Modified parareal method for solving the two-dimensional nonlinear shallow water equations using finite volumes

Joao Guilherme Caldas Steinstraesser, Vincent Guinot, Antoine Rousseau

Journal of Computational Mathematics
Vol. 00, Pages 000–000 (XXXX)

Modified parareal method for solving the two-dimensional nonlinear shallow water equations using finite volumes

Joao G. Caldas Steinstraesser [1]
Vincent Guinot [2]
Antoine Rousseau [3]

[1] Inria, IMAG, Univ Montpellier, CNRS, Montpellier, France
Email address: joao-guilherme.caldas-steinstraesser@inria.fr
[2] Univ Montpellier, HSM, CNRS, IRD, Inria, Montpellier, France
Email address: vincent.guinot@inria.fr
[3] Inria, IMAG, Univ Montpellier, CNRS, Montpellier, France
Email address: antoine.rousseau@inria.fr.

Abstract. In this work, the POD-DEIM-based parareal method introduced in [6] is implemented for the resolution of the two-dimensional nonlinear shallow water equations using a finite volume scheme. This method is a variant of the traditional parareal method, first introduced by [19], that improves the stability and convergence for nonlinear hyperbolic problems, and uses reduced-order models constructed via the Proper Orthogonal Decomposition - Discrete Empirical Interpolation Method (POD-DEIM) applied to snapshots of the solution of the parareal iterations. We propose a modification of this parareal method for further stability and convergence improvements. It consists in enriching the snapshots set for the POD-DEIM procedure with extra snapshots whose computation does not require any additional computational cost. The performances of the classical parareal method, the POD-DEIM-based parareal method and our proposed modification are compared using numerical tests with increasing complexity. Our modified method shows a more stable behaviour and converges in fewer iterations than the other two methods.

Keywords. Parareal method, POD-DEIM, reduced-order model, finite volume, shallow water equations.

Math. classification. 68W10; 76B07.

1. Introduction

The trade-off between high-fidelity results and affordable computational costs is a real challenge encountered by many mathematical models for physical problems. Computing approximate solutions close enough to analytical or experimental results usually requires fine temporal and/or spatial discretizations, which may be linked by stability conditions, and high-order numerical schemes. The combination of these factors may lead the computational time and memory cost to a prohibitive magnitude, limiting potential applications of these mathematical models.

Over the past two decades, the parareal methods have arisen as an effective strategy for overcoming part of such challenge. First developed by [19], this problem-independent class of numerical method, that stands for "parallel in real-time", consists of an iterative predictor-corrector algorithm that uses two numerical discretizations (the so-called propagators): a fine one (that gives the desired precision, but is too expensive) and a coarse one (that is less precise but much cheaper). The key point of the method is that the expensive computation of the fine propagator is done within time windows (chosen as the time steps of the coarse propagator) that can be parallelized.

The parareal method has been applied to a wide range of problems. Examples are fluid dynamics [27, 14], molecular dynamics [4], chemistry [11], geodynamics [26] and finance [22]. The popularity of the method is explained by the fast convergence and substantial reduction of the computational cost in many problems, especially parabolic ones. However, when hyperbolic problems are dealt with,

even very simple ones such as the one-dimensional advection equation, the method suffers from a slow convergence and a lack of stability. It is thus of limited practical interest for this type of applications [24].

A number of alternatives are available from the literature. One of them is the Krylov-subspace-enhanced parareal method, introduced in [16] and implemented in [25] for solving a linear acoustic-advection system. This parareal variant enriches the coarse predictor by computing subspaces generated by the solutions of the previous parareal iterations, thus keeping the information generated during the simulation. It is an effective way for improving the stability and the convergence for hyperbolic problems. However, it is restricted to linear problems, since the fine propagator is introduced in the sequential predictor step and needs a linearity assumption to be efficiently computed [25, 6].

An alternative was proposed in [6], inspired by the Krylov-subspace-enhanced parareal method but using reduced-order models (ROMs). ROMs are, in themselves, an approach to the challenge presented in the beginning of this introduction. Their purpose is to replace an expensive, high-dimensional problem by a low-dimensional one. Proper Orthogonal Decomposition (POD) is widely used in ROM approaches. It consists in projecting the problem onto a truncated low-dimensional space spanned by snapshots of the solution. POD is very effective for linear problems; combining it with the Discrete Empirical Interpolation Method (DEIM) allows for its successful extension to nonlinear problems [5]. In the parareal method proposed in [6], the coarse propagator is replaced with a POD-DEIM-based ROM.

In this work, we solve the nonlinear two-dimensional shallow water equations (SWE), discretized with a finite volume scheme, using parareal methods. This set of depth-averaged flow equations, used in many applications, from urban floods [17] to atmospheric circulation [29], may have prohibitive computational costs due to the Courant-Friedrichs-Lewy (CFL) stability condition in explicit schemes. We propose a slight modification of the POD-DEIM-based method proposed in [6] for improving the speed of convergence and the stability. This modification consists in enriching the snapshot sets used as inputs for the POD-DEIM procedures. It does not require extra computional cost for computing the additional snapshots. However, it increases the cost of the POD-DEIM itself, and the method should provide convergence in very few iterations.

This paper is structured as follows: in Section 2, we present an overview of some parareal methods, focusing in their evolution towards the algorithm adapted for nonlinear hyperbolic problems (the POD-DEIM-based parareal method); in Section 3, we briefly discuss the impact of the snapshot set on the quality of the reduced-order models and we propose our modification of the POD-DEIM-based parareal method for enriching the computed ROMs; a speedup estimation is presented in 4; numerical tests are reported in Section 5, focusing on the comparison of three parareal methods, the classical one, the POD-DEIM-based and our proposed modification, in terms of convergence and computational cost; and a discussion and a conclusion are presented in Section 6. Finally, the detailed formulation of a POD-DEIM ROM for a finite volume discretization and its application to the parareal method (firstly for the inviscid scalar two-dimensional Burgers equation as a didactic example, and then for the 2D nonlinear SWE) is described in the Appendices.

2. **Overview of parareal methods**

The parareal method, first developed by [19], is an iterative predictor-corrector algorithm that involves parallel computation in time for solving time-dependent differential equations with a smaller computational cost. It is based on two time-integration solvers (also called propagators). The coarser of the two, acting as a predictor, is computed sequentially along the entire time domain. It yields an approximate solution at a relatively low computational cost. The finer propagator produces corrections

for the predicted solution. Being more expensive, it is computed in parallel within each time step of the coarse propagator [14].

The acceleration of the computational time by the parareal method is based on the joint use of a coarse and a fine time steps, since the fine computation can be parallelized using as many processors as there are coarse time steps. Note that the concept of "coarser predictor" may take many different forms in the parareal algorithm. It may also include other aspects than the temporal discretization [2], such as using a coarser spatial mesh, a less accurate numerical method, or a simplified mathematical model (by simplifying the governing equations).

The parareal method has a very simple problem-independent implementation. It is known to be computationally efficient when applied to complex diffusive problems. In contrast, it is known to be inefficient (*i.e.* it is not faster than a full fine simulation) and may become unstable when applied to hyperbolic or advection-dominated problems. This is true even for the simplest possible hyperbolic problems, such as the 1D linear advection equation with a constant speed. This problem has been identified and studied in many works, *e.g.* [15, 9, 24]; [24] indicates that this lack of stability is due to the different discrete phase speeds in the coarse and fine propagators.

Many works, *e.g.* [9, 25, 6, 12], propose adaptations and developments of the parareal method in order to improve its performance when applied to hyperbolic problems. The present work focuses on the method introduced by [25] and its adaptation proposed by [6]. [25] presents a variation of the algorithm whereby the coarse propagator is enriched by constructing Krylov subspaces generated by the solution computed during the iterations. This method, known as Krylov-subspace-enhanced parareal method, is able to improve the stability and performance for hyperbolic problems. This, however, is true only for linear ones, since the fine propagator is introduced in the sequential prediction step and can only be parallelized by decomposing it as a linear combination of the fine propagation of the Krylov space's basis vectors. In order to cope with nonlinear hyperbolic problems, [6] modified the method by introducing reduced order models (ROMs) for replacing the coarse propagator of the Krylov-subspace-enhanced method.

In this section we briefly present the evolution of the parareal algorithms through time, focusing on the methods mentioned above and that led to the application to nonlinear hyperbolic problems. In each algorithm presented, we highlight the main modifications w.r.t. to the previous algorithm.

2.1. **The original parareal method**

We follow [14] by presenting the original or classical parareal algorithm applied to a simple problem.

We consider the following time-dependent problem

$$\begin{cases} \frac{d}{dt}\boldsymbol{y}(t) + A\boldsymbol{y}(t) = 0, \quad \text{in } [0,T] \\ \boldsymbol{y}(0) = \boldsymbol{y}_0 \end{cases} \tag{2.1}$$

where A is assumed time-independent for the sake of simplicity.

Let us define two discrete integration schemes $\mathcal{F}_{\delta t}$ and $\mathcal{G}_{\Delta t}$, called *fine* and *coarse propagators* respectively, that solve (2.1) numerically. They use fixed time steps (denoted respectively by δt and Δt) for the temporal integration of the problem. $\Delta t \gg \delta t$, therefore a full time integration over $[0,T]$ is much cheaper using the coarse propagator than using the fine one. We also assume that $p := \Delta t/\delta t$ is an integer. As mentioned above, $\mathcal{G}_{\Delta t}$ can also be characterized by a coarser spatial discretization, a cheaper integration scheme or a simplified mathematical model. When convenient, we denote by M_f and M_c the size of the spatial discretization (*e.g.* the number of cells in a finite volume scheme) of $\mathcal{F}_{\delta t}$ and $\mathcal{G}_{\Delta t}$, respectively.

Consider a homogeneous temporal discretization of $[0,T]$ using a constant time step Δt and resulting in $N_{\Delta t}+1$ discrete instants:

$$t_n = n\Delta t, \qquad n = 0, \ldots, N_{\Delta t}$$

with $T = t_{N_{\Delta t}}$. Following the notation in [25], we denote the propagation of a solution $\boldsymbol{y}$ from t_0 to t_1 by $\mathcal{F}_{\delta t}$, using a constant time step δt, and by $\mathcal{G}_{\Delta t}$, using a constant time step Δt, respectively by

$$\mathcal{F}_{\delta t}(\boldsymbol{y}, t_1, t_0), \qquad \mathcal{G}_{\Delta t}(\boldsymbol{y}, t_1, t_0)$$

The computation of approximate fine solutions $\boldsymbol{y}_n \approx \boldsymbol{y}(t_n)$ defined on the instants of the coarse temporal discretization, with $\boldsymbol{y}_n := \mathcal{F}_{\delta t}(\boldsymbol{y}_{n-1}, t_n, t_{n-1}) = \mathcal{F}_{\delta t}(\boldsymbol{y}_0, t_n, 0)$, can be very expensive in terms of computational time, since it must be done sequentially along the fine time steps. The parareal algorithm allows this problem to be overcome by introducing an iterative predictor-corrector procedure that uses the coarser propagator $\mathcal{G}_{\Delta t}$:

$$\boldsymbol{y}_{n+1}^{k+1} = \underbrace{\mathcal{G}_{\Delta t}(\boldsymbol{y}_n^{k+1}, t_{n+1}, t_n)}_{\text{Prediction}} + \underbrace{\mathcal{F}_{\delta t}(\boldsymbol{y}_n^k, t_{n+1}, t_n) - \mathcal{G}_{\Delta t}(\boldsymbol{y}_n^k, t_{n+1}, t_n)}_{\text{Correction}} \tag{2.2}$$

where the superscript k indicates the iteration number.

Notice that, given a previously computed solution $(\boldsymbol{y}_n^k)_{n=0}^{N_{\Delta t}}$ in all instants of the coarse temporal discretization during iteration k, the only term in (2.2) that must be done sequentially for the computation of the next iteration's solution is the coarse prediction (which is supposed to be cheaper). On the other hand, the correction terms in each coarse time step, including the expensive fine propagator, can be done in parallel, since they depend on $(\boldsymbol{y}_n^k)_{n=0}^{N_{\Delta t}}$. Therefore, assuming that there are as much processors as coarse time steps for the simulation, each parallel processor performs per iteration only p fine time steps using the fine propagator $\mathcal{F}_{\delta t}$ (instead of $pN_{\Delta t}$ time steps as in the full fine sequential simulation).

To make it clearer, the iterative process (2.2) is described in detail in Algorithm 1. The steps of the algorithm are named with "prediction" and "correction" interpreted as in (2.2), following [14]. It differs from the nomenclature adopted in [25] and [6]. This same remark is made to the other parareal algorithms presented in this paper, and also presented in [25] and [6]. A discussion on the different interpretations of the predictor-corrector procedure in the parareal method is proposed in [20].

The convergence criterion mentioned in Algorithm 1 can be based on the distance between the correction term and the iteration's solution, as proposed by [2]:

$$\varepsilon_n^k := \frac{\left\| \widetilde{\boldsymbol{y}}_n^k - \boldsymbol{y}_n^k \right\|}{\|\boldsymbol{y}_n^k\|} < \varepsilon_{\text{TOL}} \tag{2.3}$$

We notice that at each iteration the number of timesteps to be computed decreases. As in [2], we find, for each iteration of Algorithm 1, the first time step t_{n_0} not satisfying the convergence criterion. In the next iteration the solution is updated only for $t \geq t_{n_0}$. Indeed, by construction, at iteration k the parareal algorithm provides exact convergence to the fine referential solution at time t_k [15]. However, the algorithm is interesting in terms of computational time only if the convergence is obtained within a number of iterations k_{cvg} that is substantially smaller than $N_{\Delta t}$: if $k_{\text{cvg}} = N_{\Delta t}$, the parareal simulation is necessarily more expensive than the full fine sequential one.

2.2. The Krylov-subspace-enhanced parareal algorithm: an adaptation for hyperbolic problems

As discussed above, the slow convergence and instability of the original parareal Algorithm 1 are well-known difficulties for its application to hyperbolic problems. Based on modified parallel implicit time-integration algorithms (PITAs) introduced in [13] and [8] for solving hyperbolic problems in structural dynamics, and on their interpretation as parareal algorithms in [16], [25] presents a stable

Initialization: initial guess given by the coarse propagator:
$\boldsymbol{y}_0^0 = \boldsymbol{y}_0$
for $n \leftarrow 0$ **to** $N_{\Delta t} - 1$ **do**
  $\boldsymbol{y}_{n+1}^0 = \mathcal{G}_{\Delta t}(\boldsymbol{y}_n^0, t_{n+1}, t_n)$
end

$n_0 = 0$
Iterations:
for $k \leftarrow 0$ **to** $N_{itermax}$ **do**
  Compute the fine term of the correction (**in parallel**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\widetilde{\boldsymbol{y}}_{n+1}^k = \mathcal{F}_{\delta t}(\boldsymbol{y}_n^k, t_{n+1}, t_n)$
  end

  Compute the coarse predictions and correct them to obtain the final solution in the iteration (**sequentially**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\boldsymbol{y}_{n+1}^{k+1} = \mathcal{G}_{\Delta t}(\boldsymbol{y}_n^{k+1}, t_{n+1}, t_n) + \widetilde{\boldsymbol{y}}_{n+1}^k - \mathcal{G}_{\Delta t}(\boldsymbol{y}_n^k, t_{n+1}, t_n)$
  end

  Find the last instant $\tilde{n} \in \{1, \ldots, N_{\Delta t}\}$ not satisfying a convergence criterion
  $n_0 \leftarrow \tilde{n} - 1$
  if *all instants converged* **then**
    break;
  end
end

Algorithm 1: Original parareal algorithm

variation of the parareal method with an accelerated convergence for this kind of problems, applying it to solve a linear acoustic-advection system.

In this variation of the method, the coarse propagator $\mathcal{G}_{\Delta t}$ in the $k-$th iteration (2.2) is replaced with the operator

$$\mathcal{K}_{\Delta t}^k(\boldsymbol{y}, t_1, t_0) := \mathcal{G}_{\Delta t}((\boldsymbol{I} - \boldsymbol{P}^k)\boldsymbol{y}, t_1, t_0) + \mathcal{F}_{\delta t}(\boldsymbol{P}^k\boldsymbol{y}, t_1, t_0) \tag{2.4}$$

where $\boldsymbol{I}$ is the identity operator and $\boldsymbol{P}^k$ is the projection operator onto the subspace $\mathcal{S}^k$ spanned by all the solutions computed in the previous iterations of the parareal algorithm:

$$\mathcal{S}^k := \text{span}\{\boldsymbol{y}_n^j,\ n = 0, \ldots, N_{\Delta t} - 1,\ j = 0, \ldots, k\}$$

This method is known as *Krylov-subspace-enhanced parareal method* because the subspace $\mathcal{S}^0$, generated by shapshots of the coarse propagator, is a Krylov subspace:

$$\mathcal{S}^0 = \text{span}\{\boldsymbol{y}_0^0, \mathcal{G}_{\Delta t}\boldsymbol{y}_0^0, \mathcal{G}_{\Delta t}^2\boldsymbol{y}_0^0 \ldots, \mathcal{G}_{\Delta t}^{N_{\Delta t}}\boldsymbol{y}_0^0\} \tag{2.5}$$

even if the the spaces $\mathcal{S}^k, k \geq 1$, have a more complicated structure [16]. In eq. (2.5) we use the simplified notation $\boldsymbol{y}_n^0 = \mathcal{G}_{\Delta t}(\boldsymbol{y}_0^0, t_n, t_0) = \mathcal{G}_{\Delta t}^n\boldsymbol{y}_0^0,\ n = 0, \ldots, N_{\Delta t}$.

Therefore, this modification improves the coarse propagator by keeping the information of the previous iterations and propagating it with the fine propagator.

An useful property of the subspaces S^k is

$$\mathcal{S}^k = \mathcal{S}^{k-1} \cup \text{span}\{\boldsymbol{y}_n^k,\ n = 0, \dots N_{\Delta t} - 1\}; \tag{2.6}$$

The parareal iteration (2.2) is thus rewritten as

$$\boldsymbol{y}_{n+1}^{k+1} = \underbrace{\mathcal{K}_{\Delta t}^k(\boldsymbol{y}_n^{k+1}, t_{n+1}, t_n)}_{\text{Prediction}} + \underbrace{\mathcal{F}_{\delta t}(\boldsymbol{y}_n^k, t_{n+1}, t_n) - \mathcal{K}_{\Delta t}^k(\boldsymbol{y}_n^k, t_{n+1}, t_n)}_{\text{Correction}}$$

The Krylov-subspace-enhanced parareal method is presented in detail in Algorithm 2. The modifications w.r.t. to the classical parareal method 1 are highlighted.

Denoting by $W^k := [\boldsymbol{s}_1^k, \dots, \boldsymbol{s}_r^k] \in \mathbb{R}^{M_f \times r}$ ($r = \max\{N_{\Delta t} k, M_f\}$, with M_f the size of the fine spatial discretization of the solution $\boldsymbol{y}$) an orthonormal basis of $\mathcal{S}^k$, the projection operator can be written as $\boldsymbol{P}^k = W^k (W^k)^T$. The coordinates of $\boldsymbol{P}^k \boldsymbol{y}$ w.r.t. $\mathcal{S}^k$ are $C^k := (W^k)^T \boldsymbol{y} = [C_1^k, \dots, C_r^k]^T \in \mathbb{R}^r$.

The computation of the subspace $\mathcal{S}^k$ can be done using the update (2.6), via a Gram-Schmidt orthonormalization process for taking into account the new input vectors. However, as pointed out by [25], this procedure is instable and a full QR decomposition is preferable.

Notice, in the definition of the propagator $\mathcal{K}_{\Delta t}^k$ (eq. 2.4) and in Algorithm 2, that the fine propagator $\mathcal{F}_{\delta t}$ is introduced in the sequential prediction step, which makes each iteration more expensive than the full fine sequential simulation. However, in the case of linear problems, we can avoid this difficulty by writing the solution in the basis $\mathcal{S}^k$:

$$\mathcal{F}_{\delta t}(\boldsymbol{P}^k \boldsymbol{y}, t_1, t_0) = \mathcal{F}_{\delta t}\left(\sum_{j=1}^{r} C_j^k \boldsymbol{s}_j^k, t_1, t_0\right) = \sum_{j=1}^{r} C_j^k \mathcal{F}_{\delta t}(\boldsymbol{s}_j^k, t_1, t_0); \tag{2.7}$$

Since the vectors $\boldsymbol{s}_j^k$ do not depend on time, eq. (2.7) provides an efficient computation in the case of linear problems since it suffices, in each parareal iteration, to propagate the vectors of the basis over one single coarse time step Δt, and update the projection coefficients at each instant of the coarse temporal discretization by $[C_1^{k,n}, \dots, C_r^{k,n}]^T =: \boldsymbol{C}^{k,n} = (W^k)^T \boldsymbol{y}_n^k$.

2.3. **The POD-DEIM-based parareal algorithm for nonlinear problems**

Although it is capable of speeding up and stabilizing the parareal solution of hyperbolic problems, the Krylov-subspace-enhanced parareal algorithm described in the previous section suffers from some limitations [6]. Firstly, the number of vectors in the basis increases linearly with the number of iterations ($\boldsymbol{S}^k$ is spanned by $N_{\Delta t} k$ vectors) and therefore the number of fine propagations $\mathcal{F}_{\delta t}(\boldsymbol{s}_j^k, t_1, t_0)$ also increases linearly. Secondly, the interest of this method in terms of computational cost w.r.t. the full fine sequential simulation relies on a linearity assumption, which allows the efficient application of Eq. (2.7). Thus, the algorithm is not efficient for nonlinear problems.

Reduced basis approaches are proposed in [6] for overcoming such limitations, by reducing the dimension of large scale ODEs (or semi-discretized time dependent PDEs). Among these approaches, one that improves the efficiency for nonlinear problems is the Proper Orthogonal Decomposition - Discrete Empirical Interpolation Method (POD-DEIM). It is briefly described hereafter, based on [5].

2.3.1. *The POD-DEIM reduction method*

Consider the following system of nonlinear ODEs, that can be obtained by the spatial discretization of a scalar PDE:

$$\frac{d}{dt}\boldsymbol{y}(t) = A\boldsymbol{y}(t) + \boldsymbol{F}(\boldsymbol{y}(t)) \tag{2.8}$$

Initialization: initial guess given by the coarse propagator:
$\boldsymbol{y}_0^0 = \boldsymbol{y}_0$
for $n \leftarrow 0$ **to** $N_{\Delta t} - 1$ **do**
  $\boldsymbol{y}_{n+1}^0 = \mathcal{G}_{\Delta t}(\boldsymbol{y}_n^0, t_{n+1}, t_n)$
end

$n_0 = 0$
Iterations:
for $k \leftarrow 0$ **to** $N_{itermax}$ **do**
  Compute the fine term of the correction (**in parallel**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\widetilde{\boldsymbol{y}}_{n+1}^k = \mathcal{F}_{\delta t}(\boldsymbol{y}_n^k, t_{n+1}, t_n)$
  end

  Compute the subspace
  $\mathcal{S}^k = \mathcal{S}^{k-1} \cup \text{span}\{\boldsymbol{y}_n^k,\ n = 0, \dots N_{\Delta t} - 1\}$

  Compute the coarse term of the correction and the final correction term(**in parallel**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\overline{\boldsymbol{y}}_{n+1}^k = \mathcal{K}_{\Delta t}^k(\boldsymbol{y}_n^k, t_{n+1}, t_n)$
    $\overline{\overline{\boldsymbol{y}}}_{n+1}^k = \widetilde{\boldsymbol{y}}_{n+1}^k - \overline{\boldsymbol{y}}_{n+1}^k$
  end

  Compute the coarse predictions and correct them to obtain the final solution in the iteration (**sequentially**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\boldsymbol{y}_{n+1}^{k+1} = \mathcal{K}_{\Delta t}^k(\boldsymbol{y}_n^{k+1}, t_{n+1}, t_n) + \overline{\overline{\boldsymbol{y}}}_{n+1}^k$
  end

  Find the last instant $\tilde{n} \in \{1, \dots, N_{\Delta t}\}$ not satisfying a convergence criterion
  $n_0 \leftarrow \tilde{n} - 1$
  if *all instants converged* **then**
    break;
  end
end

Algorithm 2: Krylov-subspace-enhanced parareal algorithm

with $\boldsymbol{y} \in \mathbb{R}^{M_f}$, $A \in \mathbb{R}^{M_f \times M_f}$ being a constant-in-time matrix and $\boldsymbol{F}$ a nonlinear function with values on $\mathbb{R}^{M_f}$. We choose here the notation M_f, referring to the fine discretization, to show that this quantity is possibly too large, in which case solving (2.8) may be expensive. Thus, one may seek to approximate it by a problem with smaller dimension.

Reduction of the linear term.

The original system (2.8) is approximated with a reduced-order system of order $q \ll M_f$, via the construction of a subspace $\mathcal{S}_q$ of $\mathbb{R}^{M_f}$ with dimension q.

Let $V_q = [\phi_1, \ldots, \phi_q] \in \mathbb{R}^{M_f \times q}$ be a matrix whose columns are the vectors of an orthonormal basis of $\mathcal{S}_q$. The reduction of the system (2.8) consists in replacing $\boldsymbol{y}(t)$ by its approximation $V_q\tilde{\boldsymbol{y}}(t)$ (with $\tilde{\boldsymbol{y}} \in \mathbb{R}^q$) and projecting the system onto $\mathcal{S}_q$:

$$\frac{d}{dt}\tilde{\boldsymbol{y}}(t) = V_q^T A V_q \tilde{\boldsymbol{y}}(t) + V_q^T \boldsymbol{F}(V_q\tilde{\boldsymbol{y}}(t)) \tag{2.9}$$

The reduced basis vectors $\{\phi_i\}_{i=1}^q$ of $\mathcal{S}_q$ can be obtained via a Proper Orthogonal Decomposition (POD) of a snapshot matrix $Y = [\boldsymbol{y}(t_1), \ldots, \boldsymbol{y}(t_{n_s})] \in \mathbb{R}^{M_f \times n_s}$, containing the solution in chosen n_s instants. The subspace $\mathcal{Y} \subset \mathbb{R}^{M_f}$ spanned by the vectors in Y has dimension $r \leq \min(M_f, n_s)$, and the subspace of dimension $q < r$ spanned by the vectors $\{\phi_i\}_{i=1}^q$ best approximates $\mathcal{Y}$ in the sense that $\{\phi_i\}_{i=1}^q$ solve the minimization problem

$$\begin{aligned} &\min_{\{\phi_i\}_{i=1}^q} \sum_{j=1}^{n_s} \left\| \boldsymbol{y}(t_j) - \sum_{i=1}^{q} ((\boldsymbol{y}(t_j))^T \phi_i)\phi_i \right\|_2^2 \\ &\phi_i^T \phi_j = \delta_{ij}, \qquad i,j = 1, \ldots, q \end{aligned} \tag{2.10}$$

where $\|\cdot\|_2$ is the Euclidian norm and δ_{ij} is the Kronecker delta function.

The solution for (2.10) is given by the left singular vectors of the snapshot matrix Y, so it can be obtained via its Singular Value Decomposition (SVD):

$$Y = V\Sigma W^T$$

where $V = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_r] \in \mathbb{R}^{M_f \times r}$ and $W = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_r] \in \mathbb{R}^{n_s \times r}$ are respectively the left and right singular vectors of V, and $\Sigma = \mathrm{diag}(\sigma_1, \ldots, \sigma_r) \in \mathbb{R}^{r \times r}, \sigma_1 \geq \cdots \geq \sigma_r > 0$ is a diagonal matrix containing their respective singular values. The SVD gives r left singular vectors, but for the POD we keep the first q ones (*i.e.* those with the largest singular values) that contain the most significant part of the information for describing $\mathcal{Y}$. A threshold $\varepsilon_{\mathrm{sv}}$ must be set for choosing the singular values to be kept.

If the nonlinear term $V_q^T \boldsymbol{F}(V_q\tilde{\boldsymbol{y}}(t))$ in (2.9) is neglected, it is effectively a reduced problem w.r.t. to the original one (2.8) ($q \ll M_f$), since the matrix $V_k^T A V_k$ does not depend on time and can be precomputed.

Nevertheless, the nonlinear term limits the reduction of the model, since it must be computed on M_f points at each time step (because $V_k\tilde{\boldsymbol{y}}(t) \in \mathbb{R}^{M_f}$). Therefore, a separate reduction approach must be defined for the nonlinear term.

Reduction of the nonlinear term.

The limitation of the reduction technique for nonlinear problems can be overcome by combining the POD method with the Discrete Empirical Interpolation Method (DEIM), introduced by [5]. This method selects a small set of $m \ll M_f$ points for computing and interpolating the approximation of the nonlinear term in a reduced subspace $\widehat{\mathcal{S}}_m$ of $\mathbb{R}^{M_f}$ with dimension m, spanned by snapshots of the nonlinear term. For the sake of simplicity, we follow [5] by describing the reduction of the nonlinear term in (2.8) considering a generic nonlinear function $\boldsymbol{f}(t)$ with values in $\mathbb{R}^{M_f}$.

Let $\widehat{\mathcal{S}}_m$ be a reduced subspace of $\mathbb{R}^{M_f}$ generated by snapshots $\widehat{Y} = [\boldsymbol{f}(t_1), \ldots, \boldsymbol{f}((t_{n_s})] \in \mathbb{R}^{M_f \times n_s}$ of the nonlinear term, with dimension $m \ll M_f$. As above, this space can be obtained via a POD technique. We denote by $\widehat{V}_m = [\widehat{\phi}_1, \ldots, \widehat{\phi}_m] \in \mathbb{R}^{M_f \times m}$ the matrix whose columns are the vectors of a orthonormal basis of $\widehat{\mathcal{S}}_m$. Then, we approximate the nonlinear function with

$$\boldsymbol{f}(t) \approx \widehat{V}_m \boldsymbol{c}(t) \tag{2.11}$$

where $\boldsymbol{c}(t) \in \mathbb{R}^m$ is a vector of coefficients that can be determined uniquely by selecting m rows with indices $\mathcal{P}_1, \ldots, \mathcal{P}_m$ from the overdetermined system

$$\boldsymbol{f}(t) = \widehat{V}_m \boldsymbol{c}(t)$$

The resulting linear systems reads

$$\widehat{P}^T \boldsymbol{f}(t) = \left(\widehat{P}^T \widehat{V}_m\right) \boldsymbol{c}(t) \tag{2.12}$$

in which we define a matrix $\widehat{P} = [\boldsymbol{e}_{\mathcal{P}_1}, \ldots, \boldsymbol{e}_{\mathcal{P}_m}] \in \mathbb{R}^{M_f \times m}$, where $\boldsymbol{e}_i$ are vectors of the canonical basis of $\mathbb{R}^{M_f}$.

Supposing $\widehat{P}^T \widehat{V}_m$ to be nonsingular, by solving (2.12) for $\boldsymbol{c}(t)$ and replacing it in (2.11), we obtain the approximation

$$\boldsymbol{f}(t) \approx \widehat{V}_m \left(\widehat{P}^T \widehat{V}_m\right)^{-1} \widehat{P}^T \boldsymbol{f}(t) \tag{2.13}$$

Equation (2.13) shows that we can approximate the nonlinear function $\boldsymbol{f}(t) \in \mathbb{R}^{M_f}$ by computing it at $m \ll M_f$ points and interpolating over the full set of M_f points.

Using (2.13) in (2.9), we obtain the POD-DEIM reduced model for (2.8):

$$\frac{d}{dt}\tilde{\boldsymbol{y}}(t) = V_q^T A V_q \tilde{\boldsymbol{y}}(t) + V_q^T \widehat{V}_m \left(\widehat{P}^T \widehat{V}_m\right)^{-1} \widehat{P}^T \boldsymbol{F}(V_q \tilde{\boldsymbol{y}}(t)) \tag{2.14}$$

We notice that the matrix $V_q^T \widehat{V}_m \left(\widehat{P}^T \widehat{V}_m\right)^{-1} \in \mathbb{R}^{q \times m}$ does not depend on time and can be precomputed, and that $\widehat{P}^T \boldsymbol{F}(V_q \tilde{\boldsymbol{y}}(t))$ is a vector of size $m \ll M_f$. If $\boldsymbol{F}$ is evaluated pointwise, we can also rewrite the reduced nonlinear term as $V_q^T \widehat{V}_m \left(\widehat{P}^T \widehat{V}_m\right)^{-1} \boldsymbol{F}(\widehat{P}^T V_q \tilde{\boldsymbol{y}}(t))$.

The indices $\mathcal{P}_1, \ldots, \mathcal{P}_m$ for computing the nonlinear term are chosen iteratively by the discrete empirical interpolation method (DEIM), described in Algorithm 3.

Input: vectors $\{\widehat{\phi}_l\}_{l=1}^m \subset \mathbb{R}^{M_f}$ of a POD reduced basis obtained from the nonlinear snapshots
Output: the indices $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_m)^T \in \mathbb{R}^m$
$\mathcal{P}_1 = \text{argmax}\{|\widehat{\phi}_1|\}$
$\widehat{V} = [\widehat{\phi}_1]$, $\widehat{P} = [\boldsymbol{e}_{\mathcal{P}_1}]$, $\mathcal{P} = [\mathcal{P}_1]$
for $l \leftarrow 2$ **to** m **do**
    Solve $(\widehat{P}^T \widehat{V})\boldsymbol{c} = \widehat{P}^T \widehat{\phi}_l$ for $\boldsymbol{c}$
    $\mathcal{P}_l = \text{argmax}\{|\widehat{\phi}_l - \widehat{V}\boldsymbol{c}|\}$
    $\widehat{V} \leftarrow [\widehat{V}\ \widehat{\phi}_l]$, $\widehat{P} \leftarrow [\widehat{P}\ \boldsymbol{e}_{\mathcal{P}_l}]$, $\mathcal{P} \leftarrow \begin{pmatrix} \mathcal{P} \\ \mathcal{P}_l \end{pmatrix}$
end

Algorithm 3: DEIM algorithm

In the iteration l of the DEIM (*i.e.* the iteration that yields the index $\mathcal{P}_l$), we first solve a $(l-1) \times (l-1)$ linear system to obtain a coefficient vector $\boldsymbol{c} \in \mathbb{R}^{l-1}$ corresponding to a linear combination of the previous basis vector $\widehat{\phi}_j,\ j = 1, \ldots, l-1$ that gives exactly $\widehat{\phi}_l$ at the points indexed by $\mathcal{P}_1, \ldots, \mathcal{P}_{l-1}$. Next, we compute the residual $\boldsymbol{r} = \widehat{\phi}_l - \widehat{V}\boldsymbol{c}$ between the l-th basis vector and its approximation $\widehat{V}\boldsymbol{c}$, in the entire spatial domain, obtained by applying the computed coefficients vector $\boldsymbol{c}$ on the previous basis vectors. The next index $\mathcal{P}_l$ is the one where the residual is maximum, which means that the algorithm chooses the point where the basis vector $\widehat{\phi}_l$ has more information w.r.t. to the previous vectors. [5] shows that the algorithm is well defined since, as the basis vectors $\{\widehat{\phi}_l\}_{l=1}^m$ are linearly

independent, $\widehat{P}^T\widehat{V}$ is always nonsingular, and, besides that, the indices $\mathcal{P}_1, \ldots, \mathcal{P}_m$ are hierarchical and nonrepeated.

2.3.2. *Application of the POD-DEIM reduction method for the parareal algorithm*

The POD-DEIM approach is introduced in the framework of the parareal methods by [6]. Algorithm 2 is modified by replacing the operator $\mathcal{K}^k_{\Delta t}$ with a reduced-order model obtained by the POD-DEIM, thus avoiding a full fine computation at each iteration of the parareal algorithm, as would be required in the case of nonlinear problems.

We recall that, in Algorithm 2, the operator $\mathcal{K}_{\Delta t}$ is defined as the sum of the coarse and fine propagators $\mathcal{G}_{\Delta t}$ and $\mathcal{F}_{\delta t}$, the former being applied to $(\boldsymbol{I}-\boldsymbol{P}^k)\boldsymbol{y}$, the latter being applied to a projection of the solution onto a subspace $\mathcal{S}^k$, $\boldsymbol{P}^k\boldsymbol{y}$.

As done in [6], we begin by dropping the coarse term of $\mathcal{K}^k_{\Delta t}$, supposing the projection error vanishes asymptotically. We then define, at each iteration of the parareal method, a new operator $\widehat{\mathcal{K}}^k_{\Delta t}$ by

$$\widehat{\mathcal{K}}^k_{\Delta t}(\boldsymbol{y}, t_1, t_0) := \mathcal{F}^k_{r,\delta t}(\boldsymbol{P}^k\boldsymbol{y}, t_1, t_0)$$

where $\mathcal{F}_{r,\delta t}$ is the reduced model (2.14) defined by the subspaces $\mathcal{S}^k_q$ and $\widehat{\mathcal{S}}^k_m$ and the same time step δt as $\mathcal{F}_{\delta t}$; and $\boldsymbol{P}^k$ is the projection onto $\mathcal{S}^k_q$. Therefore, we replace the expensive computation of the operator $\mathcal{K}^k_{\Delta t}$ (which consists in the resolution of a problem with M_f degrees of freedom) by the resolution of a reduced order model with $q \ll M_f$ degrees of freedom, $m \ll M_f$ components to be computed in the nonlinear term and expensive matrices that do not depend on time and can be computed only once per iteration. In the sequel, we omit, for the sake of clearness, the subscript denoting the dimension of the subspaces, and we keep only the superscript indicating the parareal iteration.

The resulting parareal method is named in [6] as reduced basis parareal method, in a more generic way, since other model reduction procedures are also considered. We name it hereafter as POD-DEIM-based parareal method (or PD method, for simplification) for focusing on the POD-DEIM technique. The algorithm is presented in Algorithm 4, in which the modifications w.r.t. the Krylov-subspace-enhanced parareal method (Algorithm 2) are highlighted. [6] proves that, in situations where the original parareal method already converges, the PD method accelerates the convergence and it converges in one iteration if there is a good reduced-order approximation space for the problem.

In lines (20) and (21) of Algorithm 4, we include in the definition of the spaces $\mathcal{S}^k$ and $\widehat{\mathcal{S}}^k$ the snapshots set and the singular value threshold they depend on. These thresholds may not be the same in the two cases, and, with some abuse of nomenclature, we call $\varepsilon_{\text{sv, linear}}$ the one used for computing the spaces spanned by snapshots of the solution, in contrast to the spaces spanned by nonlinear snapshots. Moreover, we remark that, depending on the problem and the formulation of the reduced model, more than one space of each type may be computed at each iteration. The reduced-order model for shallow water equations, presented in Appendix B, is an example.

Finally, we notice that in Algorithm 4, the coarse propagator is used exclusively in the first iteration $k = 0$, for constructing the initial prediction. In the following iterations, the predictions are computed using the reduced fine propagator $\mathcal{F}^k_{r,\delta t}$, instead of the coarse propagator as in the original parareal Algorithm 1.

Initialization: initial guess given by the coarse propagator:
$\boldsymbol{y}_0^0 = \boldsymbol{y}_0$
for $n \leftarrow 0$ **to** $N_{\Delta t} - 1$ **do**
  $\boldsymbol{y}_{n+1}^0 = \mathcal{G}_{\Delta t}(\boldsymbol{y}_n^0, t_{n+1}, t_n)$
end

$n_0 = 0$
Iterations:
for $k \leftarrow 0$ **to** $N_{itermax}$ **do**
  Compute the fine term of the correction (**in parallel**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\widetilde{\boldsymbol{y}}_{n+1}^k = \mathcal{F}_{\delta t}(\boldsymbol{y}_n^k, t_{n+1}, t_n)$
  end

  Define the snapshots sets:
  $\boldsymbol{Y}^k = \{\widetilde{\boldsymbol{y}}_n^j,\ j = 0, \ldots, k;\ n = 0, \ldots, N_{\Delta t}\}$
  $\widehat{\boldsymbol{Y}}^k = \{\boldsymbol{F}(\widetilde{\boldsymbol{y}}_n^j),\ j = 0, \ldots, k;\ n = 0, \ldots, N_{\Delta t}\}$

  Compute the spaces and define the reduced model $\mathcal{F}_{r,\delta t}^k$:
  $\mathcal{S}^k(\boldsymbol{Y}^k, \varepsilon_{\text{sv,linear}})$ (using POD)
  $\widehat{\mathcal{S}}^k(\widehat{\boldsymbol{Y}}^k, \varepsilon_{\text{sv,nonlinear}})$ (using POD-DEIM)

  Compute the coarse term of the correction and the final correction term(**in parallel**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\overline{\boldsymbol{y}}_{n+1}^k = \mathcal{F}_{r,\Delta t}^k(\boldsymbol{P}^k \boldsymbol{y}_n^k, t_{n+1}, t_n)$
    $\overline{\overline{\boldsymbol{y}}}_{n+1}^k = \widetilde{\boldsymbol{y}}_{n+1}^k - \overline{\boldsymbol{y}}_{n+1}^k$
  end

  Compute the coarse predictions and correct them to obtain the final solution in the iteration (**sequentially**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\boldsymbol{y}_{n+1}^{k+1} = \mathcal{F}_{r,\Delta t}^k(\boldsymbol{P}^k \boldsymbol{y}_n^{k+1}, t_{n+1}, t_n) + \overline{\overline{\boldsymbol{y}}}_{n+1}^k$
  end

  Find the last instant $\tilde{n} \in \{1, \ldots, N_{\Delta t - 1}\}$ not satisfying a convergence criterion
  $n_0 \leftarrow \tilde{n}$
  if *all instants converged* **then**
    break;
  end
end

Algorithm 4: POD-DEIM-based parareal algorithm

3. A modification of the POD-DEIM-based parareal algorithm for improving the reduced model

3.1. Influence of the snapshot sets in the definition of the reduced-order models

The choice of the snapshot set is a key factor in the formulation of reduced-order models. Indeed, for a given snapshot set, the reduced basis constructed via the POD is optimal, *i.e.* it minimizes an approximation error concerning the snapshots [5], but if the snapshots are not a meaningful representation of the solution of a problem, the reduced-order model will be a low-quality approximation for this problem.

Moreover, as discussed in [23], even if the POD is optimal in approximating a given dataset, it does not necessarily provide the best possible description for the dynamics that generate the snapshots. In this case, including more POD modes (*i.e.* , retaining more basis vectors) may lead to degrade the quality of the reduced model.

Many approaches are presented in the literature for improving snapshot selection and/or overcoming misrepresentations and instabilities of the POD-based reduced order models. Examples are: subtracting the mean from the snapshots so as to consider only the fluctuations in the solution [21]; enriching the snapshot set with finite difference quotients [21] or with the gradient [1] of the solution; greedy approaches for snapshot selection [7]; and the balanced POD method [23], that requires the resolution of a dual problem.

As shown in the numerical tests presented in Section 5, the POD-DEIM-based parareal is unstable for a number of test cases, specially when the time step and/or spatial discretization of the coarse propagator $\mathcal{G}_{\Delta t}$ is a much coarser than that of the fine propagator $\mathcal{F}_{\delta t}$ ($\Delta t \gg \delta t$). This suggests two possible causes for the observed instability:

(1) Since the snapshots are selected at every coarse time step Δt, if $\Delta t \gg \delta t$ there are not enough snapshots for properly representing the fine solution;

(2) The errors due to the coarse spatial and temporal discretization make the coarse solution a poor approximation for the fine one.

We propose hereafter a slight modification of Algorithm 4 for minimizing the first cause. The algorithm, called Modified POD-DEIM-based parareal (MPD) method hereafter, is presented in Algorithm 5, with a highlight on the modifications w.r.t. Algorithm 4. It consists in enriching the snapshots set with intermediary snapshots of the fine correction term (computed in line 12 of Algorithm 5) between the instants of the coarse temporal mesh.

The intermediary snapshots are taken every $\widehat{\delta t}$, $\delta t < \widehat{\delta t} < \Delta t$, and stored in the set Υ_k^n defined for each iteration and coarse time step. In line 18 of the algorithm, $\boldsymbol{F}\left(\Upsilon_n^k\right)$ is the set of the nonlinear term computed at each element of Υ_n^k.

We notice that this modification of the algorithm does not require any extra computational cost for generating the snapshots. Indeed, in Algorithm 4, the intermediary snapshots are computed but not used. Due to this non-intrusive aspect, we propose this approach instead of focusing on the second cause of instability mentioned above (*i.e.* , modifying the coarse propagator, what in some applications one may not be able to choose freely, or may be limited by stability conditions).

However, as discussed in the speedup estimation in Section 4, the cost of the POD procedure grows quadratically with the number of snapshots. We also recall that the number of snapshots is proportional to the number of iterations. Therefore, the number of intermediary snapshots may not be excessive. By defining $\widehat{\delta t} = \Delta t/\alpha$, $\alpha \in \mathbb{N}$, one should naturally begin by setting $\alpha = 2$.

Initialization: initial guess given by the coarse propagator:
$\boldsymbol{y}_0^0 = \boldsymbol{y}_0$
for $n \leftarrow 0$ **to** $N_{\Delta t} - 1$ **do**
  $\boldsymbol{y}_{n+1}^0 = \mathcal{G}_{\Delta t}(\boldsymbol{y}_n^0, t_{n+1}, t_n)$
end

$n_0 = 0$
Iterations:
for $k \leftarrow 0$ **to** $N_{itermax}$ **do**
  Compute the fine term of the correction (**in parallel**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\widetilde{\boldsymbol{y}}_{n+1}^k = \mathcal{F}_{\delta t}(\boldsymbol{y}_n^k, t_{n+1}, t_n)$
    Store intermediary solutions defined at each $\widehat{\delta t},\ \delta t \leq \widehat{\delta t} \leq \Delta t$:
      $\Upsilon_n^k = \{\mathcal{F}_{\delta t}(\boldsymbol{y}_n^k, t_n + l\widehat{\delta t}, t_n),\ l \geq 1 \text{ s.t. } l\widehat{\Delta t} < \Delta t\}$
  end

  Define the snapshots sets:
  $\boldsymbol{Y}^k = \{\widetilde{\boldsymbol{y}}_n^j,\ j = 0, \ldots, k;\ n = 0, \ldots, N_{\Delta t}\} \bigcup_{n=0}^{N_{\Delta t}-1} \bigcup_{j=0}^{k} \Upsilon_n^k$
  $\widehat{\boldsymbol{Y}}^k = \{\boldsymbol{F}(\widetilde{\boldsymbol{y}}_n^j),\ j = 0, \ldots, k;\ n = 0, \ldots, N_{\Delta t}\} \bigcup_{n=0}^{N_{\Delta t}-1} \bigcup_{j=0}^{k} \boldsymbol{F}\left(\Upsilon_n^k\right)$

  Compute the spaces and define the reduced model $\mathcal{F}_{r,\delta t}^k$:
  $\mathcal{S}^k(\boldsymbol{Y}^k, \varepsilon_{\text{sv,linear}})$ (using POD)
  $\widehat{\mathcal{S}}^k(\widehat{\boldsymbol{Y}}^k, \varepsilon_{\text{sv,nonlinear}})$ (using POD-DEIM)

  Compute the coarse term of the correction and the final correction term(**in parallel**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\overline{\boldsymbol{y}}_{n+1}^k = \mathcal{F}_{r,\Delta t}^k(\boldsymbol{P}^k \boldsymbol{y}_n^k, t_{n+1}, t_n)$
    $\overline{\overline{\boldsymbol{y}}}_{n+1}^k = \widetilde{\boldsymbol{y}}_{n+1}^k - \overline{\boldsymbol{y}}_{n+1}^k$
  end

  Compute the coarse predictions and correct them to obtain the final solution in the iteration (**sequentially**):
  for $n \leftarrow n_0$ **to** $N_{\Delta t} - 1$ **do**
    $\boldsymbol{y}_{n+1}^{k+1} = \mathcal{F}_{r,\Delta t}^k(\boldsymbol{P}^k \boldsymbol{y}_n^{k+1}, t_{n+1}, t_n) + \overline{\overline{\boldsymbol{y}}}_{n+1}^k$
  end

  Find the last instant $\tilde{n} \in \{1, \ldots, N_{\Delta t}\}$ not satisfying a convergence criterion
  $n_0 \leftarrow \tilde{n} - 1$
  if *all instants converged* **then**
    break;
  end
end

Algorithm 5: Modified POD-DEIM-based parareal algorithm

4. **Speedup estimation**

We estimate in this section the speedup provided by the modified POD-DEIM-based parareal Algorithm 5.

We recall that Δt and δt are the time steps respectively in the coarse ($\mathcal{G}_{\Delta t}$) and in the fine models ($\mathcal{F}_{\delta t}$), and that the fine reduce model ($\mathcal{F}^k_{r,\delta t}$) also propagates with the fine time step δt. For a simulation with total length of T, we have $N_{\Delta t} = T/\Delta t$ and $N_{\delta t} = T/\delta t$, with $N_{\Delta t}$, $N_{\delta t}$ and $p = N_{\delta t}/N_{\Delta t}$ supposed to be integers. Concerning the spatial discretization, we denote by M_f and M_c the number of degrees of freedom in the fine and coarse propagators respectively. $m(k)$ is the number of degrees of freedom of $\mathcal{F}^k_{r,\delta t}$, in iteration k of the parareal algorithm.

Let τ_c be the computational time for advancing one time step Δt on the coarse model $\mathcal{G}_{\Delta t}$; and τ_f and $\tau_r(k)$ the computational times for advancing one time step δt using respectively $\mathcal{F}_{\delta t}$ and $\mathcal{F}^k_{r,\delta t}$. Since the reduced model $\mathcal{F}^k_{r,\delta t}$ is reformulated at each iteration k, its computational time depends on the iteration.

We also denote by $\tau_{\mathcal{S}}(k)$ the computational time for computing a subpace $\mathcal{S}^k$ (obtained from snapshots of the solution) or $\widehat{\mathcal{S}}^k$ (obtained from snapshots of the nonlinear terms). Since the former corresponds to a POD (via a SVD) and the latter corresponds to a POD and a subsequent DEIM, we majorate $\tau_{\mathcal{S}}(k)$ by the computational time for computing the subspaces $\widehat{\mathcal{S}}^k$. We denote by N_{spaces} the number of subspaces to be computed in each iteration, and by $\hat{m}$ the highest dimension among all subspaces and all iterations. Since the subspaces $\mathcal{S}$ and $\widehat{\mathcal{S}}$ are independent, their computation can also be parallelized. We supposed that that are $N_{p,s}$ processors, or groups of processors, for doing the POD and the POD-DEIM procedures.

The referential computational time (*i.e.* for performing a full fine sequential simulation) is $T_{\text{ref}} = N_f \tau_f$. Concerning the MPD parareal simulation with $\hat{k}$ iterations and using N_p processors, its total computational time $T_{\text{par}}(\hat{k}, N_p)$, as detailed in Algorithm 5, involves:

- an initial full coarse prediction taking $T_c = N_{\Delta t}\tau_c$;
- within each iteration k:
  - the parallel computation of the fine term of the correction, taking
$$T_{corr,f} = \frac{N_{\Delta t}}{N_p}\frac{N_{\delta t}}{N_{\Delta t}}\tau_f$$
  - the possibly parallel computation of the subspaces, taking
$$T_{\text{spaces}} = \frac{N_{\text{spaces}}}{N_{p,s}}\tau_{\mathcal{S}}(k)$$
  - the parallel computation of the coarse term of the correction, taking
$$T_{corr,c} = \frac{N_{\Delta t}}{N_p}\frac{N_{\delta t}}{N_{\Delta t}}\tau_r(k)$$
  - the sequential computation of the predictions, taking
$$T_{pred} = N_{\delta t}\tau_r(k)$$

Therefore,

$$\begin{aligned} T_{\text{par}}(\hat{k}, N_p) &\approx T_c + \hat{k}\left(T_{corr,f} + T_{\text{spaces}} + T_{corr,c} + T_{pred}\right) \\ &\approx N_{\Delta t}\tau_c + \hat{k}\left[\frac{N_{\delta t}}{N_p}\left(\tau_f + \tau_r(\hat{k})\right) + \frac{N_{\text{spaces}}}{N_{p,s}}\tau_{\mathcal{S}}(\widehat{k}) + N_{\delta t}\tau_r(\hat{k})\right] \end{aligned}$$

Notice that $\tau_{\mathcal{S}}(k)$ and $\tau_r(k)$ are supposed to increase along iterations, since the spaces are constructed using a increasing number of snapshots and may have an increasing dimension. Therefore, both these parameters are majorated by their value at iteration $\hat{k}$.

Therefore, the speedup provided by the POD-DEIM parareal algorithm can be estimated as

$$\begin{aligned} s(\hat{k}, N_p) &\approx \frac{T_{\text{ref}}}{T_{\text{par}}(\hat{k}, N_p)} \\ &\approx \frac{1}{\dfrac{N_{\Delta t}\tau_c}{N_{\delta t}\tau_f} + \dfrac{\hat{k}}{N_p}\dfrac{\tau_f + \tau_r(\hat{k})}{\tau_f} + \hat{k}\dfrac{\tau_r(\hat{k})}{\tau_f} + \hat{k}\dfrac{N_{\text{spaces}}\tau_{\mathcal{S}}(\widehat{k})}{N_{p,s}N_{\delta t}\tau_f}} \end{aligned} \tag{4.1}$$

As done in [25], we derive some bounds for (4.1) in order to obtain some guidelines on the configuration of the MPD parareal method to optimize its speedup. Moreover, we introduce the number of degrees of freedom of each model by considering $\tau_f = \mathcal{O}(M_f)$, $\tau_c = \mathcal{O}(M_c)$ and $\tau_r = \mathcal{O}(\hat{m})$ (for the latter, we majorate with the largest subspace's dimension computed in the algorithm).

We firstly have

$$s(\hat{k}, N_p) \leq \frac{\tau_f}{\hat{k}\tau_r(\hat{k})} \approx \frac{M_f}{\hat{k}\hat{m}} \tag{4.2}$$

The bound (4.2) shows the importance of formulating a reduced model with a much smaller dimension than that of the fine model. It also shows that the speedup depends strongly on the speed of convergence (*i.e.* on the value of $\hat{k}$).

Next, we have

$$s(\hat{k}, N_p) \leq \frac{N_p}{\hat{k}}\frac{\tau_f}{\tau_f + \tau_r(\hat{k})} \approx \frac{N_p}{\hat{k}}\frac{M_f}{M_f + \hat{m}} \approx \frac{N_p}{\hat{k}} \tag{4.3}$$

which makes explicit the strong dependence on the number of iterations to convergence and the impossibility of a perfect speedup.

From the coarse initial prediction, we obtain the bound

$$s(\hat{k}, N_p) \leq \frac{N_{\delta t}\tau_f}{N_{\Delta t}\tau_c} \approx \frac{N_{\delta t}M_f}{N_{\Delta t}M_c} \tag{4.4}$$

showing that the full coarse simulation must be much cheaper than the fine one, which can be obtained by increasing Δt (and thus reducing $N_{\Delta t}$) and/or using a coarser mesh (and thus reducing M_c). However, these improvements on the bound (4.4) may slow down the convergence of the method, and thus affecting negatively the bounds (4.2) and (4.3).

Lastly, the computational time needed to compute the subspaces from the solution and the nonlinear terms yields

$$s(\hat{k}, N_p) \leq \frac{N_{p,s}N_{\delta t}\tau_f}{\hat{k}N_{\text{spaces}}\tau_{\mathcal{S}}(\hat{k})} \tag{4.5}$$

Let us develop the estimation for the computational time $\tau_{\mathcal{S}}(\hat{k})$. As said above, it is majorated by the computational time for computing the subspaces $\hat{\mathcal{S}}$, since this computation is more expensive than the one for the subspaces $\mathcal{S}$. Moreover, in the framework of the modified POD-DEIM-based parareal algorithm, we enrich the snapshots set by keeping intermediate snapshots at every $\widehat{\delta t} = \Delta t/\alpha,\ \alpha \in \mathbb{N}$. Within a given iteration k, the computation of $\hat{\mathcal{S}}$ involves:

(1) A POD (via a SVD) for obtaining the subspace's basis, using as input a matrix of size $M_f \times k\alpha N_{\Delta t}$. The obtained subspace has dimension $m \ll M_f$.

(2) A DEIM for obtaining the m spatial indices for the reduced model, consisting in the sequential resolution of $m-1$ linear systems and m lookups for the maximum element in vectors of size M_f. The i-th linear system has a size $i \times i$;

Both the SVD and the resolution of the linear systems are performed using Functions *dgesvd* and *dgesv* respectively of the *MKL-LAPACK* suite. The lookup for the maximum element is performed using Function *max_element* from the *C++ Standard Library*.

The *MKL-LAPACK* function *dgesvd* applied to a matrix in $\mathbb{R}^{q\times n}$ has a complexity $\mathcal{O}(qn^2+n^3)$ [10]. Concerning *dgesv*, the complexity for a matrix in $\mathbb{R}^{n\times n}$ is $\mathcal{O}(n^3)$ [1]. For the function *max_element*, the complexity for looking up a vector in $\mathbb{R}^n$ is $\mathcal{O}(n)$ [2].

Therefore, the computational cost for subspace computation is approximately

$$\begin{aligned}\tau_{\mathcal{S}}(\hat{k}) &\sim M_f\hat{k}^2\alpha^2N_{\Delta t}^2+\hat{k}^3\alpha^3N_{\Delta t}^3+\sum_{i=2}^{\hat{m}} i^3+\hat{m}M_f\\ &\sim \hat{k}^2\alpha^2N_{\Delta t}^2(M_f+\hat{k}\alpha N_{\Delta t})+\left(\frac{\hat{m}^2(\hat{m}+1)^2}{4}-1\right)+\hat{m}M_f\\ &\sim M_f\hat{k}^2\alpha^2N_{\Delta t}^2+\frac{\hat{m}^4}{4}+\hat{m}M_f\end{aligned}$$

with a rough estimate for the term on $\hat{m}^4$. We also assume $M_f \gg \hat{k}\alpha N_{\Delta t}$ (which is expected in parareal algorithms) by keeping α reasonably small and achieving a fast convergence.

Therefore, from (4.5), we have

$$s(\hat{k},N_p) \leq \frac{N_{p,s}N_{\delta t}M_f}{\hat{k}N_{\text{spaces}}\left(M_f\hat{k}^2\alpha^2N_{\Delta t}^2+\frac{\hat{m}^4}{4}+\hat{m}M_f\right)} \tag{4.6}$$

The bound (4.6) shows firstly a strong dependency of the speedup on the dimension of the reduced model. Assume indeed that this dimension is small, so that $\hat{m}^4/4$ can be neglected w.r.t. $M_f\hat{k^2}\alpha^2N_{\Delta t}^2$, we conclude that this bound does not depend on the spatial dimension of the problem, but is strongly affected by the number of iterations (by a factor $1/\hat{k}^3$). Finally, this bound can be minimized by computing the subspaces in parallel, thus approaching $N_{p,s}/N_{\text{spaces}}$ to 1. However, since a parallel speedup is also expected in each execution of the *MKL-LAPACK* functions, a trade-off may be found in the distribution of processors and depends on the problem size.

5. Numerical results

We present in this section some numerical tests for the resolution of the 2D nonlinear SWE using parareal methods. Our objective is to compare the performance of the original parareal method (Algorithm 1), the POD-DEIM-based parareal method (Algorithm 4) and its proposed variation (Algorithm 5), in terms of computational time and convergence to the reference solution, which is given by a full fine sequential simulation:

$$\boldsymbol{y}_{\text{ref},n} := \mathcal{F}_{\delta t}(\boldsymbol{y}_0,t_n,0), \qquad n=1,\dots,N_{\Delta t}$$

We consider here the 2D nonlinear SWE as formulated in [17]. In fact, in this work a porosity-based shallow water model is proposed, in which the fine discretization of a spatial domain with obstacles (*e.g.* an urban zone) can be replaced by a coarse discretization of an analogous porous media. A

[1] http://www.netlib.org/lapack/lug/node71.html

[2] https://en.cppreference.com/w/cpp/algorithm/max_element#Complexity

porosity parameter is defined for each cell and inserted in the equations, representing the presence of obstacles. We consider here an unitary porosity, which leads to the classical SWE. They read

$$\frac{\partial}{\partial t}\boldsymbol{U}(t) + \frac{\partial}{\partial x}\boldsymbol{F}(\boldsymbol{U}(t)) + \frac{\partial}{\partial y}\boldsymbol{G}(\boldsymbol{U}(t)) = \boldsymbol{S}(\boldsymbol{U}(t)) \tag{5.1}$$

with

$$\boldsymbol{U} = \begin{pmatrix} h \\ hu_x \\ hu_y \end{pmatrix}, \qquad \boldsymbol{F} = \begin{pmatrix} hu_x \\ hu_x^2 + gh^2/2 \\ hu_x u_y \end{pmatrix},$$
$$\boldsymbol{G} = \begin{pmatrix} hu_y \\ hu_x u_y \\ hu_y^2 + gh^2/2 \end{pmatrix}, \qquad \boldsymbol{S} = \begin{pmatrix} 0 \\ S_{0,x} + S_{f,x} \\ S_{0,y} + S_{f,y} \end{pmatrix}$$

where g is the gravitational acceleration, h is the water depth, u_x and u_y are respectively the velocities in the x and y directions, $S_{0,x}$ and $S_{0,y}$ are source terms related to spatial variations of the topography, and $S_{f,x}$ and $S_{f,y}$ are friction source terms.

For the sake of clearness, the formulation of the finite-volume scheme, the reduced model and the POD-DEIM-based parareal algorithm for the SWE equations are left to the Appendices. The reduced model is constructed by computing three reduced subspaces $\mathcal{S}_{q_i}^{(i)}, i = 1, 2, 3$ of dimension $q_i \ll M_f$ spanned by snapshots of the solution, and five reduced subspaces $\hat{\mathcal{S}}_{m_l}^{(l)}, l = 1, \ldots, 5$ of dimension $m_l \ll I_f$ (with I_f the number of interfaces of the mesh) spanned by snapshots of the nonlinear term (fluxes and source terms). Due to the relatively complex structure of the SWE reduced model, we propose, in Appendix A and as a didactic example, the derivation of a reduced model for the inviscid scalar two-dimensional Burgers equation. The derivation of the POD-DEIM reduced-model for this equation is completely analogous to the SWE case, since they share the same structure, but the formulated model is much simpler, with only one subspace spanned by snapshots of the solution and one subspace spanned by snapshots of the nonlinear term. The procedure for the SWE is detailed in Appendix B.

In this section, we propose an one-dimensional test case (in fact, a 2D one but with only one direction of propagation) and two-dimensional cases, the second of them using a coarse propagator also coarsen in space. We intend to observe empirically the influence of the initial coarse prediction on the construction of the reduced-order models and, as a consequence, on the convergence and stability behaviour of the POD-DEIM-based parareal Algorithm 4. Then, the eventual need for enriching the snapshots sets (Algorithm 5) can be assessed.

Using the same notation as in the parareal algorithms presented in Section 2, $\boldsymbol{y}_n^k$ represents the parareal solution in the instant t_n and iteration k. For the 2D nonlinear SWE, we have

$$\boldsymbol{y}_n^k = \begin{pmatrix} (\boldsymbol{U}^{(1)})_n^k \\ (\boldsymbol{U}^{(2)})_n^k \\ (\boldsymbol{U}^{(3))})_n^k \end{pmatrix} \in \mathbb{R}^{3M_f}$$

with $\boldsymbol{U}^{(1)}, \boldsymbol{U}^{(2)}, \boldsymbol{U}^{(3)} \in \mathbb{R}^{M_f}$ containing respectively the values of h, hu_x and hu_y in all the cells of the mesh associated to the fine propagator $\mathcal{F}_{\delta t}$.

By denoting as $[\boldsymbol{y}]_i$ the $i-th$ component of the vector $\boldsymbol{y}$, we define the following errors between the parareal and the referential solution

- Error at instant t_n and iteration k, computed in the entire fine mesh:

$$e_n^k := \frac{\sum_{i=1}^{\overline{M}_f} |[\boldsymbol{y}_n^k]_i - [\boldsymbol{y}_{\text{ref, n}}]_i|}{\sum_{i=1}^{\overline{M}_f} |[\boldsymbol{y}_{\text{ref},n}]_i|}$$

- Error at iteration k, computed in the entire fine mesh:

$$\overline{e}^k := \max_{n=0,\ldots,N_{\Delta t}} e_n^k$$

where $\overline{M}_f = 3M_f$ for the SWE.

The code is parallelized using *OpenMP* directives and computational times are measured using the *OpenMP* function *omp_get_wtime*. Linear Algebra operations are performed using the *Intel MKL* suite, and, when the coarse and fine spatial meshes are different, the interpolation weights are given by a piecewise linear interpolation using Delaunay triangulations, which is made by the open source library *delaunay linterp* [3]. These interpolations weights are precomputed to be reused in several simulations, such that the computational times presented in this section does not include the time to compute them. I/O operations are also excluded from the computational times. Each simulation, including the referential ones, are performed five times and their computational times are averaged to give the results presented here.

For all the simulations, the convergence criterion (2.3) is used, with $\varepsilon_{\text{TOL}} = 10^{-10}$, and we set a maximal number of iterations $N_{\text{itermax}} = 5$. For the POD-DEIM based parareal simulations, we use $\varepsilon_{\text{sv,linear}} = \varepsilon_{\text{sv,nonlinear}} = 10^{-3}$.

All the simulations are performed in a dual-Xeon E5-2680 v2 with a frequency of 2.80GHz, with 256GB of RAM capacity. For taking advantage of the shared memory parallelization, all the parallel processors are in the same node, with 20 cores available per node. Therefore, in each simulation the number of processors is set to $N_p = \min\{N_{\Delta t}, 20\}$. In each iteration k, the computation of the subspaces $\mathcal{S}^{(i),k}$, $i = 1, 2, 3$ and $\widehat{\mathcal{S}}^{(l),k}$, $l = 1, \ldots, 5$, is done sequentially (relatively to each other), which corresponds to take $N_{p,s} = 1$ as defined in Section 4. Therefore, the parallelization is done inside the *MKL-LAPACK* functions.

In order to have an easier identification of the results presented in the sequel, we propose the following nomenclature for the test cases:

SWE{nb.dimensions}D-{method}-{$N_{\Delta t}$}-[c]

where

- *nb.dimensions* can be 1 (for the 1D problem) or 2 (for the 2D problem);
- *method* can be *cl* (for the classical parareal method, presented in Algorithm 1), *pd* (for the POD-DEIM-based parareal method, presented in Algorithm 4) or *mpd* (for modified the POD-DEIM-based parareal method, presented in Algorithm 5);
- $N_{\Delta t}$ is the number of coarse time steps, as defined previously;
- the last term, *c*, is used when there is spatial coarsening between the fine and the coarse meshes.

For example, *SWE1D-cl-25* stands for a 1D test case using the classical parareal method with 25 coarse time steps and no spatial coarsening, and *SWE2D-mpd-20-c* stands for the a 2D simulation using the modified POD-DEIM-based parareal method with 20 coarse time steps and spatial coarsening. Moreover, when necessary, the reference solution is denoted, for example, as *SWE1D-ref*.

[3] http://rncarpio.github.io/delaunay_linterp/

5.1. **First test case: 1D problem (SWE1D)**

We consider in this test case a rectangular domain $\Omega = [0, 20]^2$. The fine propagator is described by a time step $\delta t = 0.001$ and a Cartesian homogeneous mesh with $\delta x = \delta y = 1$, containing 400 cells and 840 interfaces. The initial solution is a lake-in-rest, with initial water depth $h = 1$ and a flat bottom. No friction is considered. An unitary and constant mass flux is imposed at the western boundary $x = 0$. The other three boundaries are closed (imposed null mass flux). The simulation length is $T = 5$, starting on $t = 0$. The average time for computing the sequential fine solution is of approximately approximately $T_{\text{ref}} = 2.2s$.

We define a coarse propagator $\mathcal{G}_{\Delta t}$ with $\Delta t = 0.2$ and no spatial coarsening. Therefore, there are $N_{\Delta t} = 25$ coarse time steps, and 20 processors are used for the simulations, such that some processors may compute two coarse time steps in the parallel computation of the correction terms. Following the nomenclature defined above, the three simulations are named *SWE1D-cl-25*, *SWE1D-pd-25* and *SWE1D-mpd-25*. The MPD method is defined by additional snapshots taken at every $\widehat{\Delta t} = 0.1$ (*i.e.* by taking $\alpha = 2$, as defined previously).

The evolution of the errors e_n^k and $\overline{e}^k$ is presented in Figure 1. We notice that the classical parareal method is very unstable, and the error increases along iterations. For the PD method, we observe a fast convergence to the referential solution and, for the MPD method, the convergence is even faster: in only one parareal iteration, the error in all the 25 timesteps is in the order of 10^{-11}.

Figure 2 exemplifies the behaviour of the methods by showing the evolution of water height h, compared with the referential solution, in a given point of the domain ($(x, y) = (10, 5)$). We observe that the solution given by the classical parareal iterations is highly oscillatory and unstable, whereas both the POD-DEIM-base methods give, in only one iteration, a visually converged solution.

The speedup provided by each method along iterations is presented in Figure 3. The $0-th$ iteration (initial coarse prediction) is not shown for the sake of readability, since this iteration is much faster than the referential solution and the following iterations. As expected, the classical parareal method is the cheapest one, and the modified POD-DEIM-based method, the most expensive one. As said above, both the PD and MPD methods give a very good result in one iteration, so we can consider for them a speedup of approximately 5.0 and 3.9, respectively.

5.2. **Second test case: 2D problem (SWE2D)**

For the 2D test case, we consider a rectangular domain $\Omega = [0, 100]^2$, with a fine propagator described by $\delta t = 0.001$ and a Cartesian homogeneous mesh with $\delta y = \delta x = 2$, containing 2500 cells and 5100 interfaces. The sequential fine solution takes approximately $T_{\text{ref}} = 12.7s$, on average. The initial solution is given by

$$\begin{cases} h(t=0,x,y) = h_0 + \exp\left(-\left(\dfrac{(x-x_0)^2}{2\sigma_x^2} + \dfrac{(y-y_0)^2}{2\sigma_y^2}\right)\right) \\ u_x(t=0,x,y) = 0, \\ u_y(t=0,x,y) = 0 \end{cases} \qquad (x,y) \in [0,100]^2 \tag{5.2}$$

with $h_0 = 1$, $x_0 = y_0 = 50$ and $\sigma_x = \sigma_y = 7.5$.

All the boundaries are closed (imposed null mass flux) and the bottom is flat, and no friction is considered. The simulation is performed from $t = 0$ to $T = 5$.

For this test case, we perform two sets of parareal methods, respectively without and with spatial coarsening.

We begin by defining a coarse propagator with $\Delta t = 0.25$ and no spatial coarsening. The tests are named *SWE2D-cl-20*, *SWE2D-pd-20* and *SWE2D-mpd-20*. For the MPD method, additional snapshots are taken at every $\widehat{\Delta t} = 0.125$ ($\alpha = 2$).

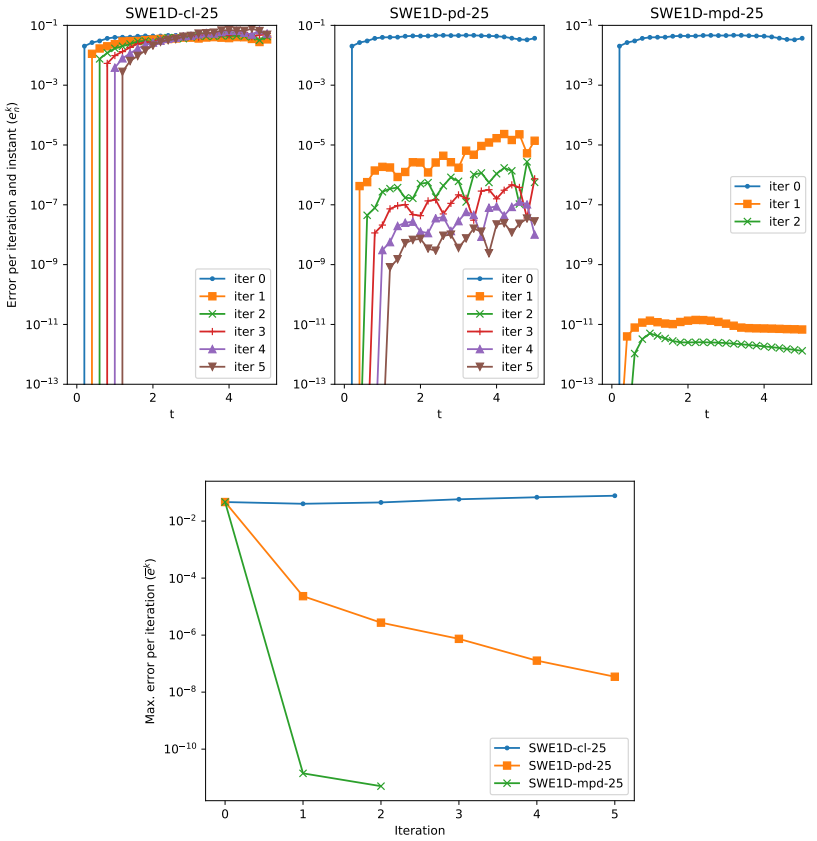

FIGURE 1. Evolution of the errors e_n^k along iterations and time for test cases *SWE1D-cl-25* (first plot), *SWE1D-pd-25* (second plot) and *SWE1D-mpd-25* (third plot); and evolution of the errors $\overline{e}^k$ along iterations for the same test cases (fourth plot). For *SWE1D-mpd-25*, the algorithm converges in 2 parareal iterations.

Figure 4 shows the relative error for three parareal methods. We notice that, in this test case, the classical parareal method behaves better than in the previous one, but it converges slower than the POD-DEIM-based methods. We also notice that the MPD, by constructing a reduced model using twice the number of snapshots, provides a great improvement on the quality of the solution in the first iteration, for almost all the the time steps.

The evolution of the speedup along iterations for each method is presented in Figure 5. We notice, for the three methods, a higher speedup in comparison to the test case *S1* presented above. As indicated in bounds (4.2)-(4.6), this may be due to the increase in the number of spatial points in the fine propagator ($M_f = 400$ in *SWE1D* and $M_f = 2500$ in *SWE2D*) and the reduction in the number of coarse time steps ($N_{\Delta t} = 25$ in *SWE1D* and $N_{\Delta t} = 20$ in *SWE2D*).

Figure 6 shows the evolution of the parareal solutions h given by each method in the point $(x, y) = (40, 40)$. For both the POD-DEIM-based methods, we obtain in one iteration a very good approximation to the referential solution in this point. For the classical parareal method, a good solution is obtained in two iterations. Therefore, in this test case, as shown in Figure 5, the classical method is able to give a fast convergence, with a speedup factor of approximately 9.2, whereas the PD and the MPD have respectively a speedup of approximately 11.0 and 8.5.

Using the same reference solution, *i.e.* the same fine propagator $\mathcal{F}_{\delta t}$, we perform a second set of parareal simulations for the test case *SWE2D*. We set now a lower-quality initial prediction, by

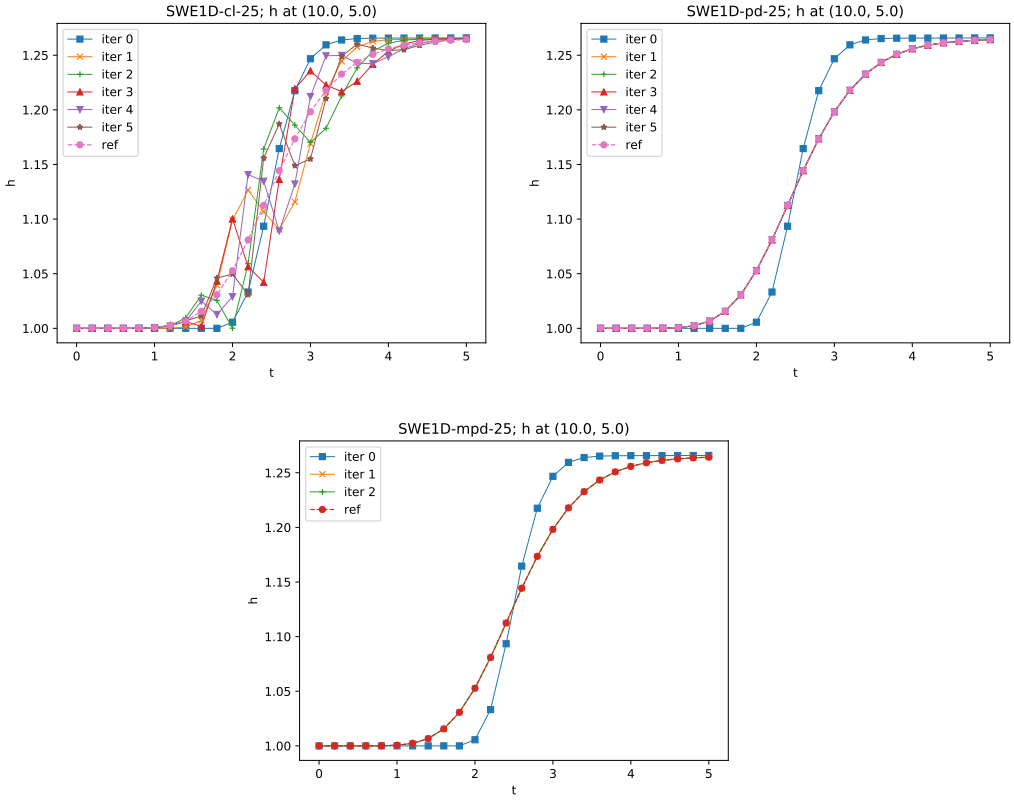

FIGURE 2. Evolution of h along time in position $(x, y) = (10, 5)$ for test cases *SWE1D-cl-25* (top left), *SWE1D-pd-25* (top right) and *SWE1D-mpd-25* (bottom). Circles and squares represent respectively the referential solution and the initial prediction. For the second and third figures, the referential solution and the parareal iterations superimpose for $k \geq 1$.

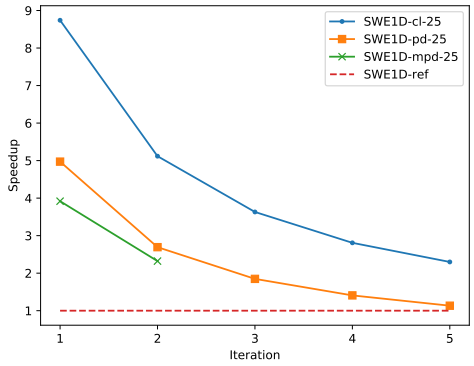

FIGURE 3. Evolution of the average speedup along iterations, given by the ratio between the referential and the parareal computational times, for test cases *SWE1D-cl-25* (circles), *SWE1D-pd-25* (squares) and *SWE1D-mpd-25* (crosses). The dashed line represents the referential speedup, equal to 1.

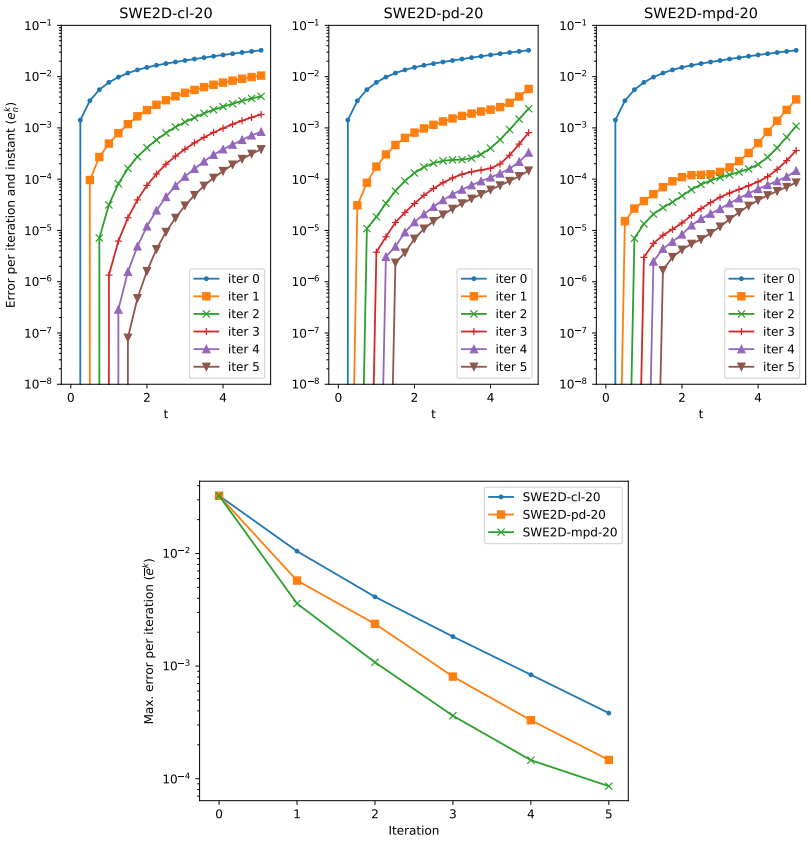

FIGURE 4. Evolution of the errors e_n^k along iterations and time for test cases *SWE2D-cl-20* (first plot), *SWE2D-pd-20* (second plot) and *S0-mpd-20* (third plot); and evolution of the errors $\overline{e}^k$ along iterations for the same test cases (fourth plot).

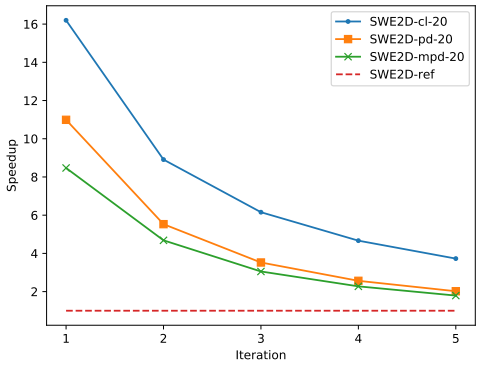

FIGURE 5. Evolution of the average speedup along iterations, given by the ratio between the referential and the parareal computational times, for test cases *SWE2D-cl-20* (circles), *SWE2D-pd-20* (squares) and *SWE2D-mpd-20* (crosses). The dashed line represents the referential speedup, equal to 1.

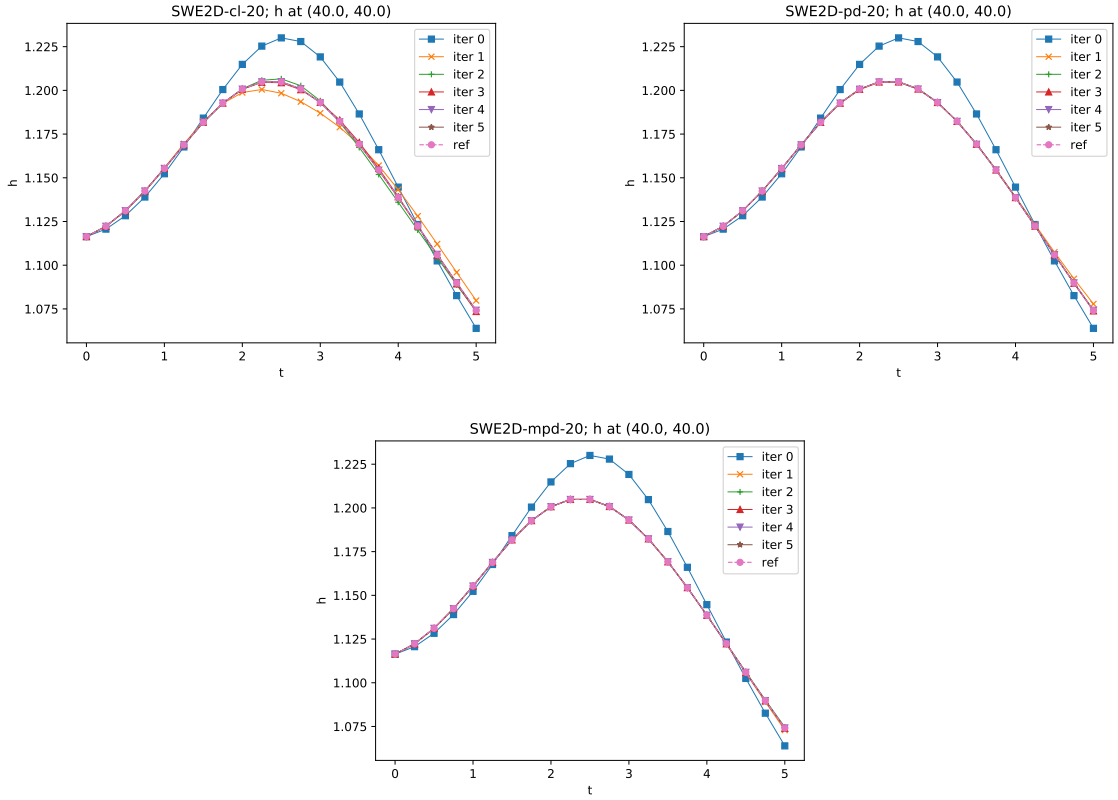

FIGURE 6. Evolution of h along time in position $(x, y) = (40, 40)$ for test cases *SWE2D-cl-20* (top left), *SWE2D-pd-20* (top right) and *SWE2D-mpd-20* (bottom). Circles and squares represent respectively the referential solution and the initial prediction. For the second and third figures, the referential solution and the parareal iterations nearly superimpose for $k \geq 1$.

defining a coarse propagator $\mathcal{G}_{\Delta t}$ with a coarser Cartesian mesh ($\Delta x = \Delta y = 5$), but the same time step $\Delta t = 0.25$ as before, such that there are $N_{\Delta t} = 20$ coarse time steps. The tests are named *SWE2D-cl-20-c*, *SWE2D-pd-20-c* and *SWE2D-mpd-20-c*. For the MPD method, additional snapshots are taken at every $\widehat{\Delta t} = 0.125$ ($\alpha = 2$).

The evolution of the errors e_n^k and $\overline{e}^k$ (Figure 7) and the evolution of the solution in position $(x, y) = (40, 40)$ (Figure 9) show that both the classical and the PD method present a troubled convergence behaviour. For the classical method, the error increases in the first iteration and in the following iterations the solution approaches slowly to the referential one. For the PD method, the first iteration provides a good error decrease, but the following iterations become unstable and the error increases. The stability can be improved by reducing the subspaces' dimensions (*i.e.* increasing $\varepsilon_{\text{sv, linear}}$ and $\varepsilon_{\text{sv, nonlinear}}$), but it may strongly affect the speed of convergence of the method. The MPD provides the better approximation among the three methods. As in the PD, we observe an error increase from the first to the second iterations in more advanced times steps, but with much smaller magnitude, and the following iterations are able to control the instabilities. Concerning the speedup (Figure 8), we can consider for the point $(40, 40)$, in the MPD method, a good convergence in two iterations (speedup factor of approximately 4.5). For the classical method, we have good convergence in four iterations (speedup factor of approximately 4.8). We notice that the classical parareal method

remains faster than the MPD, and further improvements of this last one must be done for reducing its computational cost. However, the gains in stability are a positive feature of the MPD.

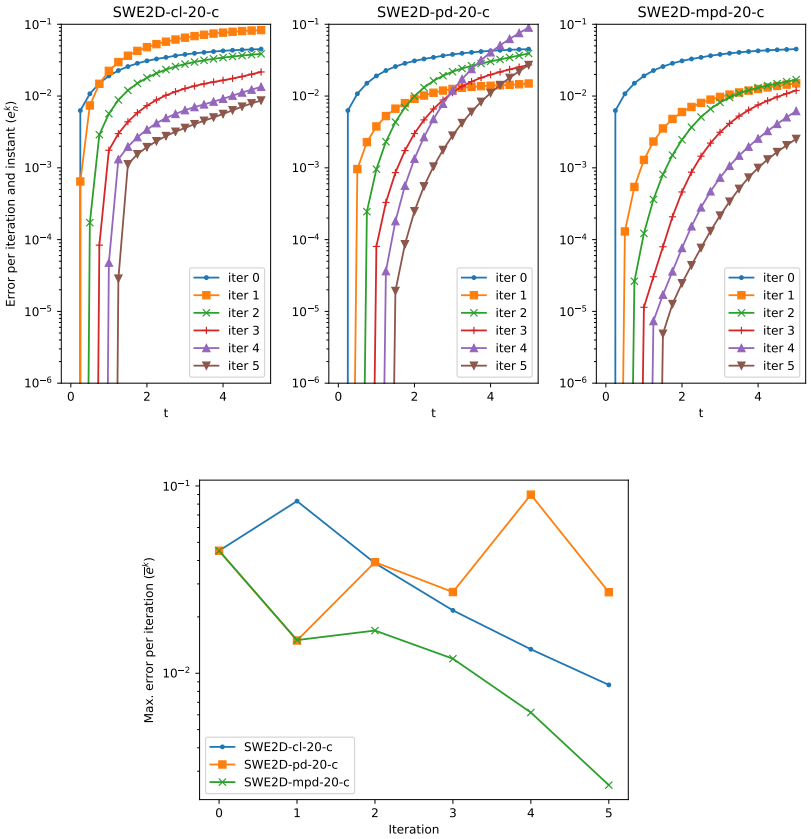

Figure 7. Evolution of the errors e_n^k along iterations and time for test cases *SWE2D-cl-20-c* (first plot), *SWE2D-pd-20-c* (second plot) and *SWE2D-mpd-20-c* (third plot); and evolution of the errors $\overline{e}^k$ along iterations for the same test cases (fourth plot).

5.3. **Discussion on the computational time of the POD-DEIM-based parareal methods**

We consider the last test case presented above (*SWE2D* with spatial coarsening) for detailing the computational cost of the POD-DEIM-based parareal method and its modification.

Figure 10 presents the average time spent in each step of PD and MPD along the parareal iterations. In these graphs, we denote:

- **FC**: time for computing the fine term of the correction (line 12 of Algorithms 4 and 5), including the storage of the snapshots;
- **SP**: time for computing the subspaces (lines 20-21 and 21-22 of Algorithms 4 and 5, respectively);
- **CC**: Time for computing the coarse term of the correction (resp. lines 25-26 and 26-27);
- **PU**: Time for computing the predictions and updating the final solution of the iteration (resp. lines 31 and 32), or, in the initial prediction (0-th iteration), the computation by the coarse propagator (line 4 of Algorithms 4 and 5);

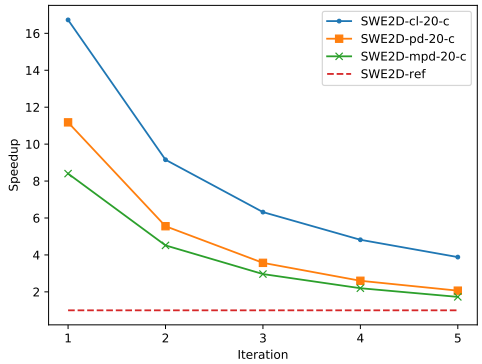

FIGURE 8. Evolution of the average speedup along iterations, given by the ratio between the referential and the parareal computational times, for test cases *SWE2D-cl-20-c* (circles), *SWE2D-pd-20-c* (squares) and *SWE2D-mpd-20-c* (crosses). The dashed line represents the referential speedup, equal to 1.

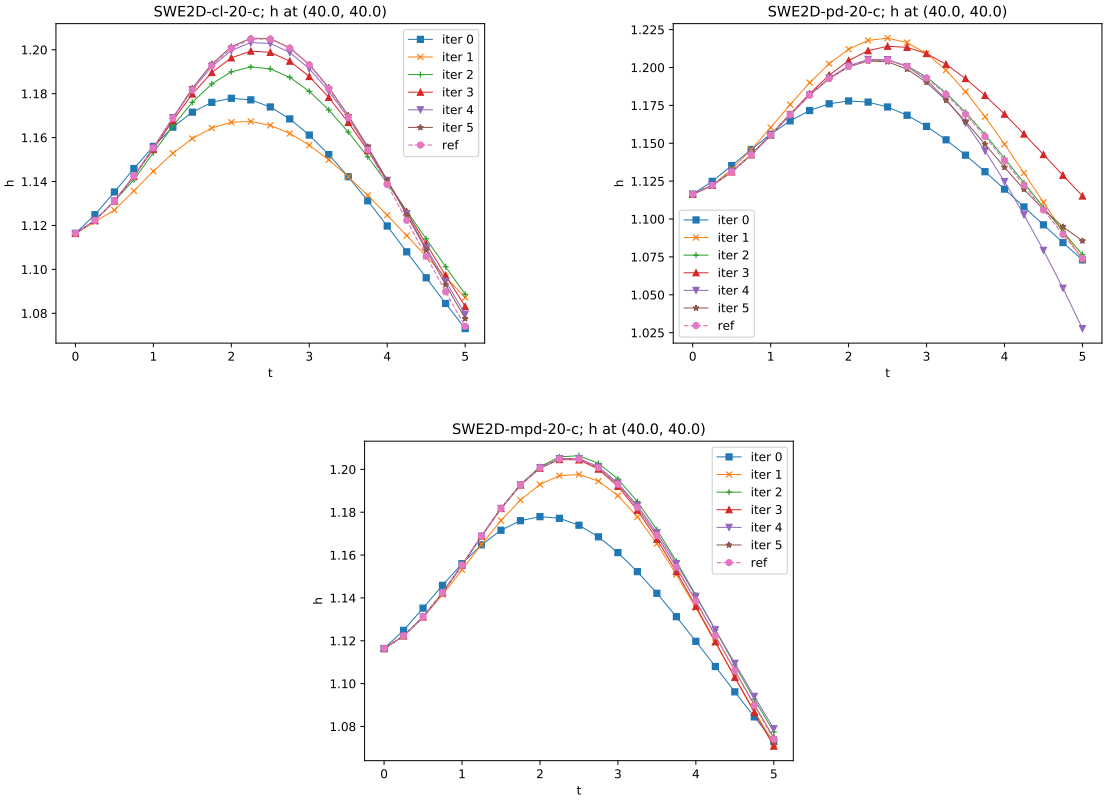

FIGURE 9. Evolution of h along time in position $(x, y) = (40, 40)$ for test cases *SWE2D-cl-20-c* (top left), *SWE2D-pd-20-c* (top-right) and *SWE2D-pd-20-c* (bottom). Circles and squares represent respectively the referential solution and the initial prediction. For the third figure, the referential solution and the parareal iterations nearly superimpose for $k \geq 2$.

- **Total**: Total time spent in the iteration.

Other iteration steps, *e.g.* the convergence checking and the computation of the matrices of the reduced-order model do not have their computational time shown explicitly in Figure 10 since they are relatively small, but are included in the total iteration time.

Several conclusions can be made from the results presented in Figure 10. We notice, as expected, that the $0-th$ iteration, consisting uniquely on a full sequential coarse simulation, has a very small computational cost. Moreover, the computation of the coarse term correction (CC), consisting on a parallel simulation of the reduced-order model, has also a negligible computational time.

Concerning the computation of the fine correction term (FC), which is done in parallel, we notice that the spent time is similar for both POD-DEIM-based parareal methods. The only exception is the first iteration in MPD, which is a little more expensive, due to the creation of the structure to store the extra snapshots. In the following iterations, the same structure is reused, and the computational time for storing the snapshots is smaller. We recall that there is no extra cost for computing these extra snapshots, since they are provided by the parallel fine simulation.

Figure 10 also provides some conclusions concerning the computation of the subspaces. As discussed in Section 4, the POD procedure for computing the basis can be estimated to have a quadratic dependence on the number of snapshots, which, in its turn, is proportional to the number of iterations and the number of time steps for taking the snapshots. Therefore, its cost increases rapidly along the parareal iteration and if we keep a large number of intermediary snapshots in the MPD. Moreover, by using more snapshots as input for the POD, we expect to have basis with higher dimensions. It affects directly the cost of the DEIM procedure, for each we estimate a dependence with exponent four on the basis dimension. The sum of this factors can be clearly observed in Figure 10: the SP curve increases much faster fort the MPD than the PD method. It reinforces the importance of having a fast convergence of the method, in very few iterations.

Finally, the influence of the number of the snapshots on the dimension of the reduced-order model can be observed in Figure 11, which represents, for the same test cases *SWE2D-pd-20-c* and *SWE2D-mpd-20-c*, the evolution, along the parareal iterations, of the subspaces $\mathcal{S}^{(l)}, l = 1, 2, 3$, computed from snapshots of the three components of the SWE solution, and the subspaces $\widehat{\mathcal{S}}^{(l)}, l = 1, 2, 3$, computed from snapshots of the three first flux terms. Concerning the two remaining flux terms, their subspaces have dimension zero since a flat bottom is considered in the simulations, so they are not represented in the graphs. In all iterations, all the subspaces computed in the MPD, using twice the number of snapshots, have a higher dimension than the subspaces computed in the PD, for the same thresholds $\varepsilon_{\text{sv,linear}}$ and $\varepsilon_{\text{sv,nonlinear}}$. As said above, it affects the computational time for the DEIM procedure. Nevertheless, the spaces remain relatively small and we do not see remarkable differences in the computational time for solving the reduced-order models (which are included in curves CC and PU in Figure 10). We recall that the dimension of the FV reduced-order is given by the dimension of the spaces $\mathcal{S}^{(i)}, i = 1, 2, 3$, with fluxes computed in a number of interfaces given by the dimension of $\widehat{\mathcal{S}}^{(l)}, l = 1, 2, 3$. Both these quantities remain in the order of tenths, whereas the mesh for the fine propagator contains 2500 cells and 5100 interfaces.

6. Conclusion

In this paper, we implement a POD-DEIM-based (PD) parareal method for solving the two-dimensional nonlinear shallow water equations, discretized using a finite volume scheme. The PD method is a further development of the Krylov-subspace-enhanced parareal method that is suitable for stabilizing and accelerating the convergence for nonlinear hyperbolic problems. We propose a modification of the method (MPD) for improving even more both these aspects.

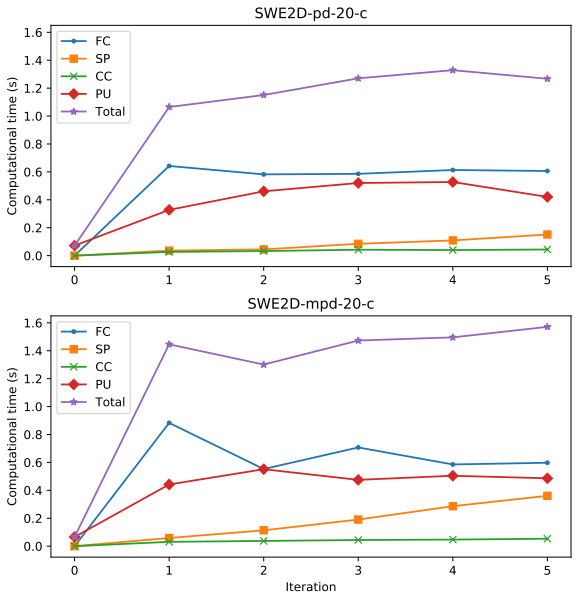

FIGURE 10. Evolution along iterations of the computational time spent in each step of the POD-DEIM-based parareal method (top, test case *SWE2D-pd-20-c*) and its modification (bottom, test case *SWE2D-mpd-20-c*). *FC*, *SP*, *CC*, *PU* and *Total* stands respectively for the computation of the fine correction term, the computation of the subspaces, the computation of the coarse correction term, the computation of the prediction and update of the solution, and the total iteration time.

This modification consists in enriching the snapshots sets used for constructing the POD-DEIM reduced-order models (ROMs). The motivation is the influence of the snapshot set on the quality of the ROM: if the snapshots are in a too few number or if they are not a good representation of the solution, the produced ROM can have a poor quality, which, in the framework of the PD parareal method, may affect its convergence and stability.

The modified POD-DEIM-based parareal method (MPD) stands out for not requiring any computational cost for obtaining the extra snapshots, since they are already computed in the fine correction step in the algorithm, but not used in the PD method. An extra cost is observed in the computation of the subspaces using the POD-DEIM procedured and can be prohibitive in too advanced parareal iterations; however, the MPD allows a convergence in very few iterations.

A preceding step for implementing the PD and the MPD is the formulation of the reduced-order model for the SWE. The nonlinearity of these equations requires a combined POD-DEIM reduction. In the context of a finite volume scheme, the obtained ROM has a dimension defined by the subspaces obtained from the POD applied to snapshots of the solution, with the nonlinear fluxes computed on interfaces chosen by the POD-DEIM applied to snapshots of the fluxes. For a test case with a reference mesh containing cells and interfaces in the order of thousands, both the ROM dimension and the number of interfaces chosen by the DEIM are in the order of tenths.

Numerical simulations are performed here for comparing the behaviour of the classical, the PD and the MPD parareal methods in terms of computational time and stability. In a first, one-dimensional test case, the classical method is unstable, whereas the PD ans specially the MPD converges exactly to reference solution in very few iterations. In a second, two-dimensional test case, the classical method presents less instabilities, but the PD and the MPD converge faster. By coarsening the spatial resolution of the coarse propagator, the PD becomes unstable, suggesting that the snapshots are not a good

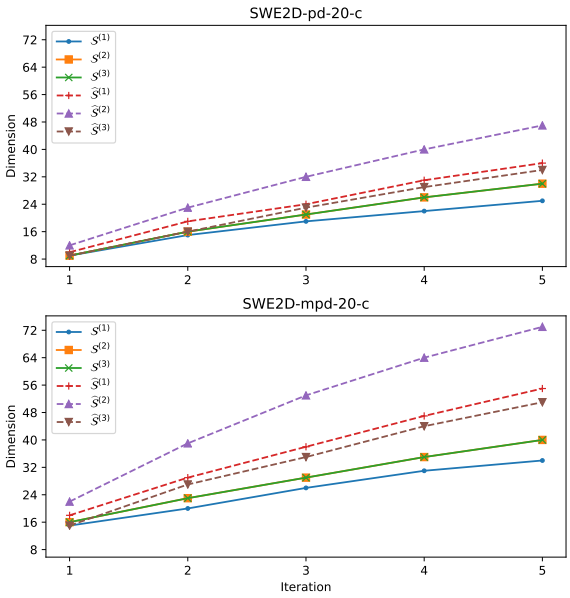

Figure 11. Evolution along iterations of the dimensions of subspaces $\mathcal{S}^{(i)}, i = 1, 2, 3$ (full lines) and $\widehat{\mathcal{S}}^{(l)}, l = 1, 2, 3$ (dashed lines) for the POD-DEIM-based parareal method (top, test case *SWE2D-pd-20-c*) and its modification (bottom, test case *SWE2D-mpd-20-c*). Subspaces $\widehat{\mathcal{S}}^{(4)}$ and $\widehat{\mathcal{S}}^{(5)}$ have zero dimension in all iterations (due to the flat bottom) and are not represented. For comparison, the fine propagator's mesh has 2500 cells and 5100 interfaces.

representation of the reference solution. By using twice the number of snapshots, the MPD is able to stabilize and accelerate the convergence.

In all the simulations, 20 parallel processors are used, and the MPD produce good approximate solutions with a good speedup factor (between 4 and 8.5 approximately, depending on the test case). However, the MPD is clearly more costful than the classical parareal method. Then, the optimization of the MPD's computational cost is a natural and necessary outlook of this work. As pointed out by [25], a task scheduling between processors as proposed in [3] is not applicable for the Krylov-subspace-enhanced parareal method (and thus for the PD and MPD methods), but similar approaches should be investigated. A coupling between spatial and temporal parallelizations, using domain decomposition and parareal techniques in an hybrid *MPI-OpenMP* implementation [18] can also be envisaged. Finally, a nested parallel computation of the subspaces, which is proposed here but not investigated, can be crucial in large problems and may be studied.

Concerning the applications of the work presented here, we look forward to simulations of urban floods using the SWE. In this case, using porosity-based shallow water models (*e.g.* the one proposed by [17]) as coarse propagator would allow to improve the quality of the initial parareal prediction and, consequently, the stability and convergence of the POD-DEIM-based parareal methods.

Appendix A. Application of the POD-DEIM-based parareal algorithm for the two-dimensional invisicid scalar Burgers equation using finite volumes

The objective of these appendices is to describe in detail the implementation of the POD-DEIM-based method (Algorithm 4) and its modification (Algorithm 5) for solving the two-dimensional nonlinear shallow water equations discretized with a finite volume (FV) scheme. For making it clearer, we

begin by formulating a reduced-order model to a simpler problem: the two-dimensional inviscid scalar Burgers equation. For the purposes of the POD-DEIM, it has the same structure as the SWE, so the reduction of this more complex problem is analogous. After the formulation of the reduced-order model, its application to the POD-DEIM-based parareal methods is straightforward.

We follow the procedure described in Section 2.3.1 for formulating the reduced-order model under the form (2.14). As, in general, the time integration in FV schemes is written separately for each spatial cell, as a very intuitive sum of flux contributions from the neighbour cells, we need first to rewrite them in a global (in space) form, in order to have a system of ODEs in time (as in eq. 2.8).

A.1. **Notations for the FV scheme**

Let us introduce some definitions for setting up the FV scheme. We consider a domain Ω discretized with a mesh $\mathcal{T}$ containing M_f cells $\Omega_i,\ i = 1, \dots, M_f$. We denote by A_i the area of Ω_i, $\mathcal{N}(i)$ the set of indices of neighbours cells to Ω_i, u_i the average of a certain function u in Ω_i, σ_{ij} the interface between the cells Ω_i and Ω_j, $j \in \mathcal{N}(i)$, and $w_{i,j}$ the length of $\sigma_{i,j}$.

We denote by $\boldsymbol{n}_{i,j} = (n^x_{i,j}, n^y_{i,j})^T$ the unit normal vector to the interface $\sigma_{i,j}$ and pointing outwards w.r.t. Ω_i. Notice that $\boldsymbol{n}_{j,i} = -\boldsymbol{n}_{i,j}$. We also define $\widetilde{\boldsymbol{n}}_{i,j} = \begin{pmatrix} \tilde{n}^x_{i,j} & \tilde{n}^y_{i,j} \end{pmatrix}^T$ as an unit normal vector to $\sigma_{i,j}$, but unique to each interface, corresponding to a chosen orientation for it (Ω_i and Ω_j would be respectively "left" and "right" cells to $\sigma_{i,j}$).

The number of interfaces in the mesh is denoted by I_f and can be indexed from 1 to I_f. We define the following maps between the interfaces indices $l = 1, \dots, I_f$ and the corresponding indices of the neighbour cells (i, j) of the interface σ_l:

$$\begin{aligned} \psi : \{1, 2, \dots, I_f\} &\to \{1, 2, \dots, M_f\}^2 \\ l &\mapsto (i, j), \qquad \sigma_l = \Omega_i \cap \Omega_j \\ \psi' : \{1, 2, \dots, M_f\}^2 &\to \{1, 2, \dots, I_f\} \\ (i, j) &\mapsto l, \qquad \sigma_l = \Omega_i \cap \Omega_j \end{aligned} \tag{A.1}$$

For each $l = 1, \dots, I_f$, $\psi(l)$ is unique and expresses the arbitrary fixed orientation for the interface. For ψ', we have $\psi'(i, j) = \psi'(j, i)$.

A.2. **Formulation of the problem and the finite volume scheme**

We consider the two-dimensional inviscid scalar Burgers equation

$$\frac{\partial u}{\partial t} + \nabla \cdot \boldsymbol{F}(u) = 0; \tag{A.2}$$

with

$$\boldsymbol{F}(u) = \frac{1}{2}u^2 \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The spatial discretization of (A.2) using a finite volumes scheme gives

$$\frac{du_i}{dt} = -\frac{1}{A_i} \sum_{j \in \mathcal{N}(i)} (\boldsymbol{F}(u_{i,j}) \cdot \boldsymbol{n}_{i,j} w_{i,j}) \tag{A.3}$$

Expression (A.3) gives the ODE satisified by each approximate solution u_i. In order to properly formulate a reduced model using the POD-DEIM procedure, we rewrite it in vector-matrix form as

$$\frac{d\boldsymbol{U}}{dt} = B\widetilde{\boldsymbol{F}}(\boldsymbol{U}) \tag{A.4}$$

with

$$\boldsymbol{U} := \begin{pmatrix} u_1 \\ \vdots \\ u_{M_f} \end{pmatrix} \in \mathbb{R}^{M_f}, \qquad \widetilde{\boldsymbol{F}}(\boldsymbol{U}) := \begin{pmatrix} \boldsymbol{F}(u_{\psi(1)}) \cdot \boldsymbol{e}_1 \\ \vdots \\ \boldsymbol{F}(u_{\psi(I_f)}) \cdot \boldsymbol{e}_1 \end{pmatrix} \in \mathbb{R}^{I_f}, \qquad \boldsymbol{e}_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

where $u_{\psi(l)}$ is an approximation to the solution on interface $\sigma_{\psi(l)}$, $l = 1, \ldots, I_f$, and $B \in \mathbb{R}^{M_f \times I_f}$ is a sparse matrix whose elements are given by

$$[B]_{i,l} = \begin{cases} -\dfrac{w_{i,j}}{A_i}(n^x_{i,j} + n^y_{i,j}), & j \in \mathcal{N}(i),\ l = \psi'(i,j), \\ 0, & \text{else} \end{cases} \tag{A.5}$$

Using an explicit Euler scheme for the temporal discretization of (A.4), we have the final numerical scheme

$$\boldsymbol{U}^{n+1} = \boldsymbol{U}^n + \Delta t B\widetilde{\boldsymbol{F}}(\boldsymbol{U}^n) \tag{A.6}$$

where $\boldsymbol{U}^n$ is the approximation of $u(t_n, \cdot)$.

Notice that the matrix B depends exclusively on geometric properties of the mesh, which are constant in time. Therefore, it can be computed only once, such that each update (A.6) only requires the computation of the flux vector $\widetilde{\boldsymbol{F}}(\boldsymbol{U})$.

A.3. **Formulation of a POD-DEIM reduced model**

We now formulate a reduced model for (A.4). The objective is to construct an approximate problem with dimension $q \ll M_f$, leading to a much smaller computational cost.

We observe that, in (A.4), the term $\widetilde{\boldsymbol{F}}(U)$ is nonlinear. Therefore, as pointed out by [5] and [6] and discussed in the previous sections, the model reduction using a POD technique is not enough for obtaining small computational costs, requiring also a DEIM approach for reducing this nonlinear term.

Let $\mathcal{U} = [\boldsymbol{U}(t_1), \ldots, \boldsymbol{U}(t_{n_s})] \in \mathbb{R}^{M_f \times n_s}$ be a matrix containing n_s snapshots of the solution of (A.4). Via a POD, we can obtain a basis of the reduced subspace $\mathcal{S}_q$ of dimension q spanned by these snapshots. We denote by $V_q \in \mathbb{R}^{M_f \times q}$ the matrix containing in its columns the orthonormal vectors of this basis.

We approximate $\boldsymbol{U}(t) \in \mathbb{R}^{M_f}$ by

$$\boldsymbol{U}(t) \approx V_q\widetilde{\boldsymbol{U}}(t) \tag{A.7}$$

with $\widetilde{\boldsymbol{U}}(t) \in \mathbb{R}^q$.

By replacing (A.7) in (A.4) and projecting the equation into $\mathcal{S}_q$, we obtain

$$\frac{d\widetilde{\boldsymbol{U}}}{dt} = V_q^T B\widetilde{\boldsymbol{F}}(V_q\widetilde{\boldsymbol{U}}) \tag{A.8}$$

The problem (A.8) has effectively a reduced dimension $q \ll M_f$, but the computation of the nonlinear term $\widetilde{\boldsymbol{F}}(V_q\widetilde{\boldsymbol{U}})$ is still expensive, since the flux need to be computed on I_f interfaces, with I_f of the same order of magnitude as M_f.

Then, let $\widehat{\mathcal{U}} = \left[\widetilde{\boldsymbol{F}}(\boldsymbol{U}(t_1)), \ldots, \widetilde{\boldsymbol{F}}(\boldsymbol{U}(t_{n_s}))\right] \in \mathbb{R}^{I_f \times n_s}$ be a matrix containing n_s snapshots of the nonlinear term of (A.4), *i.e.* the fluxes computed in n_s instants. Also using a POD technique, we obtain from these snapshots a reduced subspace $\widehat{\mathcal{S}}_m$ of dimension m and whose basis vectors are the columns of the matrix $\widehat{V}_m \in \mathbb{R}^{I_f \times m}$.

Using $\widehat{V}_m$ as input for the DEIM algorithm 3, we obtain a matrix $\widehat{P} = [\boldsymbol{e}_{\mathcal{P}_1}, \ldots, \boldsymbol{e}_{\mathcal{P}_m}] \in \mathbb{R}^{I_f \times m}$, where $\boldsymbol{e}_{\mathcal{P}_i}$ are vectors of the canonical basis and the indices $\mathcal{P}_i$ indicate a choice of $m \ll I_f$ interfaces on which the flux will be computed in the reduced problem.

Thus, the nonlinear term can be approximated by

$$\widetilde{\boldsymbol{F}}(V_q\widetilde{\boldsymbol{U}}) \approx \widehat{V}_m \left(\widehat{P}^T\widehat{V}_m\right)^{-1} \widehat{P}^T\widetilde{\boldsymbol{F}}(V_q\widetilde{\boldsymbol{U}})$$

and, by replacing in (A.8), we obtain the final reduced model

$$\frac{d\widetilde{\boldsymbol{U}}}{dt} = V_q^T B\widehat{V}_m \left(\widehat{P}^T\widehat{V}_m\right)^{-1} \widehat{P}^T\widetilde{\boldsymbol{F}}(V_q\widetilde{\boldsymbol{U}}) \tag{A.9}$$

which we denote as

$$\frac{d\widetilde{\boldsymbol{U}}}{dt} = \widehat{B}\widehat{P}^T\widetilde{\boldsymbol{F}}(V_q\widetilde{\boldsymbol{U}}) \tag{A.10}$$

where $\widehat{B} := V_q^T B\widehat{V}_m \left(\widehat{P}^T\widehat{V}_m\right)^{-1} \in \mathbb{R}^{q\times m}$ does not depend on time and can be precomputed, and $\widehat{P}^T\widetilde{\boldsymbol{F}}(V_q\widetilde{\boldsymbol{U}}) \in \mathbb{R}^m$ contains the fluxes computed on the m interfaces chosen by the DEIM algorihtm.

Appendix B. Application of the POD-DEIM-based parareal algorithm for the two-dimensional nonlinear SWE using finite volumes

B.1. Formulation of the problem and the finite volume scheme

We recall the SWE equations (5.1) as formulated by [17] and considered in this paper. For the resolution of (5.1) with a finite volume scheme, we reformulate it as a local problem to each interface. In the local coordinate system (ξ, η) attached to each interface, (5.1) reduces to

$$\frac{\partial}{\partial t}\widetilde{\boldsymbol{U}}(t) + \frac{\partial}{\partial x}\widetilde{\boldsymbol{F}}(\boldsymbol{U}(t)) = \widetilde{\boldsymbol{S}}_0(\boldsymbol{U}(t)) \tag{B.1}$$

where

$$\widetilde{\boldsymbol{U}} = \begin{pmatrix} h \\ hu \\ hv \end{pmatrix}, \qquad \widetilde{\boldsymbol{F}} = \begin{pmatrix} hu \\ hu^2 + gh^2/2 \\ huv \end{pmatrix}, \qquad \widetilde{\boldsymbol{S}}_0 = \begin{pmatrix} 0 \\ S_{0,\xi} \\ 0 \end{pmatrix}$$

with u and v as, respectively, the velocities in the directions ξ (normal to the interface) and η (tangent to the interface), and $S_{0,\xi}$ is the topography variation source term in the direction ξ. As explained in [17], the friction source term is computed in a further step and is not considered in (B.1).

For the sake of clarity, the tildes over $\boldsymbol{U}$, $\boldsymbol{F}$ and $\boldsymbol{S}$ are omitted in the following.

Using the same notation introduced in Section A.1 of the previous appendix, a finite volume discretization of the spatial variables in (B.1) leads to the system of ODEs

$$\frac{d}{dt}\boldsymbol{U}_i(t) = -\frac{1}{A_i}\sum_{j\in\mathcal{N}(i)} w_{i,j}\left[P_{i,j}\boldsymbol{F}_{i,j} + \widetilde{P}_{i,j}\left(\mathbb{1}_{\{\boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}>0\}}\boldsymbol{S}^L_{i,j} + \mathbb{1}_{\{\boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}<0\}}\boldsymbol{S}^R_{i,j}\right)\right] \tag{B.2}$$

where

$$\boldsymbol{U}_i := \begin{pmatrix} h_i \\ h_i u_i \\ h_i v_i \end{pmatrix} \in \mathbb{R}^3, \qquad i = 1, \ldots, M_f$$

and $\mathbb{1}$ is the indicator function, $\boldsymbol{F}_{i,j}$ is the average flux vector in the direction normal to the interface $\sigma_{i,j}$ and $P_{i,j}$ is a rotation matrix from the Cartesian coordinates (x, y) to the coordinates (ξ, η) attached to the interface $\sigma_{i,j}$, given by

$$P_{i,j} = \begin{pmatrix} \boldsymbol{n}_{i,j}\cdot\widetilde{\boldsymbol{n}}_{i,j} & 0 & 0 \\ 0 & n^x_{i,j} & -n^y_{i,j} \\ 0 & n^y_{i,j} & n^x_{i,j} \end{pmatrix}$$

and $\widetilde{P}_{i,j}$ is defined in a similar way:

$$\widetilde{P}_{i,j} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \tilde{n}^x_{i,j} & -\tilde{n}^y_{i,j} \\ 0 & \tilde{n}^y_{i,j} & \tilde{n}^x_{i,j} \end{pmatrix}$$

Notice that $P_{j,i} = -P_{i,j}$, In (B.2), it indicates that, for a given interface $\sigma_{i,j}$, the distribution of the flux $\boldsymbol{F}_{i,j}$ for the left and the right cells (where "left" and "right" stand for the fixed chosen orientation) have same absolute value but opposite sign. On the other hand, $\widetilde{P}_{i,j}$ is unique for each interface.

The flux $\boldsymbol{F}_{i,j}$ is defined by

$$\boldsymbol{F}_{i,j} = \begin{pmatrix} F^{[1]}_{i,j} \\ F^{[2]}_{i,j} \\ F^{[3]}_{i,j} \end{pmatrix} \tag{B.3}$$

with

$$F^{[1]}_{i,j} = \frac{1}{\lambda^+ - \lambda^-}\left[\lambda^+(uh)_L - \lambda^-(uh)_R + \lambda^-\lambda^+(z_R - z_L)\right]$$
$$F^{[2]}_{i,j} = \frac{1}{\lambda^+ - \lambda^-}\left[\lambda^+\left(hu^2 + \frac{g}{2}h^2\right)_L - \lambda^-\left(hu^2 + \frac{g}{2}h^2\right)_R + \lambda^-\lambda^+\left((hu)_R - (hu)_L\right)\right]$$
$$F^{[3]}_{i,j} = \frac{F^{[1]}_{i,j} + |F^{[1]}_{i,j}|}{2}v_L + \frac{F^{[1]}_{i,j} - |F^{[1]}_{i,j}|}{2}v_R$$

where L and R stand respectively for the arbitrary "left" and "right" cells w.r.t. interface $\sigma_{i,j}$, z_L and z_R are their respective surface elevations and the wave celerities λ^- and λ^+ are estimated by

$$\lambda^- = \min\{u_L - c_L, u_R - c_R, 0\}, \qquad \lambda^+ = \max\{u_L + c_L, u_R + c_R, 0\}$$

with $c_L = \sqrt{gh_L}$ and $c_R = \sqrt{gh_R}$.

The separation of the source term $\boldsymbol{S}_0$ into two vectors in (B.2) indicates that, contrary to the flux term $\boldsymbol{F}$, the contribution of the source term for the left and right cell to each interface are not computed in the same way, namely:

$$\boldsymbol{S}_{i,j}^{L} = \begin{pmatrix} 0 \\ \dfrac{-\lambda^{-}}{\lambda^{+}-\lambda^{-}}\Delta S_{i,j} \\ 0 \end{pmatrix}, \qquad \boldsymbol{S}_{i,j}^{R} = \begin{pmatrix} 0 \\ \dfrac{\lambda^{+}}{\lambda^{+}-\lambda^{-}}\Delta S_{i,j} \\ 0 \end{pmatrix}$$

where

$$\Delta S_{i,j} = \frac{g}{2}(h_L + h_R)(z_L - z_R + h_R - h_L)$$

In order to formulate a reduced problem for the shallow water equations, we follow the same procedure as in the case of the Burgers equations and we begin by rewriting (B.2) under a global, matricial form, corresponding to a system of ODEs in time for all the cells in the mesh.

We first define five flux vectors, corresponding to the three components of $\boldsymbol{F}_{i,j}$ and the non-zero components of $\boldsymbol{S}_{i,j}^{L}$ and $\boldsymbol{S}_{i,j}^{R}$:

$$\begin{aligned} \boldsymbol{F}^{(1)} &:= \begin{pmatrix} \boldsymbol{F}_{\psi(1)} \cdot \boldsymbol{e}_1 \\ \vdots \\ \boldsymbol{F}_{\psi(I_f)} \cdot \boldsymbol{e}_1 \end{pmatrix}, \qquad \boldsymbol{F}^{(2)} := \begin{pmatrix} \boldsymbol{F}_{\psi(1)} \cdot \boldsymbol{e}_2 \\ \vdots \\ \boldsymbol{F}_{\psi(I_f)} \cdot \boldsymbol{e}_2 \end{pmatrix}, \\ \boldsymbol{F}^{(3)} &:= \begin{pmatrix} \boldsymbol{F}_{\psi(1)} \cdot \boldsymbol{e}_3 \\ \vdots \\ \boldsymbol{F}_{\psi(I_f)} \cdot \boldsymbol{e}_3 \end{pmatrix}, \qquad \boldsymbol{F}^{(4)} := \begin{pmatrix} \boldsymbol{S}^{L}_{\psi(1)} \cdot \boldsymbol{e}_2 \\ \vdots \\ \boldsymbol{S}^{L}_{\psi(I_f)} \cdot \boldsymbol{e}_2 \end{pmatrix}, \\ \boldsymbol{F}^{(5)} &:= \begin{pmatrix} \boldsymbol{S}^{R}_{\psi(1)} \cdot \boldsymbol{e}_2 \\ \vdots \\ \boldsymbol{S}^{R}_{\psi(I_f)} \cdot \boldsymbol{e}_2 \end{pmatrix} \end{aligned} \tag{B.4}$$

where $\boldsymbol{e}_i, i = 1, 2, 3$ are the vectors of the canonical basis of $\mathbb{R}^3$ and $\boldsymbol{F}^{(l)} \in \mathbb{R}^{I_f}, l = 1, \ldots, 5$. In the sequel, $\boldsymbol{F}_{i,j}^{(l)}$ refers to the $\psi'(i,j)$-th component of $\boldsymbol{F}^{(l)}$.

We also define three vectors containing the three components of the solution in each cell:

$$\boldsymbol{U}^{(1)} := \begin{pmatrix} \boldsymbol{U}_1 \cdot \boldsymbol{e}_1 \\ \vdots \\ \boldsymbol{U}_{M_f} \cdot \boldsymbol{e}_1 \end{pmatrix}, \qquad \boldsymbol{U}^{(2)} := \begin{pmatrix} \boldsymbol{U}_1 \cdot \boldsymbol{e}_2 \\ \vdots \\ \boldsymbol{U}_{M_f} \cdot \boldsymbol{e}_2 \end{pmatrix}, \qquad \boldsymbol{U}^{(3)} := \begin{pmatrix} \boldsymbol{U}_1 \cdot \boldsymbol{e}_3 \\ \vdots \\ \boldsymbol{U}_{M_f} \cdot \boldsymbol{e}_3 \end{pmatrix} \tag{B.5}$$

with $\boldsymbol{U}^{(l)} \in \mathbb{R}^{M_f}, l = 1, 2, 3$.

Using these definitions, (B.2) can be rewritten, for each variable separately, as

$$
\begin{aligned}
\frac{d}{dt}\boldsymbol{U}_i^{(1)}(t) &= -\frac{1}{A_i}\sum_{j\in\mathcal{N}(i)} && (\boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j})w_{i,j}\boldsymbol{F}_{i,j}^{(1)} \\
\frac{d}{dt}\boldsymbol{U}_i^{(2)}(t) &= -\frac{1}{A_i}\sum_{j\in\mathcal{N}(i)} && \Big[n_{i,j}^x\boldsymbol{F}_{i,j}^{(2)} - n_{i,j}^y\boldsymbol{F}_{i,j}^{(3)} \\
& && + \tilde{n}_{i,j}^x\mathbb{1}_{\{\boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}>0\}}\boldsymbol{F}_{i,j}^{(4)} \\
& && +\tilde{n}_{i,j}^x\mathbb{1}_{\{\boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}<0\}}\boldsymbol{F}_{i,j}^{(5)}\Big]\, w_{i,j} \\
\frac{d}{dt}\boldsymbol{U}_i^{(3)}(t) &= -\frac{1}{A_i}\sum_{j\in\mathcal{N}(i)} && \Big[n_{i,j}^y\boldsymbol{F}_{i,j}^{(2)} + n_{i,j}^x\boldsymbol{F}_{i,j}^{(3)} \\
& && + \tilde{n}_{i,j}^y\mathbb{1}_{\{\boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}>0\}}\boldsymbol{F}_{i,j}^{(4)} \\
& && +\tilde{n}_{i,j}^y\mathbb{1}_{\{\boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}<0\}}\boldsymbol{F}_{i,j}^{(5)}\Big]\, w_{i,j}
\end{aligned}
\tag{B.6}
$$

Let us define the sparse matrices $B^{(1)}, B^{(2)}, B^{(3)}, B^{(4,x)}, B^{(4,y)}, B^{(5,x)}, B^{(5,y)} \in \mathbb{R}^{M_f\times I_f}$ by

$$
[B^{(1)}]_{i,l} = \begin{cases} -\left(\boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}\right)\dfrac{w_{i,j}}{A_i}, & j\in\mathcal{N}(i),\ l=\psi'(i,j), \\ 0, & \text{else} \end{cases}
$$

$$
[B^{(2)}]_{i,l} = \begin{cases} -n_{i,j}^x\dfrac{w_{i,j}}{A_i}, & j\in\mathcal{N}(i),\ l=\psi'(i,j), \\ 0, & \text{else} \end{cases}
$$

$$
[B^{(3)}]_{i,l} = \begin{cases} n_{i,j}^y\dfrac{w_{i,j}}{A_i}, & j\in\mathcal{N}(i),\ l=\psi'(i,j), \\ 0, & \text{else} \end{cases}
$$

$$
[B^{(4,x)}]_{i,l} = \begin{cases} n_{i,j}^x\dfrac{w_{i,j}}{A_i}, & j\in\mathcal{N}(i),\ \boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}>0,\ l=\psi'(i,j), \\ 0, & \text{else} \end{cases}
$$

$$
[B^{(4,y)}]_{i,l} = \begin{cases} n_{i,j}^y\dfrac{w_{i,j}}{A_i}, & j\in\mathcal{N}(i),\ \boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}>0,\ l=\psi'(i,j), \\ 0, & \text{else} \end{cases}
$$

$$
[B^{(5,x)}]_{i,l} = \begin{cases} n_{i,j}^x\dfrac{w_{i,j}}{A_i}, & j\in\mathcal{N}(i),\ \boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}<0,\ l=\psi'(i,j), \\ 0, & \text{else} \end{cases}
$$

$$
[B^{(5,y)}]_{i,l} = \begin{cases} n_{i,j}^y\dfrac{w_{i,j}}{A_i}, & j\in\mathcal{N}(i),\ \boldsymbol{n}_{i,j}\cdot\tilde{\boldsymbol{n}}_{i,j}<0,\ l=\psi'(i,j), \\ 0, & \text{else} \end{cases}
$$

Then, (B.6) can be written under a global form as

$$\begin{aligned}
\frac{d}{dt}\boldsymbol{U}^{(1)} &= B^{(1)}\boldsymbol{F}^{(1)}(\boldsymbol{U}) \\
\frac{d}{dt}\boldsymbol{U}^{(2)} &= B^{(2)}\boldsymbol{F}^{(2)}(\boldsymbol{U}) + B^{(3)}\boldsymbol{F}^{(3)}(\boldsymbol{U}) + \\
&\quad B^{(4,x)}\boldsymbol{F}^{(4)}(\boldsymbol{U}) + B^{(5,x)}\boldsymbol{F}^{(5)}(\boldsymbol{U}) \\
\frac{d}{dt}\boldsymbol{U}^{(3)} &= -B^{(3)}\boldsymbol{F}^{(2)}(\boldsymbol{U}) + B^{(2)}\boldsymbol{F}^{(3)}(\boldsymbol{U}) + \\
&\quad B^{(4,y)}\boldsymbol{F}^{(4)}(\boldsymbol{U}) + B^{(5,y)}\boldsymbol{F}^{(5)}(\boldsymbol{U})
\end{aligned} \tag{B.7}$$

Notice that the matrices $B^{(1)}, B^{(2)}, B^{(3)}, B^{(4,x)}, B^{(4,y)}, B^{(5,x)}, B^{(5,y)}$ depend only on mesh properties, which are constant in time. Therefore, for computing the time evolution of (B.7), these matrices can be computed only once, in the beginning of the simulation. We use an explicit Euler scheme for the temporal discretization of (B.7).

B.2. **Formulation of the reduced POD-DEIM model**

Reduced-order models using the POD-DEIM technique are developed for the SWE by [29] and for the spherical SWE by [28], both in the framework of finite difference methods. We formulate in this section a reduced model for the system of finite volume discretization (B.7). We notice that these equations have a similar structure than the Burgers equation (A.4): they are ODEs in which the right-hand side is a nonlinear term (or a sum of nonlinear terms). Therefore, the procedure used here is completely analogous.

Let $\mathcal{S}_{q_i}^{(i)}, i = 1, 2, 3$, be respectively the reduced subspaces of dimension $q_i \ll M_f$ spanned by snapshots of the solutions $\boldsymbol{U}^{(i)}$ defined in (B.5), and $V_{q_i}^{(i)} \in \mathbb{R}^{M_f \times q_i}$ their respective matrices of orthonormal basis vectors, obtained via the POD.

Let also $\hat{\mathcal{S}}_{m_l}^{(l)}, l = 1, \ldots, 5$ be respectively the reduced subspaces of dimension $m_l \ll I_f$ generated by snapshots of the nonlinear terms $\boldsymbol{F}^{(l)}(\boldsymbol{U})$ defined in (B.4), $\widehat{V}_{m_l}^{(l)} \in \mathbb{R}^{I_f \times m_l}$ their respective matrices of orthonormal basis vectors, obtained via the POD, and $\widehat{P}^{(l)} \in \mathbb{R}^{I_f \times m_l}$ their respective indices matrix obtained via the DEIM algorithm.

The reduced variables are denoted, for $i = 1, 2, 3$, with $\widetilde{\boldsymbol{U}}^{(i)} \in \mathbb{R}^{q_i}$, and we use the approximation $\boldsymbol{U}^{(i)} \approx V_{q_i}^{(i)} \widetilde{\boldsymbol{U}}^{(i)}$. We also define

$$\widetilde{\boldsymbol{U}} = \begin{pmatrix} \widetilde{\boldsymbol{U}}^{(1)} \\ \widetilde{\boldsymbol{U}}^{(2)} \\ \widetilde{\boldsymbol{U}}^{(3)} \end{pmatrix} \in \mathbb{R}^{q_1+q_2+q_3}$$

and, with some abuse of notation, we denote

$$V_q \widetilde{\boldsymbol{U}} = \begin{pmatrix} V_{q_1}^{(1)} \widetilde{\boldsymbol{U}}^{(1)} \\ V_{q_2}^{(2)} \widetilde{\boldsymbol{U}}^{(2)} \\ V_{q_3}^{(3)} \widetilde{\boldsymbol{U}}^{(3)} \end{pmatrix} \in \mathbb{R}^{3M_f} \tag{B.8}$$

Then, the reduced model for (B.7) reads

$$
\begin{aligned}
\frac{d}{dt}\widetilde{\boldsymbol{U}}^{(1)} &= \widehat{B}^{(1)}\left(\widehat{P}^{(1)}\right)^T \boldsymbol{F}^{(1)}\left(V_q\widetilde{\boldsymbol{U}}\right) \\
\frac{d}{dt}\widetilde{\boldsymbol{U}}^{(2)} &= \widehat{B}^{(2,1)}\left(\widehat{P}^{(2)}\right)^T \boldsymbol{F}^{(2)}\left(V_q\widetilde{\boldsymbol{U}}\right) + \widehat{B}^{(2,2)}\left(\widehat{P}^{(3)}\right)^T \boldsymbol{F}^{(3)}\left(V_q\widetilde{\boldsymbol{U}}\right) + \\
&\quad \widehat{B}^{(2,3)}\left(\widehat{P}^{(4)}\right)^T \boldsymbol{F}^{(4)}\left(V_q\widetilde{\boldsymbol{U}}\right) + \widehat{B}^{(2,4)}\left(\widehat{P}^{(5)}\right)^T \boldsymbol{F}^{(5)}\left(V_q\widetilde{\boldsymbol{U}}\right) \\
\frac{d}{dt}\widetilde{\boldsymbol{U}}^{(3)} &= \widehat{B}^{(3,1)}\left(\widehat{P}^{(2)}\right)^T \boldsymbol{F}^{(2)}\left(V_q\widetilde{\boldsymbol{U}}\right) + \widehat{B}^{(3,2)}\left(\widehat{P}^{(3)}\right)^T \boldsymbol{F}^{(3)}\left(V_q\widetilde{\boldsymbol{U}}\right) + \\
&\quad \widehat{B}^{(3,3)}\left(\widehat{P}^{(4)}\right)^T \boldsymbol{F}^{(4)}\left(V_q\widetilde{\boldsymbol{U}}\right) + \widehat{B}^{(3,4)}\left(\widehat{P}^{(5)}\right)^T \boldsymbol{F}^{(5)}\left(V_q\widetilde{\boldsymbol{U}}\right)
\end{aligned}
\tag{B.9}
$$

where the matrices $\widehat{B}^{(*)}$ are constant in time and are defined by

$$
\begin{aligned}
\widehat{B}^{(1)} &:= \left(V_{q_1}^{(1)}\right)^T B^{(1)} \widehat{V}_{m_1}^{(1)} \left[\left(\widehat{P}^{(1)}\right)^T \widehat{V}_{m_1}^{(1)}\right]^{-1} \in \mathbb{R}^{q_1\times m_1} \\
\widehat{B}^{(2,1)} &:= \left(V_{q_2}^{(2)}\right)^T B^{(2)} \widehat{V}_{m_2}^{(2)} \left[\left(\widehat{P}^{(2)}\right)^T \widehat{V}_{m_2}^{(2)}\right]^{-1} \in \mathbb{R}^{q_2\times m_2} \\
\widehat{B}^{(2,2)} &:= \left(V_{q_2}^{(2)}\right)^T B^{(3)} \widehat{V}_{m_3}^{(3)} \left[\left(\widehat{P}^{(3)}\right)^T \widehat{V}_{m_3}^{(3)}\right]^{-1} \in \mathbb{R}^{q_2\times m_3} \\
\widehat{B}^{(2,3)} &:= \left(V_{q_2}^{(2)}\right)^T B^{(4,x)} \widehat{V}_{m_4}^{(4)} \left[\left(\widehat{P}^{(4)}\right)^T \widehat{V}_{m_4}^{(4)}\right]^{-1} \in \mathbb{R}^{q_2\times m_4} \\
\widehat{B}^{(2,4)} &:= \left(V_{q_2}^{(2)}\right)^T B^{(5,x)} \widehat{V}_{m_5}^{(5)} \left[\left(\widehat{P}^{(5)}\right)^T \widehat{V}_{m_5}^{(5)}\right]^{-1} \in \mathbb{R}^{q_2\times m_5} \\
\widehat{B}^{(3,1)} &:= -\left(V_{q_3}^{(3)}\right)^T B^{(3)} \widehat{V}_{m_2}^{(2)} \left[\left(\widehat{P}^{(2)}\right)^T \widehat{V}_{m_2}^{(2)}\right]^{-1} \in \mathbb{R}^{q_3\times m_2} \\
\widehat{B}^{(3,2)} &:= \left(V_{q_3}^{(3)}\right)^T B^{(2)} \widehat{V}_{m_3}^{(3)} \left[\left(\widehat{P}^{(3)}\right)^T \widehat{V}_{m_3}^{(3)}\right]^{-1} \in \mathbb{R}^{q_3\times m_3} \\
\widehat{B}^{(3,3)} &:= \left(V_{q_3}^{(3)}\right)^T B^{(4,y)} \widehat{V}_{m_4}^{(4)} \left[\left(\widehat{P}^{(4)}\right)^T \widehat{V}_{m_4}^{(4)}\right]^{-1} \in \mathbb{R}^{q_3\times m_4} \\
\widehat{B}^{(3,4)} &:= \left(V_{q_3}^{(3)}\right)^T B^{(5,y)} \widehat{V}_{m_5}^{(5)} \left[\left(\widehat{P}^{(5)}\right)^T \widehat{V}_{m_5}^{(5)}\right]^{-1} \in \mathbb{R}^{q_3\times m_5}
\end{aligned}
$$

Notice that (B.9) is a problem of dimension q_1, q_2 and q_3 respectively on the three components of the solution, instead of a dimension M_f in each component, as in the full problem (B.7). Moreover, the fluxes $F^{(l)}, l = 1, \ldots, 5$ are computed respectively on m_l interfaces, instead of I_f interfaces as in (B.7).

B.3. **Formulation of a POD-DEIM parareal method**

Finally, we formulate the POD-DEIM-based parareal method (Algorithm 4 or 5) by defining the following propagators:

- The fine propagator $\mathcal{F}_{\delta t}$: discretization of (B.7), with M_f cells and time step δt;
- The coarse propagator $\mathcal{G}_{\Delta t}$: discretization of (B.7), with M_c cells and time step Δt;

- The reduced fine propagator $\mathcal{F}_{r,\delta t}$: discretization of (B.9), with $q_i \ll M_f,\ i = 1, 2, 3,\ m_l \ll I_f,\ l = 1, \ldots, 5$, and time step δt, to be reformulated in each iteration of the algorithm.

Acknowledgments

The authors are grateful to the OPAL infrastructure from Université Côte d'Azur and Inria Sophia Antipolis - Méditerranée "NEF" computation platform for providing resources and support.

Bibliography

[1] Nissrine Akkari, Renaud Mercier, Ghislain Lartigue, and Vincent Moureau. Stable POD-Galerkin Reduced Order Models for unsteady turbulent incompressible flows. In *55th AIAA Aerospace Sciences Meeting*, Grapevine, United States, 2017.

[2] Matteo Astorino, Franz Chouly, and Alfio Quarteroni. Multiscale coupling of finite element and lattice Boltzmann methods for time dependent problems. Technical report, October 2012.

[3] Eric Aubanel. Scheduling of tasks in the parareal algorithm. *Parallel Computing*, 37(3):172 – 182, 2011.

[4] Christophe Audouze, Marc Massot, and Sebastian Volz. Symplectic multi-time step parareal algorithms applied to molecular dynamics. Submitted to SIAM Journal of Scientific Computing, February 2009.

[5] Saifon Chaturantabut and Danny C. Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32(5):2737–2764, 2010.

[6] Feng Chen, Jan S. Hesthaven, and Xueyu Zhu. *On the Use of Reduced Basis Methods to Accelerate and Stabilize the Parareal Method*, pages 187–214. Springer International Publishing, Cham, 2014.

[7] Wang Chen, Jan S. Hesthaven, Bai Junqiang, Yasong Qiu, Yang Tihao, and Zhang Yang. Greedy non-intrusive reduced order model for fluid dynamics. *AIAA Journal*, 56:12, 2018.

[8] Julien Cortial and Charbel Farhat. A time-parallel implicit method for accelerating the solution of non-linear structural dynamics problems. *International Journal for Numerical Methods in Engineering*, 77(4):451–470, 2009.

[9] Xiaoying Dai and Yvon Maday. Stable parareal in time method for first- and second-order hyperbolic systems. *SIAM Journal on Scientific Computing*, 35(1):A52–A78, 2013.

[10] Jack Dongarra, Mark Gates, Azzam Haidar, Jakub Kurzak, Piotr Luszczek, Stanimire Tomov, and Ichitaro Yamazaki. The singular value decomposition: Anatomy of optimizing an algorithm for extreme scale. *SIAM Review*, 60(4):808–865, 2018.

[11] Max Duarte, Marc Massot, and Stéphane Descombes. Parareal operator splitting techniques for multi-scale reaction waves: Numerical analysis and strategies. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 45(5):825–852, 2011.

[12] Araz Eghbal, Andrew G. Gerber, and Eric Aubanel. Acceleration of unsteady hydrodynamic simulations using the parareal algorithm. *Journal of Computational Science*, 19:57 – 76, 2017.

[13] Charbel Farhat and Marion Chandesris. Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid–structure applications. *International Journal for Numerical Methods in Engineering*, 58(9):1397–1434, 2003.

[14] Paul F. Fischer, Frédéric Hecht, and Yvon Maday. A parareal in time semi-implicit approximation of the Navier-Stokes equations. In Timothy J. Barth, Michael Griebel, David E. Keyes, Risto M. Nieminen, Dirk Roose, Tamar Schlick, Ralf Kornhuber, Ronald Hoppe, Jacques Périaux, Olivier Pironneau, Olof Widlund, and Jinchao Xu, editors, *Domain Decomposition Methods in Science and Engineering*, pages 433–440, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[15] Martin J. Gander. Analysis of the parareal algorithm applied to hyperbolic problems using characteristics. *Boletín de la Sociedad Española de Matemática Aplicada*, 42:21–35, 2008. ID: unige:6268.

[16] Martin J. Gander and Madalina Petcu. Analysis of a Krylov subspace enhanced parareal algorithm for linear problems. *ESAIM: Proc.*, 25:114–129, 2008.

[17] Vincent Guinot and Sandra Soares-Frazão. Flux and source term discretization in two-dimensional shallow water models with porosity on unstructured grids. *International Journal for Numerical Methods in Fluids*, 50(3):309–345, 2006.

[18] Ronald Haynes. MPI–OpenMP algorithms for the parallel space–time solution of time dependent PDEs. *Lecture Notes Computational Sciences and Engineering*, 98, 01 2014.

[19] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'edp par un schéma en temps 'pararéel'. *Comptes Rendus de l'Académie des Sciences - Series I - Mathematics*, 332(7):661 – 668, 2001.

[20] Yvon Maday. The 'parareal in time' algorithm. In Frédéric Magoulès, editor, *Substructuring Techniques and Domain Decomposition Methods*, Computational science, engineering and technology series, chapter 2, pages 19–44. Saxe-Coburg Publications, 2010.

[21] Markus Müller. *On the POD method: an abstract investigation with applications to reduced-order modeling and suboptimal control*. PhD thesis, Georg-August-Universität, Göttingen, 2008.

[22] Gilles Pagès, Olivier Pironneau, and Guillaume Sall. The parareal algorithm for american options. *Comptes Rendus Mathematique*, 354(11):1132 – 1138, 2016.

[23] Clarence W. Rowley. Model reduction for fluids, using balanced proper orthogonal decomposition. *International Journal of Bifurcation and Chaos*, 15(03):997–1013, 2005.

[24] Daniel Ruprecht. Wave propagation characteristics of Parareal. *Computing and Visualization in Science*, 19:1–17, june 2018.

[25] Daniel Ruprecht and Rolf Krause. Explicit parallel-in-time integration of a linear acoustic-advection system. *Computers Fluids*, 59:72–83, Apr 2012.

[26] Henri Samuel. Time domain parallelization for computational geodynamics. *Geochemistry, Geophysics, Geosystems*, 13(1), 2012.

[27] Joana M. F. da Trindade and José F. Pereira. Parallel-in-time simulation of the unsteady Navier–Stokes equations for incompressible flow. *International Journal for Numerical Methods in Fluids*, 45(10):1123–1136, 2004.

[28] Pengfei Zhao, Cai Liu, and Xuan Feng. POD-DEIM based model order reduction for the spherical shallow water equations with Turkel-Zwas finite difference discretization. *J. Appl. Math.*, 2014:10 pages, 2014.

[29] Razvan Ştefănescu and Ionel M. Navon. POD/DEIM nonlinear model order reduction of an ADI implicit shallow water equations model. *Journal of Computational Physics*, 237:95–114, Mar 2013.