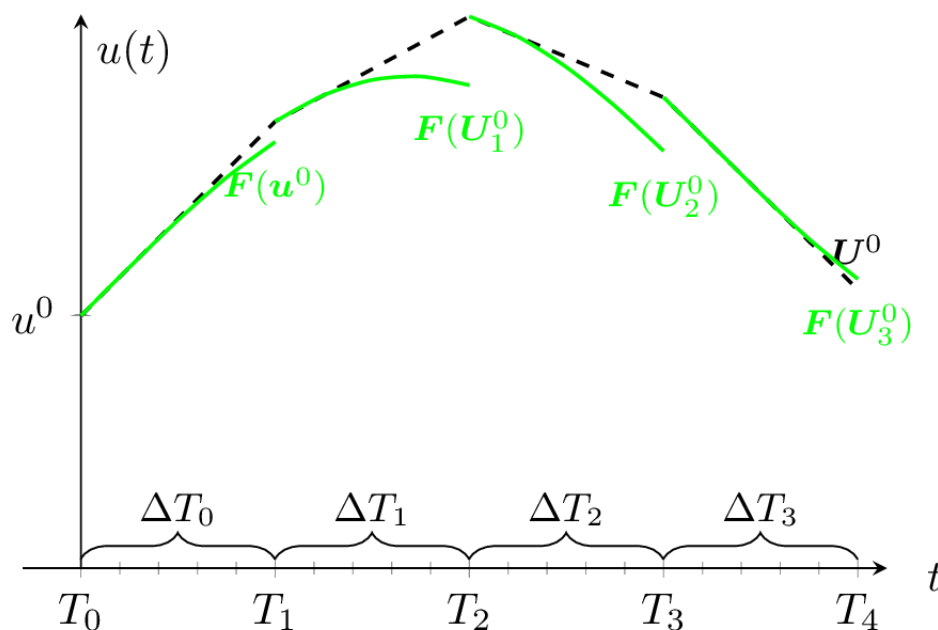


EPSRC Centre for Doctoral Training in Industrially Focused Mathematical Modelling




Choosing a Fast Initial Propagator for Rapid Convergence of the Parareal Algorithm in the Context of Simple Model Problems

Thomas Roy



Contents

1. Introduction.....	2
Glossary of Terms.....	2
2. The Parareal Algorithm.....	3
Convergence of Parareal.....	4
Other properties and options for Parareal	4
Choice of the coarse solver.....	5
3. Numerical Results	6
4. Discussion, Conclusions and Recommendations	7
High-order coarse solvers.....	7
Convergence for nonlinear problems ...	7
Further analysis for advanced Parareal versions.....	7
5. Potential Impact.....	7



Parallel Computing is a type of computation where multiple calculations are carried out simultaneously.

1. Introduction

Advancements in hardware in the last decades have made possible the numerical solution of increasingly complex models. However, these advancements are limited by the more recent stagnation in CPU clock speed. As a result, the best way to quicken calculations is by developing better algorithms and allowing calculations to be performed on multiple processors (e.g. CPUs, GPUs). This has justified the recent focus on efficient parallel hardware and algorithms. Parallel computing is a type of numerical computation where multiple calculations are carried out simultaneously, usually on different processors. In general, the parallelisation of numerical solvers for differential equations is done through the spatial variables, i.e. by separating the spatial domain into independent subdomains that are assigned to different processors. There have been several successful efforts to extend parallelisation to the time domain in the case of time-dependent ordinary differential equations (ODEs), or time-dependent partial differential equations (PDEs) where spatial parallelisation has reached its limit. These parallel-in-time methods are intrinsically more challenging due to a causality principle; the later solution depends on the earlier one. Over the past 50 years, a variety of different time-parallel time integration methods have been introduced. These different strategies include multiple shooting methods, domain decomposition and waveform relaxation, space-time multigrid, and direct time parallel methods.

One such method is the Parareal algorithm, introduced in [1]. The algorithm combines a fast initial propagator that gives a coarse approximation of the solution on the whole time domain with a fine solver to obtain more accurate solutions on independent smaller subdomains. The choice of these components affects the rate of convergence or non-convergence of the overall Parareal algorithm. The Culham Centre for Fusion Energy (CCFE) is interested in Parareal for complicated plasma physics simulations, particularly in the behaviour of plasmas at the edge of the experiment where neutral transport becomes important. Previous attempts have been made by CCFE to characterise the behaviour of the algorithm for these applications. Our key issue is understanding how the fast initial propagator affects the outcome and performance of the Parareal algorithm.

Our aim is to determine which factors affect the convergence of the Parareal algorithm in simpler model problems. In Section 2, we detail the Parareal algorithm in a very general formulation. We will introduce the necessary concepts of numerical analysis and present details about pre-existing theoretical results. Next, we will discuss the different choices of parameters for the algorithm, including the order of convergence of the coarser solver. In Section 3, we will compare theoretical results with numerical results for our test model, the Lorenz system. Finally, in Section 4, we will discuss our observations and provide recommendations based on our results.

Glossary of Terms

- **Time-stepping method:** A numerical method used to approximate the time derivative in a time-dependent differential equation. These result in a system where the earlier numerical solution is needed to evaluate the later solution. At each iteration, the numerical solution advances in time by a predetermined or adaptive time-step.
- **Truncation error:** A measure of the local discrepancy between the numerical solution given by a time-stepping method, and the exact solution of the problem.
- **Convergence:** A time-stepping method is said to be convergent if the numerical solution approaches the exact solution as the time-step goes to 0. The Parareal

algorithm is said to be convergent if the numerical solution approaches the numerical solution obtained from an accurate serial solver as the Parareal iterations increase.

- **Linear stability:** A time-stepping method is said to be stable (in the linear sense) for a chosen time-step and model if the numerical solution remains bounded. Stability is necessary for convergence.
- **Order of convergence:** A time-stepping method is of order m if the truncation error of the numerical solution is proportional to the m th power of the time-step, as the time-step goes to zero
- **Linear convergence:** An algorithm converges linearly if, after each iteration, the error is multiplied by a constant factor smaller than one.
- **Superlinear convergence:** An algorithm converges superlinearly, if after each iteration, the error is multiplied by a factor that goes to 0 as the iterations progress.

2. The Parareal Algorithm

The Parareal algorithm is a parallel algorithm used for the solution of initial value problems. In contrast to other solvers which use multiple steps (e.g Runge-Kutta methods or multi-step methods), some of the computations in the Parareal algorithm can be performed in parallel.

The time interval is decomposed into N time slices, each assigned to a different processor.

To allow computations to be undertaken in parallel, the time interval on which the initial value problem is to be solved is decomposed into N time slices. Each slice is assigned to a different processor when the parallel computations are performed, so N is also equal to the number of processors. The initial value problem is then separated into N different initial value problems with a matching condition such that the final solution of a time slice corresponds to the initial value of the next.

The fine solver F gives accurate solutions for the time slices, while the coarse solver G gives an estimate of the overall solution.

The Parareal algorithm is based on the use of two different solvers for differential equations. One of the solvers, denoted by F , gives a very accurate approximation of the solution while the other solver, denoted G , gives a less accurate approximation, but at a much cheaper computational cost. Typically, the fine solver F uses a higher-order time-stepping method than for the coarse solver G , which may also use larger time-steps. The algorithm generates an initial guess of the solution (a natural choice is the coarse solution given by the coarse solver G). This first approximate solution, formed from the solutions U_n^0 for different time-steps n , provides initial values for each time slice and its corresponding initial value problem. Each of those N problems is then solved using the fine solver F . This then leads to a sequential process, which starts at the first time slice, where the coarse solver uses the initial value to obtain a solution for that time slice. The solution is then corrected by the difference between the fine solution and coarse solution from the previous iteration. This process is called a Parareal iteration. Denoting u^0 as the initial value of the problem, and U_k^n as the solution at the n th time slice and k th iteration, we obtain the following recurrence relation:

$$\begin{cases} U_0^{k+1} = u^0, \\ U_{n+1}^{k+1} = G(U_n^{k+1}) + F(U_n^k) - G(U_n^k). \end{cases} \quad (1)$$

By denoting H as the difference between F and G , we can illustrate the recurrence relation (1) in Figure 1.

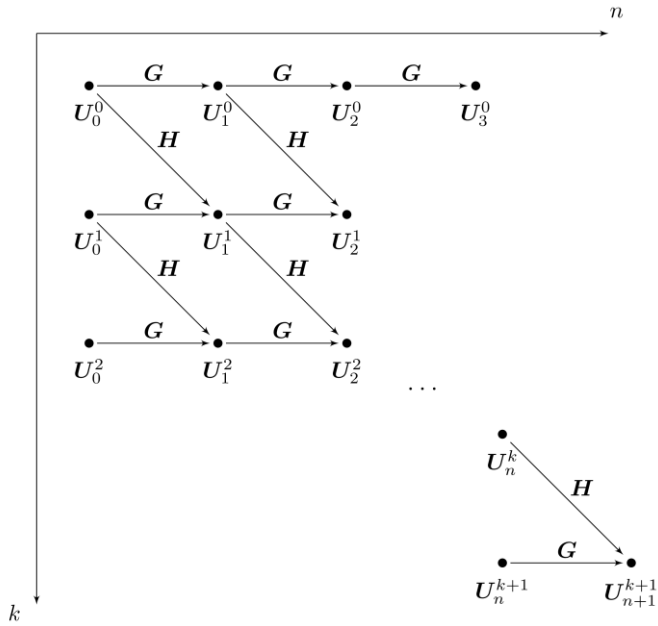


Figure 1: Recurrence Relation from (1), where U_n^k is the approximate solution at the end of the n^{th} time slice and k^{th} iteration. The coarse approximation with G is done in serial, while the fine approximation and correction with H is done in parallel.

Convergence of the Parareal algorithm

Several convergence results for the Parareal algorithm exist in the literature. We will focus on the results presented in [2]. These results are based on a very simple initial value problem: the linear scalar problem $u'(t) = a u(t)$, where a is a constant, t is time, and $u'(t)$ is the time derivative of $u(t)$. We start by quickly introducing one-step time-stepping methods, where only the solution at the current time-step is needed to update the solution at the following time-step. A one-step method is associated with a stability function $R(z)$, which updates the solution to the next time-step. Given a time-step Δt , the one-step method is stable for the linear scalar problem if $R(a\Delta t) < 1$.

In theory, the Parareal algorithm has superlinear and linear convergence on bounded and unbounded interval,

As is commonly done in studies of the Parareal algorithm, we suppose that the fine solver F is so accurate that it returns the exact solution of the problem. Additionally, we suppose that G is a stable one-step method. If the linear scalar case is solved on a bounded interval the convergence of the Parareal algorithm is superlinear. Alternatively, if the problem is solved on a long unbounded interval, the convergence is linear.

Other properties and options for Parareal

We notice from (1) that, since $U_0^0 = U_0^1 = u^0$, we have $U_1^2 = F(u^0)$, which implies convergence to the fine solution on the first time slice after a single iteration. Similarly, after k iterations, the solution at the n^{th} time slice has converged for all n smaller or equal to k . Therefore, after $k = N$ iterations the solution will have converged over the whole time domain. This property can be thought of as the information from the initial condition travelling to later time slices. Obviously, the Parareal algorithm provides no computational gain if it converges in $k = N$ iterations. In fact, each processor would do the same work as a single processor would by solving the problem serially, without considering the cost of the coarse solver. Therefore, the algorithm is only efficient as long as a lower number of iterations is needed for a sufficient convergence (sufficient relative to a chosen tolerance and stopping criteria). In practice, sufficient convergence can be achieved for k much

smaller than n . We seek to determine what parameter options favour a fast convergence of the algorithm.

The parameters for the Parareal algorithm include the choice of the solvers F and G and the length of the time slices. The time-step used for the solvers may be chosen as equal to the size of the time slice or alternatively, one or both solvers may use smaller time-steps. The general idea for the choice of F and G is that F should be accurate, and G should be cheap to evaluate. One simple choice for F and G is using the same time-stepping method, but with smaller time-steps for F . In addition, or alternatively, one could also use a higher-order method for F .

For PDEs, the coarse solver G could solve the equations with a coarser spatial discretisation than the one used with F . This is, of course, cheaper to compute, but may be necessary for the stability of the numerical solution. Furthermore, the coarse solver G could solve a simpler problem than the original problem solved by F .

Choice of the coarse solver

In practice, after each iteration the error between the solution at the current iteration and the initial guess is multiplied by some *convergence factor*. If this number is smaller than one, the algorithm will converge. In both the bounded and unbounded cases, the convergence factor is proportional to the truncation error of the coarse solver, which is the difference between the exact solution and the solution given by the coarse solver, for a time slice of size ΔT .

The convergence factors are proportional to the truncation error of the coarse solver G .

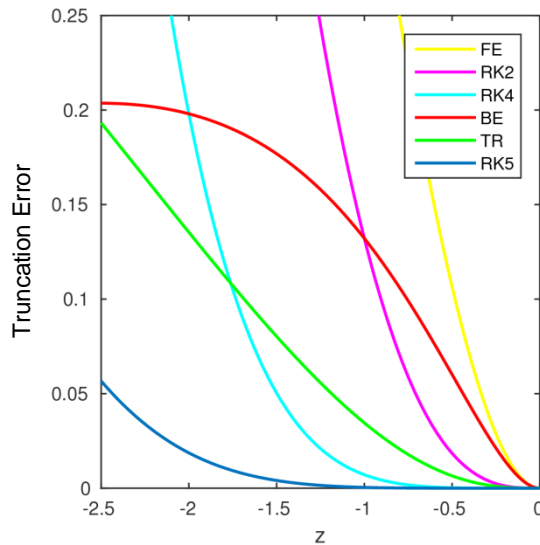


Figure 2: Truncation error of various time-stepping methods for the scalar linear problem $u'(t) = a u(t)$, where $z = a\Delta T$.

In Figure 2, we illustrate the truncation error of various time-stepping methods. The second, fourth, and fifth-order Runge-Kutta methods are denoted by RK2, RK4 and RK5, respectively. We also study two first-order methods, forward Euler (FE) and backward Euler (BE), as well as one second-order method, the trapezoidal rule (TR). We observe that, for small magnitudes of $z = a\Delta T$, the truncation errors of the higher-order methods are much smaller, resulting in smaller convergence factors. As z increases, the truncation error becomes unbounded for some of the methods. This corresponds to the coarse solver being unstable for larger magnitudes of z . We conclude from Figure 2 that, for z small enough, the Parareal algorithm will converge faster when using higher-order coarse solvers.

For small enough time-steps, higher-order coarse solvers result in the faster convergence of the Parareal algorithm.

3. Numerical Results

Most results from the literature only consider linear scalar problems. In addition to this simple case, we will consider other models to verify whether the theory can be applied to more complex equations. We consider the Lorenz system, a system of nonlinear ODEs, as well as the wave equation, a linear PDE (results not shown). The Lorenz system is a system of nonlinear ODEs introduced by Lorenz in 1963. It is known for having chaotic solutions for certain parameter values and initial conditions.

In this section, we investigate the behaviour of the error at different Parareal iterations for the Lorenz system. We compare the numerical results with the theoretical superlinear and linear bounds presented in [2]. In order to compare the numerical results with the theoretical linear and superlinear bounds, we need a value for the equivalent of the constant α in the linear scalar case. These can be obtained through a linearization of the system at each time slice. However, these values will be different throughout the time domain. In this case, we will take α as the average of these different values, which is a lenient and less formal way of choosing this constant.

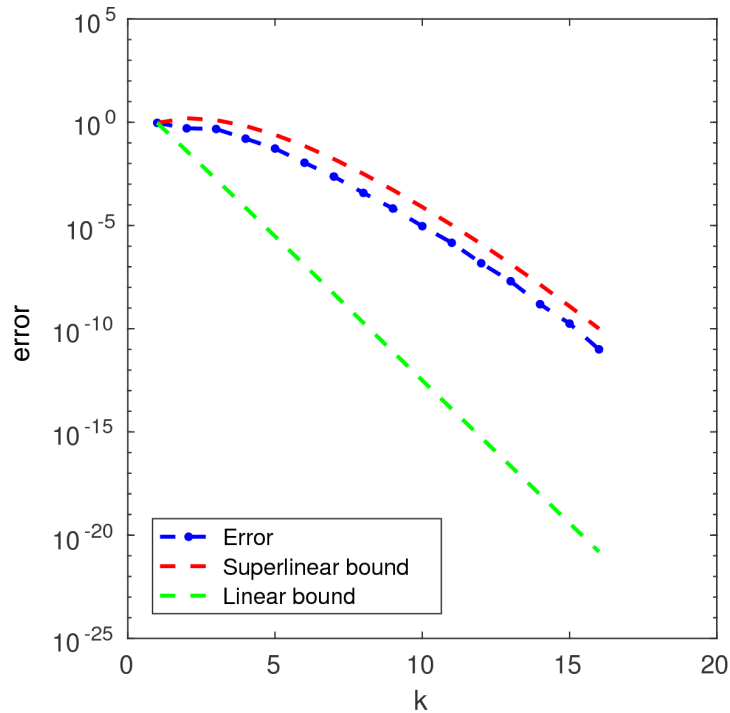


Figure 3: Convergence of the Parareal algorithm applied to the Lorenz system using the trapezoidal rule as the coarse solver. We compare the observed numerical error (shown in blue) at time $T = 4$ with the theoretical bounds.

In Figure 3, we illustrate how the observed errors compare to the theoretical bounds when using the trapezoidal rule as the coarse solver with $N = 64$ time slices. For each variable, we calculate the maximal absolute error over the different time slices. We observe that the algorithm converges superlinearly, very closely to the theoretical bound. However, this bound is usually more restrictive than the observed errors, especially for more conservative choices of α .

4. Discussion, Conclusions & Recommendations

We have investigated the different options which have an influence on the convergence of the Parareal algorithm.

High-order coarse solvers

Based on theoretical results from the literature, we determined that the choice of the coarse solver was key in the resulting rate of convergence (or non-convergence) of the Parareal algorithm. From our observations, using higher-order time-stepping methods for the coarse solver produces a faster convergence of the Parareal algorithm. This result was then confirmed by numerical results for a simple scalar linear problem, the Lorenz system, and the wave equation (using finite differences for the spatial discretisation). However, the faster convergence for the high-order coarse solvers is also accompanied by coarse solves which require additional computational time. We recommend that CCFE test the performance of higher-order coarse solvers such as the fifth-order Runge-Kutta method. Other more complex one-step high-order methods could be considered.

Convergence for nonlinear problems

Theoretical results from the literature provide linear and superlinear convergence bounds for the Parareal algorithm, for unbounded and bounded intervals, respectively, in the case of a simple scalar linear problem. It is not straightforward to what degree these results apply to nonlinear systems. In the observed cases at least, the theoretical bounds are satisfied by the numerical errors of the algorithm. However, these bounds tend to be more restrictive than the observed errors. Nonetheless, the qualitative behaviour of the error corresponds with superlinear convergence. Therefore, further work is needed to get better convergence results for the nonlinear cases. The theoretical error bounds will potentially be very specific to the application as well as the spatial discretisation in the case of PDEs.

Further analysis for advanced Parareal versions

Additionally, further analysis of the more advanced versions of the Parareal algorithm is necessary. In fact, most theoretical results focus on simple implementations of the algorithm, as we have presented here. A formal analysis of the use of a coarser spatial grid for the coarse solver is required, including the manner in which the interpolation between the coarse and fine grids is performed. Further analysis on the use of simpler physical models for the coarse solver is also needed.

5. Potential Impact

CCFE now has a better understanding of how the fast initial propagator affects the outcome and performance of the Parareal algorithm. We have provided new insights that help choosing a coarser solver for the faster convergence of the algorithm.

Debasmita Samaddar, Computational Plasma Physicist, CCFE, said, “*Great job! Thomas's work lays a solid foundation for better simulating complex systems that we deal with at CCFE.*”

References

1. JL Lions, Y Maday, G Turinici (2001) *Résolution d'EDP par un schéma en temps «pararéel»*. Comptes Rendus de l'Académie des Sciences-Series I-Mathematics 332.7, 661-668.
2. MJ Gander, S Vandewalle (2007) *Analysis of the parareal time-parallel time-integration method*. SIAM Journal on Scientific Computing 29(2), 556-578.