A parareal-based parallel-in-time method for explicit time-marching schemes

Yen-Chen Chen[1,a)] Kengo Nakajima[1,b)]

Abstract: Renowned Parallel-in-Space/Time (PinST) methods such as the parareal method and the MGRIT method are compelling with implicit time-marching schemes. However, PinST methods with explicit time-marching have been challenging due to the restriction of the Courant-Friedrichs-Lewy (CFL) condition. Moreover, explicit schemes are highly scalable in the space dimension, which is more efficient than existing PinST methods. In this research, we propose a method based on the parareal algorithm, which is highly scalable and complies with the CFL condition. We also demonstrate the performance utilizing two explicit time-marching models.

Keywords: Parallel-in-time, PinT, PinST, Multigrid, parareal, PDE, explicit scheme

1. Introduction

Parallel-in-time methods have been a hot topic in recent years. As the performance of supercomputers grows exponentially, spatial parallelization is reaching its limit and is no longer enough. Thus, researchers turn to the time dimension for more parallelization. Renowned methods such as the parareal method[2] and the MGRIT method[1] has been used on various of applications such as heat transfer and parabolic model. In recent studies[5], even hyperbolic equations such as one-dimensional advection have been accelerated with the MGRIT method.

Despite the active researches in parallel-in-time method, not many methods for direct, explicit time-marching schemes are proposed. Since explicit schemes are not time-dependent, one can efficiently distribute computing nodes into separate processors, and boundary values are exchanged through communication. Thus, explicit schemes are highly scalable in the spatial dimension, and we usually prefer parallel-in-space than parallel-in-time when it comes to parallelizing explicit schemes, however, for applications such as weather simulation, which has an enormous amount of time steps, the performance of spatial parallelization is limited, and the bottleneck becomes the time dimension. Therefore parallel-in-time methods for explicit schemes could still be useful for such applications with many time steps, which is the target of this paper.

In Section 2, we will first give a short overview of the parareal method, which is the base algorithm of our proposed method. Then, in Section 3, we will talk about what are the challenges for developing a parallel-in-time and then introduce the proposed multilevel parareal method in Section 3.2. In Section 4 we will analyze the result and performance of the proposed method and finally in Section 5 and 6 the conclusion and future work.

[1] The University of Tokyo, Bunkyo-ku, Tokyo 133–8656, Japan
[a)] yenchen_chen@mist.i.u-tokyo.ac.jp
[b)] nakajima@cc.u-tokyo.ac.jp

2. The Parareal Method[2]

The parareal method[2] is an iterative two-level parallel-in-time method. The parareal method is composed of a high-precision solver and a low precision solver. First, the whole timeline is divided into segments by the number of processors. The main concept of the parareal method is to update the difference between high and low precision solver in each segment with a sequential low precision solver. After sufficient iterations, the result at the last time step would converge to the high precision result.

Before iteration, we perform a sequential solve with the low precision solver to get an initial value.

$$y^0_{j+1} = \mathcal{G}(y^0_j, t_j, t_{j+1}) \quad \forall j = 0, \ldots, P$$

From this initial value, we then perform iterative parareal algorithm until the result converges to the high precision result. In each iteration, we first compute results by both high and low precision in each time segment.

$$y^k_{f,j} = \mathcal{F}(y^k_j, t_j, t_{j+1}), \quad y^k_{c,j} = \mathcal{G}(y^k_j, t_j, t_{j+1})$$

Then, we use the low precision solver to pass down the difference of the high and low solver sequentially to the last time step.

$$y^{k+1}_{j+1} = \mathcal{G}(y^{k+1}_j, t_j, t_{j+1}) + \mathcal{F}(y^k_j, t_j, t_{j+1}) - \mathcal{G}(y^k_j, t_j, t_{j+1})$$

Figure 1 shows a simple example of solving a parabola line by the parareal method. Assume that eight processors are available, the timeline is first divided into eight segments.

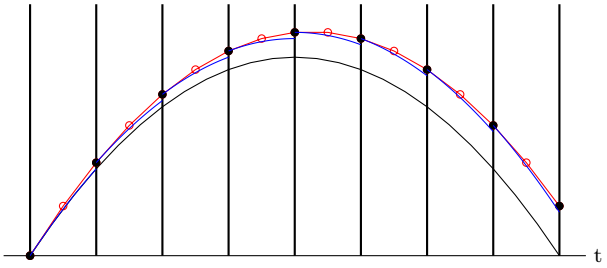


Fig. 1: Illustration for solving a parabola line by the parareal method.

In each processor, we solve the results with a high precision solver and a low precision solver. In each time segment, the red line shows the results of the low precision, and the blue line shows that of the high precision solver. Then, we update the differences between the two results in each processor from the first processor to the last one. Repeat the process iteratively, and we would get the desired parabola result as the black line.

As we could see from the algorithm, the parareal method works faster if the low precision solver is a lot cheaper than the high precision solver. Given a different application, the iteration number until convergence would also widely differ. Linear equations converge with a single iteration. However, non-linear equations such as hyperbolic equations are proven to converge slow using the parareal algorithm[7]. The iteration number required to converge for the parareal method changes according to the number of processors. As the number of processors grows, the iteration number also grows. Thus, depending on the application, the parareal method could be very costly with many processors.

3. Proposed Method

We propose a multilevel parareal method that parallelizes explicit schemes in the time dimension. In Section 4, we show the performance of the proposed algorithm with a one-dimensional advection problem.

3.1 Challenges

Despite various developments of parallel-in-time methods[1], [2], there is not yet a efficient one for explicit schemes. Explicit time-marching schemes are very efficient ways to solve PDE problems. For specific applications, we tend to use explicit schemes rather than implicit ones. However, since explicit schemes are highly scalable in the spatial dimension, there is not yet an efficient parallel-in-time method that could reach better or even similar scalability at the parallelization in the time dimension. Due to its high scalability, there are not many pieces of research on parallel-in-time methods for explicit time-marching schemes either. One goal of the research is to develop a parallel-in-time method which provides greater scalability in the time dimension parallelization.

While using explicit schemes, one has to satisfy the Courant-Friedrichs-Lewy (CFL) condition[3]. The CFL condition is the necessary condition for convergence. For example, the CFL condition for one-dimensional problems has the following form:

$$C = \frac{u\,\Delta t}{\Delta x} \leq C_{\max}$$

where C is called the Courant number, u is the wave velocity, Δx is the mesh size, and Δt is the time step. For explicit schemes, $C_{\max}$ is typically 1.

As most parallel-in-time methods involve coarse time grids, coarser time step Δt directly leads to larger Courant number. Thus, direct coarsening in the time step would result in a convergence problem. A direct solution is to coarsen the x mesh-size at the same time, which is also what we are going to use. However, the coarser x grid also leads to less accuracy in the coarse grid and therefore affects the convergence speed. This is the second challenge, and trade-off we have to overcome.

At last, we choose a hyperbolic equation, the advection problem, as our numerical experiment. Hyperbolic equations are known to have convergence trouble with parallel-in-time problems[7], which is our third challenge.

3.2 A Multilevel Parareal Method

We propose a multilevel parareal method, which is a parallel-in-time method for explicit time-marching schemes. Similar to the parareal method, we also divide the timeline into segments by the number of processors. Each processor is in charge of a time segment. We then further construct multiple levels of coarser time grids. In the coarse time grids, to prevent the violation of the CFL-condition, we coarsen the space grid at the same time. For a one-dimension explicit scheme, as long as we coarsen the time grid and the space grid in the same ratio, the Courant number stays the same, and therefore the algorithm does not violate the CFL-condition on each coarse level.

$$C = \frac{v\Delta t}{\Delta x}$$
$$C' = \frac{v(r\Delta t)}{r\Delta x} = \frac{v\Delta t}{\Delta x} = C$$

In order to move results between levels, we have to define restriction and prolongation operations. Since only want the result of the last time step. As long as the last time step is included in every level, we do not have to define restriction and prolongation for the time dimension. However, since we are also coarsening the space grid, and we want the result

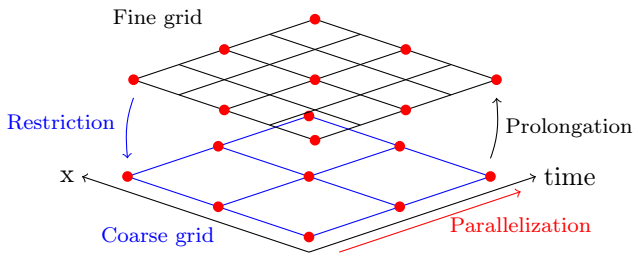


Fig. 2: Out proposed method extract coarse grid by coarsening both the time grid and the x grid.

on the whole space grid, we have to define restriction and prolongation for the space dimension. Here we assume that the coarsen ratio between two levels is 2 for each level.

Assume that y_i represents the point results on a specific time step. We restrict it to a coarser level with a restriction operator. Here, we take values from every other point.

$$y'_i = y_{2i-1}$$

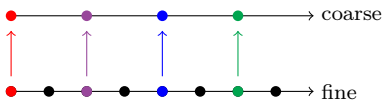


Fig. 3: Restriction takes coarse grid values direct from the fine grid.

To move values from a coarse level to a fine level, we define the following prolongation operator. The center point, which is not on the coarse grid, is updated by the average update value of its neighbor points.

$$y'_i = y_i + \frac{(y'_{i-1} - y_{i-1}) + (y'_{i+1} - y_{i+1})}{2}$$

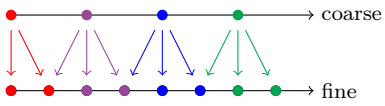


Fig. 4: Prolongation at fine points takes average update value from tow neighbor coarse points.

We choose a non-linear prolongation method here since linear interpolation has been shown to provide weak convergence in previous research[6].

Figure 2 shows the multilevel structure with two levels. We first use restriction

The main idea of the proposed multilevel parareal method is to start from the coarsest level and performance the parareal algorithm to get an initial for the next level such that with smaller computation, we could get a better initial value and therefore reduce iteration number. The algorithm is shown in the following Algorithm 1.

Algorithm 1: Multilevel parareal (MPI parallelized)

Explicit time-marching on the coarsest level L.
for *level l = L-1 to 1* **do**
  for *iterate until residual tolerance* **do**
    On current processor:
    Solve on the current level
    $y_f = \mathcal{F}_l(y_j, t_j, t_{j+1})$.
    Solve on the coarsest level $y_c = \mathcal{G}(y_j, t_j, t_{j+1})$.
    for *Processor p = 1 to P* **do**
      Solve on the coarsest level
      $y_{j+1}^{k+1} = \mathcal{G}(y_j^{k+1}, t_j, t_{j+1}) +$
      $\mathcal{F}_l(y_j^k, t_j, t_{j+1}) - \mathcal{G}(y_j^k, t_j, t_{j+1})$
      Update values to level l with prolongation
      $y'_i = y_i + \frac{(y'_{i-1} - y_{i-1}) + (y'_{i+1} - y_{i+1})}{2}$
    end
  end
end

Similar to the parareal method, we start by computing an initial value on the coarsest level. Then, we start from coarse to fine, use explicit solver on each level as the high precision solver compare to the coarsest grid as the low precision solver, and perform the parareal method until convergence. Then, we use the results as the initial values for the next level. Different from the MGRIT method, we have separate iteration numbers on each level, and we move to a finer level only when the convergence criteria is reached.

In order to reduce the number of computations, we want to reduce the iteration numbers at finer levels as much as possible. Thus, we set smaller tolerance (stricter criteria) in coarser levels.

3.3 Computation and Communication Costs

Before moving into the numerical result, we would like first to analyze the computation and communication costs of our method compared to the spatial parallelization method. The following computation cost is the parallelized computation cost. In the parallel part, only computation cost on one processor would be counted, and in the sequential part, all computation costs will be computed.

Assume that we are using the Lax-Wendroff method as the target explicit time-marching scheme. For simplification, we say that the full computation cost of a sequential Lax-Wendroff method is $O(LW)$. Also, assume that we have P processors, and we make full use of them, the number of space points is N, and the number of time steps is T. We then note the iteration number of the multilevel parareal method in each level as m_l, $l = L-1, \ldots, 0$ from coarse to fine. The computation cost for the spatial parallelization is

$$\frac{1}{P} O(LW)$$

and the computation cost for the multilevel parareal method is

$$\frac{1}{P}(m_{L-1} \times \frac{1}{2^{n-1}} O(LW) + \ldots + m_0 \times O(LW))$$

$$+\frac{1}{2^L}O(LW) \times (1 + m_{L-1} + \ldots + m_0)$$

We can easily see that the multilevel parareal method has a much higher computation complexity than straightforward spatial parallelization. However, if we could reduce the iteration in the fine levels, especially m_0, then the computation complexity would not differ so much. As we can see in the numerical result at Section 4, with our method m_0 iteration number is typically 1 with enough levels.

Similarly, we could also estimate the communication cost of both the spatial parallelization method and the multilevel parareal method. Lax-Wendroff only takes values of itself and its neighbor points from the previous time step. Thus, only two values are exchanged between each neighbor processor. The communication cost is as the following:

$$2(P-1) \times T$$

For the multilevel parareal method, for each iteration, and between each processor, we pass a whole space grid of the current level to the next processor. The communication cost is the following:

$$(m_0\frac{1}{2}N + m_1\frac{1}{2^2}N + \ldots + m_{L-1}\frac{1}{2^n}N) \times (P-1)$$

For a generally large one-dimensional problem with similar number of space grid and time grid, we could write the following inequation.

$$m_i \leq P \ll N \approx T \quad \forall i$$

We could see that theoretically, the communication cost is also larger with the parallel-in-time method. However, another critical factor that affects the runtime for parallel computation is the number of synchronizations. The total synchronization number required for spatial parallelization is:

$$T$$

and the total synchronization number for the multilevel parareal method is:

$$\sum_{i=0}^{L-1} m_i$$

We see that for large problems, the multilevel parareal method clearly has fewer synchronization numbers. This implicates that the multilevel parareal method might achieve higher scalability than spatial parallelization, assuming that the iteration number m_is does not overgrow according to the number of processors.

4. Numerical Results

For the numerical experiment, we choose the one-dimensional advection problem, which is a hyperbolic PDE problem. As mentioned in Section 3.1, hyperbolic problems are harder to converge than other problems while applying parallel-in-time methods.

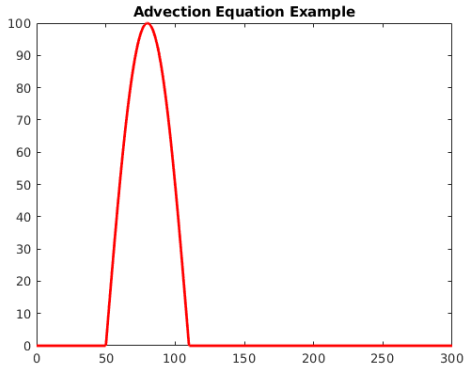


Fig. 5: Initial condition for the advection problem.

4.1 One-Dimensional Advection Problem

We apply our method to the following one-dimensional advection problem.

$$\frac{\partial u}{\partial t} = -1 \times \frac{\partial u}{\partial x}$$

This advection function represents a wave moving towards the +x direction with a velocity of 1. We set the initial condition as the following sine wave, as shown in Figure 5.

$$u(x,0) = \begin{cases} 0 & 0 \leq x \leq 50,\ 110 \leq x \leq 300 \\ 100[\sin(\pi\frac{x-50}{60})] & 50 \leq x \leq 110 \end{cases}$$

We choose the Lax-Wendroff method as the explicit time-marching scheme to solve the advection problem.

$$u_i^{n+1} = u_i^n + \frac{\partial u}{\partial x}\Delta t + \frac{\partial^2 u}{\partial x^2}\frac{\Delta t^2}{2!} + O(\Delta t^3)$$

$$u_i^{n+1} = u_i^n - \frac{\Delta t}{2\Delta x}(u_{i+1}^n - u_{i-1}^n) + \frac{\Delta t^2}{2\Delta x^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

For the multilevel parareal method, we apply the Lax-Wendroff method on the target grid level as low and high precision solvers.

We compare the result of our multilevel parareal method to the sequential Lax-Wendroff method after 128 time steps. With average error tolerance 0.3, as shown in Figure 6a, we can see that we achieve very similar results for the sequential Lax-Wendroff method and the multilevel parareal method. Most error, as shown in Figure 6b concentrates on the discontinuous points of the sine wave, which is inevitable due to the Gibbs phenomenon. Overall, the proposed method serves as a decent approximating parallelization method. Furthermore, the multilevel parareal method is very scalable and efficient, which we will discuss in the following sections.

4.2 Convergence and Iteration Number

We also observe that with a sufficient number of levels, the iteration number of the finest level could be reduced to just one iteration. Table 1 shows the iteration required to converge to a reasonable result with a different number of levels and different error tolerance. We could see that overall, with deeper levels, the numbers of iteration required to converge on the finest level are fewer. Of course, too many levels would also cause redundant computations. Thus, choosing

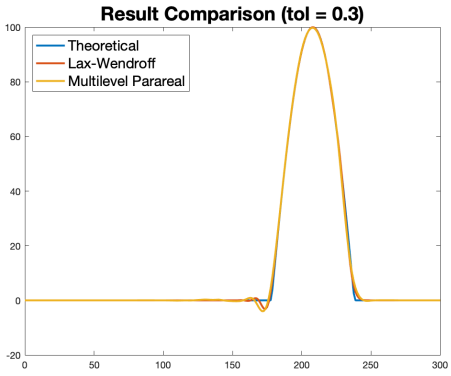
the right number of levels is essential for this method.

The result we show in Figure 6a uses three levels, eight processors, and with average tolerance 0.3. The tolerance on the table is the tolerance of the finest level. As described in Section 3, we use smaller tolerance in coarser levels. Therefore, the tolerance on level 1 would be 0.15, so on so forth. The tolerance values are relatively large, but as we could see from the result, most error cams from the discontinuous points and on fine grids that are computed by prolongation.

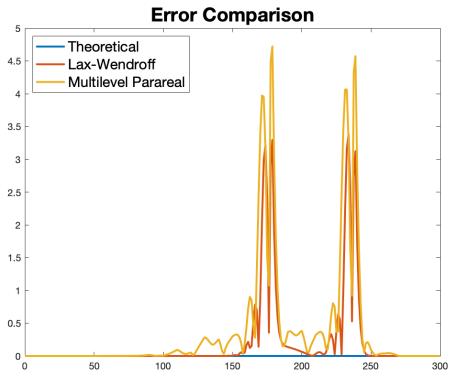
4.3 Runtime Comparison

In order to get the scalability of the multilevel parareal method, we texted on a huge example with $2^{14} + 1$ x grid points and 2^{14} number of time steps. We construct a multilevel parareal method with seven levels and compare the runtime result to that of the spatial parallelization.

We record results with the number of processors from 1 to 64. The tests are run on Oakbridge-CX. Oakbridge-CX is an Intel Xeon Platinum 8280 system with 28 CPUs per node and two cores per CPU. The memory size is 96 GB, and the memory bandwidth 101 GB/s.



(a) Result comparison



(b) Error comparison

Fig. 6: (a)Result and (b)error comparison of the multilevel parareal method to the sequential Lax-Wendroff method.

Error tol	1.0			0.5			0.3			0.2		
level 3			8			8			8			8
level 2		4	8		6	8		7	8		7	8
level 1	3	3	1	3	5	5	4	6	8	4	8	8
level 0	1	1	1	1	1	1	2	2	1	2	4	1

Table 1: Required iteration to converge with different error tolerance and with different number of levels.

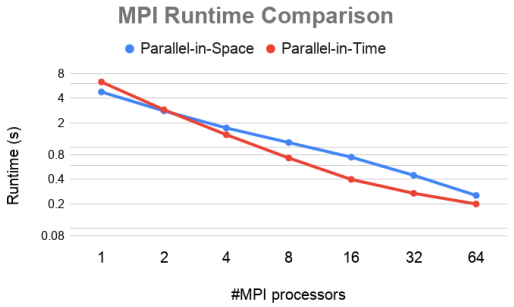


Fig. 7: Runtime comparison of spatial parallelization and parallel-in-time (our method) with different number of MPI processors.

We could see that the multilevel parareal method has higher scalability than the spatial parallelization of the Lax-Wendroff method. As discussed in Section 3.3, spatial parallelization synchronizes T times, much larger than $\sum m_i$ times for the multilevel parareal method. The huge difference in synchronization number is what causes the multilevel parareal method to scale better than spatial parallelization.

We could also observe that the scalability for the multilevel parareal method decreases after 16 processors. This is because we are using a 7 level structure. With more than 16 processors, the number of data points at the coarsest level in each processor is too less that it affects the convergence rate. However, if we have a larger example, the higher scalability should be maintained even with more than 16 processors.

5. Conclusion

We propose a multilevel parareal method to perform parallel-in-time for explicit time-marching schemes. The multilevel parareal method is mainly based on the parareal method but has a hierarchy similar to the MGRIT method. We coarsen both space and time grid at the same time to prevent the violation of the CFL-condition and to improve the convergence of the method. We also use a non-linear prolongation operation to increase the convergence rate. With not so small tolerance, we could achieve similar results with a small iteration number, which leads to less computation time.

The proposed method has higher scalability compare to spatial parallelization because it has much less synchronization to perform. We have also shown the previous results with a one-dimensional advection example, which is hyperbolic.

Despite that the method has high scalability, the accuracy is yet to be improved. Errors occur on discontinuous points, and fine points suggest that we might have to use a better prolongation method. We have also not yet tested out the method on sufficiently enough number of examples, or higher dimension examples. So the versatility of the method is yet to be proven.

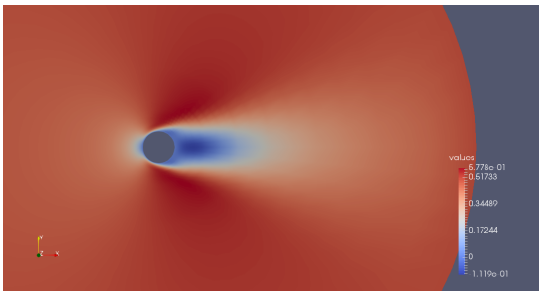


Fig. 8: Two dimension compressible fluid CFD with a obsticle in the center.

6. Future Work

As shown in the numerical result, the result of our multilevel parareal method could not perfectly fit that of the sequential Lax-Wendroff method. We are currently taking a larger tolerance and use the method as a fast approximation method. For future work, we would like to improve the accuracy of the method. One direct idea is to use higher-order prolongation methods.

Another problem with our current method is that since we have a deep level structure, there are no enough points at the coarsest level, which causes iteration to grow rapidly with the number of processors at coarse levels. This could be solved by changing the number of levels or by improving the convergence, which is our future work.

Since parallel-in-time and parallel-in-space each has its limits and advantages, it would be reasonable to apply both at the same time and find out the best parallel strategy with a given computation resource.

We are also working on applying our method on two-dimension compressible computational fluid dynamics (CFD) problems. CFD simulation, especially shock wave simulation, are important applications of explicit time-marching schemes. Apart from that, a parallel-in-time method for complicated systems such as CFD has not yet been tried because of its huge complexity. As our method is relatively simple and highly scalable, it would be critical if we could apply our multilevel parareal method on CFD problems.

References

[1] Falgout, R.D., Friedhoff, S., Kolev, T.V., MacLachlan, S.P. and Schroder, J.B.: Parallel time integration with multigrid, *SIAM Journal on Scientific Computing*, vol.36,No.6,pp.C635–C661 (2014).

[2] Lions, J., Maday, Y. and Turinici, G.: Résolution d'EDP par un schéma en temps «pararéel», *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, vol.332,No.7,pp.661–668 (2001).

[3] Lewy, H, Friedrichs, K and Courant, R: Über die partiellen Differenzengleichungen der mathematischen Physik, *Mathematische annalen*, vol.100,pp.32–74 (1928).

[4] Gander, M. J., Kwok, F. and Zhang, H.: Multigrid interpretations of the parareal algorithm leading to an overlapping variant and MGRIT, *Computing and Visualization in Science*, vol.19,No.3-4,pp.59–74 (2018).

[5] Krzysik, O. A., De Sterck, H., MacLachlan, S. P. and Friedhoff, S.: On selecting coarse-grid operators for Parareal and MGRIT applied to linear advection, *arXiv* , (2019).

[6] Ruprecht, D.: Convergence of Parareal with spatial coarsening, *PAMM*, vol.14,No.1,pp.1031–1034 (2014).

[7] Iizuka, M. and Ono, K.: Investigation of Convergence of Parareal Method for Advection Equation using Accurate Phase Calculation Method, *7th Workshop on Parallel-in-Time methods*, (2018).