Université de Strasbourg

cemosis
Centre de modélisation et de simulation de Strasbourg

UFR de mathématique et d'informatique
Université de Strasbourg

Master 1 CSMI : Scientific Computation and Mathematics of Innovation

Parareal Algorithm: An Advanced Analysis

Host Organization: Cemosis

Author:
Oussama BOUHENNICHE

Supervisor:
M. Christophe
PRUD'HOMME

August 20, 2024

Abstract : The parareal algorithm is a method to solve time-dependent problems in parallel. This report presents an analysis of the Parareal algorithm, focusing on its convergence properties, order of convergence, and performance evaluation in a parallel computing context. Various solver combinations and test cases, including the Lorenz system, are analyzed. Solvers tested include explicit Euler, implicit Euler, explicit RK2, implicit RK2, explicit RK4, and implicit RK4. Results indicate that solver choice significantly affects the parallel efficiency, number of iterations, and convergence behavior of the Parareal algorithm. These findings provide valuable insights into optimizing solver selection for parallel-in-time computations.
keywords : Parareal, Euler, Runge-Kutta, Lorenz system, convergence order, speedup

Contents

List of Figures

List of Algorithms

1 Introduction

The Parareal algorithm is a parallel-in-time integration method designed to solve time-dependent differential equations efficiently by exploiting parallel computation. This report builds upon a previous project where the Parareal algorithm was studied and implemented, along with various ODE solvers. This internship continues and expands upon that work, aiming to further explore the capabilities and performance of the Parareal algorithm.

The primary objectives of this report include:

1. Analyzing the convergence of the Parareal algorithm by evaluating the number of iterations required for different solver combinations.
2. Investigating the order of convergence of the Parareal algorithm based on solver choices.
3. Evaluating the performance of the Parareal algorithm in relation to the number of processes used.

Solvers tested include explicit Euler, implicit Euler, explicit RK2, implicit RK2, explicit RK4, and implicit RK4. These solvers represent a range of explicit and implicit methods with varying computational complexities and stability properties. The performance metrics considered in this study provide a comprehensive evaluation of the Parareal algorithm's efficiency and effectiveness with different solver combinations.

2 Background

2.1 Parareal Algorithm

The parareal algorithm, presented by Lions, Maday, and Turinici in 2001 [1], is an iterative method designed to parallelize the solution of time-dependent differential equations. It aims to solve problems of the form:

$$\frac{du}{dt} = f(t,u) \quad , \; u(t_0) = u_0$$

The key idea is to decompose the time domain into smaller sub-intervals, solve these intervals in parallel using a coarse solver for an initial approximation, and iteratively refine the solution using a fine solver. The algorithm leverages parallel computation to achieve significant speedup, making it particularly useful for long-time integration problems.

2.1.1 Parareal Algorithm Steps

1. **Initialization:**
   - Divide the time interval $[t_0, T]$ into N sub-intervals:
     $$t_0 < t_1 < ... < t_N = T$$
     .
   - Set the initial guess for the solution at each time point using a coarse solver G:
     $$u_0^0 = u_0, U_n^0 = G(u_{n-1}, \Delta t) \text{ for } n = 1, 2, ..., N$$
2. **Iterative Refinement:**
   - For each iteration k:
     $$u_0^{k+1} = u_0$$
     $$U_{n+1}^k = G(t_n, t_{n+1}, U_n^k) + F(t_n, t_{n+1}, U_n^{k-1}) - G(t_n, t_{n+1}, U_n^{k-1}) \text{ for } n = 1, 2, ..., N$$
     Here, G is the coarse solver and F is the fine solver. The fine solver F provides a more accurate solution over the sub-interval Δt.
3. **Convergence:**
   - The iterations continue until the difference between successive iterations meets a specified tolerance:
     $$||U_n^{k+1} - U_n^k|| < tol \text{ for all } n$$

2.1.2 Parallelization

The Parareal algorithm leverages parallelism to enhance the efficiency of time integration by processing different time intervals simultaneously. This approach is particularly effective for large-scale simulations involving time-dependent problems.

The following steps outline how the Parareal algorithm achieves parallelization:

1. **Time Decomposition:**

   Split the total time interval $[0, T]$ into N sub-intervals: $[T_0, T_1], [T_1, T_2], ..., [T_{N-1}, T_N]$. Each sub-interval can be processed independently once the initial conditions are set.

2. **Coarse Propagation:**

   Use a coarse solver G to generate an initial approximation of the solution over the entire time domain. This step is performed sequentially but is relatively quick due to the use of larger time steps.

3. **Parallel Fine Propagation:**

   Solve the problem within each sub-interval using a fine solver F in parallel. This solver provides a more accurate solution with smaller time steps, tailored to each sub-interval.

4. **Iterative Correction:**

   Improve the solution iteratively by refining the combined coarse and fine solutions. Each iteration updates the previous solution based on the fine solver's results. This step requires communication and synchronization among processors to adjust the initial conditions for subsequent iterations.

The following algorithm outlines the parallel implementation of the Parareal method:

Algorithm 1 Parareal Algorithm (parallel)

1: **Given:** Coarse function G, fine function F, number of time steps N, maximum number of iterations K, ϵ tolerance, number of process.
2: **Initialize: in sequential**

- $U_0^0 = u_0$,
- $U_{j+1}^0 = G(t_j, t_{j+1}, U_j^0) \quad j = 0, 1, \ldots, N-1$

3: **while** $|U^k - U^{k-1}| > \epsilon$ **do**

- run the fine solver on each process and given a sub-interval of the time slices, in parallel: $F(t_j, t_{j+1}, U_j^{k-1}) \qquad j = 0, 1, ..., N-1.$
- communicate fine solution and update u in sequential and share it to all process: $\qquad j = 0, 1, ..., N-1.$
  $U_{j+1}^k = G(t_j, t_{j+1}, U_j^k) + F(t_j, t_{j+1}, U_j^{k-1}) - G(t_j, t_{j+1}, U_j^{k-1})$

4: **end while**
5: **Return:** U^k

2.2 ODEs

2.2.1 Lorenz System

The Lorenz system [2] is used to measure execution time and speedup. It is governed by the following equations:

$$\begin{cases} \frac{dx}{dt} = \sigma(y - x) \\ \frac{dy}{dt} = x(\rho - z) - y \\ \frac{dz}{dt} = xy - \beta z \end{cases} \tag{1}$$

where:

- x is the rate of convective overturning,
- y is the horizontal temperature variation,
- z is the vertical temperature variation,
- σ is the Prandtl number,
- ρ is the Rayleigh number,
- β is a geometrical factor.

The equations describe the behavior of a two-dimensional fluid layer that is heated uniformly from below and cooled from above.

- Parameters: $\sigma = 10$, $\rho = 28$, $\beta = \frac{8}{3}$.
- Initial conditions: $u_0 = [5, -5, 20]$.
- Time span: $[0, 10]$.

We used the parameters and initial condition proposed by H Batmalle, PE Bécue, V Le Gallic (2011),[3]

2.2.2 Other Test Systems

These systems are used to measure the number of Parareal iterations required for convergence and the convergence order.

- **System 1:**

$$\frac{dy}{dt} = -ty^2 \tag{2}$$

  - Exact solution: $y(t) = \frac{2}{1+t^2}$.
  - Initial condition: $y(0) = 2$.
  - Time span: $[0, 5]$.

- **System 2:**

$$\frac{dy}{dt} = -y + cos(t) \tag{3}$$

  - Exact solution: $y(t) = \frac{-1}{2}e^{-t} + \frac{1}{2}cos(t) + \frac{1}{2}sin(t)$.
  - Initial condition: $y(0) = 0$.
  - Time span: $[0, 5]$.

- **System 1:**

$$\frac{dy}{dt} = -2y \tag{4}$$

  - Exact solution: $y(t) = e^{-2t}$.
  - Initial condition: $y(0) = 1$.
  - Time span: $[0, 5]$.

2.3 Solvers

Six different solvers were tested, representing a mix of explicit and implicit methods with varying computational complexities and stability properties [4].

2.3.1 Explicit Euler (feuler)

- **Order:** First-order.
- **Stability:** Conditionally stable.
- **Scheme:**

$$y_{n+1} = y_n + h * f(t_n, y_n) \tag{5}$$

2.3.2 Implicit Euler (beuler)

This requires solving a nonlinear equation at each step, typically using fixed-point iteration or Newton's method.

- **Order:** First-order.
- **Stability:** Unconditionally stable.
- **Scheme:**

$$y_{n+1} = y_n + h * f(t_{n+1}, y_{n+1}) \tag{6}$$

2.3.3 Explicit RK2 (rk2)

- **Order:** Second-order.
- **Stability:** Conditionally stable..
- **Scheme:**

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + h, y_n + k_1) \\ y_{n+1} &= y_n + \frac{1}{2}(k_1 + k_2) \end{aligned} \tag{7}$$

2.3.4 Implicit RK2 (brk2)

This also involves solving a nonlinear system at each step.

- **Order:** Second-order.
- **Stability:** Unconditionally stable.
- **Scheme:**

$$y_{n+1} = y_n + hf(t_n + 2h, y_n + 2hf(t_n, yn)) \tag{8}$$

2.3.5 Explicit RK4 (rk4)

- **Order:** Fourth-order.
- **Stability:** Conditionally stable..
- **Scheme:**

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\ k_4 &= hf(t_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \end{aligned} \tag{9}$$

2.3.6 Implicit RK4 (brk4)

This involves solving a complex system of nonlinear equations.

- **Order:** Fourth-order.
- **Stability:** Unconditionally stable.
- **Scheme:**

$$\begin{aligned} yn + 1 = y_n + \frac{h}{6}(f(t_n, y_n) + 2f(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n)) \\ + 2f(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n))) \\ + f(t_n + h, y_n + hf(t_n + \frac{h}{2}, y_n + \frac{h}{2}f(t_n, y_n)))) \end{aligned} \tag{10}$$

3 Methodology

3.1 Host Organization

3.1.1 Overview

The internship was hosted by Cemosis, a research laboratory specializing in scientific computing, mathematical modeling, and industrial applications. Cemosis collaborates closely with academic institutions and industry partners to develop innovative solutions in computational science and engineering.

3.1.2 Access to Resources

Access to high-performance computing resources and specialized software tools facilitated the implementation and testing of the Parareal algorithm across different solver configurations and test cases. This infrastructure enabled rigorous experimentation and performance evaluation under varying computational loads.

3.2 Previous Works

In a prior project, I developed a Python package that implements the Parareal algorithm for solving time-dependent differential equations. This package served as the foundation for the current work, allowing for further exploration and refinement of the algorithm's performance and capabilities.

3.2.1 Overview of the Python Package

The Python package developed in the previous project provides a flexible and modular implementation of the Parareal algorithm. It is designed to facilitate experimentation with various ordinary differential equation (ODE) solvers and to allow easy customization of solver configurations and convergence criteria.

- **Core Features:**
  - **Solver Integration:** The package supports multiple ODE solvers, including explicit and implicit Euler methods, and Runge-Kutta methods of different orders as well as scipy ode solvers. These solvers can be used interchangeably within the Parareal framework.
  - **Parallelization Support:** The package leverages Python's multiprocessing library **MPI** to parallelize computations across multiple cores. This enables efficient execution of the Parareal algorithm on multi-core systems.
  - **Customization:** Users can define their own coarse and fine solvers, adjust time intervals, and set convergence tolerances and max number of parareal iteration, making the package highly adaptable to different problem domains.

3.3 Experimental Setup

3.3.1 Parallel Environment

The experiments were conducted on a computing cluster utilizing many threads to parallelize the Parareal algorithm across multiple cores.

3.3.2 Solver Configurations

Each test case employed a combination of coarse and fine solvers. These solvers were selected based on their computational complexity, stability characteristics, and order of convergence to provide a comprehensive evaluation of the Parareal algorithm's performance.

3.3.3 Convergence Criteria

The convergence of the Parareal algorithm was assessed based on the ℓ^2 norm of the difference between successive iterations of the solution vector. Convergence was deemed achieved when the difference fell below a predefined tolerance level *tol*.

3.4 Tests Setup

3.4.1 Number of Parareal Iterations and Convergence Order Tests

To assess the number of iterations required for convergence and to determine the convergence order, the different systems (System 1, System 2, System 3) were tested. These tests were conducted using all possible combinations of coarse and fine solvers. The steps for these tests were:

1. **Initial Setup:**
    - For each system, the initial condition was set as specified.
    - The time span $[0, 5]$ was discretized into appropriate time intervals.
2. **Execution:**
    - For each system, numerical solutions were obtained using different solver combinations (Explicit Euler, Implicit Euler, Explicit RK2, Implicit RK2, Explicit RK4, Implicit RK4).
    - The number of Parareal iterations required to reach a specified tolerance *tol*. was recorded for each solver combination.

3. **Convergence Order:**

   - **Convergence Order Calculation:** the convergence order was computed by the following formula:

$$P = \frac{1}{n} \sum_{i=0}^{n-1} \frac{log\left(\frac{e_{i+1}}{e_i}\right)}{log\left(\frac{\Delta_{i+1}}{\Delta_i}\right)} \tag{11}$$

     were:
     - e_i is the error after the i iteration
     - Δ_i is the time step used in the i iteration.

   For each combination of coarse and fine solvers, the convergence order P was calculated by running the Parareal algorithm on all tests systems. The error was measured as The difference between the numerical and exact solutions obtained analytically.

3.4.2 Execution Time and Speedup Tests

To evaluate the execution time and speedup, the Lorenz system was used for all possible combinations of coarse and fine solvers available. The steps for these tests were:

1. **Initial Setup:**
   - The initial condition and parameters were set as specified.
   - he time span $[0, 10]$ was discretized into appropriate time intervals.
2. **Parallel Execution:**
   - The Lorenz system was solved using different numbers of P processes (up to 128) with MPI for parallel execution.
   - Execution times (T_P) were recorded for each P.
3. **Speedup Calculation:**
   - Speedup (S) was calculated using the formula:

$$S = \frac{T_1}{T_p} \tag{12}$$

     where T_1 is the execution time with 1 process, and T_P is the execution time with P processes.

4 Implementation

In this section, we describe the technical details of the implementation of our methodology. The project is implemented in Python, leveraging several libraries and tools to facilitate the development of the Parareal algorithm analysis.

4.1 Software and Libraries

4.1.1 Python Environment:

The implementation was carried out using Python, chosen for its extensive libraries and ease of use in numerical computations.

4.1.2 Libraries:

- **NumPy:** For efficient numerical operations and array manipulations.
- **mpi4Py:** For parallel execution using MPI (Message Passing Interface), allowing the Parareal algorithm to be executed across multiple processes.
- **Matplotlib:** For generating plots and visualizations of the results.
- **Seaborn:** for creating informative and attractive statistical graphics.
- **Pandas:** for data manipulation and analysis.

4.2 Parallel Test Execution

To enhance the efficiency of testing, especially when dealing extensive test suites, a shell scripts has been created to execute tests in a parallel environment. This approach significantly reduces the overall test execution time.

The shell scripts automates the execution of test cases, leveraging parallel processing to optimize performance. This ensures that the tests are executed more efficiently, especially in environments with a large number of tests.

For each analysis test, a corresponding shell script and Python script have been created to handle specific test cases and custom configurations.

These scripts typically include the following elements:

- **Test Configuration:** Define parameters specific to the test, such as test file names, parameters, and configurations.

- **Test Execution**: Commands to execute the test, calling a Python script with appropriate arguments.
- **Result Handling:** Save results, and clean up temporary files.

4.3 Code Structure

The project is available on github and is structured as shown in the picture below 1:

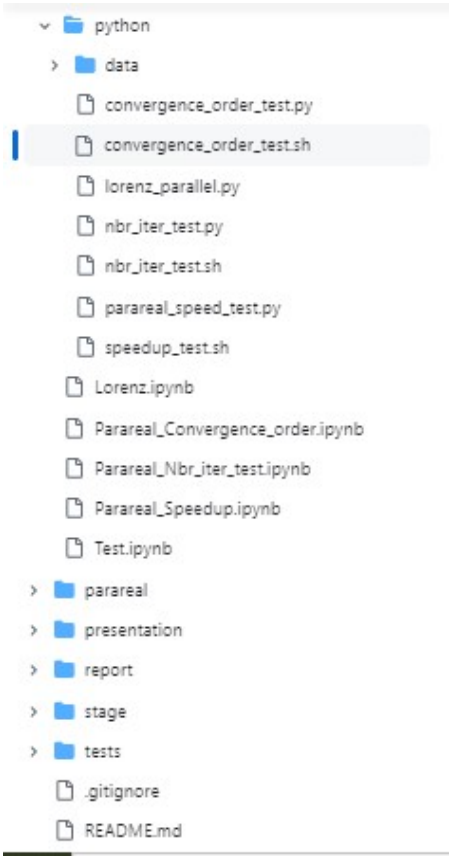

Figure 1: Code structure

- **convergence_order_test.sh:** This shell script automates the execution of the convergence order test using the Python script convergence_order_test.py. To run the script, use the following command in the terminal:

```
./convergence_order_test.sh <nbr_of_mpi_processes>
```

Replace <nbr_of_mpi_processes> with the desired number of MPI processes.

- **nbr_iter_test.sh:** This shell script automates the execution of the number of parareal itertation test using the Python script nbr_iter_test.py. To run the script, use the following command in the terminal:

```
./nbr_iter_test.sh.sh <nbr_of_mpi_processes>
```

Replace <nbr_of_mpi_processes> with the desired number of MPI processes.

- **speedup_testt.sh:** This shell script automates the execution of the number of parareal speedup test using the Python script speedup_test.py. To run the script, use the following command in the terminal:

```
./speedup_test.sh
```

For each test analysis, a dedicated Jupyter notebook was created to visualize and interpret the results. These notebooks utilize libraries such as Seaborn and Pandas to generate comprehensive plots, aiding in the detailed examination of the test outputs. Each notebook is designed to provide insights into the test results through visualizations, making it easier to understand and analyze the data.

5 Results and Analysis

This section presents the results from the tests conducted using the Parareal algorithm with various solver combinations. The focus is on convergence analysis, order of convergence, and performance evaluation in a parallel computing environment.

5.1 Convergence Analysis

To evaluate the convergence of the Parareal algorithm, three systems were tested: System 1, System 2, and System 3. Each system was solved using different solver combinations, and the number of iterations required to reach a specified tolerance (*tol*) was recorded.

5.1.1 Number of Parareal Iterations for Convergence

The results for the number of iterations required by the Parareal algorithm to converge using different combinations of coarse and fine solvers are presented below for the three different systems. The results are depicted in the provided bar charts.

System 1 Results:

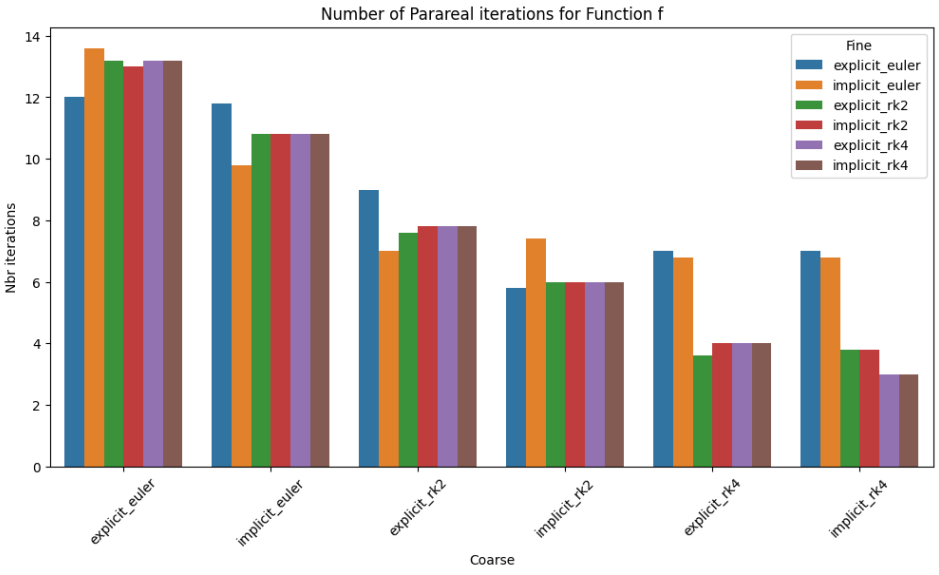

Figure 2: Number of Parareal iterations for system 1

System 2 Results:

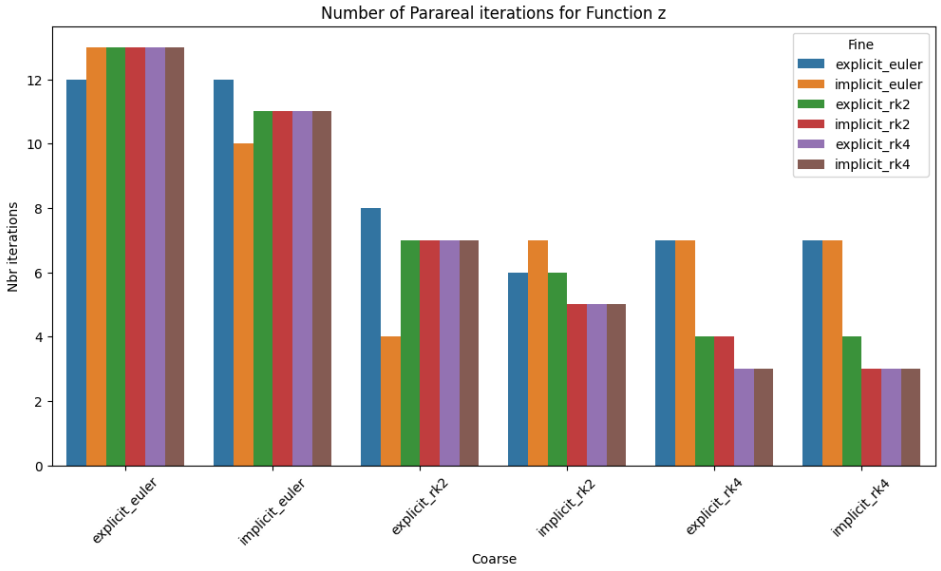

Figure 3: Number of Parareal iterations for system 2

System 3 Results:

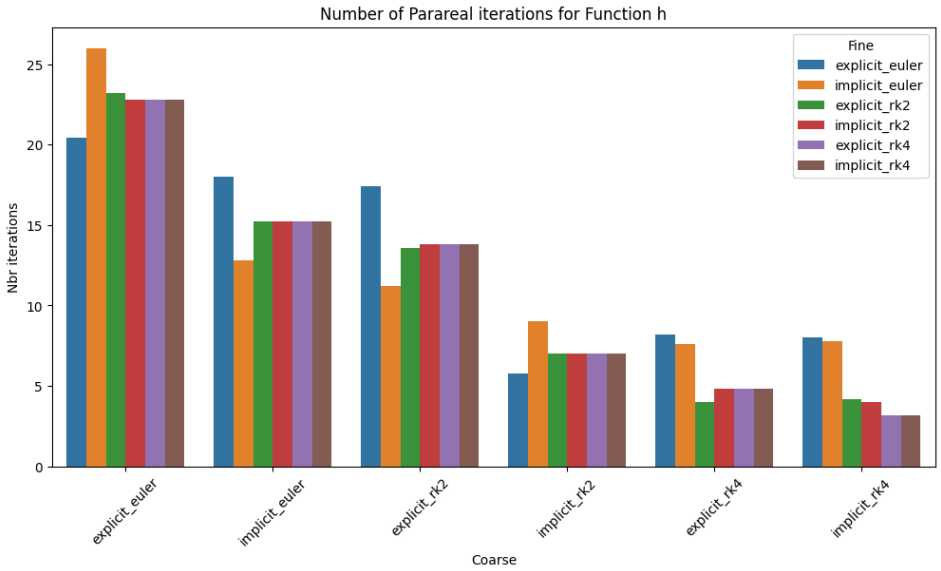

Figure 4: Number of Parareal iterations for system 3

Analysis:

- The number of iterations required for convergence varied significantly depending on the solver combination used.
- Higher-order coarse solvers tended to require fewer iterations for convergence compared to lower-order solvers.
- Explicit Euler required the most iterations across all systems, indicating lower efficiency in the Parareal framework.
- Implicit methods generally led to faster convergence, with implicit RK4 requiring the least number of iterations.
- The choice of the coarse solver affect the number of iterations required for convergence and Parareal using Higher-order coarse solvers tended to converge faster than lower-order.

5.2 Order of Convergence

The order of convergence was analyzed by comparing the numerical solutions obtained using the Parareal algorithm to the exact solutions of the test systems.

5.2.1 Convergence Order Results

The following figures illustrate the results of convergence order test for the different systems using various solvers combinations. The results are presented in the provided bar charts.

System 1 Results:

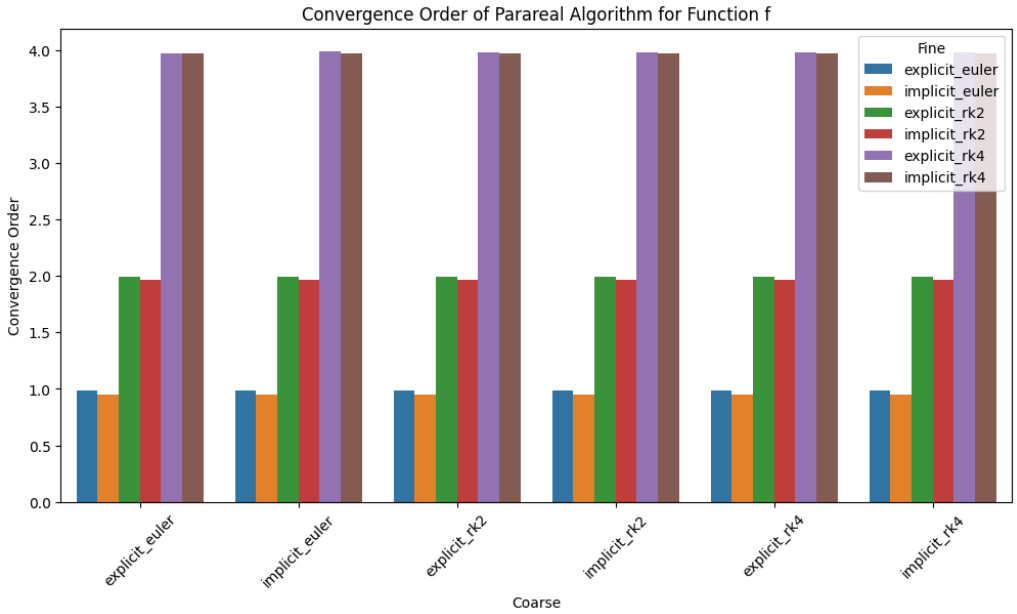

Figure 5: Convergence Order of Parareal Algorithm for system 1

System 2 Results:

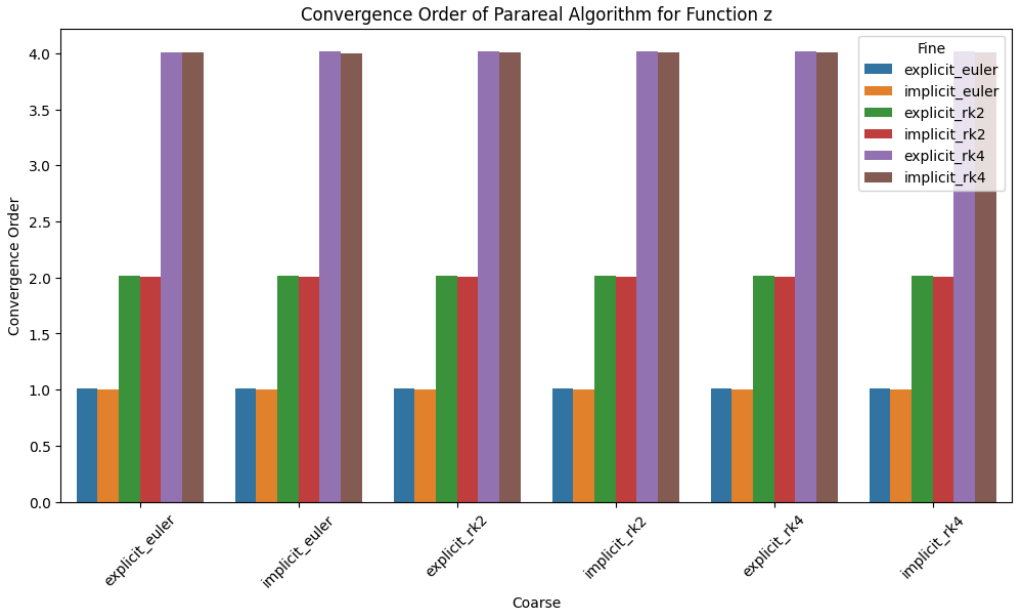

Figure 6: Convergence Order of Parareal Algorithm for system 2

System 3 Results:

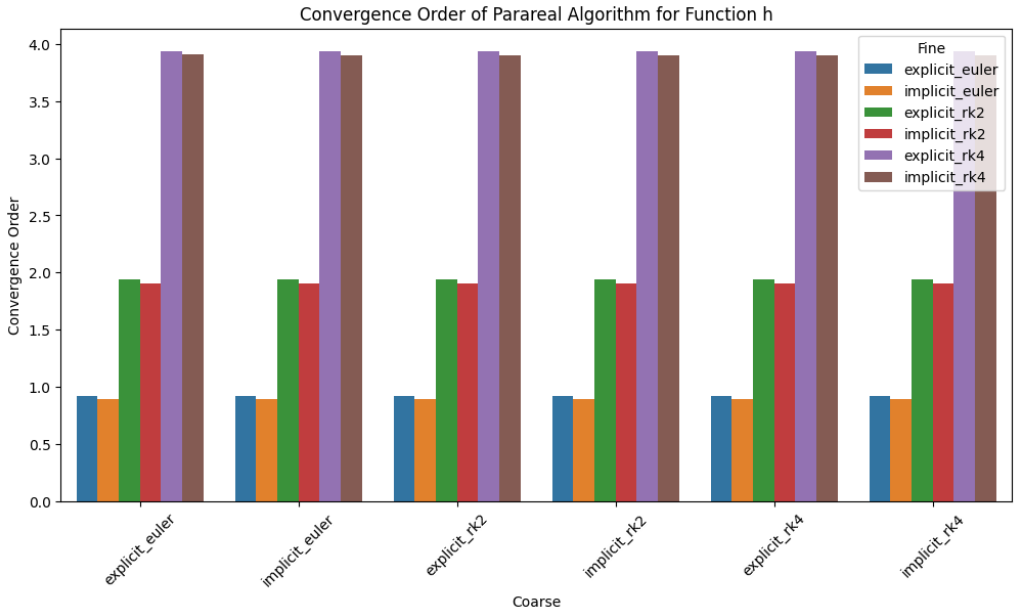

Figure 7: Convergence Order of Parareal Algorithm for system 3

Analysis:

- The order of convergence of the Parareal algorithm is primarily set by the fine solver used in the iterative process. This means that the accuracy and efficiency of the Parareal algorithm are directly influenced by the characteristics of the fine solver.
- Higher-order fine solvers lead to a higher order of convergence for the overall algorithm.

5.3 Performance Evaluation

The performance of the Parareal algorithm was evaluated using the Lorenz system to measure execution time and speedup achieved with parallel execution.

5.3.1 Execution Time and Speedup

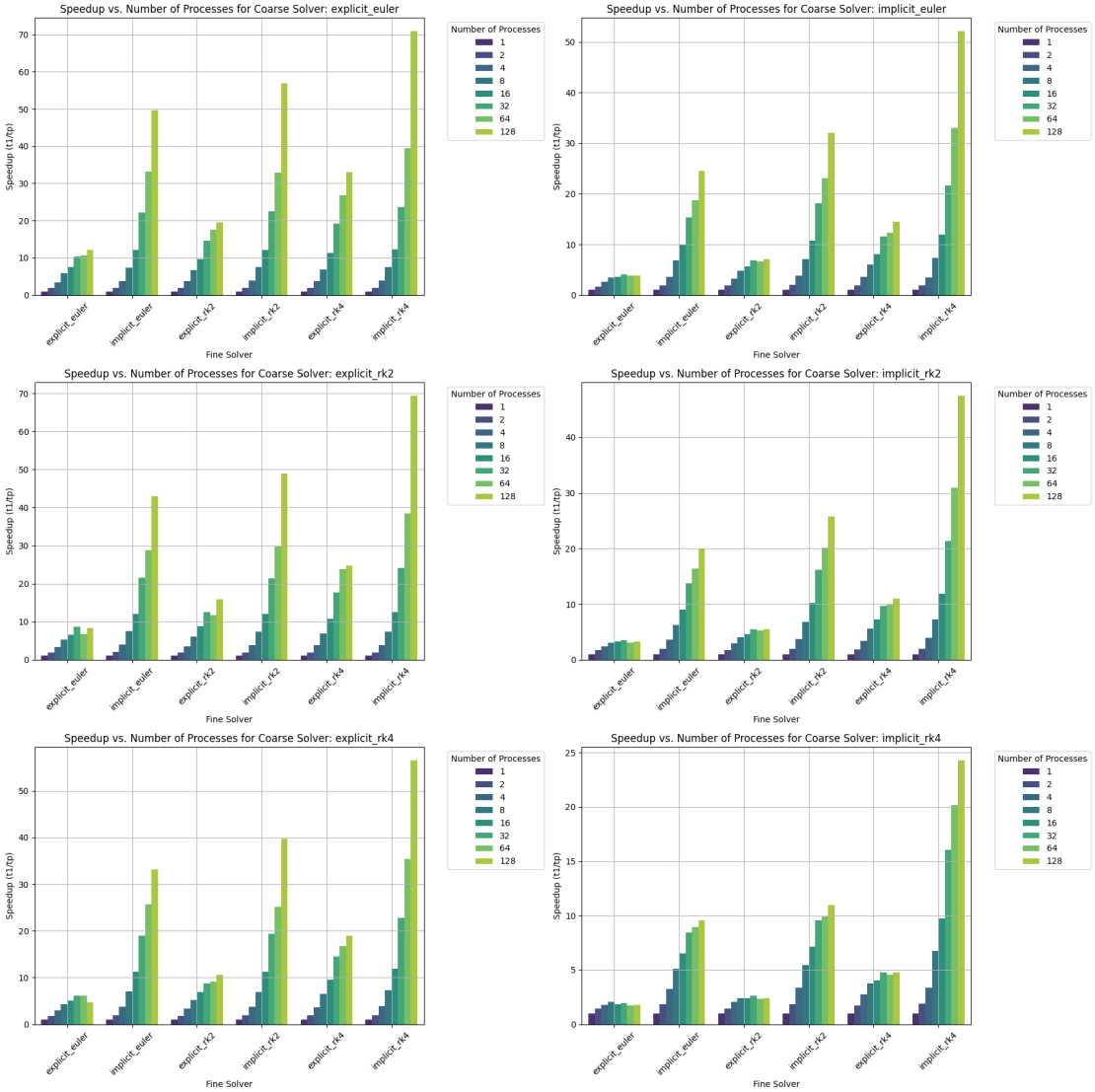

Figure 8: Speedup vs. Number of Processes for Different solvers combination

Analysis:

- The execution time decreased significantly as the number of processes increased, demonstrating the efficiency of parallel execution.
- The speedup achieved was nearly linear for a smaller number of processes but exhibited diminishing returns as the number of processes increased,

particularly for explicit solvers. This is likely due to communication overhead and the shorter execution time of these methods, which limits further speed increases.

6 Discussion

The results of this study offer valuable insights into the performance and efficiency of the Parareal algorithm when applied to various systems with different solver combinations. Key findings and their implications are discussed below:

6.1 Convergence Analysis

The convergence analysis revealed that the number of iterations required for the Parareal algorithm to reach the specified tolerance varies significantly based on the solver combination used.

Higher-order coarse solvers were shown to generally require fewer iterations for convergence compared to lower-order solvers. This indicates that the choice of coarse solver is crucial for optimizing the convergence rate of the Parareal algorithm. The efficiency gains from using higher-order coarse solvers, however, must be balanced against their increased computational complexity.

6.2 Order of Convergence

The analysis of the order of convergence demonstrated that the fine solver predominantly determines the Parareal algorithm's convergence order, as discussed in Gunnar A. Staff's article (2002) [5].

Higher-order fine solvers, such as RK4, resulted in a higher overall convergence order for the algorithm. This underlines the importance of selecting an appropriate fine solver to maximize the accuracy and efficiency of the Parareal algorithm. The results affirm that while higher-order solvers improve convergence properties, they also introduce greater computational demands. Therefore, the trade-off between accuracy and computational cost needs careful consideration when choosing solvers for the Parareal algorithm.

6.3 Performance Evaluation

Performance evaluation using the Lorenz system showed a significant decrease in execution time as the number of processes increased, indicating the efficiency of parallel execution.

The speedup achieved was nearly linear for a smaller number of processes but exhibited diminishing returns as the number of processes increased, particularly for explicit solvers. This is likely due to communication overhead and the shorter execution time of explicit methods, which limits further speed increases. These findings suggest that while the Parareal algorithm can achieve substantial computational savings through parallelization, the efficiency gains may be limited by communication costs as the number of processes increases. Optimizing the balance between computation and communication is essential for maximizing the performance of the Parareal algorithm in large-scale parallel environments.

7 Conclusion

This study, hosted by Cemosis, extended previous work on the Parareal algorithm by thoroughly investigating its convergence properties, order of convergence, and performance in a parallel computing environment. The following conclusions can be drawn from the research:

1. **Convergence Analysis:** The number of iterations required for convergence is significantly influenced by the choice of coarse solver. Higher-order coarse solvers generally lead to faster convergence. However, this improvement must be weighed against the increased computational demands of these solvers.

2. **Order of Convergence:** The fine solver dictates the overall convergence order of the Parareal algorithm. Higher-order fine solvers enhance accuracy and improve the convergence rate. This finding highlights the importance of selecting an appropriate fine solver to optimize the algorithm's performance.

3. **Performance Evaluation:** Parallel execution significantly reduces execution time, demonstrating near-linear speedup with a smaller number of processes. However, as the number of processes increases, the speedup shows diminishing returns, especially for explicit solvers, due to communication overhead and the shorter execution times of these methods. This underscores the need for balancing computation and communication to maximize the efficiency of the Parareal algorithm.

These findings underscore the importance of selecting appropriate solvers to optimize the performance and accuracy of the Parareal algorithm. Future work could further explore the application of adaptive solver strategies and investigate the algorithm's performance on more complex, real-world problems. Additionally, addressing communication overhead in large-scale parallel environments remains a crucial area for further research.

Overall, this study contributes to a deeper understanding of how different solvers impact the Parareal algorithm's efficiency and effectiveness, providing a foundation for optimizing parallel-in-time computations in scientific and engineering applications.

References

[1] 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.

[2] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of atmospheric sciences*, 20(2):130–141, 1963.

[3] Hadrien Batmalle, Pierre-Elliott Bécue, and Vincent Le Gallic. Accélération des méthodes de parallélisation en temps par approches de type quasi-newton. Technical report, Technical report, Ecole Nationnale Supérieure de Cachan, France, 2011.

[4] Endre Süli. Numerical solution of ordinary differential equations. *Mathematical Institute, University of Oxford*, 2010.

[5] Gunnar Staff. Convergence and stability of the parareal algorithm: A numerical and theoretical investigation. Technical report, SIS-2003-312, 2003.