GParareal: (Towards) A Probabilistic Time-Parallel ODE Solver

K. Pentland[1] M. Tamborrino[1] **T. J. Sullivan**[1,2] J. Buchanan[3] L. C. Appel[3]

[1]**University of Warwick, UK**
[2]**Alan Turing Institute, UK**
[3]Culham Centre for Fusion Energy, UK

Gaussian Process Summer School
Sheffield, UK, 15 September 2022

Overview

- We are interested in solving ordinary differential equation (ODE) initial value problems (IVPs) of the form

$$\frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} = \boldsymbol{f}\big(t, \boldsymbol{u}(t)\big) \quad \text{over} \quad t \in [t_0, T], \quad \text{with} \quad \boldsymbol{u}(t_0) = \boldsymbol{u}_0 \in \mathcal{U} \subseteq \mathbb{R}^d, \tag{1}$$

  with the aid of two time-stepping schemes, an expensive and accurate fine solver $\mathcal{F}$ and a cheap and inaccurate coarse solver $\mathcal{G}$.
- We seek numerical solutions $\boldsymbol{U}_j \approx \boldsymbol{u}(t_j)$ to the IVP in (1) on a pre-defined mesh $\boldsymbol{t} = (t_0, \dots, t_J)$, where $t_{j+1} = t_j + \Delta T$ for fixed $\Delta T = (T - t_0)/J$.
- Our computational budget does not allow $\mathcal{F}$ to be run over the whole interval $[t_0, T]$, but does allow it to be run in parallel over the many subintervals $[t_j, t_{j+1}]$.
- **Key takeaway message:** Training a Gaussian process (GP) for $\mathcal{F} - \mathcal{G}$ enables a time-parallel numerical solution of (1) that is competitive with the well-established "parareal" method introduced by Lions et al. (2001), and has significant advantages in robustness and flexibility.

Motivation for Time Parallelism

- Expensive vector fields $\boldsymbol{f}$ and time-steppers $\mathcal{F}$ can arise when (spatially) discretising partial differential equations (PDEs) into a system of ODEs.
- These issues also arise when solving IVPs with spatial or other non-temporal dependencies in the sense that, even though highly efficient domain decomposition methods exist (Dolean et al., 2015), the parallel speed-up of such methods on high performance computers (HPCs) is still constrained by the serial nature of the time-stepping scheme.
- With exascale HPCs on the horizon (Mann, 2020), there has been renewed interest in developing more efficient and robust *time-parallel* algorithms to reduce wallclock runtimes for IVP simulations in applications such as numerical weather prediction (Hamon et al., 2020), kinematic dynamo modelling (Clarke et al., 2020), and plasma physics (Samaddar et al., 2010, 2019).

Motivation for GPs and Probabilistic Approaches

- The last decade of has seen a renewed interest in the interplay of probability/statistics and the design and analysis of numerical algorithms — under the name probabilistic numerics (Hennig et al., 2015; Cockayne et al., 2019, e.g).
- This general research direction builds on over a century of previous ideas, arguably ones that could not be computationally realised at the time (Oates and Sullivan, 2019).
- The general idea is that many numerical tasks can also be seen as statistical inference tasks — although in practice one often sacrifices statistical exactness for computational cost.
- Example: the trapezoidal rule is the statistically optimal (minimum posterior variance) univariate quadrature scheme under a Brownian motion prior on the integrand (Sul′din, 1959, 1960).
- Example: the RK4 time-stepping ODE solver results from the statistically optimal (minimum posterior variance) placement of observations of the vector field $\boldsymbol{f}$ over each integration interval (Schober et al., 2014).

Parareal

Parareal

- Let's recall the parareal algorithm (Lions et al., 2001), first describing the fine- and coarse-grained numerical solvers it uses, then the algorithm itself, and finally some remarks on complexity, numerical speed-up, and choice of solvers.
- For a full mathematical derivation and exposition of parareal, see Gander and Vandewalle (2007). To simplify notation, we describe parareal for solving a scalar-valued ODE, i.e. $\boldsymbol{f}(t, \boldsymbol{u}(t)) := f(t, u(t))$ in (1), without loss of generality.
- Extending parareal to the multivariate case in (1) is straightforward: see (Gander and Vandewalle, 2007) for notation and (Pentland et al., 2022) for pseudocode.

Parareal: Parallel in Time???

- What does "time parallelism" even mean?
- We start with a trivial observation, that — assuming existence and uniqueness of integral curves for the vector field f — the solutions at time $t > t_i$ of the IVPs

$$\frac{\mathrm{d}u}{\mathrm{d}t} = f\big(t, u(t)\big) \qquad\qquad u(t_0) = u_0$$

  and

$$\frac{\mathrm{d}v}{\mathrm{d}t} = f\big(t, v(t)\big) \qquad\qquad v(t_j) = v_j$$

  are the same *if* we are lucky enough to start the second system off at $v_j = u(t_j)$.
- In other words, if we *could* (simultaneously!) know the terminal state of the ODE on each subinterval $[t_j, t_{j+1}]$ and use it as an initial condition for a solve on $[t_{j+1}, t_{j+2}]$, then we would be in a position to solve the ODE on all J subintervals independently and in parallel.
- The issue, of course, is obtaining these terminal/initial states!

Parareal: The Solvers

- The (expensive) fine solver $\mathcal{F}$ propagates an initial value at t_j, over an interval of length ΔT, and returns a solution with high numerical accuracy at t_{j+1}.
- We assume that $\mathcal{F}$ provides accuracy to solve (1) for the solution to be considered 'exact', i.e. $U_j \approx u(t_j)$. The objective is to calculate these exact solutions

$$U_{j+1} = \mathcal{F}(U_j) \quad \text{for} \quad j = 0, \dots, J-1, \quad \text{where} \quad U_0 = u_0, \tag{2}$$

  without running $\mathcal{F}$ J times sequentially, as this calculation is assumed to be computationally intractable. To avoid this, parareal locates iteratively improved approximations U_j^k, where $k = 0, 1, 2, \dots$ is the iteration number, that converge toward U_j; note that $U_0^k = U_0 = u_0$ for all k.
- To do this, parareal uses a second numerical integrator $\mathcal{G}$, referred to as the coarse solver.
- $\mathcal{G}$ propagates an initial value at t_j over an interval of length ΔT, however, it has lower numerical accuracy and is computationally inexpensive to run compared to $\mathcal{F}$.
- $\mathcal{G}$ can be run serially across a number of time slices to provide relatively cheap low accuracy solutions whilst $\mathcal{F}$ is permitted only to run in parallel over multiple slices.

Parareal: The Algorithm

- To begin (iteration $k = 0$), approximate solutions to (1) are calculated sequentially using $\mathcal{G}$, on a single processor, such that

$$U_{j+1}^{0} = \mathcal{G}(U_j^0) \quad j = 0, \dots, J-1. \tag{3}$$

- Following this (iterations $k \geqslant 1$), the fine solver propagates each approximation in (3) *in parallel*, on J processors, to obtain $\mathcal{F}(U_j^0)$ for $j = 0, \dots, J-1$. These values are then used in the predictor-corrector

$$U_{j+1}^{k} = \underbrace{\mathcal{G}(U_j^k)}_{\text{predict}} + \underbrace{\mathcal{F}(U_j^{k-1}) - \mathcal{G}(U_j^{k-1})}_{\text{correct}} \quad \text{for} \quad j = 0, \dots, J-1. \tag{4}$$

  Here, $\mathcal{G}$ is applied sequentially to predict the solution at the next time step, before being corrected by the residual between coarse and fine values found during the previous iteration (note that (4) cannot be run in parallel).
- The predictor-corrector discretised approximation of the Newton–Raphson method for locating the true roots U_j with initial guess (3) (Gander and Vandewalle, 2007).

Parareal: Convergence Criteria

- For a pre-defined tolerance $\varepsilon > 0$, the parareal solution U_j^k has converged up to time t_I if

$$|U_j^k - U_j^{k-1}| < \varepsilon \quad \forall j \leqslant I. \tag{5}$$

- This criterion is standard for parareal (Garrido et al., 2006; Gander and Hairer, 2008), however, other criteria such as taking the average relative error between fine solutions over a time slice (Samaddar et al., 2010, 2019) or measuring the total energy of the system could be used instead.
- Unconverged solution values, i.e. U_j^k for $j > I$, are improved in future iterations ($k > 1$) by initiating further parallel $\mathcal{F}$ runs on each U_j^k, followed by an update using (4).
- The algorithm stops once $I = J$, converging in k (out of J) iterations. The version of parareal we implemented does not iterate over solutions that have already converged, avoiding the waste of computational resources (Elwasif et al., 2011; Pentland et al., 2022; Garrido et al., 2006).

Parareal: Convergence and Complexity

- After k iterations, the first k time slices (at minimum) are converged, as the exact initial condition (u_0) has been propagated by $\mathcal{F}$ at least k times.
- If parareal converges in $k = J$ iterations, the solution will be equal to the one found by calculating (2) serially, at an even higher computational cost! Convergence in $k \ll J$ iterations is necessary if significant parallel speed-up is to be realised.
- Without loss of generality, assume running $\mathcal{F}$ over any $[t_j, t_{j+1}]$, $j \in \{0, \dots, J-1\}$, takes wallclock time $T_\mathcal{F}$ (denote time $T_\mathcal{G}$ similarly for $\mathcal{G}$). Therefore, calculating (2) using $\mathcal{F}$ serially, takes approximately $T_{\text{serial}} = JT_\mathcal{F}$ seconds. Using parareal, the total wallclock time (in the worst case, excluding any serial overheads) can be approximated by

$$T_{\text{para}} \approx \underbrace{JT_\mathcal{G}}_{\text{Iteration } 0} + \sum_{i=1}^{k} \underbrace{\left(T_\mathcal{F} + (J-i)T_\mathcal{G}\right)}_{\text{Iterations 1 to } k} = kT_\mathcal{F} + (k+1)\left(J - \frac{k}{2}\right)T_\mathcal{G}. \tag{6}$$

- The approximate parallel speed-up is therefore

$$S_{\text{para}} \approx \frac{T_{\text{serial}}}{T_{\text{para}}(k)} = \left[\frac{k}{J} + (k+1)\left(1 - \frac{k}{2J}\right)\frac{T_\mathcal{G}}{T_\mathcal{F}}\right]^{-1}. \tag{7}$$

Parareal: Task Scheduling

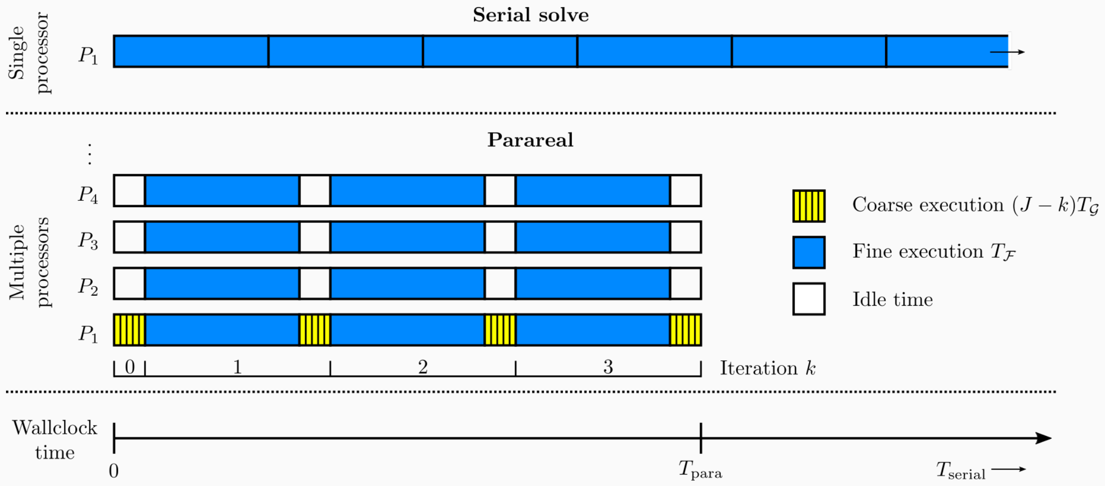


Figure 1: Computational task scheduling during three iterations of parareal as compared with a full serial integration. The coloured blocks represent the wallclock time any given processor spends on a task. Coarse runs are shown in yellow, fine runs in blue, and any idle time in white. The wallclock time is given on the axis at the bottom, indicating both T_{para} and T_{serial}.

GParareal

GParareal: Overview

- The GParareal algorithm uses a GP emulator in place of parareal's predictor-corrector.
- Suppose we seek the same high resolution numerical solutions to (1) as expressed in (2), denoted now as V_j instead of U_j. Furthermore, we denote the iteratively improved approximations in GParareal at iteration k as V_j^k (as before, $V_0^k = V_0 = u_0$).
- In parareal, the predictor-corrector (4) updates the numerical solutions at iteration k using a correction term based on information calculated during the *previous* iteration $k-1$. We propose the following refinement rule, which instead refines solutions using information from the *current* iteration k, rather than $k-1$:

$$V_{j+1}^k = \mathcal{F}(V_j^k) = \underbrace{(\mathcal{F} - \mathcal{G})(V_j^k)}_{\text{correction}} + \underbrace{\mathcal{G}(V_j^k)}_{\text{prediction}} \quad k \geqslant 1,\ j = 0, \dots, J-1. \tag{8}$$

- We propose using a GP emulator to model the correction term, trained on *all* previously-obtained evaluations of $\mathcal{F}$ and $\mathcal{G}$. The emulator returns a Gaussian distribution over $(\mathcal{F} - \mathcal{G})(V_j^k)$ from which we can extract an explicit value and carry out the refinement in (8).

GParareal: Setting Up the Emulator

- Before solving (1), we define a GP prior (Rasmussen and Williams, 2006) to emulate the unknown correction function $\mathcal{F} - \mathcal{G}$. More formally, we define the GP prior

$$\mathcal{F} - \mathcal{G} \sim \mathcal{GP}(m, \kappa), \tag{9}$$

  with mean function $m \colon \mathcal{U} \to \mathbb{R}$ and covariance kernel $\kappa \colon \mathcal{U} \times \mathcal{U} \to \mathbb{R}$.
- Given some vectors of initial values, $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{U}^J$, the corresponding mean vector is denoted $\mu(\boldsymbol{x}) = (m(x_j))_{j=0,\dots,J-1}$ and the covariance matrix $K(\boldsymbol{x}, \boldsymbol{x}') = (\kappa(x_i, x_j'))_{i,j=0,\dots,J-1}$. The correction term is expected to be small, depending on the accuracy of both $\mathcal{F}$ and $\mathcal{G}$, hence we define a zero-mean process, i.e. $m(x_j) = 0$.
- Ideally, κ will be chosen based on any prior knowledge of the solution to (1), e.g. regularity/smoothness. For simplicity, we use here the isotropic square exponential (SE) kernel with amplitude $\sigma > 0$ and length scale $\ell > 0$; these hyperparameters are referred to collectively in the vector $\boldsymbol{\theta}$ and need to be initialised prior to simulation.

GParareal: Iteration $k = 0$

Firstly, run $\mathcal{G}$ sequentially from the exact initial value, on a single processor, to locate the coarse solutions $\boldsymbol{x} = (V_0^0, \dots, V_{J-1}^0)^\intercal$ with

$$V_{j+1}^0 = \mathcal{G}(V_j^0) \quad j = 0, \dots, J-1. \tag{10}$$

GParareal: Iteration $k = 1$ — Propagate and Condition 1

- Use $\mathcal{F}$ to propagate these values (10) on each time slice in *parallel*, on J processors, to obtain the following values at t_{j+1}

$$\mathcal{F}(V_j^0) \quad j = 0, \ldots, J-1. \tag{11}$$

  At this stage, we diverge from the parareal method. Given the initially propagated values $\boldsymbol{x} = (V_0^0, \ldots, V_{J-1}^0)^\intercal$, store the values of $\mathcal{F} - \mathcal{G}$, using (10) and (11), in the vector

$$\boldsymbol{y} := \big((\mathcal{F} - \mathcal{G})(x_j)\big)_{j=0,\ldots,J-1}. \tag{12}$$

- At this point, the inputs $\boldsymbol{x}$ and evaluations $\boldsymbol{y}$ are used to optimise the kernel hyperparameters $\boldsymbol{\theta}$ via maximum likelihood estimation.

GParareal: Iteration $k = 1$ — Propagate and Condition 2

- Conditioning the prior (9) using the acquisition data $\boldsymbol{x}$ and $\boldsymbol{y}$, the GP posterior over $(\mathcal{F} - \mathcal{G})(x')$, where $x' \in \mathcal{U}$ is some initial value in the state space, is given by

$$(\mathcal{F} - \mathcal{G})(x') \mid \boldsymbol{x}, \boldsymbol{y} \sim \mathcal{N}\big(\hat{\mu}(x'), \hat{K}(x', x')\big), \tag{13}$$

with mean

$$\hat{\mu}(x') = \underbrace{\mu(x')}_{=\mathbf{0}} + K(x', \boldsymbol{x})[K(\boldsymbol{x}, \boldsymbol{x})]^{-1}\big(\boldsymbol{y} - \underbrace{\mu(\boldsymbol{x})}_{=\mathbf{0}}\big) \tag{14}$$

and variance

$$\hat{K}(x', x') = K(x', x') - K(x', \boldsymbol{x})[K(\boldsymbol{x}, \boldsymbol{x})]^{-1}K(\boldsymbol{x}, x'). \tag{15}$$

GParareal: Iteration $k = 1$ — Predict and Correct 1

- Now we wish to determine refined solutions V_j^1 at each mesh point.
- Given $\mathcal{F}$ has been run once, the exact solution is known up to time t_1. Specifically, at t_0 we know $V_0^1 = V_0$ and at t_1 we know $V_1^1 = V_1 = \mathcal{F}(V_0^1)$.
- At t_2, the exact solution $V_2 = \mathcal{F}(V_1^1)$ is unknown, hence we need to calculate its value without running $\mathcal{F}$ again. To do this, we re-write the exact solution using (8):

$$V_2^1 = \mathcal{F}(V_1^1) = (\mathcal{F} - \mathcal{G} + \mathcal{G})(V_1^1) = \underbrace{(\mathcal{F} - \mathcal{G})(V_1^1)}_{\text{correction}} + \underbrace{\mathcal{G}(V_1^1)}_{\text{prediction}} . \tag{16}$$

Both terms in (16) are initially unknown, but the prediction can be calculated rapidly at low computational cost while the correction can be inferred using the GP posterior (13) with $x' = V_1^1$.

GParareal: Iteration $k = 1$ — Predict and Correct 2

- Therefore, we obtain a Gaussian distribution over the solution

$$V_2^1 \sim \mathcal{N}\big(\hat{\mu}(V_1^1) + \mathcal{G}(V_1^1), \hat{K}(V_1^1, V_1^1)\big), \tag{17}$$

  with variance stemming from uncertainty in the GP emulator.

- Repeating this process to determine a distribution for the solution at t_3 by attempting to propagate the random variable V_2^1 using $\mathcal{G}$ is computationally infeasible for nonlinear IVPs. To tackle this and be able to propagate V_2^1, we do something rather crude: we approximate the distribution by taking its mean value,

$$V_2^1 = \hat{\mu}(V_1^1) + \mathcal{G}(V_1^1). \tag{18}$$

  This approximation is a convenient way of minimising computational cost, at the price of ignoring uncertainty in the GP emulator — we'll discuss possible alternatives later.

GParareal: Iteration $k = 1$ — Predict and Correct 3

- The refinement process, applying (8) and then approximating its Gaussian distribution by taking its expectation, is repeated sequentially for later t_j, yielding the approximate solutions

$$V_{j+1}^1 = \hat{\mu}(V_j^1) + \mathcal{G}(V_j^1) \quad \text{for} \quad j = 2, \ldots, J-1. \tag{19}$$

- Finally, we impose stopping criteria (5), identifying which V_j^1 for $j \leqslant I$ have converged. Using the same stopping criteria as parareal will allow us to compare the performance of both algorithms.

GParareal: Iteration $k = 1$

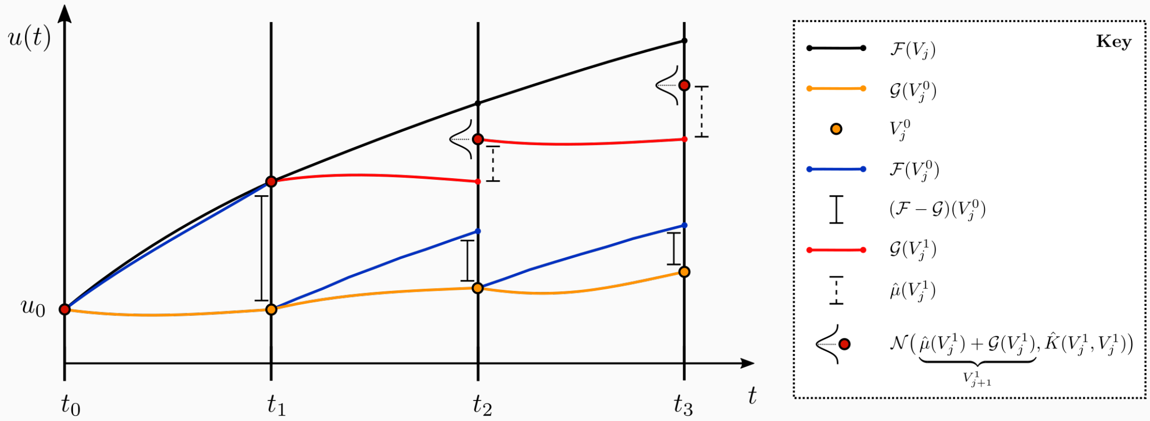


Figure 2: Schematic of the first iteration of GParareal. The 'exact' solution over $[t_0, t_3]$ is shown in black, with the first coarse and fine (parallel) runs given in yellow and blue respectively. Solid bars represent the residual between these solutions (12). The predictions, i.e. the second coarse runs, are shown in red and the corresponding corrections from the GP emulator are represented by the dashed bars. The refined solutions (20) at the end of the iteration are represented by the red dots.

GParareal: Iteration $k \geqslant 2$

- If the stopping criteria are not met, i.e. $I < J$, we can iteratively refine any unconverged solutions by re-running the previous steps.
- This means calculating $\mathcal{F}(V_j^{k-1})$, $j = I, \dots, J-1$, in parallel and generating new evaluations $\hat{\boldsymbol{y}} = \left((\mathcal{F} - \mathcal{G})(V_j^{k-1})\right)^\intercal_{j=I,\dots,J-1}$, with corresponding inputs $\hat{\boldsymbol{x}} = (V_I^{k-1}, \dots, V_{J-1}^{k-1})^\intercal$. Hyperparameters are then re-optimised and the GP is re-conditioned using *all* prior acquisition data, i.e. $\boldsymbol{x} = [\boldsymbol{x}; \hat{\boldsymbol{x}}]$ and $\boldsymbol{y} = [\boldsymbol{y}; \hat{\boldsymbol{y}}]$, generating an updated posterior. Here, $[\boldsymbol{a}; \boldsymbol{b}]$ denotes the vertical concatenation of column vectors $\boldsymbol{a}$ and $\boldsymbol{b}$.
- The refinement step is then applied such that we obtain

$$V_{j+1}^k = \hat{\mu}(V_j^k) + \mathcal{G}(V_j^k) \quad \text{for} \quad j = I+1, \dots, J-1. \tag{20}$$

- Once $I = J$, the solution, the number of iterations k taken to converge, and the acquisition data $\boldsymbol{x}$ and $\boldsymbol{y}$ are returned. A key benefit of GParareal is that this acquisition data can be used in future GParareal simulations as “legacy data” to pre-train the GP emulator and provide additional speedup — see the numerical experiments later.

Numerics

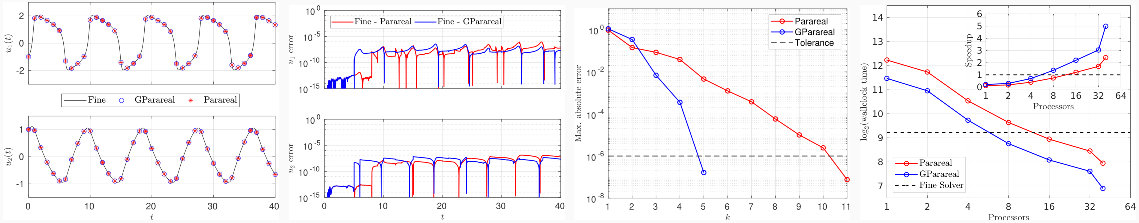
FitzHugh–Nagumo

In this experiment, we consider the FitzHugh–Nagumo (FHN) model (FitzHugh, 1961; Nagumo et al., 1962) given by

$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = c\left(u_1 - \frac{u_1^3}{3} + u_2\right), \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = -\frac{1}{c}(u_1 - a + bu_2), \tag{21}$$

where we fix parameters $(a, b, c) = (0.2, 0.2, 3)$. We integrate (21) over $t \in [0, 40]$, dividing the interval into $J = 40$ slices, and set the tolerance for both GParareal and parareal to $\varepsilon = 10^{-6}$. We use solvers $\mathcal{G} = \text{RK2}$ and $\mathcal{F} = \text{RK4}$ with $N_{\mathcal{G}} = 160$ and $N_{\mathcal{F}} = 1.6 \times 10^8$ steps respectively. Note that the large value of $N_{\mathcal{F}}$ is required to ensure that $\mathcal{F}$ is expensive to run and that parallel speedup can be realised (as both algorithms require $T_{\mathcal{G}}/T_{\mathcal{F}} \ll 1$).

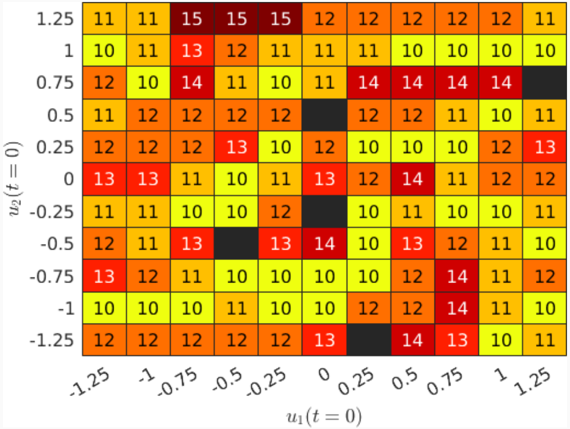
FitzHugh–Nagumo



(a) **(b)** **(c)** **(d)**

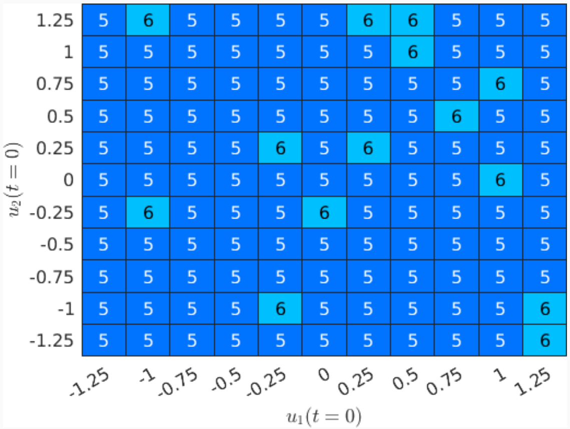
Figure 3: Numerical results obtained solving (21) for $\boldsymbol{u}_0 = (-1, 1)^\intercal$. (a) Time-dependent solutions using the fine solver, GParareal, and parareal — both GParareal and parareal plotted only at times $\boldsymbol{t}$ for clarity. (b) The corresponding absolute errors between solutions from GParareal and parareal vs. the fine solution. (c) Maximum absolute errors from (5) of each algorithm at successive iterations k until tolerance $\varepsilon = 10^{-6}$ is met. (d) Median wallclock times (taken over 5 runs) of both algorithms against the number of processors (up to 40). The inset plot displays the corresponding parallel speedup.

FitzHugh–Nagumo



$u_2(t=0)$ \ $u_1(t=0)$	-1.25	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1	1.25
1.25	11	11	15	15	15	12	12	12	12	12	11
1	10	11	13	12	11	11	11	10	10	10	10
0.75	12	10	14	11	10	11	14	14	14	14	■
0.5	11	12	12	12	12	■	12	12	11	10	11
0.25	12	12	12	13	10	12	10	10	10	12	13
0	13	13	11	10	11	13	12	14	11	12	12
-0.25	11	11	10	10	12	■	10	11	10	10	11
-0.5	12	11	13	■	13	14	10	13	12	11	10
-0.75	13	12	11	10	10	10	10	12	14	11	12
-1	10	10	10	11	10	10	12	12	14	11	10
-1.25	12	12	12	12	12	13	■	14	13	10	11

(a) Parareal



$u_2(t=0)$ \ $u_1(t=0)$	-1.25	-1	-0.75	-0.5	-0.25	0	0.25	0.5	0.75	1	1.25
1.25	5	6	5	5	5	5	6	6	5	5	5
1	5	5	5	5	5	5	5	6	5	5	5
0.75	5	5	5	5	5	5	5	5	5	6	5
0.5	5	5	5	5	5	5	5	5	6	5	5
0.25	5	5	5	5	6	5	6	5	5	5	5
0	5	5	5	5	5	5	5	5	5	6	5
-0.25	5	6	5	5	5	6	5	5	5	5	5
-0.5	5	5	5	5	5	5	5	5	5	5	5
-0.75	5	5	5	5	5	5	5	5	5	5	5
-1	5	5	5	5	6	5	5	5	5	5	6
-1.25	5	5	5	5	5	5	5	5	5	5	6

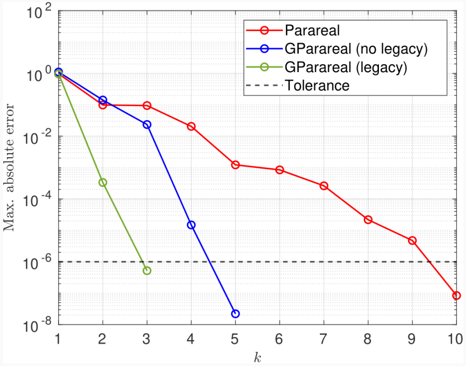
(b) GParareal

Figure 4: Heat maps displaying the number of iterations taken until convergence k of (a) parareal and (b) GParareal when solving the FitzHugh–Nagumo system (21) for different initial values $\boldsymbol{u}_0 \in [-1.25, 1.25]^2$. Black boxes indicate where parareal returned a `NaN` value during simulation.

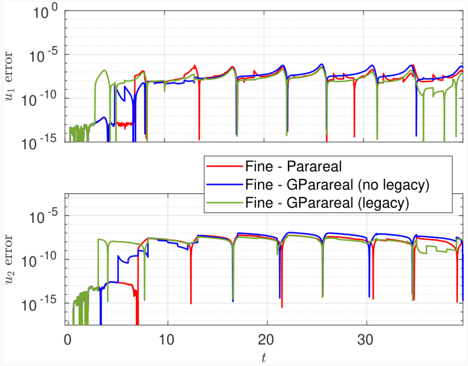
FitzHugh–Nagumo: Legacy Data

- GParareal can use both acquisition and legacy data to converge in fewer iterations than without the legacy data, and can evaluate the dynamics of the FHN model in significantly lower wallclock time than parareal.
- Approximately $kJ = 5 \times 40 = 200$ legacy data points, obtained solving (21) for $\boldsymbol{u}_0 = (-1, 1)^\intercal$, are stored and used to condition the GP emulator prior to solving (21) for alternate initial values $\boldsymbol{u}_0 = (0.75, 0.25)^\intercal$. In Figure 5(a), we can see that convergence takes two fewer iterations with the legacy data than without. Accuracy of the solutions obtained from these simulations is again shown to be of the order of the parareal solution in both cases — see Figure 5(b). Repeating the experiment from Figure 4(b) with the same legacy data for a range of initial values we see that k is either unchanged or improved in all cases, see Figure 6.
- Conditioning the GP and optimising hyperparameters using the legacy data comes at extra (serial) computational cost and checks should be made to ensure the parallel computations, i.e. $T_{\mathcal{F}}$, still dominate all serial parts of the simulation.

FitzHugh–Nagumo: Legacy Data



(a)



(b)

Figure 5: Numerical simulations solving (21) for $\boldsymbol{u}(0) = (0.75, 0.25)^\intercal$ using GParareal with and without access to legacy data, i.e. $\mathcal{F} - \mathcal{G}$ data obtained solving (21) for $\boldsymbol{u}(0) = (-1, 1)^\intercal$. The parareal simulation of the same problem is also shown for comparison. (a) Maximum absolute errors from (5) against iteration number k until tolerance $\varepsilon = 10^{-6}$ met. (b) Time-dependent errors of the corresponding numerical solutions from each simulation vs. the fine solution.

FitzHugh–Nagumo: Legacy Data

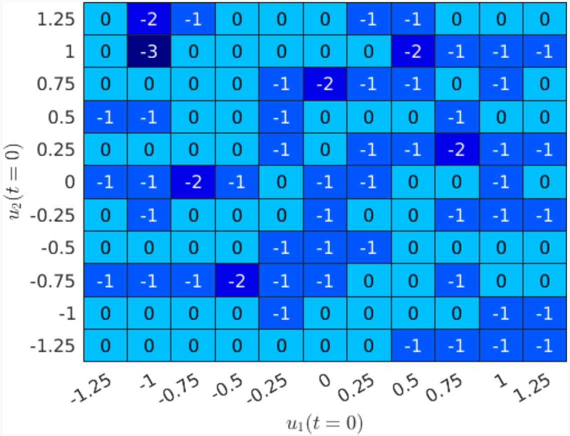


Figure 6: Heat map displaying the decrease in the number of iterations taken until convergence of GParareal when solving (21) for different initial values $\boldsymbol{u}_0 \in [-1.25, 1.25]^2$ *with* legacy data compared to without, i.e. compared to Figure 4(b). Legacy data was obtained by solving (21) for $\boldsymbol{u}_0 = (-1, 1)^\intercal$.

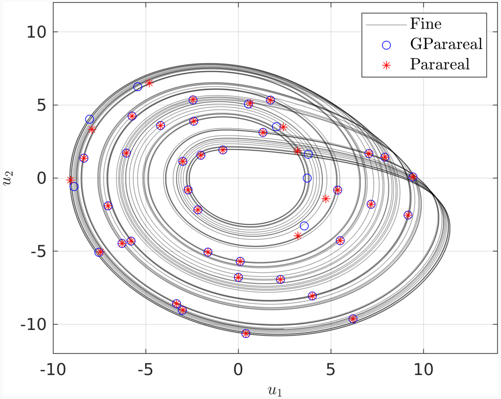
Rössler

- Next we solve the Rössler system,

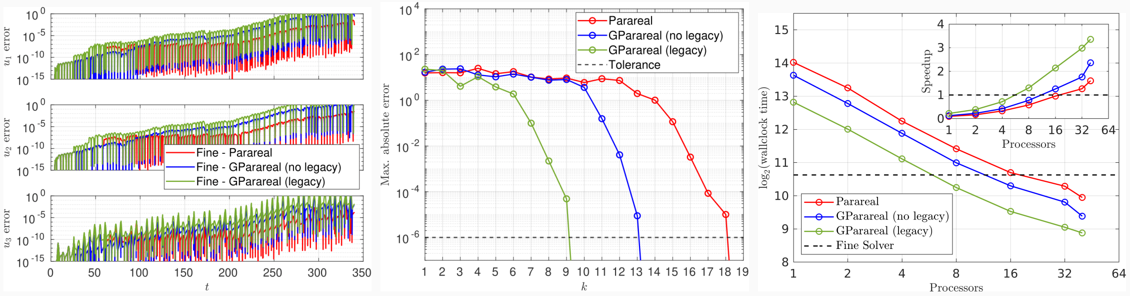
$$\frac{\mathrm{d}u_1}{\mathrm{d}t} = -u_2 - u_3, \quad \frac{\mathrm{d}u_2}{\mathrm{d}t} = u_1 + \hat{a}u_2, \quad \frac{\mathrm{d}u_3}{\mathrm{d}t} = \hat{b} + u_3(u_1 - \hat{c}), \tag{22}$$

with parameters $(\hat{a}, \hat{b}, \hat{c}) = (0.2, 0.2, 5.7)$ that cause the system to exhibit chaotic behaviour (Rössler, 1976).

- Suppose we wish to integrate (22) over $t \in [0, 340]$ with initial values $\boldsymbol{u}_0 = (0, -6.78, 0.02)^\intercal$ and solvers $\mathcal{G} = \text{RK1}$ and $\mathcal{F} = \text{RK4}$. The interval is divided into $J = 40$ time slices, $N_{\mathcal{G}} = 9 \times 10^4$ coarse steps, and $N_{\mathcal{F}} = 4.5 \times 10^8$ fine steps.
- The convergence tolerance is set to $\varepsilon = 10^{-6}$.



Rössler



(a) **(b)** **(c)**

Figure 7: Numerical results obtained solving the Rössler system (22) over $t \in [0, 340]$. (a) The corresponding absolute errors between solutions from GParareal and parareal vs. the fine solution. (b) Maximum absolute errors from (5) of each algorithm at successive iterations k until tolerance $\varepsilon = 10^{-6}$ is met. (c) Median wallclock times (taken over 5 runs) of each simulation against the number of processors (up to 40). Inset: The corresponding parallel speedup vs. the serial wallclock time.

Rössler

- In this experiment, rather than obtaining legacy data by solving (22) using alternative initial values, we instead generate such data by integrating over a shorter time interval. This is particularly useful if we are unsure how long to integrate our system for, i.e. to reach some long-time equilibrium state or reveal certain dynamics of the system, as is the case in many real-world dynamical systems.
- The legacy simulation, integrating over $[0, 170]$, takes nine iterations to converge using GParareal (ten for parareal), giving us approximately $kJ^{(2)} = 9 \times 20 = 180$ legacy evaluations of $\mathcal{F} - \mathcal{G}$ (results not shown).
  Integrating (22) over the full interval $[0, 340]$, GParareal converges in four iterations sooner with the legacy data than without — refer to Figure 7(b). In Figure 7(c) we can see that using the legacy data achieves a higher numerical speedup ($3.4\times$) compared to parareal ($1.6\times$).
- Figure 7(a) illustrates GParareal retaining a similar numerical accuracy to parareal with and without the legacy data. Note the steadily increasing errors for both algorithms is due to the chaotic nature of the Rössler system.

Discussion

GParareal

- In this paper, we present a time-parallel algorithm (GParareal) that iteratively locates a numerical solution to a system of ODEs.
- It does so using a predictor-corrector, comprised of numerical solutions from coarse ($\mathcal{G}$) and fine ($\mathcal{F}$) integrators. However, unlike the classical parareal algorithm, it uses a Gaussian process (GP) emulator to infer the correction term $\mathcal{F} - \mathcal{G}$.
- The numerical experiments reported above demonstrate that GParareal performs favourably compared to parareal, converging in fewer iterations and achieving increased parallel speedup for a number of low-dimensional ODE systems.
- We also demonstrate how GParareal can make use of legacy data, i.e. prior $\mathcal{F}$ and $\mathcal{G}$ data obtained during a previous simulation of the same system (using different ICs or a shorter time interval), to pre-train the emulator and converge even faster — something that existing time-parallel methods cannot do.

Parareal v. GParareal

- For the FitzHugh–Nagumo model, using just the data obtained during simulation (acquisition data), GParareal achieves an almost two-fold increase in speedup over parareal.
- Simulating over a range of initial values, GParareal converged in fewer than half the iterations taken by parareal and, in some cases, managed to converge when the coarse solver was too poor for parareal. When using legacy data, GParareal could converge in even fewer iterations.
- Similar results were illustrated for the Rössler system in but with legacy data obtained from a prior simulation over a shorter time interval — beneficial when one does not know how long to integrate a system for.
- We tested GParareal on a larger number of processors (up to 512), verifying that the cost of the GP needs to be much smaller than the cost of the fine solver in order for speedup to be maximised. In all cases, the solutions generated by GParareal were of a numerical accuracy comparable to those found using parareal.

Dimensionality (kJ and d)

- In its current implementation, GParareal may suffer from the curse of dimensionality in two ways.
- An increasing number of data points, $\mathcal{O}(kJ)$, is problematic for the standard cubic complexity GP implemented here. Possible fixes include using a more sophisticated (non-cubic complexity) emulator (e.g. Schäfer et al., 2021) or a neural network emulator instead.
- Second, trying to emulate a d-dimensional function $\mathcal{F} - \mathcal{G}$ is difficult if the number of evaluation points is not sufficient. One possible fix is to obtain more acquisition data by launching more $\mathcal{F}$ and $\mathcal{G}$ runs using the idle processors to further train the emulator at little additional computational cost. However, the accuracy of the GP emulator for $\mathcal{F} - \mathcal{G}$ is strongly controlled by the fill distance of the set of evaluation points, which is generally difficult to restrict when d is large.

Towards a Fully Probabilistic Approach?

- In equation (17), we approximate a Gaussian distribution by taking its expected value, ignoring uncertainty in the GP posterior for $\mathcal{F} - \mathcal{G}$.
- In this setting, the GP emulator is used to interpolate the $\mathcal{F} - \mathcal{G}$ data, hence it is perfectly acceptable to swap it out for any other sufficiently accurate regression method, e.g. kernel ridge regression (Kanagawa et al., 2018).
- Ideally, one would keep and propagate the full GP posterior uncertainty using the coarse solver to the next time step and then continue. While this would produce a fully probabilistic version of GParareal, this would be a computationally expensive process that we wish to avoid at this stage.
- Instead of approximating (17) by its expected value, one could simply draw a sample, leading to a stochastic solution to the ODE, much like the stochastic parareal algorithm (Pentland et al., 2022). It is unclear how this algorithm would perform vs. parareal (or even stochastic parareal), but it could still make use of legacy data following successive independent simulations.

Further Future Work 1

- Follow-up work will focus on extending GParareal, using some of the methods suggested above, to solve higher-dimensional systems of ODEs in parallel.
- It will also be of interest to investigate whether one can provide a rigorous proof of convergence to substantiate the numerical results reported here. (In the oven and nearly ready to taste!)
- In the longer term, we aim to develop a truly probabilistic time-parallel numerical method that can account for the inherent uncertainty in the GP emulator, returning a probability distribution rather than point estimates over the solution.

Thank You!

References 1

A. Clarke, C. Davies, D. Ruprecht, and S. Tobias. Parallel-in-time integration of kinematic dynamos. *J. Comput. Phys. X*, 7, 2020. doi:10.1016/j.jcpx.2020.100057.

J. Cockayne, C. J. Oates, T. J. Sullivan, and M. Girolami. Bayesian probabilistic numerical methods. *SIAM Rev.*, 61(4): 756–789, 2019. doi:10.1137/17M1139357.

V. Dolean, P. Jolivet, and F. Nataf. *An Introduction to Domain Decomposition Methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2015. doi:10.1137/1.9781611974065.

W. R. Elwasif, S. S. Foley, D. E. Bernholdt, L. A. Berry, D. Samaddar, D. E. Newman, and R. Sanchez. A dependency-driven formulation of parareal: Parallel-in-time solution of PDEs as a many-task application. In *MTAGS'11 - Proceedings of the 2011 ACM International Workshop on Many Task Computing on Grids and Supercomputers, Co-Located with SC'11*, pages 15–24, New York, NY, 2011. ACM Press. doi:10.1145/2132876.2132883.

R. FitzHugh. Impulses and physiological states in theoretical models of nerve membrane. *Biophys. J.*, 1:445–466, 1961. doi:10.1016/S0006-3495(61)86902-6.

M. J. Gander and E. Hairer. Nonlinear convergence analysis for the parareal algorithm. In *Lecture Notes in Computational Science and Engineering*, volume 60, pages 45–56. Springer, 2008. doi:10.1007/978-3-540-75199-1_4.

M. J. Gander and S. Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM J. Sci. Comput.*, 29:556–578, 2007. doi:10.1137/05064607X.

I. Garrido, B. Lee, G. E. Fladmark, and M. S. Espedal. Convergent iterative schemes for time-parallelization. *Math. Comput.*, 75(255):1403–1428, 2006. doi:10.1090/S0025-5718-06-01832-1.

References 2

F. P. Hamon, M. Schreiber, and M. Minion. Parallel-in-time multi-level integration of the shallow-water equations on the rotating sphere. *J. Comput. Phys.*, 407:109210, 2020. doi:10.1016/j.jcp.2019.109210.

P. Hennig, M. A. Osborne, and M. Girolami. Probabilistic numerics and uncertainty in computations. *Proc. R. Soc. Math. Phys. Eng. Sci.*, 471:20150142, 2015. doi:10.1098/rspa.2015.0142.

M. Kanagawa, P. Hennig, D. Sejdinovic, and B. K. Sriperumbudur. Gaussian processes and kernel methods: A review on connections and equivalences, 2018. arXiv:1807.02582.

J. L. Lions, Y. Maday, and G. Turinici. Résolution d'EDP par un schéma en temps ≪pararéel≫. *Comptes Rendus Acad. Sci. Ser. I Math.*, 332(7):661–668, 2001. doi:10.1016/S0764-4442(00)01793-6.

A. Mann. Core Concept: Nascent exascale supercomputers offer promise, present challenges. *PNAS*, 117:22623–22625, 2020. doi:10.1073/pnas.2015968117.

J. Nagumo, S. Arimoto, and S. Yoshizawa. An active pulse transmission line simulating nerve axon. *Proc. IRE*, 50: 2061–2070, 1962. doi:10.1109/JRPROC.1962.288235.

C. J. Oates and T. J. Sullivan. A modern retrospective on probabilistic numerics. *Stat. Comput.*, 29:1335–1351, 2019. doi:10.1007/s11222-019-09902-z.

K. Pentland, M. Tamborrino, D. Samaddar, and L. C. Appel. Stochastic parareal: An application of probabilistic methods to time-parallelization. *SIAM J. Sci. Comput.*, pages S82–S102, 2022. ISSN 1064-8275. doi:10.1137/21M1414231.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006. ISBN 026218253X.

O. E. Rössler. An equation for continuous chaos. *Phys. Lett. A*, 57:397–398, 1976. doi:10.1016/0375-9601(76)90101-8.

References 3

D. Samaddar, D. E. Newman, and R. Sánchez. Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm. *J. Comput. Phys.*, 229:6558–6573, 2010. doi:10.1016/j.jcp.2010.05.012.

D. Samaddar, D. P. Coster, X. Bonnin, L. A. Berry, W. R. Elwasif, and D. B. Batchelor. Application of the parareal algorithm to simulations of ELMs in ITER plasma. *Comput. Phys. Commun.*, 235:246–257, 2019. doi:10.1016/j.cpc.2018.08.007.

F. Schäfer, T. J. Sullivan, and H. Owhadi. Compression, inversion, and approximate PCA of dense kernel matrices at near-linear computational complexity. *Multiscale Model. Simul.*, 19(2):688–730, 2021. doi:10.1137/19M129526X.

M. Schober, D. Duvenaud, and P. Hennig. Probabilistic ODE solvers with Runge–Kutta means. In *Advances in Neural Information Processing Systems 27*, pages 739–747. Curran Associates, Inc., 2014. URL `http://papers.nips.cc/paper/5451-probabilistic-ode-solvers-with-runge-kutta-means.pdf`.

A. V. Sul′din. Wiener measure and its applications to approximation methods. I. *Izv. Vysš. Učebn. Zaved. Mat.*, 6(13): 145–158, 1959.

A. V. Sul′din. Wiener measure and its applications to approximation methods. II. *Izv. Vysš. Učebn. Zaved. Mat.*, 5(18): 165–179, 1960.