maday. Parareal in time algorithm for kinetic systems based on model reduction. *High-dimensional partial differential equations in science and engineering*, 41:183–194, 2007.

[33] A. J. Christlieb and B. W. Ong. Implicit parallel time integrators. *J. Sci. Comput.*, 49(2):167–179, 2011.

[34] A. J. Christlieb, R. D. Haynes, and B. W. Ong. A parallel space-time algorithm. *SIAM J. Sci. Comput.*, 34(5):C233–C248, 2012.

[35] A. Blumers, Z. Li, and G. E. Karniadakis. Supervised parallel-in-time algorithm for long-time Lagrangian simulations of stochastic dynamics: Application to hydrodynamics. *J. Comput. Phys.*, 393:214–228, 2019.

[36] R. Livni, S. Shalev-Shwartz, and O. Shamir. On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, pages 855–863, 2014.

[37] L. Torrey and J. Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.

[38] M. Shu. Deep learning for image classification on very small datasets using transfer learning. 2019.

[39] X. Meng and Z. Guo. Localized lattice Boltzmann equation model for simulating miscible viscous displacement in porous media. *Int. J. Heat Mass Tran.*, 100:767–778, 2016.

[40] M. M. Meerschaert, H.-P. Scheffler, and C. Tadjeran. Finite difference methods for two-dimensional fractional dispersion equation. *J. Comp. Phys.*, 211(1):249–261, 2006.

[41] J. Sanders and E. Kandrot. *CUDA by example: an introduction to general-purpose GPU programming*. Addison-Wesley Professional, 2010.

[42] T. Haut and B. Wingate. An asymptotic parallel-in-time method for highly oscillatory pdes. *SIAM J. Sci. Comput.*, 36(2):A693–A713, 2014.

[43] F. Legoll, T. Lelievre, and G. Samaey. A micro-macro parareal algorithm: application to singularly perturbed ordinary differential equations. *SIAM J. Sci. Comput.*, 35(4):A1951–A1986, 2013.

[44] G. Beylkin and N. Coult. A multiresolution strategy for reduction of elliptic PDEs and eigenvalue problems. *Appl. Comput. Harmon. Anal.*, 5(2):129–155, 1998.