An Investigation into Parareal

Abhijit Chowdhary
New York University

May 21, 2019

Contents

1 Introduction

The parareal algorithm was originally proposed by Lions, Maday, and Turinici [4] as a numerical method to solve ordinary differential equations that was parallel capable over the time domain. Here we intend to given a overview of the method, and it's properties. In section 2 we describe the parareal iteration, which later in 3 we discuss how to implement efficiently using the parallel library OpenMP. Then in 4 we discuss the parallel efficiency of this implementation, and measure the scalability of it in a high performance computing context. Then in 5 and 6 we examine the analytic properties of the method, deriving stability criteria and measuring and deriving theoretical convergence rates.

2 Parareal

We would like to solve the autonomous ordinary differential equation:

$$\begin{cases} u'(t) = f(u), \quad t \in [t_0, t_f] \\ u(t_0) = u_0 \end{cases}$$

where $f : \mathbb{R}^d \to \mathbb{R}^d$ and $u : \mathbb{R} \to \mathbb{R}^d$.

Parareal is an iterative scheme to approximate u for the above, which can be derived from the concept of *multiple shooting* methods. The idea behind these methods are to turn the problem into a nonlinear optimization problem, which then can be solved via Newton-Raphson. [3]

Take our time domain $[t_0, t_f]$ and partition it into N pieces $0 = t_0 < \cdots < t_N = t_f$. Looking specifically at the interval $[t_n, t_{n+1}]$, we pose the new ODE:

$$\begin{cases} u_n' = f(u_n), \quad t \in [t_n, t_{n+1}] \\ u(t_n) = U_n \end{cases}$$

where U_n is such that $U_n - u_{n-1}(t_n, U_{n-1}) = 0$, i.e. the initial condition satisfies the solution to the previous time slice. The conditions on U_i form a nonlinear system $F(U) = 0$, which we can approximate using Newton-Raphson, so we recieve the iteration $U^{k+1} = U^k - J_F^{-1}(U^k)F(U^k)$. As it turns out, as seen in Gander and Vandewalle [3], that we can actually change this into the form:

$$\begin{cases} U_0^{k+1} = u_0 \\ U_{n+1}^{k+1} = u_n(t_{n+1}, U_n^k) + \frac{\partial u_n}{\partial U_n}(t_{n+1}, U_n^k)(U_n^{k+1} - U_n^k) \end{cases}$$

If we were to then, call $u_n(t_{n+1}, U_n^k) = \mathcal{F}(t_{n+1}, t_n, U_n^k)$ where F is some near truth integrator and were to approximate the second term with another integrator $\mathcal{G}(t_{n+1}, t_n, U_n^{k+1}) - \mathcal{G}(t_{n+1}, t_n, U_n^k)$, then we would recieve the iteration:

$$\begin{cases} U_0^{k+1} = u_0 \\ U_{n+1}^{k+1} = \mathcal{F}(t_{n+1}, t_n, U_n^k) + \mathcal{G}(t_{n+1}, t_n, U_n^{k+1}) - \mathcal{G}(t_{n+1}, t_n, U_n^k) \end{cases}$$

This is precisely what we call parareal iteration. Alternatively, you can think about this as the predictor-corrector scheme:

$$\begin{cases} U_0^{k+1} = u_0 \\ U_{n+1}^{k+1} = \mathcal{G}(t_{n+1}, t_n, U_n^{k+1}) + \left(\mathcal{F}(t_{n+1}, t_n, U_n^k) - \mathcal{G}(t_{n+1}, t_n, U_n^k)\right) \end{cases}$$

Regardless, we notice that this iteration, which is an approximation to our solution as $k \to \infty$, has a term which is decoupled from the current iteration step, $\mathcal{F}$. Furthermore, it satisfies a *first same as last* property with the coarse integrators $\mathcal{G}$. The idea behind the parareal method is to take advantage of both of these properties.

3 Implementation

Here we discuss two main implementations of the Parareal algorithm using OpenMP.

3.1 Naive OpenMP

From the basic description of Parareal, we see that the fine propagator is able to be computed in parallel, so the main idea here is to parallelize the for loop corresponding to the fine propagator. See algorithm 1 for the pseudocode. A quick description is as follows:

(1) Lines $(1-2)$ describe the initial coarse approximation of the system. It's important that these solutions are loaded into both y_c and y, since they're needed for the next bit to satisfy the first same as last property.

(2) Lines $(4-8)$ are the computation of the fine approximation, and the construction of the corrector term δy. There are a few implicit assumptions here:

- We assume that $y(n)$ hold the previous λ_n^k.
- We assume that $y_c(n)$ already holds the value $\mathcal{G}(t_{n+1}, t_n, \lambda_n^k)$.

Both of these assumptions are true atleast for the first iteration due to our considerations in the first part. Furthermore, this part is completely dependent on previously computed information, and has no previous dependence on previous iterates, and therefore is embarrassingly parallel.

(3) Lines $(9-12)$ actually do the parareal iteration, we predict via line 10, and we correct via our previously computed $\delta y(n)$.

Algorithm 1 Naive Parallel Parareal Algorithm

Require: y_0 and coarse and fine solvers $\mathcal{G}$, $\mathcal{F}$.

1: $y_c \leftarrow \mathcal{G}(t_f, t_0, y_0)$. ▷ Coarsely approximate solution
2: $y \leftarrow y_c$.
3: **while** iter $<$ max_iter && not converged **do**
4: #pragma omp parallel for
5: **for** $n = 0 \to P$ **do**

6: $y_f(n) = \mathcal{F}(t_{n+1}, t_n, y(n))$.
7: $\delta y(n) = y_f(n) - y_c(n)$. ▷ corrector term. FSAL
8: **end for**
9: **for** $n = 0 \to P$ **do**
10: $y_c(n) = \mathcal{G}(t_{n+1}, t_n, y(n))$. ▷ Predict.
11: $y(n) = y_c(n) + \delta y(n)$. ▷ Correct.
12: **end for**
13: **end while**

3.2 Pipelined OpenMP

There's some clear problems with the naive implementation of Parareal. Primarily, let's examine the coarse computation portions, lines 1 and 10 of algorithm 1. Both of these portions iteratively compute the coarse approximation to the solution, but they lock up all processors while they do so. In fact, once we've computed the approximation for the nth node, we can immediately continue, instead of waiting for the rest of the serial computation to finish. This is referred to in literature (Ruprecht [6]) as *pipelining*. In fact, this would happen naturally in the MPI implementation, but since I had decided to use OpenMP, we have to tackle this problem.

The solution here is actually to mimic an MPI enviornment by wrapping the entire parareal algorithm in a parallel region. The idea is that thread p has the job of processing the fine and coarse operators, and later solution at position t_{p+1}. While we don't have the cost of MPI communication across processes, we still need to appropriately place locks around our reads and writes to prevent race conditions. A brief description of this new algorithm, whose pseudocode is provided in algorithm 2, follows:

(1) Line 1 encases the whole algorithm in a parallel region, essentially mimicing a MPI centric algorithm, except in shared memory.

(2) Lines $(2-8)$ are the initial coarse approximation of the solution. Note, that for the pth process, we actually integrate from $t_0 \to t_p$, in order to compute the coarse solution at t_{p+1}. We could instead have process 0 do the whole integration, and then communicate those values to all processes, but it was observed then that this would be no better than the serial computation in the naive algorithm. Note that the nowait statement allows to continue immediately, which is the *pipelining* that we desire.

(3) Lines $(12-16)$ describe the parallel computation of $\mathcal{F}$ and δy. Note, it's important to have locks around the points from which we read and write, since across this algorithm thread p reads from position p and writes into position $p+1$. A point of note is that if we could refine this algorithm so that it read and wrote into only position p, save for one location, it would vastly improve.

(4) Lines $(17-22)$ describe the predictor and corrector step of parareal, accompanied with the necessary locks. Note that this is surrounded in a ordered directive, which is necessary since to compute the predictor step for $p+1$, we need $y(p)$. Therefore, we have no choice but to do this iteratively. However, because of the nowait clause on line 10, we aren't forced to wait for the other threads to finish before continuing on.

Algorithm 2 Pipelined Parallel Parareal Algorithm (source Ruprecht [6])

Require: y_0 and coarse and fine solvers $\mathcal{G}$, $\mathcal{F}$.

#pragma omp parallel ▷ Enclose whole algorithm in parallel region
#pragma omp for nowait
for $p = 0 \to P$ **do**
  if $p \neq 0$ **then**
    $y(p) = \mathcal{G}(t_p, t_0, y_0)$. ▷ Repeated work across threads, but $\mathcal{G}$ is cheap.
  end if
  $y_c(p+1) = \mathcal{G}(t_{p+1}, t_p, y(p))$.
end for
while iter $<$ max_iter && not converged **do**
  #pragma omp for ordered nowait ▷ nowait critical to avoid waiting for threads $> p$
  for $p = 0 \to P$ **do**
    omp_set_lock(p) ▷ Read lock.
    temp $= \mathcal{F}(t_{p+1}, t_p, y(p))$.
    omp_unset_lock(p) and then omp_set_lock($p+1$) ▷ Write lock.
    $y_f(p+1) =$ temp, $\delta y(p+1) = y_f(p+1) - y_c(p+1)$.
    omp_unset_lock($p+1$)
    #pragma omp ordered ▷ ordered because $y(p)$ depends on $y(p-1)$.
      omp_set_lock(p) ▷ Read lock.
      temp $= \mathcal{G}(t_{p+1}, t_p, y(p))$.
      omp_unset_lock(p) and then omp_set_lock($p+1$) ▷ Write lock.
      $y_c(p+1) =$ temp, $y(p+1) = y_c(p+1) + \delta y(p+1)$.
      omp_unset_lock($p+1$)
  end for
end while

4 Efficiency Analysis

Here we try to analyze the speedup of Parareal analytically, and then through scalability studies conducted on the supercomputer Prince, here at NYU.

4.1 Theoretical Results

We try to directly estimate the speedup of the Naive OpenMP implementation. We denote our speedup S by: ([2] slide 17)

$$S = \frac{\text{Time taken by fine solver}}{\text{Parareal time}}$$

Let the time taken by the fine and coarse solvers respectively be denoted by T_f and T_g. Furthermore, suppose we have the optimal amount of processors P, such that $\Delta t = T/P$, where T is the time to integrate up to.

Considering this, each step parareal makes a predictor and corrector computation. The predictor is just a run of the coarse solver, costing PT_g. The corrector has two parts, the fine and coarse portion. The fine portion is parallelized optimally, and therefore costs just T_f, instead of PT_f. In addition, the coarse portion of the corrector takes advantage of the *first same as last* property, and doesn't have to be counted sepreately. Recall, also, before the parareal iteration, we make an initial guess using one coarse solve, PT_g. Therefore, for a parareal with k iterations it would have speedup:

$$S = \frac{PT_f}{PT_g + k(PT_g + T_f)} = \frac{1}{\frac{T_g}{T_f} + k(\frac{T_g}{T_f} + \frac{1}{P})} = \frac{1}{\frac{T_g}{T_f}(1+k) + \frac{k}{P}}$$

Notice, under the limit:

$$\lim_{P\to\infty} S = \frac{1}{\frac{T_g}{T_f}(1+k)} = \frac{T_f}{T_g(1+k)}$$

Supposing we have the idealized case of $k = 1$, then we would find that this would result in speedup $\frac{T_f}{2T_g}$, which, given that $T_g << T_f$, could be significant. However, in practice, k has to be taken to be large enough so that S isn't too great. This is one of the first indicators we have of why $T_g << T_f$ must be true.

4.2 Scalability

Since this algorithm is designed to take our numerical ODE techniques to a high performance computing context, we would like to perform the standard scalability tests using this algorithm, measuring how it scales to massively parallel hardware.

All tests are performed on a single Prince node, having requested 28 processors. We would be worried about other processes running on this node if this problem was memory bound, however since the fine computation is performed in place and the coarse operator has few points, the computation is compute bound.

4.2.1 Strong Scaling Study

A strong scaling study consists of fixing a total problem size, and increasing the number of processors necessary. See figure 1 for our performed analysis. Note that figure 1 presents a very interesting, and dismal, phenomena. Notice how from processors $14 \to 27$ the speedup recieved is negligable from the previous. Why does this occur? Recall that the optimal number of processors for parareal iteration is the number of coarse steps, so that we can process all of the $\mathcal{F}$ integrators in one step. However, supposing we have one less than optimal, the entire compute time is bounded by that one thread that has to process two $\mathcal{F}$ integrators. This is troublesome. In fact, suppose we desire to cut our time domain $[0, T]$ into $2N$ pieces. Then any computation with number of processors P taking value $N \to 2N - 1$ will have the same computation time, theoretically.

Notice, however, that the general speedup does follow the theoretical linear speedup we hope to see with P processors, which is great.

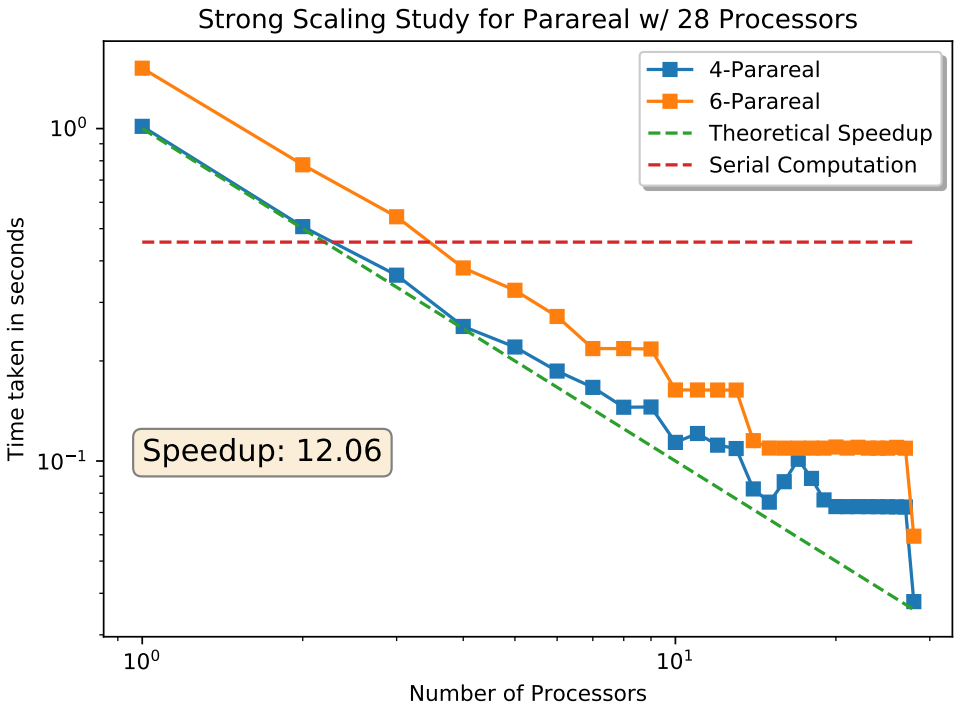

Figure 1: This is a strong scaling study performed on the ODE $u' = u, u(0) = 1$ integrating on time scale $[0, 4]$. Here we divided our time scale into pieces of size $4/P$, and then ran Parareal restricting the number of processors. Notice how the time taken doesn't improve from $14 \to 28$.

4.2.2 Weak Scaling Study

The weak scaling study examines how the runtime increases as we increase the amount of processors, working on a similar sized problem. See figure 2 for the results. In total, we see that the weak scaling of Parareal is very poor. This is likely by Amadahl's law. When we increase the amount of processors working, what happens is that the fine portion is computed in the same amount of time, but the amount of points in general increase. This implies that the serial portion of the computation will increase linearly in the number of points, which is exactly what we see in the figure. This implies that this problem scales poorly in the number of coarse points chosen. Ideally we would like to choose this to be as small as possible, i.e. we want the Δt of $\mathcal{G}$ to be large.

5 Stability Analysis

With regards to the stability analysis, we aim to try and prove some results on the "stability function" $R(z)$ as seen in Levecue's [5] chapter five through eight by following the arguments in Staff [7].

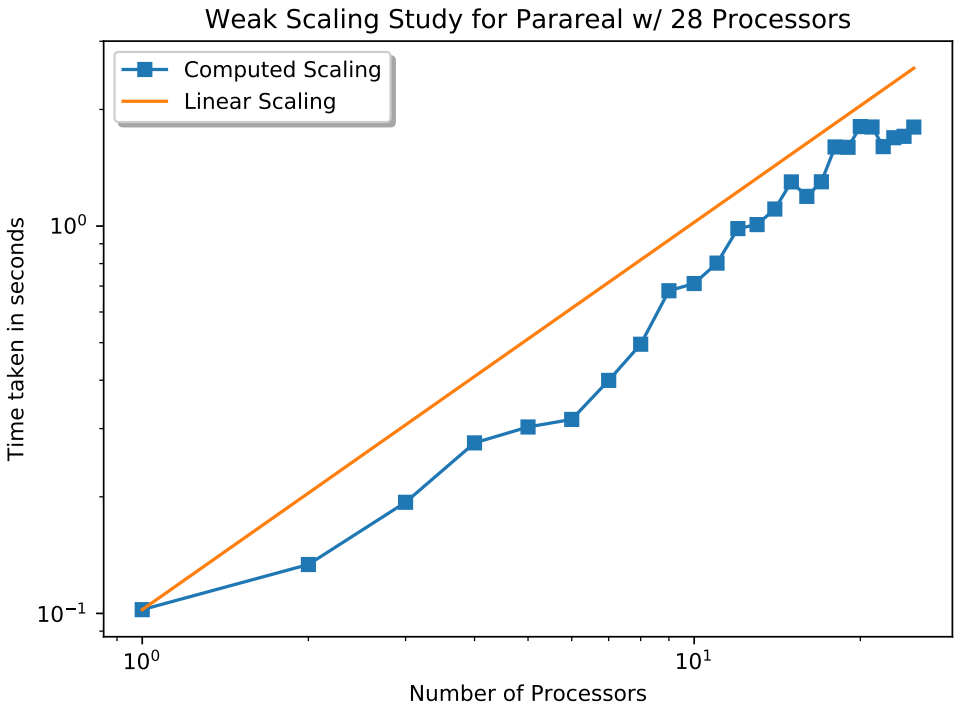

Figure 2

Suppose we have the following ordinary differential equation:

$$\begin{cases} u' = \mu u, \quad \mu < 0, t > 0 \\ u(0) = u_0 \end{cases} \tag{1}$$

We would like to analyze the stability of parareal on this system with a coarse operator $\mathcal{G}$ and a fine operator $\mathcal{F}$.

5.1 Deriving a Stability Function

With respect to Parareal, let's try and write our iteration in the form $\lambda_n^k = H(n,k)\lambda_0$. Suppose that we have some integrators $\mathcal{G}$ and $\mathcal{F}$, such that they're actually the same method, but with different time scales. In addition, suppoes that they're iteration can be described by they're own stability function $R(z)$. Then our parareal iteration goes to:

$$\begin{aligned} \lambda_{n+1}^{k+1} &= \mathcal{G}(t^{n+1}, t^n, \lambda_n^{k+1}) + \mathcal{F}(t^{n+1}, t^n, \lambda_n^k) - \mathcal{G}(t^{n+1}, t^n, \lambda_n^k) \\ &= R(\mu\Delta t)\lambda_n^{k+1} + R(\mu\delta t)^s \lambda_n^k - R(\mu\Delta t)\lambda_n^k \end{aligned}$$

where we say $s = \Delta t/\delta t$, i.e. how many fine steps are needed to make one coarse step. Combining like terms results in:

$$\lambda_{n+1}^{k+1} = R(\mu\Delta t)\lambda_n^{k+1} + \left(R(\mu\delta t)^s - R(\mu\Delta t)\right)\lambda_n^k$$

Now, if we were to note the terms on λ, we note that we have something very similar to the recurrence relation on combinations $\binom{n}{k} = \binom{n}{k-1} + \binom{n-1}{k-1}$. Exploiting that relationship, we can unroll our recursion into:

$$\lambda_{n+1}^{k+1} = \left(\sum_{i=0}^{k} \binom{n}{i} \left[R(\mu\delta t)^s - R(\mu\Delta t) \right]^i R(\mu\Delta t)^{n-i} \right) \lambda_0 = H(\mu, n, k, \delta t, \Delta t)\lambda_0$$

We would like to see when it's true that $|H| \leq 1$. As it turns out, this function is very difficult to analyze by itself, none of my plots of H by itself generated useful stability regions. Instead we try to analyze this under a few easy cases, and see exactly how parareal transforms the stability region of known temporal integrators.

5.2 Trivial case: $k = 0$

First, we make a sanity check, suppose we make no Parareal iterations. We would assume that in this case that the stability function of Parareal should devolve to the stability function of $\mathcal{G}$, since that's the only thing computed. Indeed:

$$\begin{aligned} H &= \left(\sum_{i=0}^{0} \binom{n}{i} \left[R(\mu\delta t)^s - R(\mu\Delta t) \right]^i R(\mu\Delta t)^{n-i} \right) \lambda_0 \\ &= \binom{n}{0} \left[R(\mu\delta t)^s - R(\mu\Delta t) \right]^0 R(\mu\Delta t)^n \lambda_0 \\ &= R(\mu\Delta t)^n \lambda_0 \end{aligned}$$

which is precisely the stability function of $\mathcal{G}$.

5.3 Trivial case $\mathcal{G} = \mathcal{F}$.

Suppose we were to choose $\mathcal{G} = \mathcal{F}$, time scales and all. This implies that $s = 1$. We would expect that in this case, our parareal algorithm should result in the stability region of $\mathcal{G}$, since the corrector step should result in a correction of 0. Indeed we see that all parareal iterations with $k > 0$ are canceled to zero by the term $[R(\mu\delta t)^s - R(\mu\Delta t)]$. Otherwise we're just left with $\binom{n}{0} R(\mu\Delta t)^n \lambda_0 = R(\mu\Delta t)^n \lambda_0$, which is exactly the stability region of $\mathcal{G}$, as expected.

5.4 Parareal Under Explicit Euler

The most iconic temporal integrator is the Euler methods, and we would like to see how Parareal changes they're stability regions. Suppose now that we are using explicit Euler for our coarse and fine integrators. Recall, that this implies that the stability function for our integrators $\mathcal{G}$ and $\mathcal{F}$ are $(1 + z)$ and $(1 + z)^s$ respectively, and remember that original the region of stability for explicit Euler on the given ODE is for z in the unit disk centered around -1.

Suppose $\mathcal{F} = \mathcal{G}$ but with the time scale squared, i.e. $\delta t = \Delta t^2$. So for example, if $\mathcal{G}$ has a time step of $1/10$, then $\mathcal{F}$ has a time step of $1/100$. Then, $s = \frac{\Delta t}{\delta t} = \Delta t^{-1}$. As an example, suppose that $\Delta t = 1/2$. Then $s = 2$, and (let $z = \Delta t$) our stability region goes to:

$$\begin{aligned} H &= \sum_{i=0}^{k} \binom{n}{i} \left[R(\mu\delta t)^s - R(\mu\Delta t)\right]^i R(\mu\Delta t)^{n-i} \\ &= \sum_{i=0}^{k} \binom{n}{i} \left[(1+\mu z^2)^2 - (1+\mu z)\right]^i (1+\mu z)^{n-i} \\ &= \sum_{i=0}^{k} \binom{n}{i} \left[\mu^2 z^4 + 2\mu z^2 - \mu z\right]^i (1+\mu z)^{n-i} \\ &= \sum_{i=0}^{k} \binom{n}{i} \left[\mu z(\mu z^3 + 2z - 1)\right]^i (1+\mu z)^{n-i} \end{aligned}$$

At this point the roots of the cubic are ugly, so I don't take the general case any further. Now we examine interesting cases under k. Recall, by arguments made above, we already know that $k = 0 \implies$ that we just have stability region $(1 + \mu z)$. For the following analysis, suppose we take $n = 2$ too.

5.4.1 One Parareal Iteration: $k = 1$

Suppose we only make one parareal iteration. Then the above goes to:

$$(1+\mu z)^2 + 2\mu z(\mu z^3 + 2z - 1)(1+\mu z) = (1+\mu z)(2\mu^2 z^4 + 4\mu z^2 - \mu z + 1)$$

5.4.2 Two Parareal Iteration: $k = 2$

Suppose we make two parareal iteration. This is a little more painful to compute:

$$(1+\mu z)^2 + 2\mu z(\mu z^3 + 2z - 1)(1+\mu z) + \mu z(\mu z^3 + 2z - 1) = (\mu z^2 + 1)^4$$

So the stability region $(1 + \mu z)^2$ transformed to $(1 + \mu z^2)^4$ under two parareal iteration, which is exactly the same region.

6 Convergence Analysis

Now down to the main point, we would like to analyze and confirm the convergence of the Parareal method. First we derive a theoretical result, and then confirm it through numerical expirements.

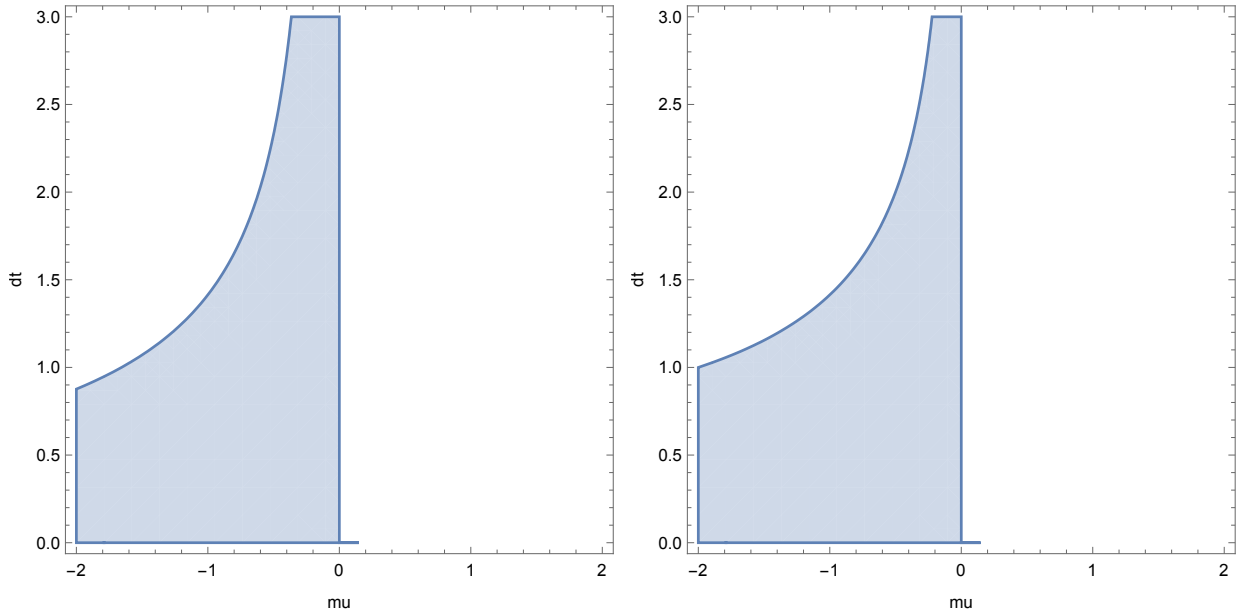

Figure 3: On the left we have the stability region for $k = 1$, and on the right we have it for $k = 2$. It's a little bit difficult to see, but for $k = 1$, the stability region is a bit smaller in Δt, but larger in μ.

6.1 Theoretical Convergence

Suppose we have the following ordinary differential equation:

$$\begin{cases} u' = f(t, u), & t > 0 \\ u(0) = u_0 \end{cases} \tag{2}$$

In addition suppose we have the coarse operator $\mathcal{G}(t^{n+1}, t^n, u^n)$ and the fine operator $\mathcal{F}(t^{n+1}, t^n, u^n)$ with the following properties:

1. On $\mathcal{G}(t^{n+1}, t^n, u^n)$:
   - Suppose this operator has order m.
   - Suppose it's Lipschitz in the initial condition:

     $$\left\|\mathcal{G}(t^{n+1}, t^n, u) - \mathcal{G}(t^{n+1}, t^n, v)\right\| \leq C\|u - v\|$$

     In particular we write $C = (1 + L\Delta t)$.
2. With respect to $\mathcal{F}(t^{n+1}, t^n, u^n)$, we suppose it's accurate enough to be assumed to be the true solution u^*. This means that if $\mathcal{G}$ is accurate with order m to the true solution, then it too will be so to $\mathcal{F}$.

Then we can prove the following theorem:

Theorem. *The parareal method with coarse operator $\mathcal{G}$ and fine operator $\mathcal{F}$ has order of accuracy mk, where $k - 1$ is the number of parareal iterations made. [1] [2]*

Proof. We proceed via induction on k and n. Suppose $k = 1$, then it is trivial, this is the coarse operator, and for $n = 0$, this is the initial condition which we know to any accuracy.

Now suppose for $k, n > 1$, that we know:

$$\left\|u(t^n) - u_k^n\right\| \leq \|u_0\| \, C(\Delta t)^{mk},$$

We want to show that:

$$\left\|u(t^n) - u_{k+1}^n\right\| \leq \|u_0\| \, C(\Delta t)^{m(k+1)}$$

To proceed, recall that $\mathcal{F}$ is assumed to be a good approximation for $u(t^n)$, so we may write:

$$\begin{aligned}
\left\|u(t^n) - u_{k+1}^n\right\| &= \left\|\mathcal{F}(u(t^{n-1})) - \mathcal{G}(u_{k+1}^{n-1}) - \mathcal{F}(u_k^{n-1}) + \mathcal{G}(u_k^{n-1})\right\| \\
&= \left\|\mathcal{G}(u(t^{n-1})) + \delta\mathcal{G}(u(t^{n-1})) - \mathcal{G}(u_{k+1}^{n-1}) - \delta\mathcal{G}(u_k^{n-1})\right\| \\
&\leq \left\|\mathcal{G}(u(t^{n-1})) - \mathcal{G}(u_{k+1}^{n-1})\right\| + \left\|\delta\mathcal{G}(u(t^{n-1})) - \delta\mathcal{G}(u_k^{n-1})\right\| \\
&\leq (1 + L\Delta t)\left\|u(t^{n-1}) - u_{k+1}^{n-1}\right\| + C(\Delta t)^{m+1}\left\|u(t^{n-1}) - u_k^{n-1}\right\| \\
&\leq (1 + L\Delta t)\left\|u(t^{n-1}) - u_{k+1}^{n-1}\right\| + C(\Delta t)^{m+1}(\Delta t)^{mk}\|u_0\| \\
&\leq (1 + L\Delta t)\left\|u(t^{n-1}) - u_{k+1}^{n-1}\right\| + C(\Delta t)^{m(k+1)+1}\|u_0\|
\end{aligned}$$

At this point, note that the left hand term is the approximation of u at the previous time step, which we can assume to also be of the order $m(k+1)$. Therefore, we can say that $\left\|u(t^n) - u_{k+1}^n\right\| = \mathcal{O}(\Delta t^{m(k+1)})$, completing our inductive step. □

6.2 Numerical Results and Validation

Here we seek to confirm the theoretical order derived in the last section (from [1] [2]). To see a clear example, consider the Euler methods. Recall that the explicit Euler method is of order 1, so theoretically if we have a fine method of high enough order, we should see a method whose order is k, for an order k Parareal. And indeed, as we expect, we do indeed see such such results, see figure 4 for the numerical plots.

Using this information, we might begin to consider how can we leverage this mk convergence rate. It's immediately clear, that if we want k to be as small as possible (as noted in the efficiency section), that we want m to be as large as possible. However, m corresponds to the order of the coarse integrator, and typically we want this integrator to be as cheap and inexpensive as possible. Therefore, we have an optimization problem here between the coarse integrator and number of parareal iterations. Of coarse, all of this is assuming that the fine integrator performs to the desired accuracy, otherwise the former is useless.

For example, see figure 5 for a examination of the errors of parareal underneath the Runge-Kutta methods. In addition, see figure 6 for a more confirmation of the mk order, for the Runge-Kutta methods.

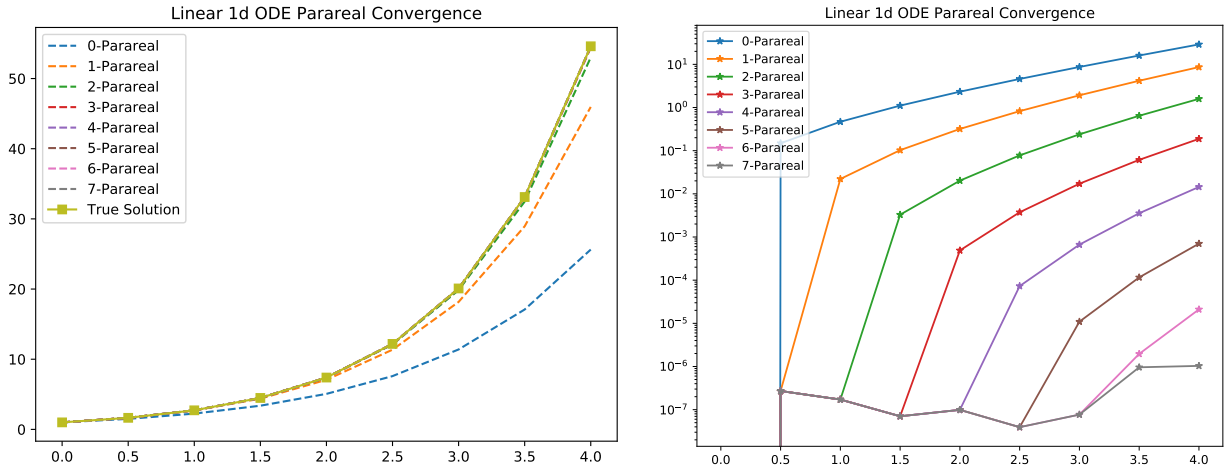

Figure 4: The figure on the left is showing the resulting solutions to the ODE $u' = u, u(0) = 1$ for $t \in [0, 4]$, and the right is showing the errors under the 1 norm. Critically, this plot reveals that our implementation of Parareal conforms to the theoreical convergence rate, which validates our method. The last iterate has strange behavior, but this is because I could not decrease the accuracy of the fine integrator (forward Euler) anymore without running out of memory.

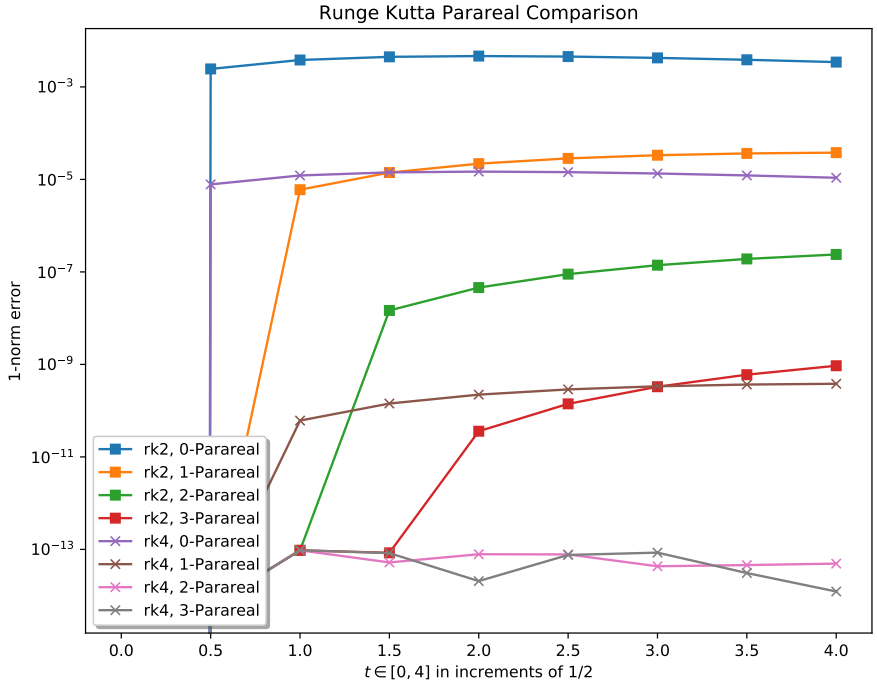

Figure 5: This figure is showing the resulting errors in the parareal approximation to to the ODE $u' = -\frac{1}{2}u, u(0) = 1$ for $t \in [0, 4]$ under the 1 norm. This plot reveals how, given a higher order coarse method, the parareal is able to make larger jumps of accuracy as the number of parareal iterations increases.

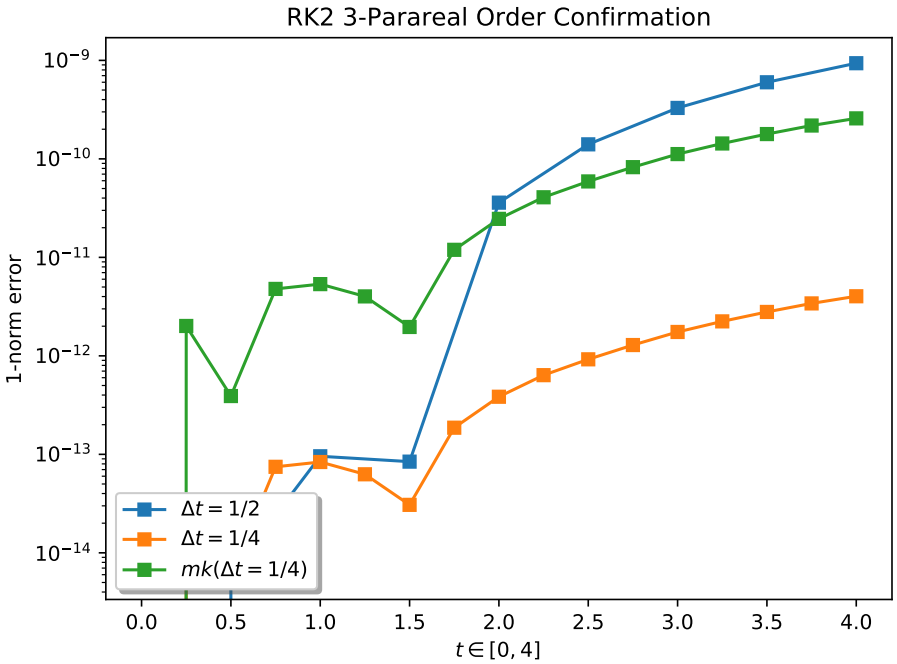

Figure 6: This figure shows that how, under refinement of Δt, the third parareal iteration of the Runge Kutta 2 method has order roughly 5. This is confirmed by taking the error of $\Delta t/2$ and shifting it by the order. They don't exactly land on top of each other, but I believe that to be an artifact of the approximation not being exact.

7 Conclusion

In conclusion, we find that Parareal is indeed capable of providing significant speedup to our general numerical methods for ordinary differential equations. However, as we scale our hardware larger and larger, we might run into runtime issues, due to the poor weak scaling of the algorithm. Furthermore, it's difficult to determine what's the correct choice of $\mathcal{F}$ and $\mathcal{G}$, since the stability function is not easy to analyze in general, and because there's an unfomfortable optimization problem between wanting $\mathcal{G}$ to be as accurate as possible, while making sure that the ratio T_f/T_g is as large as possible.

Some points to personally reflect on this project:

- I should not have used OpenMP to implement parareal. In the end, I was limited in the number of processors I could test on due to it, and the pipelined efficient implementation was just mimicing MPI-like behavior anyway. If I had to do this again, I would use a combination of MPI and OpenMP, MPI would split the domain over different compute nodes, and OpenMP would internally handle the parallel computation of the fine integrations assigned to that node.

- I ran out of time to test implicit integrators, and how their stability regions functioned. I found it difficult to implement them, since the way I had written my code isolated

the actual f of the ODE $u' = f(t, u)$, and thus if I wanted implicit solvers, I had to solve a nonlinear optimization problem, which was too much trouble.

- I wanted to look at generalizations and improvements of parareal, such as PFASST, which takes a spectral deffered corrections approach to the iteration, and another which adapts it to a multigrid scheme, but I ran out of time.

References

[1] Bal. *On the Convergence and the Stability of the Parareal Algorithm to solve Partial Differential Equations*. Columbia University, APAM

[2] Fields. *Parareal Methods*. http://www.cfm.brown.edu/people/jansh/page5/page10/page40/assets/Field_Talk.pdf

[3] Gander and Vandewalle. *Analysis of the Parareal Time-Parallel Time-Integration method* https://doi.org/10.1137/05064607X

[4] Lions, Maday, and Turinici. *A parareal in time procedure for the control of partial differential equations*. https://doi.org/10.1016/S1631-073X(02)02467-6

[5] Levecue. *Finite Difference Methods for Ordinary and Partial Differential Equations*. https://epubs.siam.org/doi/abs/10.1137/1.9780898717839.fm.

[6] Ruprecht. *A shared memory implementation of pipelined Parareal*. DOI:10.1007/978-3-319-64203-1_48.

[7] Staff and Rønquist. *Stability of the Parareal Algorithm*. Norweigian University of Science and Technology, Department of Mathematical Sciences.