Space-time parallel scaling of Parareal with a physics-informed Fourier Neural Operator coarse propagator applied to the Black-Scholes equation

Abdul Qadir Ibrahim
abdul.ibrahim@tuhh.de
Chair Computational Mathematics,
Institute of Mathematics
Hamburg, Germany

Sebastian Götschel
sebastian.goetschel@tuhh.de
Chair Computational Mathematics,
Institute of Mathematics
Hamburg, Germany

Daniel Ruprecht
daniel.ruprecht@tuhh.de
Chair Computational Mathematics,
Institute of Mathematics
Hamburg, Germany

Abstract

Iterative parallel-in-time algorithms like Parareal can extend scaling beyond the saturation of purely spatial parallelization when solving initial value problems. However, they require the user to build coarse models to handle the unavoidable serial transport of information in time. This is a time-consuming and difficult process since there is still limited theoretical insight into what constitutes a good and efficient coarse model. Novel approaches from machine learning to solve differential equations could provide a more generic way to find coarse-level models for multi-level parallel-in-time algorithms. This paper demonstrates that a physics-informed Fourier Neural Operator (PINO) is an effective coarse model for the parallelization in time of the two-asset Black-Scholes equation using Parareal. We demonstrate that PINO-Parareal converges as fast as a bespoke numerical coarse model and that, in combination with spatial parallelization by domain decomposition, it provides better overall speedup than both purely spatial parallelization and space-time parallelization with a numerical coarse propagator.

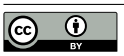
CCS Concepts

• **Computing methodologies** → **Distributed algorithms**; **Neural networks**; • **Mathematics of computing** → **Partial differential equations**.

Keywords

Parareal, parallel-in-time integration, physics-informed neural operator, machine learning, space-time parallelization, Black-Scholes equation

ACM Reference Format:
Abdul Qadir Ibrahim, Sebastian Götschel, and Daniel Ruprecht. 2025. Space-time parallel scaling of Parareal with a physics-informed Fourier Neural Operator coarse propagator applied to the Black-Scholes equation. In *Platform for Advanced Scientific Computing Conference (PASC '25), June 16–18, 2025, Brugg-Windisch, Switzerland.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3732775.3733574



PASC '25, June 16–18, 2025, Brugg-Windisch, Switzerland

ACM ISBN 979-8-4007-1886-1/2025/06
https://doi.org/10.1145/3732775.3733574

1 Introduction

Space-time parallelization, combining spatial parallelization via domain decomposition with parallel-in-time integration, has been shown to be able to extend parallel scaling beyond what parallelization in space alone can provide [43]. However, a key challenge when deploying iterative, multi-level methods like Parareal [32], MGRIT [9] or PFASST [8] remains the need to define one or multiple coarse level models that deal with the inevitable serial information propagation in the time direction. A good coarse-level model must be accurate enough to ensure quick convergence of the method but also needs to be computationally cheap, as it forms a serial bottleneck and limits achievable speedup. Physics-informed neural operators [31], an approach from machine learning to solve partial differential equations, have two properties that suggest they might be attractive as coarse-level models for parallel-in-time methods: first, they are very fast to evaluate once trained and second they are generic, requiring just the PDE residual in the loss function, although some tuning of training parameters might be needed. Neural operators can solve initial-boundary value problems over some time horizon in a single step. Together with an appropriate discretization of the PDE in space, the Neural Operator can be trained to map the state at time t to the state at a later point $t + \Delta t$ [17]. Hence, the neural operator performs exactly the mapping required from a coarse model in Parareal. In contrast to sparse, mesh-based numerical algorithms, neural networks also run efficiently on GPUs, suggesting that a combination of both could help to better utilise heterogeneous architectures [21]. We investigate the effectiveness of a physics-informed neural operator (PINO) as coarse level model for the parallel-in-time integration method Parareal to solve the two-asset Black-Scholes equation, a PDE used in computational finance. The key novelty in this paper is the demonstration that Parareal with an ML-based coarse model not only provides speedup over serial time stepping but can extend scaling beyond the saturation point of space-only parallelization and that a PINO provides a more effective coarse model than a PINN-based propagator.

2 Related Work

This paper is an extension of Ibrahim et al. 2023 [21] where we study a physics-informed neural network propagator (PINN-P) for the one-asset Black-Scholes equation as a coarse model for Parareal. The PINN-P propagator was inspired by classical physics-informed neural networks (PINNs), but, instead of solving the differential equation directly, was used to learn the mapping from a given initial value to the solution of the Black-Scholes equation at a future time

step. PINN-P therefore used a PINN-like architecture to create a simple Neural Operator (NO). NOs, like the Fourier Neural Operator (FNO) [29] or DeepONets [33], are methods to learn such solution operators, i.e., the mapping from parameters or initial conditions to the solution of a partial differential equation. This is in contrast to an actual PINN that would learn the solution of a PDE for a fixed initial condition.

Here, we extend our previous results in three ways: (i) we show that a Physics Informed Neural Operator (PINO), based on the Fourier neural operator, is a better coarse propagator than PINN-P as it requires fewer parameters and a significantly shorter training time, (ii) we consider the more complex two-asset Black-Scholes equations and combine Parareal with spatial parallelization, (iii) we analyze performance of the full space-time parallelization in contrast to studying the performance of Parareal alone.

We have discussed the main works studying combinations of machine learning with parallel-in-time integration in more detail in our previous paper [21]. Therefore, we only provide a quick summary whilst discussing publications that have been published since then in more detail below. Using ML to build coarse propagators was first studied by Yalla and Enquist [45]. The first reports of speedups were made by Agboh et al. [2] for a system of ordinary differential equations related to robotic control. Nguyen and Tsai use ML not to replace the coarse model but to enhance it, in order to improve its accuracy [37]. Gorynina et al. use ML to provide fast approximations of force fields in molecular dynamics simulations with Parareal [15].

A few relevant papers on the combination of data-based techniques with Parareal have been published since our previous paper. Motivated by machine learning approaches, Jin et al. [22] recently proposed an optimization method to construct optimized coarse propagators, based on error estimates, that exhibit favourable numerical properties. Focused on wave-propagation problems, Kaiser et al. [24] extend their previous work [37] to provide a more general end-to-end framework. Instead of learning a coarse propagator, they combine a coarse numerical solver with a convolutional autoencoder (JNet) to augment the under-resolved wave fields. Their approach tries to improve coarse level accuracy to get Parareal to converge in fewer iterations but not to reduce the computational cost of the coarse level. By contrast, we aim to reduce the computational cost of the coarse model to improve the scaling of Parareal but not necessarily to reduce the number of iterations. An approach similar to the one by Kaiser et al. is pursued by Fang and Tsai [10] to stabilize Parareal for Hamiltonian systems. They suggest using either a neural network to correct phase information in the coarse model or to replace the fine model. Both approaches are tested for the Fermi-Pasta-Ulam problem. A combination of Parareal with a Neural Operator for fusion MHD simulations was very recently investigated [39] but without analyzing speedups from space-time parallelization. Betcke et al. use a Random Projection Neural Network as coarse propagator for Parareal [5]. They provide a theoretical analysis of convergence properties of the resulting method but do not measure speedups. Gattiglio et al. [13] propose a combination of Random Neural Networks with Parareal called RandNet-Parareal. They show a significant improvement in speedup over classical Parareal, in line with PINN-Parareal [21], but do not explore space-time parallelization.

While our approach leverages machine learning to improve parallel-in-time integration algorithms, there are also studies that go the other way, using Parareal to improve training of neural networks. Lee et al. [27] further develop their approach to use a Parareal-like technique to accelerate the training of very deep networks, similar to previous works by Guenther et al. [20] or Meng et al. [34]. They interpret a deep neural network as what they call a "parareal neural network", which consists of coarse and fine structures, similar to the propagators in Parareal. A Parareal-style iteration is then used to parallelize the training of the DNN across multiple GPUs.

One issue when using machine learning to construct neural operators is the required amount of training data. To reduce this, physics-informed variants add the PDE residual and other constraints to the loss function, and thus can even be trained without any training data. Physics-informed neural operators [31] based on the Fourier neural operator [29], as used here, were originally constructed to approximate PDE solution operators on cartesian, periodic domains, but can be extended to more general geometries [28]. Deep operator networks, so-called DeepONets [33], have been extended to physics-informed variants as well [16, 44]. Alternative architectures for operator learning include, among others, nonlinear manifold decoders [11, 41] and wavelet-based approaches [19, 36].

We use the Black-Scholes equation from computational finance with two assets as a test problem [6, 35]. The performance of classical Parareal for option pricing has been studied before [4, 18, 38] but not with ML-based coarse propagators.

3 Algorithms and Benchmark Problem

Section §3.1 briefly explains the Parareal algorithm. In §3.2 we introduce our benchmark problem, the two-asset Black-Scholes equation, and describe a numerical solution approach based on finite differences. Then, §3.3 describes the spatial parallelization while §3.5 explains the physics-informed neural operator we use to construct a coarse-level model for Parareal.

3.1 Parareal

Parareal is an algorithm for the parallel solution of an initial value problem

$$u'(t) = f(u(t)),\ u(t_0) = u_0 \tag{1}$$

for $0 \leq t \leq T$ which here stems from the spatial discretization of a time-dependent PDE with finite differences, see §3.2. To parallelize integration of (1), Parareal needs a decomposition of the temporal domain $[0, T]$ into P_{time} time slices $[T_n, T_{n+1}]$, $n = 0, \ldots, P_{\text{time}} - 1$. Although it is possible to assign multiple time slices to a processor, we always assume that every processor handles only one-time slice so that P_{time} is also equal to the number of processors used in the temporal direction.[1] Let $\mathcal{F}$ denote a time integration method of suitable accuracy such that

$$u_{n+1} = \mathcal{F}(u_n) \tag{2}$$

is the approximation delivered at the end of a slice $[T_n, T_{n+1}]$ when starting from an initial value u_n at T_n. Classical time stepping would

[1] Note that when Parareal is combined with spatial parallelization, every time slice is assigned not to a single processor but to P_{space} many processors, where P_{space} is the number of processors over which a single instance of the solution u_n is distributed in space. The total number of processors is then $P_{\text{time}} \times P_{\text{space}}$.

compute (2) sequentially for $n = 0, \ldots, P_{\text{time}} - 1$. Parareal replaces this serial procedure with the iteration

$$\begin{aligned} u_{n+1}^{k+1} &= \mathcal{G}(u_n^{k+1}) + \mathcal{F}(u_n^k) - \mathcal{G}(u_n^k), \\ & n = 0, \ldots, P_{\text{time}} - 1, \\ & k = 0, \ldots, K - 1 \end{aligned} \tag{3}$$

where K is the number of iterations and $\mathcal{G}$ a coarse solver for (1). Note that both fine and coarse propagator in Parareal are treated as black-boxes and can encompass explicit or implicit methods as well as arbitrary direct or iterative linear or nonlinear solvers [3].

Although the serial computation of iteration (3) is computationally more expensive than serial time stepping by computing (2) for $n = 0, \ldots, P_{\text{time}-1}$, the costly evaluation of $\mathcal{F}$ can be parallelized over all time slices. Therefore, if the coarse model is cheap enough and the number of iterations small, iteration (3) can solve (1) in less wallclock time than serially evaluating (2). For Parareal, speedup using P_{time} timeslices is bounded by

$$S(P_{\text{time}}) \leq \frac{1}{\left(1 + \frac{K}{P_{\text{time}}}\right)\frac{c_{\text{coarse}}}{c_{\text{fine}}} + \frac{K}{P_{\text{time}}}} \leq \min\left\{\frac{c_{\text{fine}}}{c_{\text{coarse}}}, \frac{P_{\text{time}}}{K}\right\} \tag{4}$$

where c_{coarse} and c_{fine} are the execution times for $\mathcal{G}$ and $\mathcal{F}$. To allow for speedup from Parareal, K has to be small, that is, the method needs to converge quickly and $c_{\text{coarse}} \ll c_{\text{fine}}$, that is, the coarse propagator must be computationally much cheaper than the fine. We will demonstrate that a PINO provides a coarse propagator that is much faster than a numerical $\mathcal{G}$, thus reducing c_{coarse}, but accurate enough to not require more iterations K. It therefore provides a significant benefit in scaling in situations where the first term in (4) is limiting speedup. Details on Parareal, including the pseudocode of a Parareal implementation, can be found in the literature [40].

3.2 Numerical solution of the two-asset Black-Scholes equation

We consider the Black-Scholes equation in the form

$$\begin{aligned} &\frac{\partial u}{\partial t} + \frac{1}{2}\sigma_1^2 x^2 \frac{\partial^2 u}{\partial x^2} + \frac{1}{2}\sigma_2^2 y^2 \frac{\partial^2 u}{\partial y^2} \\ &\quad + \rho\sigma_1\sigma_2 xy \frac{\partial^2 u}{\partial x \partial y} + rx\frac{\partial u}{\partial x} + ry\frac{\partial u}{\partial y} - ru = 0 \end{aligned} \tag{5}$$

where $u(t, x(t), y(t))$ is the option price, t is time, x the first and y the second asset value, σ_1 and σ_2 are the volatilities of the two assets, ρ is the correlation of the two assets and r is the risk-free interest rate. Note that while (5) is a model from computational finance, its mathematical structure is that of a quasi-linear advection-diffusion-reaction equation. Therefore, the presented approach will hopefully generalize to other applications.

Since the PDE is typically augmented with a final time condition, we change the variable $\tau = T - t$ to turn that into an initial condition for convenience. Under this change of variable, the equation becomes

$$\begin{aligned} &\frac{\partial u}{\partial \tau} - \frac{1}{2}\sigma_1^2 x^2 \frac{\partial^2 u}{\partial x^2} - \frac{1}{2}\sigma_2^2 y^2 \frac{\partial^2 u}{\partial y^2} \\ &\quad - \rho\sigma_1\sigma_2 xy \frac{\partial^2 u}{\partial x \partial y} - rx\frac{\partial u}{\partial x} - ry\frac{\partial u}{\partial y} + ru = 0 \end{aligned} \tag{6}$$

This can be extended to a more general n-dimensional PDE

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sum_{i,j=1}^{n} \rho_{ij}\sigma_i\sigma_j x_i x_j \frac{\partial^2 u}{\partial x_i \partial x_j} + r\sum_{i=1}^{n} x_i \frac{\partial u}{\partial x_i} - ru = 0 \tag{7}$$

for n assets but we focus on the two-asset case with $n = 2$ here. We use a centered finite difference for spatial discretization and an implicit Euler scheme for time discretization [23], resulting in the discretization

$$\begin{aligned} &\frac{u_{i,j}^{n+1} - u_{i,j}^n}{\Delta\tau} - \frac{1}{2}\sigma_1^2 x_i^2 \frac{u_{i-1,j}^{n+1} - 2u_{i,j}^{n+1} + u_{i+1,j}^{n+1}}{\Delta x^2} \\ &- \frac{1}{2}\sigma_2^2 y_j^2 \frac{u_{i,j-1}^{n+1} - 2u_{i,j}^{n+1} + u_{i,j+1}^{n+1}}{\Delta y^2} \\ &- \rho\sigma_1\sigma_2 x_i y_j \frac{u_{i+1,j+1}^{n+1} + u_{i-1,j-1}^{n+1} - u_{i+1,j-1}^{n+1} - u_{i-1,j+1}^{n+1}}{4\Delta x \Delta y} \\ &- rx_i \frac{u_{i+1,j}^{n+1} - u_{i-1,j}^{n+1}}{2\Delta x} - ry_j \frac{u_{i,j+1}^{n+1} - u_{i,j-1}^{n+1}}{2\Delta y} + ru_{i,j}^{n+1} = 0 \end{aligned} \tag{8a}$$

The subscripts i and j stand for horizontal and vertical node index and the superscript n for the time step index. For simplicity, we consider a uniform grid with $\Delta x = \Delta y = 1$ where the nodal distances for the two axes are same. The time step size for the coarse propagator is chosen as $\Delta\tau = 0.1$. For the fine propagator, this interval is subdivided into three smaller steps. We consider the two-asset cash-or-nothing opion since the closed form solution for this is known [25] and can be used to verify the accuracy of PINO-Parareal.

The closed-form solution for the cash-or-nothing option reads:

$$u(x, y, \tau) = ce^{-r\tau} B(d_x, d_y; \rho), \tag{9}$$

$$d_x = \frac{\ln\left(\frac{x}{S_1}\right) + (r - 0.5\sigma_x^2)\tau}{\sigma_x\sqrt{\tau}}, \tag{10}$$

$$d_y = \frac{\ln\left(\frac{y}{S_2}\right) + (r - 0.5\sigma_y^2)\tau}{\sigma_y\sqrt{\tau}}, \tag{11}$$

where the Bivariate cumulative normal distribution function [14] is given by

$$B(d_x, d_y; \rho) = \frac{1}{2\pi\sqrt{1-\rho^2}} \int_{-\infty}^{d_x}\int_{-\infty}^{d_y} e^{-\frac{\xi_1^2 - 2\rho\xi_1\xi_2 + \xi_2^2}{2(1-\rho^2)}} d\xi_2 d\xi_1. \tag{12}$$

The computational domain is $[0, 300] \times [0, 300]$ for $t \in [0, 1]$ and (9) is used to set the Dirichlet boundary condition. The exercise price is set to $S_1 = 100$ and $S_2 = 100$, the risk-free interest rate to $r = 1.0$, the volatility of the first asset to $\sigma_1 = 0.3$, the volatility of the second asset to $\sigma_2 = 0.3$, the correlation to $\rho = 0.5$ and the maturity is $T = 1$ [25]. These parameters are chosen to balance practical relevance and the need to test Parareal in a regime where achieving good convergence is a challenge. Therefore, the risk-free interest rate $r = 1.0$ chosen higher than typical market rates to magnify the drift terms in (6), since these make achieving good Parareal convergence more difficult [12]. Volatilities $\sigma_1 = \sigma_2 = 0.3$ are moderate and ensure that the diffusion terms contribute noticeably to the dynamics without making the problem diffusion-dominated, a regime in which Parareal is already known to converge very

quickly. A correlation of $\rho = 0.5$ introduces interdependence between the assets, which is crucial for capturing realistic multi-asset interactions. In real-world applications, these parameters depend on market conditions: volatilities typically range from 0.1 to 0.5, interest rates normally vary between 0.01 and 0.05, and correlations must be in $[-1, 1]$. The selected values test Parareal as well as the PINO model's robustness without being too unrealistic.

Strike prices $S_1 = S_2 = 100$ represent an at-the-money option, which produces meaningful dynamics around the most sensitive region of the option price. The strike prices S_1, S_2 do not explicitly appear in the Black-Scholes equation but are incorporated into the terminal or boundary conditions. For this problem, the terminal payoff function of cash-or-nothing options with wo assets is given by

$$u_T(x, y) = \begin{cases} c, & \text{if } x \geq S_1,\ y \geq S_2, \\ 0, & \text{otherwise.} \end{cases}$$

where S_1, S_2 are the strike prices of the two assets and c is the cash amount received at expiration.

3.3 Spatial parallelization

We use the `dask` Python framework [7] to parallelize the computation of the finite difference approximation (8) in space. This library offers high-level abstractions for distributed computing and uses mpi4py for communication and data exchange. Note that while a shared memory parallelization in time might offer advantages, the global interpreter lock in Python makes this very difficult to realize. We therefore rely on distributed memory parallelization in both space and time. Spatial parallelization is done by distributing the iterative solver step in each time step of implicit Euler by decomposing the spatial domain. To do this, we implemented a conjugate gradient (CG) method using `dask` arrays to parallelize the matrix-vector and vector-vector products arising in CG, which are by far the most computationally costly part of the algorithm. For spatial parallelization using P_{space} processors, speedup is bounded by

$$S(P_{\text{space}}) \leq P_{\text{space}}. \tag{13}$$

For a combined space-time parallelization, in the best case the achieved speedup is multiplicative so that

$$S(P_{\text{space}}, P_{\text{time}}) \leq \frac{P_{\text{space}}}{\left(1 + \frac{K}{P_{\text{time}}}\right) \frac{c_{\text{coarse}}}{c_{\text{fine}}} + \frac{K}{P_{\text{time}}}} \tag{14}$$

Note that the total number of processors is $P_{\text{total}} = P_{\text{space}} \times P_{\text{time}}$ since computations on every time slice are parallelized using P_{space} many processes. Also, since this model neglects overheads from communication or competition for resources, it is only an upper bound and not necessarily predictive of actual speedup.

3.4 Fourier Neural Operator (FNO)

The Fourier Neural Operator (FNO) [29] is a recently developed deep learning framework for approximating solution operators of partial differential equations (PDEs). Unlike traditional numerical methods that compute the solution for a fixed discretization, FNO learns a mapping between function spaces, enabling generalization across different resolutions and input conditions. At its core, FNO leverages the Fast Fourier Transform (FFT) to express the integral kernel of the operator in the frequency domain. In each Fourier layer, the input function is first transformed to Fourier space, where a learnable multiplier is applied to a truncated set of Fourier modes. An inverse FFT then returns the modified data to physical space. This procedure allows the network to capture long-range dependencies and global features while keeping the number of learnable parameters modest. The typical FNO architecture involves three main steps:

- Lifting: The input function is projected into a higher-dimensional feature space using a pointwise fully connected layer.
- Fourier Layers: A sequence of layers is applied where each layer computes the FFT of the feature maps, performs a modal truncation and a corresponding multiplication with trainable weights in Fourier space, and finally transforms the result back with an inverse FFT.
- Projection: The resulting representation is mapped back to the target function space via another pointwise layer.

Key advantages of the FNO framework include its mesh invariance, which permits the application of the trained operator to inputs with different discretizations, and its computational efficiency stemming from the FFT operations. In our work, the FNO serves as the backbone for the physics-informed neural operator (PINO). By embedding the physics of the Black-Scholes PDE into the loss function, the PINO retains the efficiency of FNO while ensuring that the learned operator respects the governing equations.

3.5 Physics-Informed Neural Operator (PINO)

To construct an efficient and accurate coarse propagator for Parareal, we use a Physics-Informed Neural Operator (PINO) based on the Fourier Neural Operator (FNO) [30, 31]. The FNO learns a solution operator that maps input states at time t_n to t_{n+1} and is enhanced here with physics-informed loss functions that incorporate knowledge of the two-asset Black-Scholes equation.

The architecture comprises three stages: (i) a *lifting layer* that maps the input tuple (t, x, y) into a 64-dimensional feature space using a pointwise fully connected layer, (ii) a sequence of $L = 4$ *Fourier integral operator (FIO) blocks* which apply FFTs, truncate to the lowest $m = 12$ modes, multiply by learnable weights in Fourier space, and return to physical space via inverse FFT, and (iii) a *projection layer* that maps the feature output to the target scalar option price. Each block uses ReLU activations and batch normalization for stability. Hyperparameters like number of layers, modes, etc. were chosen in an ad-hoc manner without extensive hyperparameter optimization.

The training objective is a composite physics-informed loss:

$$\text{MSE}_{\text{total}} = \text{MSE}_f + \text{MSE}_{\text{b}} + \text{MSE}_{\text{exp}}, \tag{15}$$

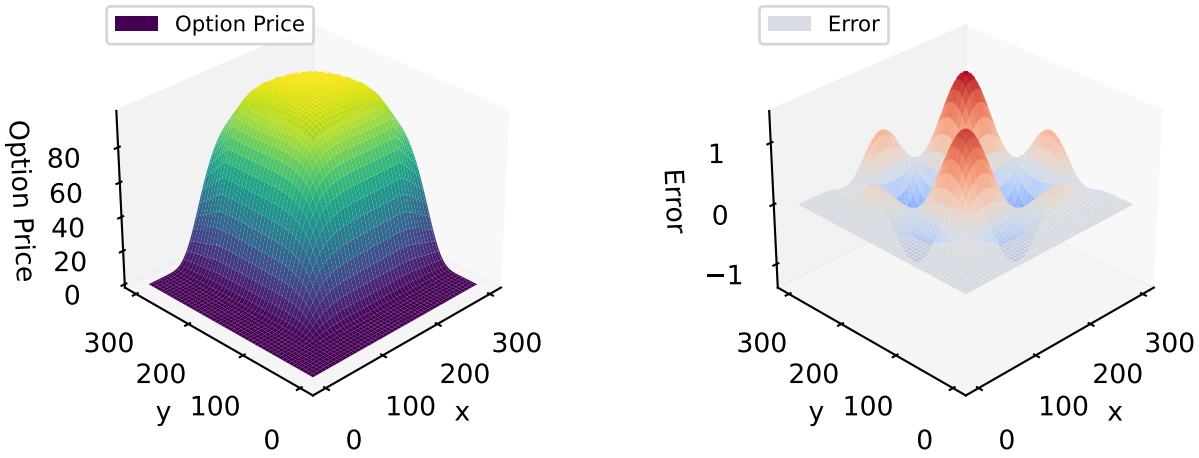


Figure 1: Numerical solution using finite differences (left) of the two-asset Black-Scholes equation and error against the closed-form solution (right).

with the components defined as:

$$\mathrm{MSE}_f = \frac{1}{N_f}\sum_{i=1}^{N_f} \left| f\left(\tilde{u}(t_i, x_i, y_i)\right)\right|^2, \tag{16}$$

$$\mathrm{MSE}_b = \frac{1}{N_b}\sum_{i=1}^{N_b} \left|\tilde{u}(t_i, x_i, y_i) - u(t_i, x_i, y_i)\right|^2, \tag{17}$$

$$\mathrm{MSE}_{\mathrm{exp}} = \frac{1}{N_{\mathrm{exp}}}\sum_{i=1}^{N_{\mathrm{exp}}} \left|\tilde{u}(T, x_i, y_i) - \max\left[\max(x_i, y_i) - S, 0\right]\right|^2. \tag{18}$$

We use N_f = 10,000 collocation points in the domain, N_b = 5,000 on the boundary, N_{exp} = 5,000 at expiration, and N_{init} = 5,000 sampled initial conditions. These were randomly drawn to represent a diverse range of system states. Training is performed using the Adam optimizer [26] for 2,500 epochs, with an initial learning rate of 0.001, a decay factor of 0.96 every 25 epochs, and batch size tuning based on available compute. Although L-BFGS has been shown to improve convergence in some settings [46], our experiments found that Adam yielded comparable final accuracy in shorter overall training times. The resulting PINO model has approximately 833,000 trainable parameters and trains in about 10 minutes on a modern CPU. It achieves a relative ℓ_2-error of 1.5×10^{-2} versus the closed-form solution (9), and inference is over 50 times faster than using a numerical coarse solver. Compared to our earlier PINN-P model [21], which required over 40 minutes of training, the PINO is both faster and more accurate. Importantly, PINO is mesh-invariant and generalizes well to changes in temporal/spatial resolution and moderate shifts in model parameters like volatility (σ_1, σ_2) or interest rate (r), making it highly robust for varied use cases in financial modeling. To use the trained PINO in Parareal, the model takes asset values at time t_n and returns the predicted solution at t_{n+1}. Figure 2 illustrates the solution error compared to the analytical benchmark and the loss decay during training.

4 Numerical Results

The system used for the tests is the Linux cluster at Hamburg University of Technology (TUHH), running AlmaLinux 8 and equipped with fast OmniPath connectivity. The system architecture supports both 32-bit and 64-bit operation modes. It features different types of nodes but we use one with two AMD EPYC 9124 32-core CPUs. The CPU frequency ranges from 1500 MHz to 3711.9141 MHz, with a base frequency of 3000 MHz. The system node has 32 KB each of L1 data and instruction cache, 1024 KB of L2 cache, and 16384 KB of L3 cache. Additionally, the system is configured with NUMA (Non-Uniform Memory Access) architecture, consisting of two NUMA nodes. Virtualization support is enabled, and the system's BogoMIPS value is 6000.00.

4.1 Convergence of Parareal: PINN-P vs PINO

Figure 2 (left) shows the difference between the PINO-generated and the closed-form solution. Figure 3 (left) shows the normalized error of Parareal with P_{time} = 12 timeslices against the fine serial solution on the y-axis and the number of iterations K on the x-axis. It shows convergence for three variants of Parareal, one with a numerical coarse model, one using PINN-P as coarse model and one using PINO. Note that because all coarse propagators run in single precision, convergence stops at a normalized error of around 10^{-8}. In order to provide a fair comparison of runtimes, we modified the numerical coarse propagator to also run in single precision.

There is very little difference in convergence speed between PINN-P and PINO Parareal. Parareal with numerical coarse models converges at the same rate but with slightly larger errors, although the differences are small. However, the PINO uses a much smaller

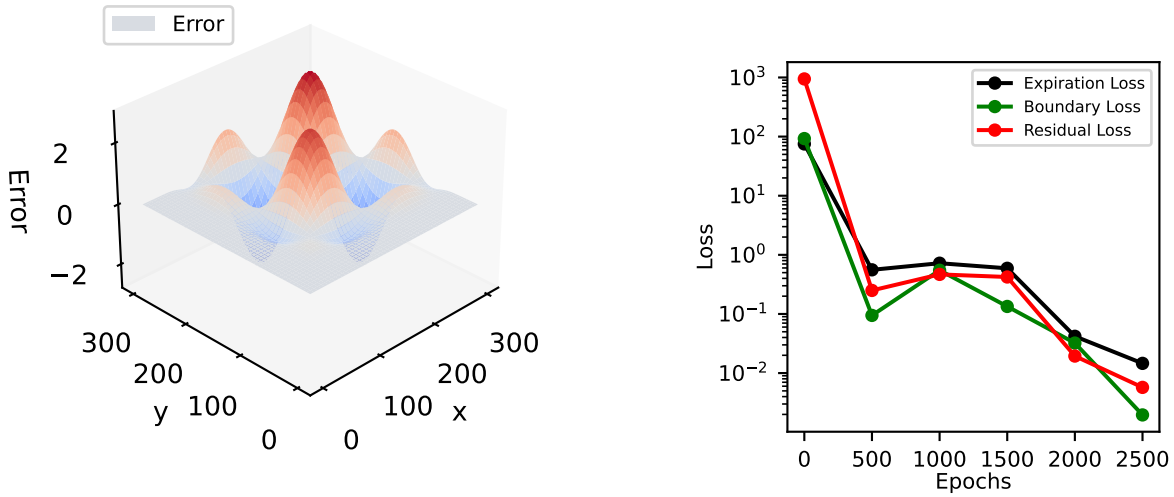


Figure 2: Error of the PINO-generated solution against the closed-form solution (left) and evolution of the physics-informed loss components over training (right). After 2500 epochs, the loss is reduced by five orders of magnitude.

Table 1: Size, training time and accuracy, measured as relative l_2-error against the closed-form solution, for the PINN-P and PINO coarse propagator.

Network	Trainable parameters	Training time	Accuracy
PINN-P	1262833	42 min	2.3×10^{-2}
PINO	832833	10 min	1.5×10^{-2}

network than the PINN-P with fewer trainable parameters and thus is significantly faster to train, see Table 1. The PINO is also marginally more accurate. For comparison, the relative error of the numerical coarse propagator is 1.0×10^{-2} while the fine propagator has a relative l_2-error of 2.2×10^{-3}. Below, we report speedups for $K = 1$ iterations, where Parareal will deliver a comparable discretization error as the fine propagator, as well as for $K = 2$, after which Parareal has converged up to a more generic tolerance[2] of 10^{-4}.

Figure 3 (right) shows convergence of PINO-Parareal if training of the coarse propagator is stopped at 500, 1500 or 2500 epochs. Interestingly, the convergence behavior changes little, that is the shapes of the curves are similar, but with more training the results become more accurate.

Table 2 shows the runtimes c_{fine} and c_{coarse} in (4) for the fine propagator and the numerical and PINO coarse propagator. Upper bounds for achievable speedup according to (4) are $S \leq 350/113 \approx 3.1$ for Parareal-Num and $S \leq 350/2.2 \approx 159$ for Parareal-PINO. Note that these are the runtimes per timeslice so that the total runtime for the fine serial propagator baseline would be $c_{\text{fine}} \times P_{\text{time}}$.

[2]In a use case, the discretization error of the fine propagator will not be known. Therefore, Parareal is normally run with some user-defined tolerance and 10^{-4} would be a reasonable value when using single precision coarse propagators.

Table 2: Runtime of the fine propagator $\mathcal{F}$, the numerical coarse propagator and the PINO coarse propagator $\mathcal{G}$ in seconds, averaged over ten runs. Timings are done in serial execution and exclude setup times. Because maximum speedup by Parareal is bounded by $c_{\text{fine}}/c_{\text{coarse}}$, speedup for Parareal with a coarse propagator is bounded by $350/113 \approx 3.1$ whereas Parareal-PINO can, in theory, achieve speedup of up to $350/2.2 \approx 159.1$. Note that all coarse propagators run in single precision, including the numerical.

Runtimes per timeslice in seconds	
Fully serial fine propagator	350.007 ± 0.00368
Space-parallel fine propagator ($P_{\text{space}} = 8$)	58.33 ± 0.00385
Space-parallel fine propagator ($P_{\text{space}} = 16$)	37.82 ± 0.002108
Numerical coarse propagator	113.011 ± 0.00118
PINO coarse propagator (after training)	2.203 ± 0.00228
PINN coarse propagator (after training)	6.504 ± 0.00157

4.2 Parareal-only scaling

Figure 4 shows parallel speedup from Parareal with $K = 1$ (left) or $K = 2$ iterations (right), without spatial parallelization, with the number of time slices P_{time} ranging from 2 to 64, filling the whole node. The serially run fine propagator is used as a baseline, in line with common practice when studying performance of Parareal. Parareal scales reasonably well, not too far from the theoretical maximum (4), even though efficiencies decrease somewhat when filling the whole node. While both Parareal with a numerical and a PINO coarse propagator deliver speedup that is relatively close to the respective theoretical maximum, Parareal-PINO provides much better speedup throughout because of the greatly improved $c_{\text{fine}}/c_{\text{coarse}}$

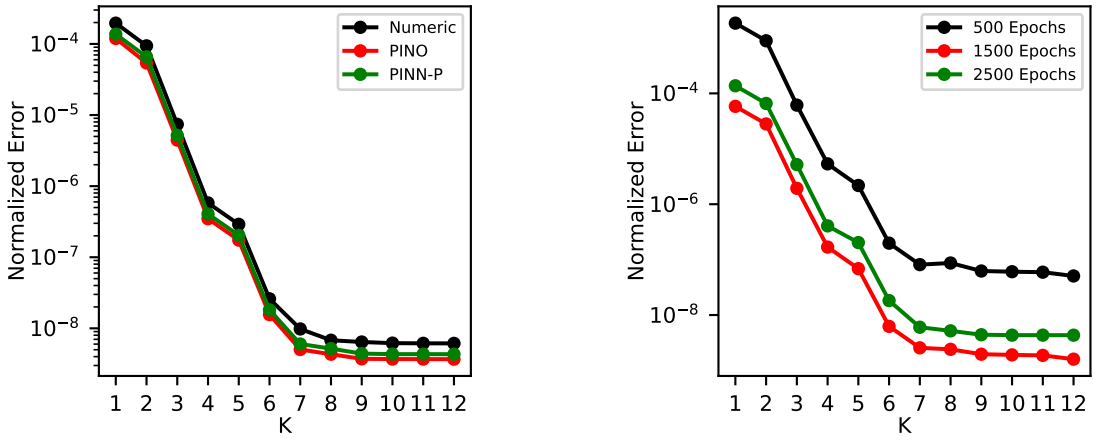


Figure 3: Convergence of Parareal against the serial fine solution with PINO, PINN-P and numerical propagator as coarse model (left). K is the number of Parareal iterations. There is no significant difference in convergence between Parareal with a PINO, PINN-P or numerical coarse propagator. Convergence of PINO-Parareal against the serial fine solution if the training process is stopped after 500, 1500 or 2500 epochs (right). Longer training produces a more accurate coarse propagator which reduces error levels. However, convergence behavior is is very similar in all three cases.

ratio in (4). Note that because speedup for PINO-Parareal is limited by the second term in (4), we see a much greater reduction of speedup compared to Parareal-Num when increasing K from 1 to 2. However, even for $K = 2$, PINO-Parareal outperforms Parareal with a numerical coarse propagator and delivers more than 10-times speedup when using 64 timeslices.

4.3 Space-time parallel strong scaling

Figure 5 shows speedup measured against a reference simulation running the fine propagator in serial without spatial parallelization versus the total number of used cores. Shown are speedups from a space-only parallelization using a serial fine integrator as well as space-time parallelization using Parareal with numerical and PINO coarse propagator and $K = 1$ (left) and $K = 2$ (right) iterations.

Spatial parallelization alone provides near-ideal speedup up to $S(P_{\text{space}}) \approx 16$ at $P_{\text{space}} = 16$ cores but, after that, speedup decreases when more cores are added due to communication overheads. Since it has been shown that it is often beneficial for space-time parallelism to use slightly fewer than the maximum number of processors up to which spatial parallelization scales [42], we fix $P_{\text{space}} = 8$ for all Parareal runs and increase P_{times} from 2 to 8 for a total number of cores ranging from $P_{\text{space}} \times P_{\text{time}} = 8 \times 2 = 16$ to $8 \times 8 = 64$.

For $K = 1$, space-time Parareal with a numerical coarse propagator scales all the way up to $8 \times 8 = 64$, close to the theoretical maximum However, total speedup, even when using the full node, is still slightly less than what space-only parallelization with $P_{\text{space}} = 16$ cores can provide. The reason is that the numerical coarse propagator is too expensive compared to the fine. In such a case, Parareal will not provide any benefits. By contrast, Parareal-PINO provides close to ideal speedup all the way up to the full node. The combination of Parareal-PINO with spatial parallelization extends scaling from a total speedup of around 16 to a total speedup of around 50 by being able to efficiently utilize the full 64 cores of the node.

For $K = 2$ iterations, the speedup of Parareal-PINO in Figure 5 is noticeably reduced due to the second bound in (4). Because speedup from Parareal is bounded by P_{time}/K and thus parallel efficiency by $1/K$, we now also see speedup that is farther away from ideal speedup. Here, around 40 cores are required for space-time parallelization to become faster than purely spatial parallelization. However, when using the full node, space-time parallelization using Parareal-PINO still extends scaling, providing speedup of around 30 in contrast to 16 from spatial parallelization alone.

Figure 6 shows runtimes of Parareal with PINO, PINN-P and numerical coarse propagator and $P_{\text{space}} = 8$ processes for parallelization in space. For comparison, the serial fine propagator with no parallelization in space requires ≈ 3150.63 seconds while at peak spatial parallelization using $P_{\text{space}} = 16$ processes takes ≈ 364 seconds. The latter value is indicated in both plots as a dashed-dotted horizontal line. In both cases, Parareal with a numerical coarse propagator struggles to reduce runtimes below what space parallelization alone can provide. However, both PINN-P-Parareal and PINO-Parareal, in combination with spatial parallelization, can reduce runtimes further. In both cases, runtimes are smaller for PINO-Parareal than for PINN-P-Parareal.

4.4 Generalization to different resolution

Figure 7 shows convergence of PINO-Parareal against the serial fine solution in a weak scaling type test. We start with $\Delta x = \Delta y = 1$ spatial resolution and $\Delta\tau = 0.1$ temporal resolution in the coarse and $\Delta\tau/3$ in the fine propagator using $P_{\text{time}} = P = 2$ time slices. We twice double both spatial and temporal resolution and the number of timeslices to $P_{\text{time}} = 2P = 4$ and $P_{\text{time}} = 4P = 8$. The PINO remains the same as before and is not retrained — this is possible because PINOs are inherently mesh-invariant, as they learn mappings between function spaces rather than finite-dimensional vectors. By

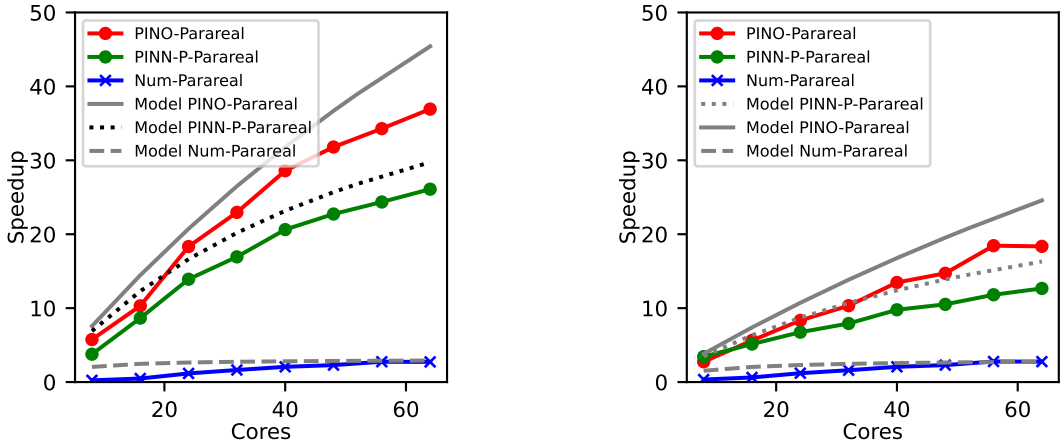


Figure 4: Parallel scaling of Parareal alone with no spatial parallelization after $K = 1$ (left) and $K = 2$ (right) iterations. PINO-Parareal scale better than PINN-P-Parareal and both outperform Parareal with a numerical coarse propagator.

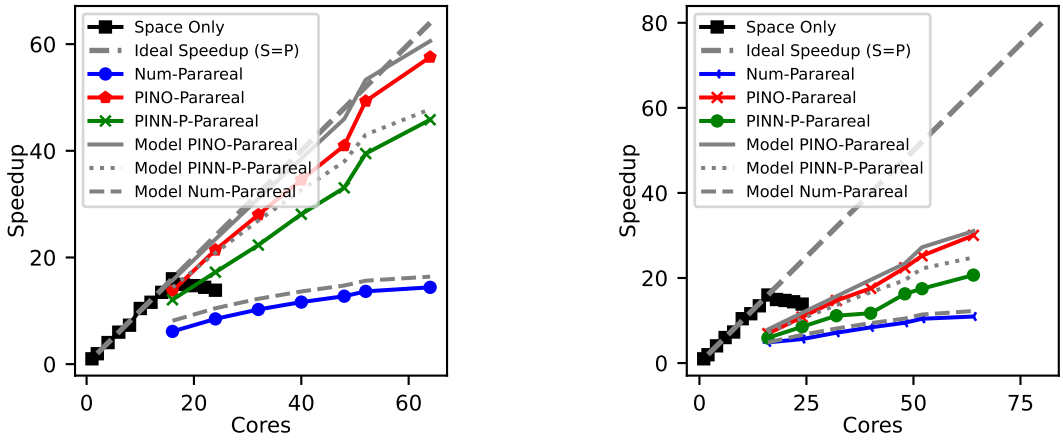


Figure 5: Space-time parallel scaling of Parareal with $K = 1$ (left) and $K = 2$ (right) iterations with PINO, PINN-P and numerical coarse propagator combined with spatial parallelization using $P_{\text{space}} = 8$ cores and speedup from spatial parallelization alone. Since runtimes for the PINO coarse propagator are smaller than for the PINN-P or numerical coarse model, Parareal-PINO provides better speedup, see also the discussion after (4).

contrast, traditional neural networks operate on fixed-size inputs and outputs.

Convergence of PINO-Parareal remains fast and error levels even decrease slightly as resolution increases. For a fixed error tolerance, the number of required Parareal iterations therefore remains constant as resolution and problem size increase. This means that, for the Black-Scholes equations, PINO-Parareal can deliver good weak scaling if implemented efficiently because its computational cost will not grow with problem size. While similar behavior can be expected for other parabolic PDES where Parareal typically performs well, this will not be true for hyperbolic-style problems, where numerical diffusion becomes weaker as $\Delta x \to 0$, which will make Parareal converge worse [12].

4.5 Generalization of Parareal-PINO to different model parameters

The PINO was trained for model parameters $\sigma_1 = 0.3$, $\sigma_2 = 0.3$ and $r = 1$ and resolution $\Delta x = \Delta y = 1$ and $\Delta\tau = 0.1$. To assess how well PINO-Parareal generalizes and how robust its convergence is against changes in model parameters, we show the convergence error of PINO-Parareal against the fine serial solution for a range of different values for r, σ_1 and σ_2 in Figure 8. The PINO is not retrained or otherwise modified - therefore, PINO-Parareal uses a coarse propagator that was trained for different parameters than the problem it is applied to. The rate of convergence remains mostly invariant but error levels change as the parameters change. Convergence is very robust against variations in r. Even for $r = 5$ the

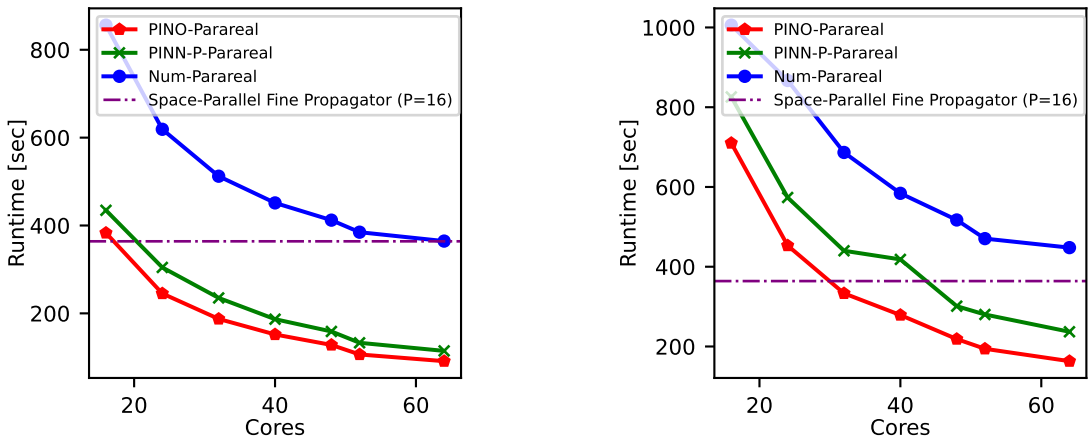


Figure 6: Runtimes of Parareal with numerical, PINO and PINN-P coarse propagator for $K = 1$ (left) and $K = 2$ (right) iterations. The runtime of the space-parallel time-serial fine propagator using $P_{\text{space}} = 16$ processes is shown as horizontal lines.

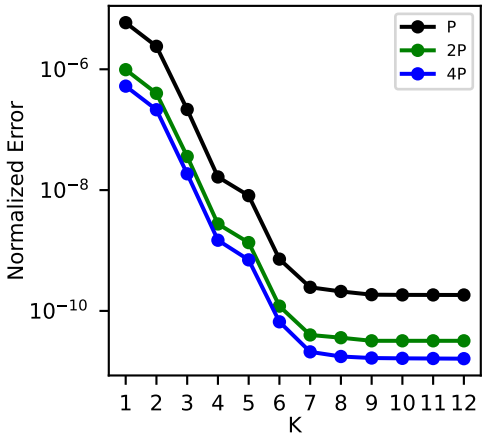


Figure 7: Convergence of PINO-Parareal when increasing spatial and temporal resolution and thus problem size as well as the number of timeslices together ("weak scaling"). The PINO coarse propagator generalizes well to different spatial and temporal resolutions.

difference in convergence compared to the training value of $r = 1$ is small. Sensitivities are larger for changes in the volatilities σ_1 and σ_2. Interestingly, values of σ_1, σ_2 that are slightly smaller than the trained value improve convergence, possibly because the dynamics of the problem become less rapid. However, in the limit $\sigma_1, \sigma_2 \to 0$, only the first order derivatives are left in (6). For very small volatilities, the equation degenerates into an advection-type problem and we see the well-documented non-monotonic convergence of Parareal for transport problems [12]. These results are not shown here but can be generated with the provided code. For values of σ_1, σ_2 larger than the training value, error levels increase until, at $\sigma_1 = 5$ or $\sigma_2 = 5$, they are about one order of magnitude larger. However, at this point, the training values have been exceeded by a factor of $5/0.2 = 25$ for σ_1 and $5/0.3 \approx 17$ for σ_2, so that some degeneration of performance is to be expected.

For our test cases, we used a range of values for both volatility σ and the risk-free interest rate r: 0.1, 0.4, 1, 3, and 5. While *sigma* and r values of 5 are large compared to typical market conditions — where volatilities usually range from 15% to 60% and interest rates are often below 2% — these values were intentionally chosen to test the models robustness and evaluate how well the PINO generalizes across a wide parameter space, including edge cases. This approach helps ensure the model can handle both realistic and high-stress scenarios.

5 Summary

We investigate convergence and parallel scaling of the Parareal parallel-in-time method using a Fourier Neural Operator as a coarse model. As a benchmark problem, we use the two-asset Black-Scholes equation from computational finance. The PINO provides accuracy comparable to a numerical coarse model and a previously studied coarse model based on a physics-informed neural network propagator (PINN-P). However, the PINO takes only about a quarter of the training time of the PINN-P, making it a better choice. Because evaluating the PINO, once trained, is about a factor of fifty faster than running the numerical coarse model and about three times faster than the PINN-P, Parareal-PINO greatly relaxes the bound on speedup for Parareal given by the ratio of runtimes of the fine to the coarse propagator. Whereas the speedup of standard Parareal is bounded by 3.1, the much faster PINO coarse model theoretically allows for Parareal speedups up to 159. We perform scaling tests of Parareal alone and of a combination of Parareal with spatial parallelization. In both cases, Parareal-PINO significantly outperforms standard Parareal. Furthermore, we demonstrate that a combined space-time parallelization using Parareal-PINO scales beyond the saturation point of spatial parallelization alone. Whereas the latter saturates at a speedup of around 16 on as many cores, the former scales to the full node with 64 cores, providing a total speedup of

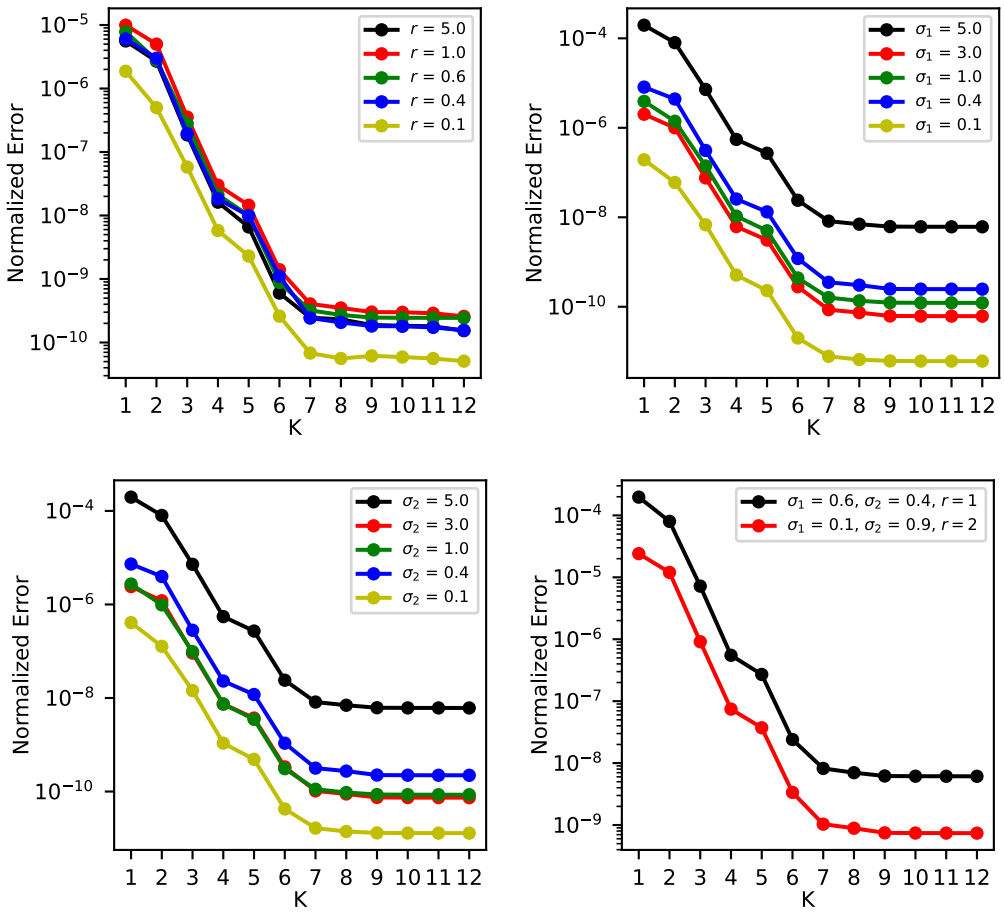


Figure 8: Convergence of PINO-Parareal for model parameters different to what the PINO was trained for. The values used for training are $\sigma_1 = 0.2$, $\sigma_2 = 0.3$ and $r = 1$. The model is very robust against changes in r. Changes in volatilities σ_1 and σ_2 have a more discernible impact but, even when far exceeding the training values, PINO-Parareal still converges fast, even though error levels are higher.

almost 60 after one and 30 after two Parareal iterations. We also show that convergence of PINO-Parareal is robust against changes in parameters or resolution without retraining the model.

Data availability.

The code for this paper is available from Zenodo [1].

Acknowledgments

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 101118139. The JU receives support from the European Union's Horizon Europe Programme.

References

[1] Abdul Qadir Ibrahim. 2024. Space-time parareal. https://doi.org/10.5281/zenodo.14334803 Accessed: May 13, 2025.

[2] Wisdom Agboh, Oliver Grainger, Daniel Ruprecht, and Mehmet Dogar. 2020. Parareal with a Learned Coarse Model for Robotic Manipulation. *Computing and Visualization in Science* 23, 8 (2020).

[3] E. Aubanel. 2011. Scheduling of Tasks in the Parareal Algorithm. *Parallel Comput.* 37 (2011), 172–182. https://doi.org/10.1016/j.parco.2010.10.004

[4] Guillaume Bal and Yvon Maday. 2002. A "Parareal" time discretization for non-linear PDE's with application to the pricing of an American Put. In *Recent Developments in Domain Decomposition Methods*, L. Pavarino and A. Toselli (Eds.). Lecture Notes in Computational Science and Engineering, Vol. 23. Springer Berlin, 189–202. https://doi.org/10.1007/978-3-642-56118-4_12

[5] Marta M. Betcke, Lisa Maria Kreusser, and Davide Murari. 2024. Parallel-in-Time Solutions with Random Projection Neural Networks. (2024). http://arxiv.org/abs/2408.09756v1

[6] Fischer Black and Myron Scholes. 1973. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy* 81, 3 (1973), 637–654. https://doi.org/10.1086/260062

[7] Dask core developers. 2022. dask. https://www.dask.org/ Accessed 25 March 2024.

[8] Matthew Emmett and Michael L. Minion. 2012. Toward an Efficient Parallel in Time Method for Partial Differential Equations. *Communications in Applied Mathematics and Computational Science* 7 (2012), 105–132. https://doi.org/10.2140/camcos.2012.7.105

[9] R. D. Falgout, S. Friedhoff, T. V. Kolev, Scott P. MacLachlan, and Jacob B. Schroder. 2014. Parallel time integration with multigrid. *SIAM Journal on Scientific Computing* 36 (2014), C635–C661. Issue 6. https://doi.org/10.1137/130944230

[10] Rui Fang and Richard Tsai. 2023. Stabilization of parareal algorithms for long time computation of a class of highly oscillatory Hamiltonian flows using data.

arXiv:2309.01225v1 [math.NA].

[11] Zhiwei Fang, Sifan Wang, and Paris Perdikaris. 2024. Learning Only on Boundaries: A Physics-Informed Neural Operator for Solving Parametric Partial Differential Equations in Complex Geometries. *Neural Computation* 36, 3 (2024), 475–498.

[12] Martin J. Gander and Stefan Vandewalle. 2007. Analysis of the Parareal Time-Parallel Time-Integration Method. *SIAM Journal on Scientific Computing* 29, 2 (2007), 556–578. https://doi.org/10.1137/05064607X

[13] Guglielmo Gattiglio, Lyudmila Grigoryeva, and Massimiliano Tamborrino. 2024. RandNet-Parareal: a time-parallel PDE solver using Random Neural Networks. (2024). http://arxiv.org/abs/2411.06225v1

[14] Alan Genz. 2004. Numerical computation of rectangular bivariate and trivariate normal and t probabilities. *Statistics and Computing* 14, 3 (Aug. 2004), 251–260. https://doi.org/10.1023/b:stco.0000035304.20635.31

[15] Olga Gorynina, Frederic Legoll, Tony Lelievre, and Danny Perez. 2022. Combining machine-learned and empirical force fields with the parareal algorithm: application to the diffusion of atomistic defects. (2022).

[16] Somdatta Goswami, Aniruddha Bora, Yue Yu, and George Em Karniadakis. 2023. Physics-informed deep neural operator networks. In *Machine Learning in Modeling and Simulation: Methods and Applications*. Springer, 219–254.

[17] Thomas J. Grady, Rishi Khan, Mathias Louboutin, Ziyi Yin, Philipp A. Witte, Ranveer Chandra, Russell J. Hewett, and Felix J. Herrmann. 2023. Model-parallel Fourier neural operators as learned surrogates for large-scale parametric PDEs. *Computers & Geosciences* 178 (2023), 105402. https://doi.org/10.1016/j.cageo.2023.105402

[18] Xian-Ming Gu, Jun Liu, and Cornelis W. Oosterlee. 2024. Parallel-in-Time Iterative Methods for Pricing American Options. (2024). http://arxiv.org/abs/2405.08280v1

[19] Gaurav Gupta, Xiongye Xiao, and Paul Bogdan. 2021. Multiwavelet-based operator learning for differential equations. *Advances in neural information processing systems* 34 (2021), 24048–24062.

[20] Stefanie Günther, Lars Ruthotto, Jacob B. Schroder, Eric C. Cyr, and Nicolas R. Gauger. 2020. Layer-Parallel Training of Deep Residual Neural Networks. *SIAM Journal on Mathematics of Data Science* 2, 1 (2020), 1–23. https://doi.org/10.1137/19m1247620

[21] Abdul Qadir Ibrahim, Sebastian Götschel, and Daniel Ruprecht. 2023. Parareal with a Physics-Informed Neural Network as Coarse Propagator. In *Euro-Par 2023: Parallel Processing*. Springer Nature Switzerland, 649–663. https://doi.org/10.1007/978-3-031-39698-4_44

[22] Bangti Jin, Qingle Lin, and Zhi Zhou. 2023. Learning Coarse Propagators in Parareal Algorithm. arXiv:2311.15320v1 [math.NA].

[23] Joonglee Jo and Yongsik Kim. 2013. Comparison of numerical schemes on multidimensional black-scholes equations. *Bulletin of the Korean Mathematical Society* 50 (2013). https://doi.org/10.4134/BKMS.2013.50.6.2035

[24] Luis Kaiser, Richard Tsai, and Christian Klingenberg. 2024. Efficient Numerical Wave Propagation Enhanced By An End-to-End Deep Learning Model. arXiv:2402.02304 [math.AP]. arXiv:2402.02304 [math.AP] arXiv:2402.02304 [math.AP].

[25] Sangkwon Kim, Darae Jeong, Chaeyoung Lee, and Junseok Kim. 2020. Finite Difference Method for the Multi-Asset Black–Scholes Equations. *Mathematics* 8, 3 (2020). https://doi.org/10.3390/math8030391

[26] Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG] https://arxiv.org/abs/1412.6980

[27] Chang-Ock Lee, Youngkyu Lee, and Jongho Park. 2022. A Parareal Architecture for Very Deep Convolutional Neural Networks. In *Domain Decomposition Methods in Science and Engineering XXVI*, Susanne C. Brenner, Eric Chung, Axel Klawonn, Felix Kwok, Jinchao Xu, and Jun Zou (Eds.). Springer International Publishing, Cham, 407–415.

[28] Zongyi Li, Daniel Zhengyu Huang, Burigede Liu, and Anima Anandkumar. 2023. Fourier neural operator with learned deformations for PDEs on general geometries. *Journal of Machine Learning Research* 24, 388 (2023), 1–26.

[29] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. 2021. Fourier Neural Operator for Parametric Partial Differential Equations. arXiv:2010.08895 [cs.LG]. arXiv:2010.08895 [cs.LG]

[30] Zongyi Li, Nikola B. Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew M. Stuart, and Anima Anandkumar. 2020. Fourier Neural Operator for Parametric Partial Differential Equations. *CoRR* abs/2010.08895 (2020). arXiv:2010.08895 https://arxiv.org/abs/2010.08895

[31] Zongyi Li, Hongkai Zheng, Nikola Kovachki, David Jin, Haoxuan Chen, Burigede Liu, Kamyar Azizzadenesheli, and Anima Anandkumar. 2021. Physics-informed neural operator for learning partial differential equations. *ACM/JMS Journal of Data Science* (2021).

[32] J.-L. Lions, Yvon Maday, and Gabriel Turinici. 2001. A "parareal" in time discretization of PDE's. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics* 332 (2001), 661–668. https://doi.org/10.1016/S0764-4442(00)01793-6

[33] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. 2021. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence* 3, 3 (2021), 218–229. https://doi.org/10.1038/s42256-021-00302-5

[34] Xuhui Meng, Zhen Li, Dongkun Zhang, and George Em Karniadakis. 2020. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *Computer Methods in Applied Mechanics and Engineering* 370 (2020), 113250. https://doi.org/10.1016/j.cma.2020.113250

[35] Robert C. Merton. 1973. Theory of Rational Option Pricing. *The Bell Journal of Economics and Management Science* 4, 1 (1973), 141–183. http://www.jstor.org/stable/3003143

[36] Navaneeth N., Tapas Tripura, and Souvik Chakraborty. 2024. Physics informed WNO. *Computer Methods in Applied Mechanics and Engineering* 418 (2024), 116546. https://doi.org/10.1016/j.cma.2023.116546

[37] Hieu Nguyen and Richard Tsai. 2023. Numerical wave propagation aided by deep learning. *J. Comput. Phys.* 475 (2023), 111828. https://doi.org/10.1016/j.jcp.2022.111828

[38] G. Pagès, O. Pironneau, and G. Sall. 2018. The Parareal Algorithm for American Options. *SIAM Journal on Financial Mathematics* 9, 3 (2018), 966–993. https://doi.org/10.1137/17M1138832

[39] S.J.P. Pamela, N. Carey, J. Brandstetter, R. Akers, L. Zanisi, J. Buchanan, V. Gopakumar, M. Hoelzl, G. Huijsmans, K. Pentland, T. James, and G. Antonucci. 2025. Neural-Parareal: Self-improving acceleration of fusion MHD simulations using time-parallelisation and neural operators. *Computer Physics Communications* 307 (2025), 109391. https://doi.org/10.1016/j.cpc.2024.109391

[40] Daniel Ruprecht. 2017. *Shared Memory Pipelined Parareal*. Springer International Publishing, 669–681. https://doi.org/10.1007/978-3-319-64203-1_48

[41] Jacob Seidman, Georgios Kissas, Paris Perdikaris, and George J Pappas. 2022. NOMAD: Nonlinear manifold decoders for operator learning. *Advances in Neural Information Processing Systems* 35 (2022), 5601–5613.

[42] Robert Speck, Daniel Ruprecht, Matthew Emmett, Matthias Bolten, and Rolf Krause. 2014. A space-time parallel solver for the three-dimensional heat equation. In *Parallel Computing: Accelerating Computational Science and Engineering (CSE) (Advances in Parallel Computing, Vol. 25)*. IOS Press, 263–272. https://doi.org/10.3233/978-1-61499-381-0-263

[43] Robert Speck, Daniel Ruprecht, Rolf Krause, Matthew Emmett, Michael L. Minion, Mathias Winkel, and Paul Gibbon. 2012. A massively space-time parallel N-body solver. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis* (Salt Lake City, Utah) *(SC '12)*. IEEE Computer Society Press, Los Alamitos, CA, USA, Article 92, 11 pages. https://doi.org/10.1109/SC.2012.6

[44] Sifan Wang, Hanwen Wang, and Paris Perdikaris. 2021. Learning the solution operator of parametric partial differential equations with physics-informed DeepONets. *Science advances* 7, 40 (2021), eabi8605.

[45] Gopal R. Yalla and Bjorn Engquist. 2018. Parallel in Time Algorithms for Multiscale Dynamical Systems Using Interpolation and Neural Networks. In *Proceedings of the High Performance Computing Symposium (HPC '18)*. Society for Computer Simulation International, Article 9, 9:1–9:12 pages.

[46] Stefano Zampini, Umberto Zerbinati, George Turkyyiah, and David Keyes. 2024. PETScML: Second-Order Solvers for Training Regression Problems in Scientific Machine Learning. In *Proceedings of the Platform for Advanced Scientific Computing Conference* (Zurich, Switzerland) *(PASC '24)*. Association for Computing Machinery, New York, NY, USA, Article 17. https://doi.org/10.1145/3659914.3659931