

ParaFlow: Parareal Acceleration of Gradient-Flow Minimization

Paolo Bellezza,
Gabriele Ciaramella^[0000-0002-5877-4426],
Camilla Macchini,
Ilario Mazzieri^[0000-0003-4121-8092],
Marco Verani^[0000-0001-8015-4947]

1 Introduction

In this paper, we introduce the ParaFlow class of algorithms and the ParaFlowS algorithm. ParaFlow is an optimization framework that uses the Parareal algorithm to accelerate the descent of the gradient flow towards a minimum point. For the Parareal algorithm and the gradient flow framework, we refer, respectively, to [2] and [5], and references therein. ParaFlowS, a specific optimization algorithm of the ParaFlow class, combines Parareal and (stochastic) gradient descent (GD) to create a purely sequential optimization algorithm - hence the "S." It is important to note that ParaFlow does not aim to accelerate the whole gradient flow trajectory calculations or improve accuracy. Rather, it aims to accelerate the optimization process to reach a minimum point. In this work, we focus on an unconstrained optimization problem associated to the training of a fully connected neural network (see, e.g., [3]):

$$\min_{\mathbf{W} \in \mathbb{R}^d} L(\mathbf{W}) := \frac{1}{N} \sum_{i=1}^N \frac{1}{2} \left\| P_i \mathbf{y}(\mathbf{x}^i) - P_i \mathbf{a}^{[L]}(\mathbf{x}^i; \mathbf{W}) \right\|_2^2. \quad (1)$$

Here, the set of pairs $\{(\mathbf{y}(\mathbf{x}^i), \mathbf{x}^i)\}_{i=1}^N \subset \mathbb{R}^O \times \mathbb{R}^I$, $I, O \in \mathbb{N}^+$, is the training data set, $\mathbf{a}^{[L]}(\mathbf{x}^i; \mathbf{W})$ is the final state of the fully connected neural network that starting with $\mathbf{a}^{[1]}(\mathbf{x}; \mathbf{W}) = \mathbf{x}$, is then defined via the usual formula [3]: $\mathbf{a}^{[\ell]}(\mathbf{x}^i; \mathbf{W}) = \sigma(W^{[\ell]} \mathbf{a}^{[\ell-1]} + \mathbf{b}^{[\ell]})$, for $\ell = 2, \dots, L$. Here, σ is an activation function, $W^{[\ell]} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ and $\mathbf{b}^{[\ell]} \in \mathbb{R}^{n_\ell}$, with $n_\ell \in \mathbb{N}_+$ for $\ell = 1, \dots, N$, are weights and biases, which we collect in the vector $\mathbf{W} \in \mathbb{R}^d$, $d = \sum_{\ell=1}^N n_\ell + \sum_{\ell=1}^{N-1} n_\ell n_{\ell+1}$, used in (1). The operators P_i in (1) represent different observations of the state variables, distinguishing between categorical loss functions used in classification problems

Paolo Bellezza · Gabriele Ciaramella · Camilla Macchini · Ilario Mazzieri · Marco Verani
MOX Lab, Dipartimento di Matematica, Politecnico di Milano, Italy, e-mail:
{gabriele.ciaramella, ilario.mazzieri, marco.verani}@polimi.it,
{paolo.bellezza, camilla.macchini}@mail.polimi.it

(where P_i are identity operators) and residual-based losses, such as those arising in PINN formulations (where P_i encode differential and boundary operators). Please note that this short manuscript focuses solely on the optimization phase of the training process; generalization errors will be addressed in future work.

This manuscript is structured as follows. Section 2 reviews the continuous and discrete gradient-flow trajectories that form the basis of our study. Section 3 introduces ParaFlow and ParaFlowS. The ParaFlowS performance is evaluated in Section 4 through extensive numerical experiments on test problems of the form (1).

2 Continuous and discrete gradient-flow trajectories

The gradient-flow trajectory $\mathbf{W}(t)$ of (1) satisfies the differential equation

$$\frac{\partial \mathbf{W}(t)}{\partial t} = -\nabla L(\mathbf{W}(t)) \quad \text{for } t \in [0, \infty), \quad (2)$$

initialized at $t = 0$ by a given $\mathbf{W}^0 \in \mathbb{R}^d$. By discretizing (2) with the Explicit-Euler method, one gets the classical GD formula [5]: $\mathbf{W}^{n+1} = \mathbf{W}^n - \eta \nabla L(\mathbf{W}^n)$, where $\eta > 0$ is the step length (also called time step, or learning rate). Notice that $\nabla L(\mathbf{W}^n) = \sum_{i=1}^N \nabla L_i(\mathbf{W}^n)$, where $L_i(\mathbf{W}^n) := \frac{1}{2} \|P_i \mathbf{y}(\mathbf{x}^i) - P_i \mathbf{a}^{[L]}(\mathbf{x}^i)\|_2^2$. Since the size N of the training set is usually large, the computation of the full gradient $\nabla L(\mathbf{W}^n)$ can be prohibitively expensive. A possible remedy is to approximate the full gradient: $\mathbf{W}^{n+1} = \mathbf{W}^n - \eta \sum_{i \in \mathcal{I}_n} \nabla L_i(\mathbf{W}^n)$, where \mathcal{I}_n is a uniformly at random selected subset of $\{1, \dots, N\}$. If \mathcal{I}_n is a singleton, then one gets the simplest version of the stochastic gradient algorithm. In alternative, one can consider a set \mathcal{I}_n of $m > 1$ elements leading to the so-called mini-batch approach. For other possibilities, we refer to [3, 1] and references therein. Notice that the computation of the step-length η is not a simple task. When the full gradient is available, one can rely on line-search algorithms. For stochastic gradient (and related methods), choosing η is nontrivial, as it must satisfy specific decay rules (see [1]). In this short contribution, η is kept constant; further investigations are left for future work.

3 ParaFlow and ParaFlowS

Parareal is an iterative algorithm designed for the parallel-in-time integration of dynamical systems. To do so, given a finite time interval $[t_0, t_{N_G}]$, where a dynamical system is defined, one considers a decomposition into N_G non-overlapping subintervals: $[t_0, t_{N_G}] := \bigcup_{i=0}^{N_G-1} [t_i, t_{i+1}]$, where $t_i < t_{i+1}$. Now, considering for example the dynamical system (2) on $[t_0, T]$, Parareal is defined through coarse and fine propagator operators defined as follows. Restricting (2) on a subinterval $[t_i, t_{i+1}]$, The coarse propagator, denoted by $G(t_i, t_{i+1}, \overline{\mathbf{W}})$, computes a rough approximation to the solution at time t_{i+1} , starting from the initial condition $\mathbf{W}(t_i) = \overline{\mathbf{W}}$. Similarly, the

fine operator, denoted by $F(t_i, t_{i+1}, \widetilde{\mathbf{W}})$, computes a more accurate approximation of $\mathbf{W}(t_{i+1})$.¹ Starting from a set $\{\mathbf{W}_{n,0}\}_{n=0}^{N_G}$ of initial approximations $\mathbf{W}_{n,0} \approx \mathbf{W}(t_n)$, Parareal updates them, for $\ell = 1, 2, 3, \dots$ and $n = 0, \dots, N_G - 1$, by the formula

$$\mathbf{W}_{n+1,\ell+1} = G(t_n, t_{n+1}, \mathbf{W}_{n,\ell+1}) + F(t_n, t_{n+1}, \mathbf{W}_{n,\ell}) - G(t_n, t_{n+1}, \mathbf{W}_{n,\ell}). \quad (3)$$

Here, ℓ and n are iteration and time indices, respectively. Notice that while the coarse propagator terms need to be computed sequentially, the fine propagator ones, that are $F(t_n, t_{n+1}, \mathbf{W}_{n,\ell})$, can be computed in parallel, since they all depend on the already available data $\mathbf{W}_{n,\ell}$. Formula (3) comes from a Newton step with a finite-difference approximation of the Jacobian of the fine propagator (see, e.g., [2]).

It is worth noting that applying the Parareal algorithm directly to the gradient-flow system (2) would require decomposing the interval $[0, +\infty)$ into an infinite number of nonoverlapping subintervals, which is clearly impractical for any implementation. A straightforward strategy is to first partition $[0, +\infty)$ into time windows of fixed length $T > 0$, namely $[0, +\infty) := \bigcup_{k=0}^{+\infty} \mathcal{T}_k$ with $\mathcal{T}_k = [kT, (k+1)T]$, and to apply Parareal sequentially on them. In particular, each time window \mathcal{T}_k is split into N_G subintervals: $\mathcal{T}_k = \bigcup_{i=0}^{N_G-1} [t_i^k, t_{i+1}^k]$, with $kT = t_0^k < t_1^k < \dots < t_{N_G}^k = (k+1)T$, and Parareal is applied on it. One moves to \mathcal{T}_{k+1} as soon Parareal converges on \mathcal{T}_k . Thus, starting from $\mathbf{W}(0) = \mathbf{W}_0$, Parareal is first executed on \mathcal{T}_0 to obtain $\mathbf{W}(T)$. Using this value as the new initial condition, Parareal is then applied to \mathcal{T}_1 to compute $\mathbf{W}(2T)$. This process is repeated iteratively to solve (2) on each \mathcal{T}_i , for $i = 0, 1, 2, \dots$.

This sequential application enables parallel computation within each time window, thereby parallelizing the overall integration of the gradient-flow trajectory. However, in the context of minimization problems such as (2), the primary interest lies not in the entire trajectory $\mathbf{W}(t)$ but rather in its limit point—that is, the minimum of the objective function corresponding to the steady state of the gradient flow.

This simple observation leads to the ParaFlow framework: instead of running Parareal until convergence on each time window \mathcal{T}_k , one performs only one single iteration on \mathcal{T}_k before moving to \mathcal{T}_{k+1} , with the hope of gaining from the Newton-type convergence of Parareal. Thus, let \mathbf{W}_k be an initialization vector for the time window \mathcal{T}_k , one first builds the initialization $(\mathbf{W}_{0,0}^k, \dots, \mathbf{W}_{N_G,0}^k)$, and then computes $(\mathbf{W}_{0,1}^k, \dots, \mathbf{W}_{N_G,1}^k) \approx (\mathbf{W}(t_0^k), \dots, \mathbf{W}(t_{N_G}^k))$ with one Parareal iteration. Notice that, it is not guaranteed that $\arg \min_{\mathbf{W} \in (\mathbf{W}_{1,1}^k, \dots, \mathbf{W}_{N_G,1}^k)} L(\mathbf{W}) = \mathbf{W}_{N_G,1}^k$, namely the minimum objective is at the right extremum of \mathcal{T}_k as would be at convergence. Thus, with the goal of minimizing L , we choose as initialization point for the next time window \mathcal{T}_k , the one that minimize L in \mathcal{T}_k , that is $\mathbf{W}_{k+1} \in \arg \min_{\mathbf{W} \in (\mathbf{W}_{0,1}^k, \dots, \mathbf{W}_{N_G,1}^k)} L(\mathbf{W})$.

The ParaFlow framework is detailed in Alg. 1, where the single Parareal iteration is obtained from line 4 to line 11, corresponding to (3). Notice that the ParaFlow framework offers freedom in line 3 in the definition of the initialization set $(\mathbf{W}_{0,0}^k, \dots, \mathbf{W}_{N_G,0}^k)$. To compute it, a typical Parareal strategy is

¹ Notice that the propagators F and G can be chosen, e.g., equal to a certain number of GD steps, corresponding to the Explicit-Euler discretization of (2) and different discrete time steps.

Algorithm 1 ParaFlow

Require: Decomposition of $[0, +\infty)$, N_G parallel sources, maximum number of iterations k_{max} , initial vector \mathbf{W}_0 , propagators F and G .

- 1: Set $k = 0$.
- 2: **while** $k \leq k_{max}$ and an optimal condition is not satisfied **do**
- 3: From \mathbf{W}_k build a set $(\mathbf{W}_{0,0}^k, \dots, \mathbf{W}_{N_G,0}^k)$.
- 4: **for** $n = 0, \dots, N_G - 1$ (Parallel loop) **do**
- 5: Compute $g_n^k = G(t_n^k, t_{n+1}^k, \mathbf{W}_{n,0}^k)$ and $f_n^k = F(t_n^k, t_{n+1}^k, \mathbf{W}_{n,0}^k)$.
- 6: **end for**
- 7: Set $\mathbf{W}_{0,1}^k = \mathbf{W}_{0,0}^k$.
- 8: **for** $n = 0, \dots, N_G - 1$ (Sequential loop) **do**
- 9: Compute $g_n^{k+1} = G(t_n^k, t_{n+1}^k, \mathbf{W}_{n,1}^k)$.
- 10: Compute $\mathbf{W}_{n+1,1}^k = f_n^k + g_n^{k+1} - g_n^k$.
- 11: **end for**
- 12: Compute $\mathbf{W}_{k+1} \in \arg \min_{\mathbf{W} \in (\mathbf{W}_{0,1}^k, \dots, \mathbf{W}_{N_G,1}^k)} L(\mathbf{W})$ and update $k = k + 1$.
- 13: **end while**
- 14: **Output:** Last \mathbf{W}_k computed.

Algorithm 2 ParaFlowS

Require: Maximum number of iterations k_{max} , initial vector \mathbf{W}_0 and propagators F and G .

- 1: Set $k = 0$.
- 2: **while** $k \leq k_{max}$ and an optimal condition is not satisfied **do**
- 3: Set $\mathbf{W}_0^k = \mathbf{W}_k$.
- 4: Compute $g_0 = G(0, \Delta T, \mathbf{W}_0^k)$ and $f_0 = F(0, \Delta T, \mathbf{W}_0^k)$.
- 5: **for** $n = 0, \dots, N_G - 1$ **do**
- 6: Compute $g_n = G(0, \Delta T, \mathbf{W}_n^k)$.
- 7: Compute $\mathbf{W}_{n+1}^k = f_0 + g_n - g_0$.
- 8: **if** $L(\mathbf{W}_{n+1}^k) \geq L(\mathbf{W}_n^k)$ **break**.
- 9: **end for**
- 10: Set $\mathbf{W}_{k+1} = \mathbf{W}_n^k$ and update $k = k + 1$.
- 11: **end while**
- 12: **Output:** Last \mathbf{W}_0 computed

$\mathbf{W}_{n+1,0}^k = G(t_n^k, t_{n+1}^k, \mathbf{W}_{n,0}^k)$, which one could use starting from $\mathbf{W}_{0,0}^k = \mathbf{W}_k$. This choice clearly requires parallel computations at each ParaFlow iteration. Another possibility is to choose the simpler update formula $\mathbf{W}_{n,0}^k = \mathbf{W}_k$ together with $t_{n+1}^k - t_n^k = \Delta T$ for all n and k . This simple choice has a direct interesting consequence. Since the considered gradient-flow system (2) is autonomous, one gets $F(0, \Delta T, \mathbf{W}_{n,0}^k) = F(t_n^k, t_{n+1}^k, \mathbf{W}_{n,0}^k) = F(t_i^k, t_{i+1}^k, \mathbf{W}_{i,0}^k)$ for all i and n . Hence, all trajectories computed in parallel in lines 4-6 are actually equal. This means that one can actually compute only one of them and just reuse it. The same happens for the term $G(t_n^k, t_{n+1}^k, \mathbf{W}_{n,0}^k)$. With this observation, one can reformulate ParaFlow (which is a parallel algorithm) in a purely sequential (though equivalent) form. This leads to ParaFlowS, where "S" stays for "Sequential", which is detailed in Alg. 2. Let us now comment on the instruction in line 8 of Alg. 2. The sequential nature of ParaFlowS, allows us to monitor the value $L(\mathbf{W}_n^k)$. Since it is not guaranteed the monotonicity $L(\mathbf{W}_{n+1}^k) \leq L(\mathbf{W}_n^k)$, ParaFlowS breaks the loop over n as soon a non-decrease in terms of L happens. This saves computations by avoiding the use of G for large n (if not needed). This would be also possible in ParaFlow. However, notice that one sin-

gle ParaFlowS iteration requires only one action of F and at most $N_G + 1$ actions of G . In contrast, ParaFlow requires N_G (parallel) actions of F and at most N_G parallel and N_G sequential actions of G . Hence, neglecting communication issues, ParaFlow and ParaFlowS would require the same sequential cost per iteration. However, this is only possible if N_G parallel resources are available for ParaFlow.

It is interesting to notice the following properties of ParaFlowS, which states that if the computational cost of F is much higher than that of the N_G actions of G , then from a given point \mathbf{W} a single ParaFlowS iteration decreases the loss function more than one gradient-flow step of F , while keeping nearly the same computational cost.

Theorem 1 (On the ParaFlowS convergence) *The ParaFlowS sequence $\{\mathbf{W}_k\}_{k \geq 0}$ generates a monotonically decreasing sequence of the objective values, that is $L(\mathbf{W}_{k+1}) \leq L(\mathbf{W}_k)$. Moreover, assuming that $\mathbf{W}(\Delta T; \mathbf{W}_{k-1}) := F(0, \Delta T, \mathbf{W}_{k-1})$ corresponds to the exact gradient-flow trajectory, this trajectory is lower-bounded by the ParaFlow sequence in the sense that $L(\mathbf{W}_k) \leq L(\mathbf{W}(\Delta T; \mathbf{W}_{k-1}))$.*

Proof. The first statement is straightforward to verify in Alg. 2. For the second statement, using (3) with $\ell = 0$, one gets for $n = 0$ (on the first subinterval) that $\mathbf{W}_1^k = G(0, \Delta T, \mathbf{W}_0^k) + F(0, \Delta T, \mathbf{W}_0^k) - G(0, \Delta T, \mathbf{W}_0^k) = F(0, \Delta T, \mathbf{W}_0^k)$. Thus, $L(\mathbf{W}_1^k) \leq L(\mathbf{W}(\Delta T; \mathbf{W}_{k-1}))$. The result follows from line 8 in Alg. 2. \square

4 Numerical experiments

To assess the efficiency of ParaFlowS, we consider three problems of the form (1):

1. Classification problem (taken from [3]). Here, a classification of two areas in a domain needs to be obtained by a 10 points data set. This is done by training a neural network with four layers, denoted [2,3,3,2]: input and output layers of two neurons and two hidden layers of three neurons. The sigmoid is used as activation function.

2. Approximation problem. We consider a neural network of four layers, [1,16,16,1]: input and output layers of one neuron and two hidden layers of 16 neurons. The activation function is the standard sigmoid. The goal is to approximate the function $f(x) = e^{-x} \sin(x)$ for $x \in [0, 10]$ using a training set $\{x_i\}_{i=1}^{30}$ obtained by 30 equispaced points in $[0, 10]$ and the corresponding values $f(x_i)$, $i = 1, \dots, 30$.

3. PINN Poisson problem (taken from [4]). The goal is to train a PINN with the structure [2,50,50,50,50,1] (an input layer of two neurons, four hidden layers of 50 neurons, and one output layer of one neuron) for solving the Poisson problem $-\Delta u = 1$ in an L-shaped domain Ω , with homogeneous Dirichlet boundary conditions. The training set is formed by 120 and 1320 points on $\partial\Omega$ and in Ω , respectively.

The three problems are solved by a stochastic GD method (with different batch sizes and constant η ; see Section 2) and by ParaFlowS using the same stochastic GD (same batch sizes) for the propagator F , and GD with full batch for the propagator G . Moreover, F performs N_F stochastic gradient iterations with step length $\eta_F = \eta$, while G a single GD (full batch) iteration with step length $\eta_G = N_F \eta_F$.

Batch size	η_F	Budget	GD	$N_F = 10$	$N_F = 50$	$N_F = 100$	$N_F = 500$	$N_F = 1000$	$N_F = 2000$
10	0.1	1e4	2.5e + 00	0.0884	0.1611	0.2708	1.0007	1.0000	–
		1e5	2.4e – 03	0.0746	0.0235	0.0273	0.4370	0.5877	0.8950
		1e6	1.4e – 04	0.1016	0.1002	0.0988	0.0919	0.0972	0.3628
		1e7	1.4e – 05	0.9736	0.9079	0.8422	0.7616	0.7771	0.5922
		1e4	5.0e + 00	0.5066	0.5138	0.9508	1.0000	1.0000	–
		1e5	2.5e + 00	0.0011	0.0008	0.0088	0.0552	0.1537	0.9703
		1e6	2.3e – 03	0.0668	0.0577	0.0474	0.0457	0.0443	0.3519
	0.01	1e7	1.5e – 04	0.7623	0.7339	0.7126	0.6927	0.6704	0.6027
		1e4	5.0e + 00	0.9994	0.9864	0.7013	1.0000	1.0000	–
		1e5	5.0e + 00	0.5062	0.0023	0.0019	0.5108	0.9231	0.9994
		1e6	2.5e + 00	0.0010	0.0004	0.0004	0.0004	0.0102	0.0224
		1e7	2.3e – 03	0.4346	0.4211	0.4118	0.3939	0.3919	0.3745
		1e4	5.0e + 00	0.9990	0.9988	0.9987	0.9983	1.0000	–
		1e5	5.0e + 00	0.9994	0.9863	0.5804	0.3118	0.2437	0.4889
	0.0001	1e6	5.0e + 00	0.5071	0.0026	0.0022	0.0021	0.0019	0.2383
		1e7	2.5e + 00	0.0045	0.0044	0.0043	0.0041	0.0038	0.0040
		1e4	2.6e + 00	1.2148	1.4462	1.1058	1.0202	1.1947	1.0317
		1e5	2.2e – 03	11.3770	1.8010	1.3648	1.0714	1.0433	1.0195
		1e6	1.3e – 04	1.0523	0.2483	0.1357	0.1013	0.1016	0.8105
		1e7	1.3e – 05	1.5669	0.9122	0.8805	0.6140	0.6067	0.6039
1e4		5.0e + 00	0.9947	0.5664	0.9943	0.9948	1.0044	1.0003	
0.01	1e5	2.6e + 00	0.9464	1.0019	0.9791	0.9822	0.9908	1.0409	
	1e6	2.1e – 03	2.0481	0.2395	0.1063	0.0797	0.8136	0.9025	
	1e7	1.3e – 04	3.0968	0.7023	0.5996	0.5490	0.5618	0.5476	
	1e4	5.0e + 00	0.9904	0.9903	0.9902	1.0003	0.9915	0.9968	
	1e5	5.1e + 00	0.9848	0.7264	0.4678	0.5242	0.9853	0.9891	
	1e6	2.5e + 00	0.9991	0.0028	0.0010	0.9796	0.8024	0.9441	
	1e7	2.1e – 03	1.9928	0.3540	0.3278	0.2531	0.3028	0.3044	
0.0001	1e4	5.0e + 00	0.9996	0.9991	0.9990	1.0000	0.9999	0.9996	
	1e5	5.1e + 00	0.9888	0.9886	0.9885	0.9911	0.9292	0.5554	
	1e6	5.1e + 00	0.9891	0.9862	0.5292	0.0029	0.0016	0.2434	
	1e7	2.5e + 00	1.0135	0.0056	0.0028	0.0028	0.0028	0.0027	

Table 1 Final loss for different batch sizes, learning rates, and budgets (classification problem).

All methods run with η_F constant along the iterations and until a *budget* is completely consumed. The *budget* is defined as the maximum number of calls to the function computing the (partial) gradient. For example, if one considers a budget of 10^4 , GD with full batch is stopped after $10^4/N$ iterations, where N is the size of the training data; see Section 1. For the same budget, a stochastic gradient with batch size equal to one is stopped after 10^4 iterations, while a stochastic gradient with batch size equal 10 is stopped after $10^4/10$ iterations. Moreover, if we denote by $d_B \in \{1, \dots, N\}$ the batch size, the cost of a single ParaFlowS iteration is equal to $d_B N_F + N N_G$ (assuming that the condition in line 8 of Alg. 2 is never fulfilled). Thus, ParaFlowS is stopped after $10^4/(d_B N_F + N N_G)$ iterations. Notice that, the computational time of each run is proportional to the budget and obtained by multiplying the budget by the time needed for a single gradient computation.

The results of our experiments are shown in Tables 1, 2, and 3. Here, we report for the three test problems the loss function values obtained for different batch sizes, different values of η and N_F , and different budgets. In particular, the column GD shows the loss function values obtained by the GD (or stochastic gradient). If we denote this value by L_{GD} and by L_{PF} the loss function value obtained by ParaFlowS, then the columns for the different N_F report the ratio $R = L_{PF}/L_{GD}$. Thus, a ratio R smaller than one means that ParaFlowS reaches a loss function value smaller than that of the GD by spending the same budget. To help the reader, we highlight in dark gray the tables' cells for $R < 0.9$ (ParaFlowS outperforms GD) and in light gray the cells for $R \in [0.9, 1.1]$ (ParaFlowS performs like GD). In bold, the reader can find the best performance for each case (each row), while “–” means that the budget

Batch size	η_F	Budget	GD	$N_F = 10$	$N_F = 50$	$N_F = 100$	$N_F = 500$	$N_F = 1000$	$N_F = 2000$		
30	0.1	1e4	2.3e-01	0.422	0.598	0.710	–	–	–		
		1e5	9.1e-02	0.059	0.137	0.281	0.804	0.873	1.001		
		1e6	1.5e-03	0.630	0.676	0.693	0.737	0.787	0.810		
		1e7	8.1e-04	0.873	0.888	0.900	0.922	0.936	0.938		
		1e4	2.9e-01	0.654	0.509	0.482	–	–	–		
		1e5	2.3e-01	0.467	0.034	0.113	0.622	0.699	1.001		
		1e6	9.2e-02	0.020	0.026	0.055	0.125	0.249	0.443		
	0.01	1e7	1.5e-03	0.598	0.594	0.662	0.669	0.696	0.719		
		1e4	3.6e-01	0.805	0.689	0.867	–	–	–		
		1e5	2.9e-01	0.804	0.419	0.416	0.357	0.485	1.000		
		1e6	2.3e-01	0.428	0.014	0.012	0.020	0.107	0.549		
		1e7	9.2e-02	0.018	0.014	0.014	0.014	0.049	0.115		
		1e4	1.3e+00	0.273	0.235	0.293	–	–	–		
		1e5	3.6e-01	0.810	0.768	0.718	0.462	0.498	1.000		
	0.0001	1e6	2.9e-01	0.814	0.423	0.386	0.024	0.200	0.086		
		1e7	2.3e-01	0.414	0.029	0.023	0.023	0.024	0.024		
		1e4	2.4e-01	0.673	1.138	1.107	1.010	0.966	0.606		
		1	0.1	1e5	9.2e-02	1.427	1.309	1.243	1.071	1.037	0.906
				1e6	1.4e-03	15.353	4.194	2.141	1.169	1.094	1.130
				1e7	8.1e-04	2.160	1.138	1.084	1.018	1.011	1.009
				1e4	2.9e-01	1.181	0.666	0.560	1.002	1.001	1.001
	0.01		1e5	2.2e-01	1.059	0.700	0.594	0.723	0.816	0.910	
			1e6	9.0e-02	1.557	0.111	0.577	0.988	0.999	0.997	
			1e7	1.4e-03	2.533	1.403	1.577	1.056	1.044	1.031	
0.001	1e4	3.5e-01	7.072	2.425	1.131	1.056	1.029	1.014			
	1e5	2.9e-01	4.535	1.712	0.889	0.556	0.506	0.850			
	1e6	2.3e-01	1.148	0.700	0.612	0.080	0.519	0.566			
0.0001	1e7	9.1e-02	1.422	0.070	0.048	0.047	0.126	0.236			
	1e4	1.3e+00	1.944	1.929	1.876	0.683	1.050	1.023			
	1e5	3.6e-01	4.260	3.859	3.458	1.486	0.814	0.630			
	1e6	2.9e-01	1.049	0.971	0.874	0.555	0.491	0.181			
		1e7	2.2e-01	1.238	0.671	0.567	0.031	0.026	0.040		

Table 2 Final loss for different batch sizes, learning rates, and budgets (approximation problem).

is not enough to perform a single full ParaFlowS iteration. For all three problems, one can see that when the full batch is used, ParaFlowS essentially outperforms GD (and in many cases by several orders of magnitude), almost all cells are dark gray or light gray, only in a few cases GD performs better, up to one single case the best performance is achieved by ParaFlowS. Notice also that the ParaFlowS performance is generally better for larger budgets but milder when very small batch sizes are used. When the batch size is half of the full size, the results (not reported for the sake of brevity) are similar to the full batch case. This indicates that the stochasticity in the propagator F affects the performance of ParaFlowS, and a further and deeper study is needed to better explore this case and resolve this issue. Notice also that, ParaFlowS performs significantly better than GD when η_F is small, also for small N_F . This is a good property since the choice of the step length is generally an issue. Thus, while a large η_F can guarantee a faster decay of the loss function, it can easily lead to divergence of the method (instability of the Explicit Euler method discretizing (2)). In contrast, a small η guarantees a more robust behavior, while the loss decays slower. In this case, the ParaFlowS acceleration is very beneficial since it improves the speed of decay, while maintaining the robustness of the method. Finally, we notice that (up to Table 3) for smaller η_F the best result (for a given budget) is obtained for increasing values of N_F . A possible explanation is that ParaFlowS works best when η_G is near the maximum stable step size for the Explicit Euler method. While the method remains stable, the G -correction (Alg. 2, line 7) continues and rapidly decreases the loss; once instability appears, it stops due to the criterion in line 8. Thus, for large N_F and η_F , Explicit Euler with step η_G becomes highly unstable, and

Batch size	η_F	Budget	GD	$N_F = 10$	$N_F = 50$	$N_F = 100$	$N_F = 500$	$N_F = 1000$	$N_F = 2000$
1448(full batch)	0.1	1e5	7.6e-02	1.382	1.339	-	-	-	-
		1e6	8.9e-03	0.652	1.030	0.992	1.002	-	-
		1e7	2.6e-03	1.040	0.967	0.964	1.065	1.030	1.016
		1e8	1.3e-03	1.059	1.453	0.931	1.390	1.543	0.917
	0.01	1e5	1.9e-01	0.635	1.076	-	-	-	-
		1e6	1.4e-02	0.490	0.587	0.702	1.005	-	-
		1e7	3.4e-03	0.832	0.839	0.853	0.896	0.934	0.933
		1e8	2.3e-03	0.857	0.858	0.862	0.923	0.925	0.927
	0.001	1e5	9.1e-01	0.241	1.008	-	-	-	-
		1e6	1.9e-01	0.075	0.204	0.449	1.007	-	-
		1e7	1.4e-02	0.259	0.422	0.421	0.531	0.689	0.820
		1e8	3.4e-03	0.673	0.810	0.796	0.840	0.855	0.887
0.0001	1e5	1.2e+00	0.761	0.714	-	-	-	-	
	1e6	9.1e-01	0.230	0.043	0.025	1.001	-	-	
	1e7	1.9e-01	0.077	0.026	0.023	0.187	0.442	0.597	
	1e8	1.4e-02	0.245	0.188	0.187	0.412	0.414	0.459	
724	0.1	1e5	2.4e-02	0.701	0.475	0.583	-	-	-
		1e6	8.7e-03	0.836	0.762	3.607	1.053	1.379	-
		1e7	3.2e-03	1.223	1.194	1.168	1.693	1.008	0.960
		1e8	1.4e-03	0.996	4.221	0.962	1.431	0.964	1.261
	0.01	1e5	1.0e-01	0.734	1.053	1.029	-	-	-
		1e6	9.0e-03	1.189	1.042	1.104	1.076	0.983	-
		1e7	4.8e-03	1.075	0.939	0.890	0.845	0.971	0.898
		1e8	3.4e-03	1.027	0.977	0.965	0.942	0.937	1.065
	0.001	1e5	8.2e-01	0.133	0.231	1.008	-	-	-
		1e6	9.9e-02	0.111	0.385	0.509	1.018	0.973	-
		1e7	9.1e-03	0.584	0.985	0.950	0.960	1.036	1.052
		1e8	4.2e-03	0.850	1.016	1.096	1.036	1.229	1.156
0.0001	1e5	1.2e+00	0.722	0.325	0.330	-	-	-	
	1e6	8.3e-01	0.125	0.025	0.015	0.156	0.998	-	
	1e7	9.9e-02	0.097	0.061	0.057	0.307	0.549	0.774	
	1e8	9.2e-03	0.513	0.452	0.505	0.928	0.963	0.959	

Table 3 Final loss for different batch sizes, learning rates, and budgets (PINN Poisson problem).

the G-correction is applied only once. This behavior and the general convergence of ParaFlow(S) (deterministic and stochastic) will be explored in future work.

5 Conclusions

This work presented the ParaFlow framework and its sequential variant, ParaFlowS, which combines Parareal with gradient-based optimization to accelerate convergence towards a minimum. The presented numerical results suggest that ParaFlowS is a promising acceleration strategy. Future work will focus on, e.g., a deeper theoretical analysis and on comparisons with existing optimization methods, more extensive statistical numerical studies on larger-scale problems, the use of mini-batch also in the coarse propagator, and the exploration of alternative optimization algorithms for both the fine and coarse propagators.

Acknowledgements GC, IM, and MV are members of the Indam GNCS group.

References

1. Bottou, L., Curtis, F.E., Nocedal, J.: Optimization methods for large-scale machine learning. *SIAM Rev.* **60**(2), 223–311 (2018)
2. Gander, M.J., Lunet, T.: Time parallel time integration. *SIAM* (2024)
3. Higham, C.F., Higham, D.J.: Deep learning: An introduction for applied mathematicians. *SIAM Rev.* **61**(4), 860–891 (2019)
4. Lu, L., Meng, X., Mao, Z., Karniadakis, G.E.: Deepxde: A deep learning library for solving differential equations. *SIAM Rev.* **63**(1), 208–228 (2021)
5. Tanabe, K.: A geometric method in nonlinear programming. *J. Optim. Theory Appl.* **30**, 181–210 (1980)