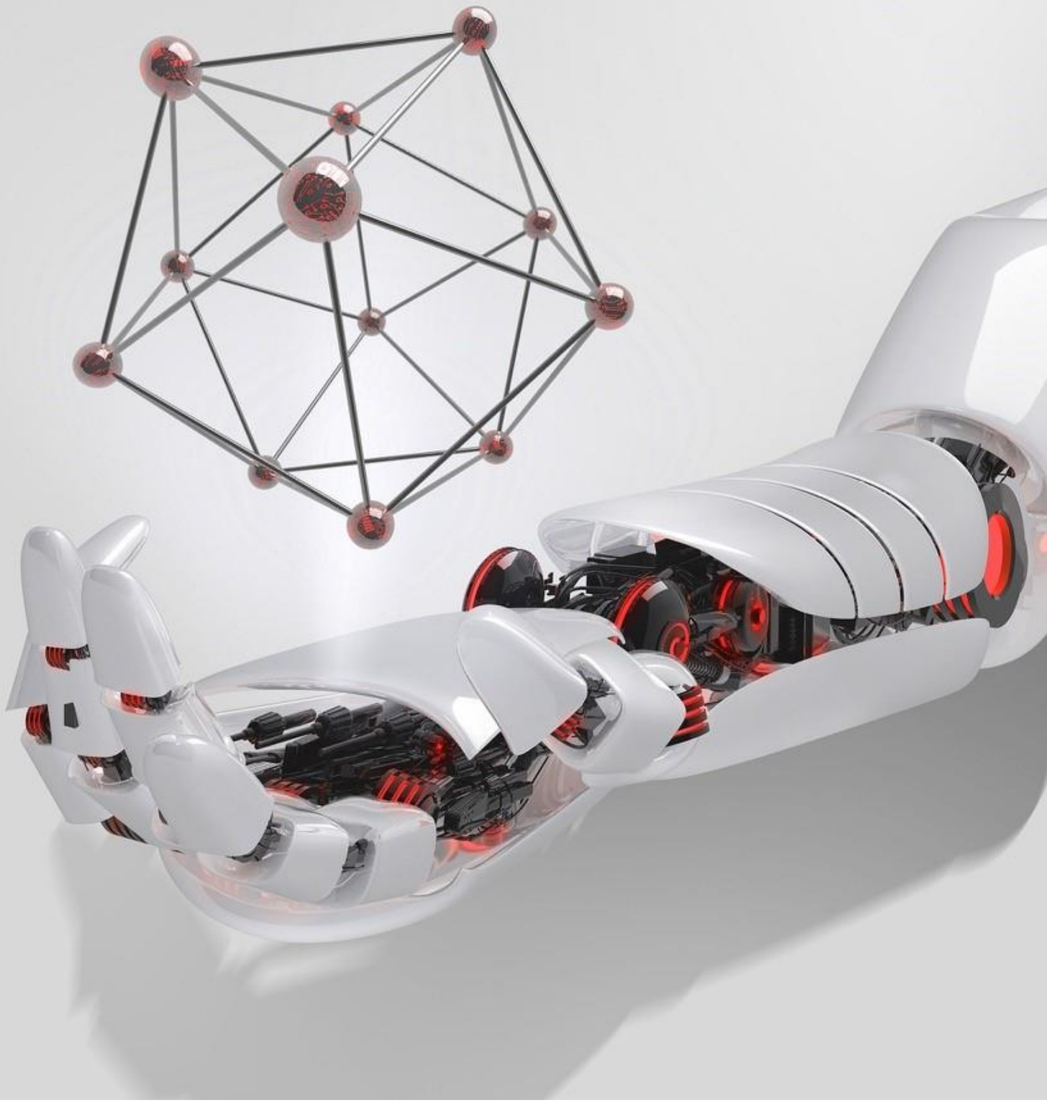
A time-parallel ordinary differential equation solver with an adaptive step size: performance assessment

Eugeniy Kazakov[1], Dmitry Efremenko[2],
Viacheslav Zemlyakov[1], Jiexing Gao[1]

[1] Huawei Technologies Co., Ltd, Russian Research Institute, Moscow, Russia
[2] Remote Sensing Technology Institute (IMF), German Aerospace Center (DLR), Oberpfaffenhofen, Germany



Content

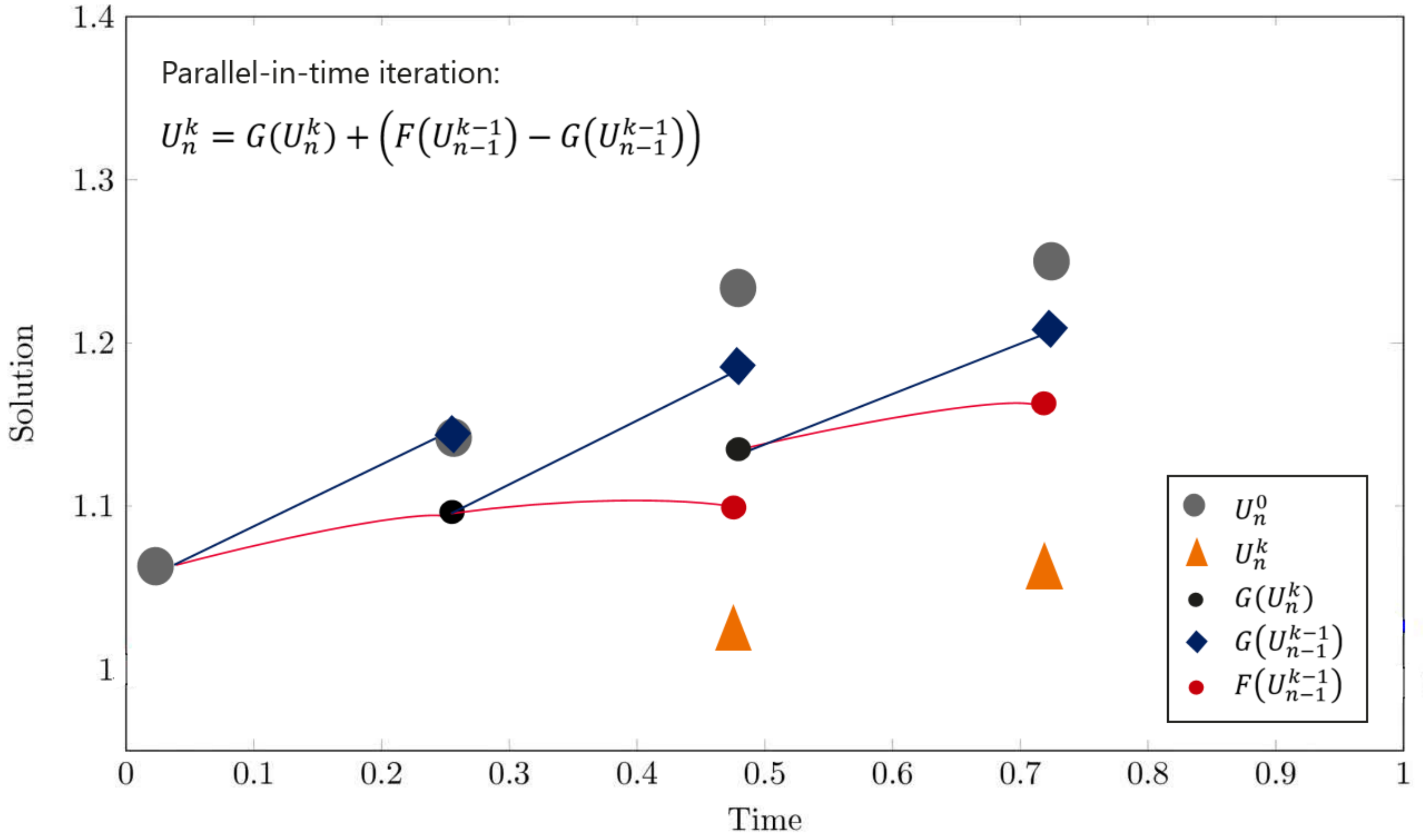
- Motivation
- Classical parareal ODE algorithm
- Parallel ODE solvers with adaptive step
- Numerical results
- Summary

Motivation

- There is a need for speeding up the ODE solvers on large time domains (with fine resolution).
- In a view of cheap multicore CPUs, it is very tempting to design parallel ODE solvers
- parallelization across the time direction is challenging due to a causality principle, namely, the ODE solution later in time depends on the solution earlier in time.
- Nevertheless, in 1950s, Nievergelt proposed parallel-in-time ODE solver - PARAREAL)
- The goals of the study are
  a. to assess the possible benefits of PARAREAL
  b. to find some ways to improve PARAREAL

Parallel-in-time ODE solver

- Parareal is based on coarse and fine ODE solvers:
- The coarse solver provides initial points across the domain to the fine solver.
- Fine solvers are performed in parallel, while the coarse solver is sequential.



U_n^0 – initial coarse integrator solution solution,

U_n^k – current solution,

$G\left(U_{n-1}^{k-1}\right)$ – coarse integrator

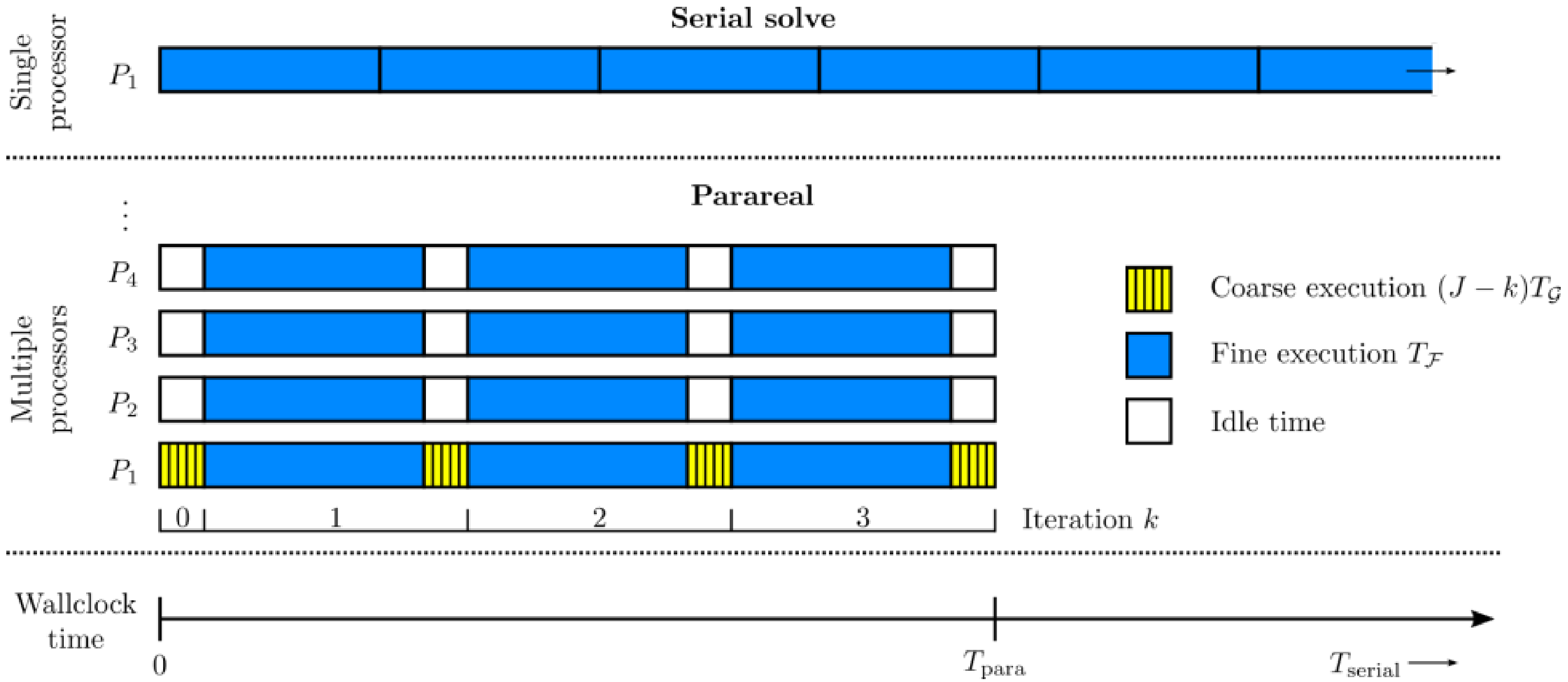
(e.g. Euler method),

$F\left(U_{n-1}^{k-1}\right)$ – fine integrator (e.g. Runge-Kutta method),

k – iteration number,

n – solution point number.

Parareal



Theoretical speedup factor for of the Parareal algorithm:

$$S = \left[k/M + (k+1)\left(1 - \frac{k}{2M}\right)\frac{W_G}{W_F}\right]^{-1}$$

k is the number of iterations,
M – number of parallel subdomains,
W_G, W_F - the total computational fine and coarse solver times.

S is large when W_G/W_F<<1 (which is a common assumption in papers about Parareal)

Iterations

- If the prediction of the fine solver is far from that of the coarse solver, then the whole solution is wrong and has to be corrected (empirically).

$$y_{n(j+1)}^{k+1} = G\left(y_{nj}^{k+1}, t_{nj}, t_{n(j+1)}\right) + F\left(y_{nj}^{k}, t_{nj}, t_{n(j+1)}\right) - G\left(y_{nj}^{k}, t_{nj}, t_{n(j+1)}\right)$$

- E.g., in Gparareal: the correction is performed using the “history” of previous iterations involving Gaussian processes (a little improvement as compared to the classical Parareal)

- The corrections are not fail-safe -> for *M* subdomains, *M*-1 iterations might be required -> no gain in performance

Ways to improve Parareal

- The majority of studies of Parareal are concerned with the performance of the fine solvers and the correction procedure, sticking to the Euler solver as the coarse one [1,2].

- In our examples, we did not find a case where this approach would be more efficient than a single-threaded ODE solver.

- A possible solution : improve the accuracy of the coarse solver -> minimize the number of iterations

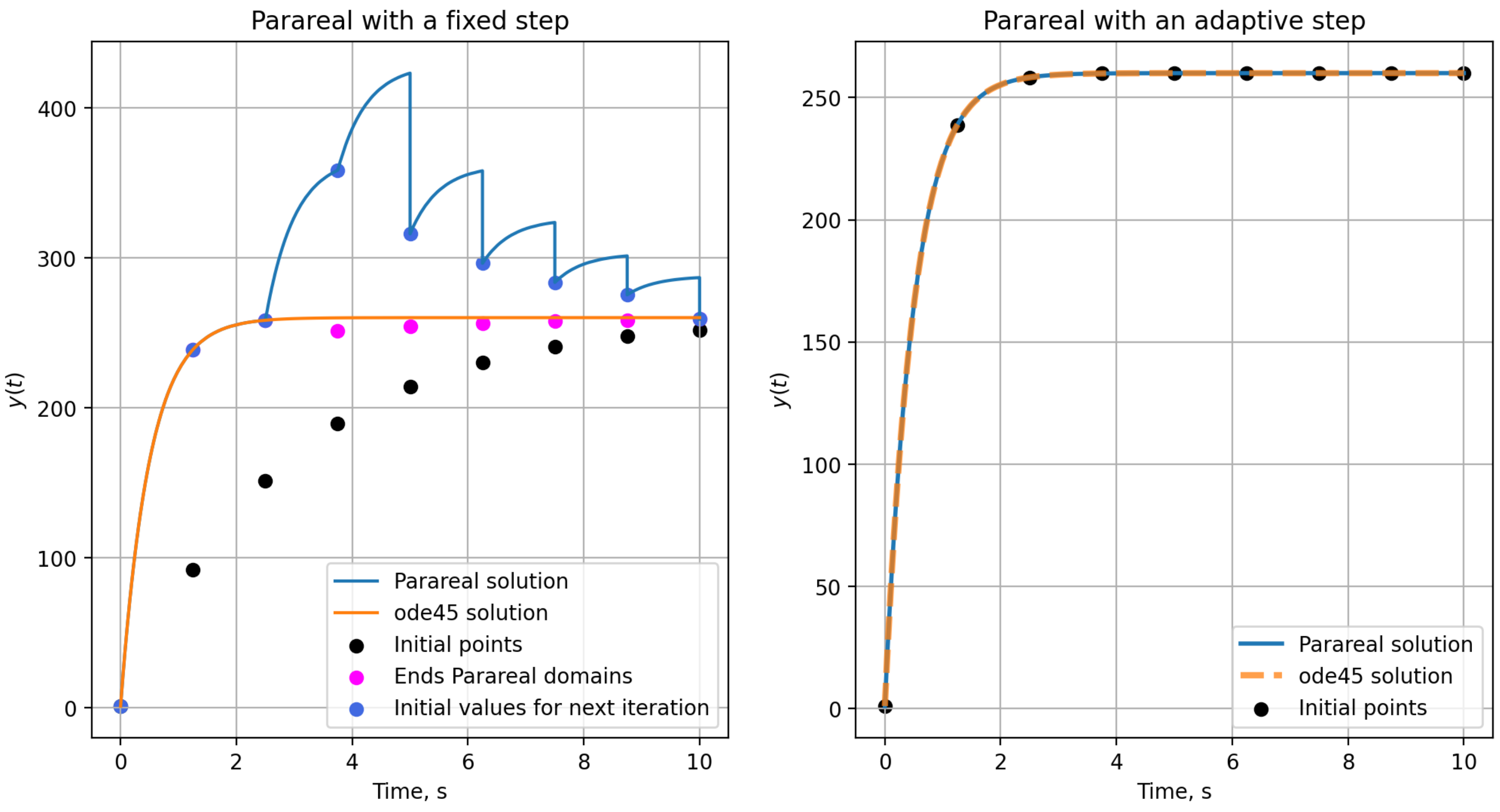
.[1] Arteaga, A., Ruprecht, D., Krause, R.: A stencil-based implementation of parareal in the C++ domain specific embedded language STELLA. (2014), https://arxiv.org/abs/1409.8563.
[2] Pentland, K., Tamborrino, M., Sullivan, T.J., Buchanan, J., Appel, L.C.: Gparareal: A time-parallel ODE solver using Gaussian process emulation. (2022), https://arxiv.org/abs/2201.13418.

Algorithm and implementation

```
Begin:
Setup input parameters.
MPI_Init:
    Begin iteration cycle:
        In master process:
            Coarse solver iteration cycle:
                Run G solver,
            End coarse solver iteration cycle,
            MPI_Send: obtained initial values another processes,
            MPI_Recv: solutions of non-zero processes,
            Error estimation.
        In other processes:
            MPI_Recv: initial value,
            Run F solver,
            MPI_Send: solution to 0 process.
    End iteration cycle.
MPI_Finalize.
End
```

OpenMP implementation is considered in Daniel Ruprech, Implementing Parareal – OpenMP or MPI? (2021), https://arxiv.org/abs/1509.06935v1.

Applying adaptive step



(a)

(b)

$$
\begin{aligned}
&\frac{dy}{dt} = \alpha - \beta y,\\
&y(0) = 1,\\
&\alpha = 520,\ \beta = 2.
\end{aligned}
\tag{1}
$$

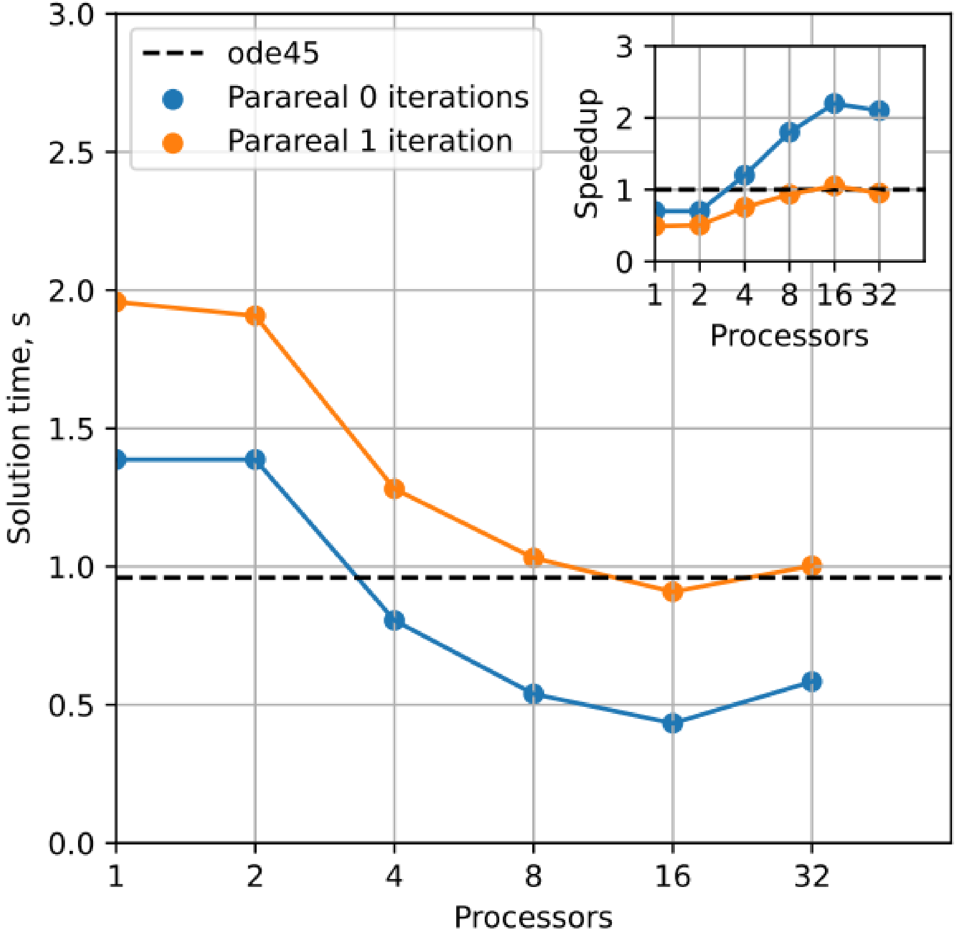
Solution after one iteration for Eq. (1):

(a) with the fix step coarse integrator;

(b) with the adaptive step coarse integrator.

Efficiency of Parareal

Solution after one and two iteration for Eq. (1), 1e7 points.

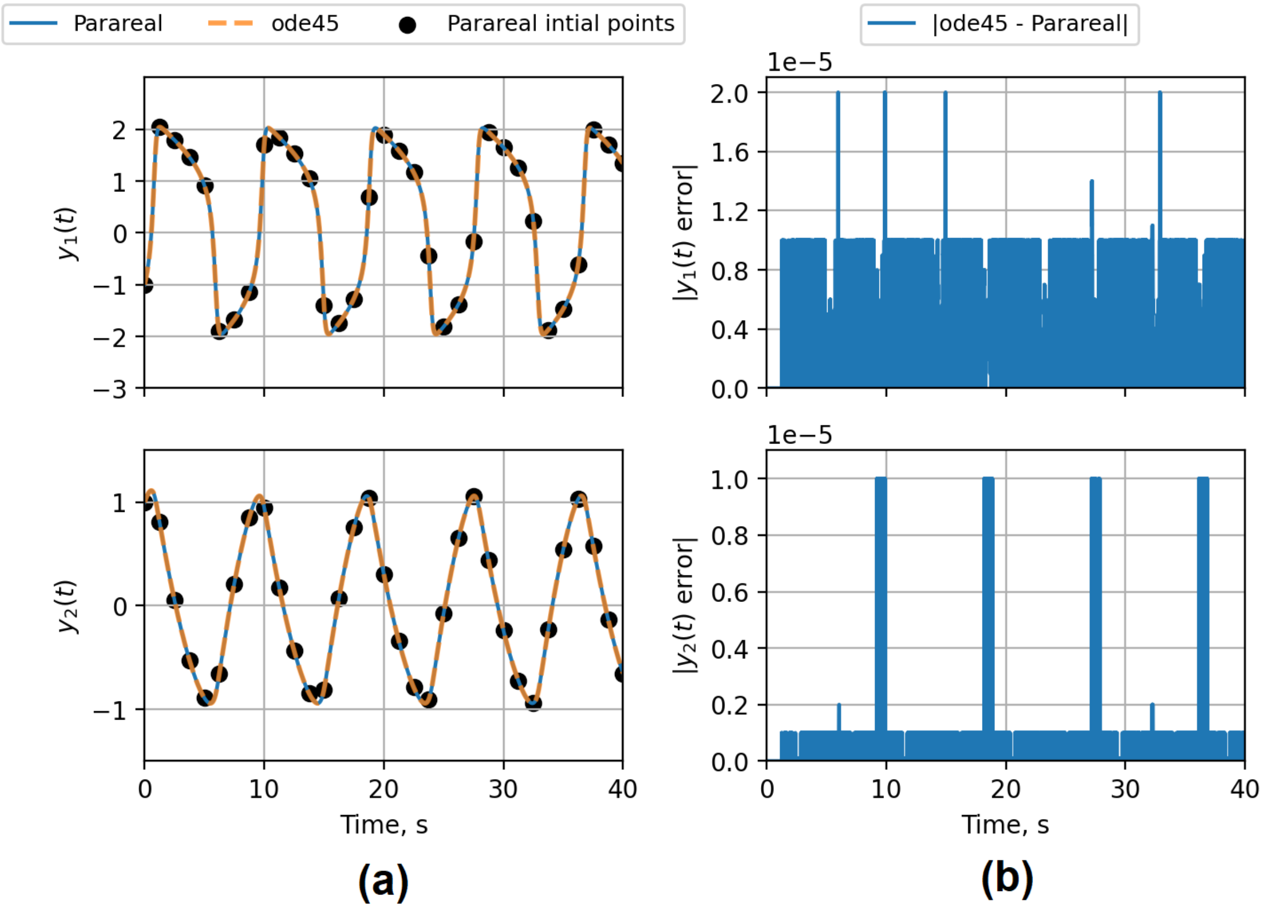


- Unlike in theoretical papers about Parareal, in real applications, iterations may nullify performance enhancement of Parareal;
  - W_G/W_F ~ 1;
  - Formula for S does not take into account overhead due to data transfer;

Number of CPUs	Speedup	Implementation	Ref.
16	2	C++ MPI	Current work
16	2	MATLAB parfor + Gaussian predictor	[1]
24	4	FORTRAN OpenMP/MPI	[2]

[1] Pentland, K., Tamborrino, M., Sullivan, T.J., Buchanan, J., Appel, L.C.: Gparareal: A time-parallel ODE solver using Gaussian process emulation. (2022), https://arxiv.org/abs/2201.13418.
[2] OpenMP implementation is considered in Daniel Ruprech, Implementing Parareal – OpenMP or MPI? (2021), https://arxiv.org/abs/1509.06935v1

Results for FitzHugh-Nagumo model



$$\frac{dy_1}{dt} = c\left(y_1 - \frac{y_1^3}{3} + y_2\right), \qquad (2)$$

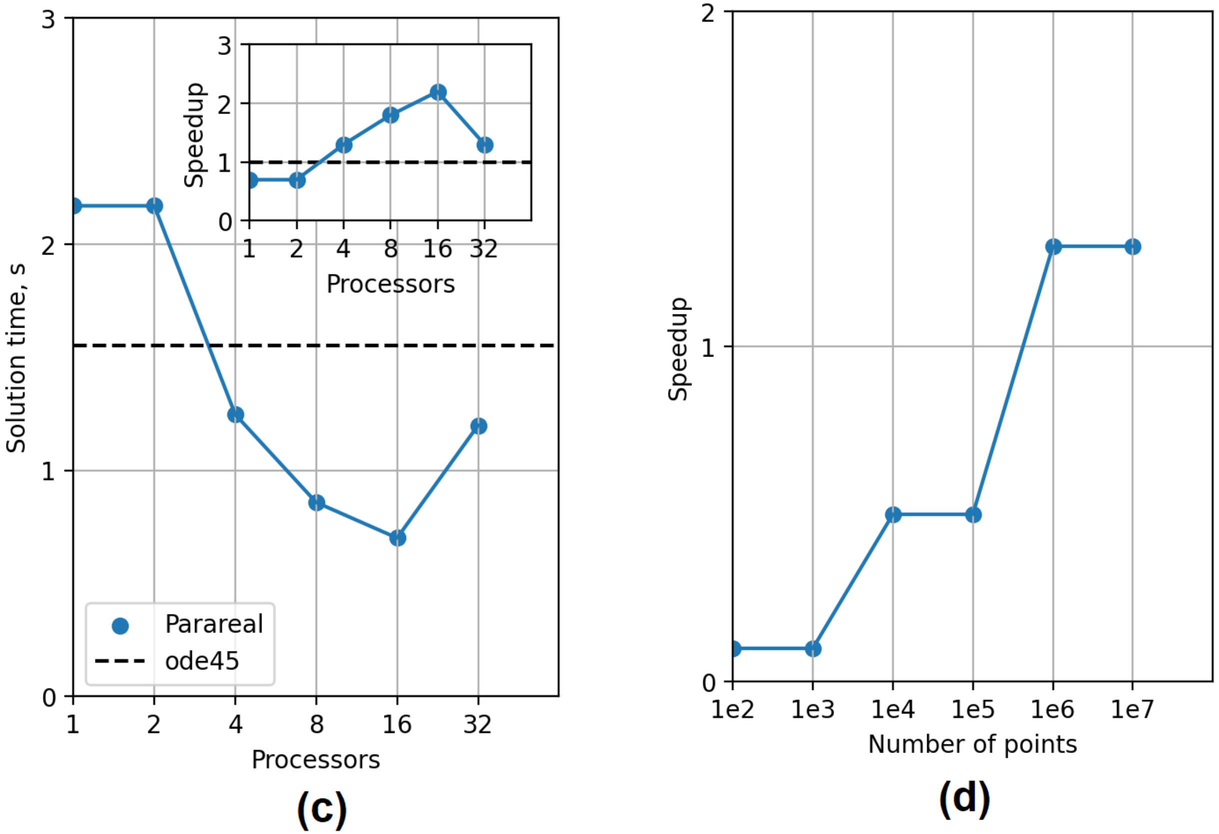
$$\frac{dy_2}{dt} = -\frac{1}{c}\left(y_1 - a + by_2\right),$$

$a = 0.2, b = 0.2, c = 3.$

(a) Solutions for Eq. (2) obtained with Parareal on 32 CPUs and serial ode45 solver;

(b) Absolute errors of the Parareal solution.

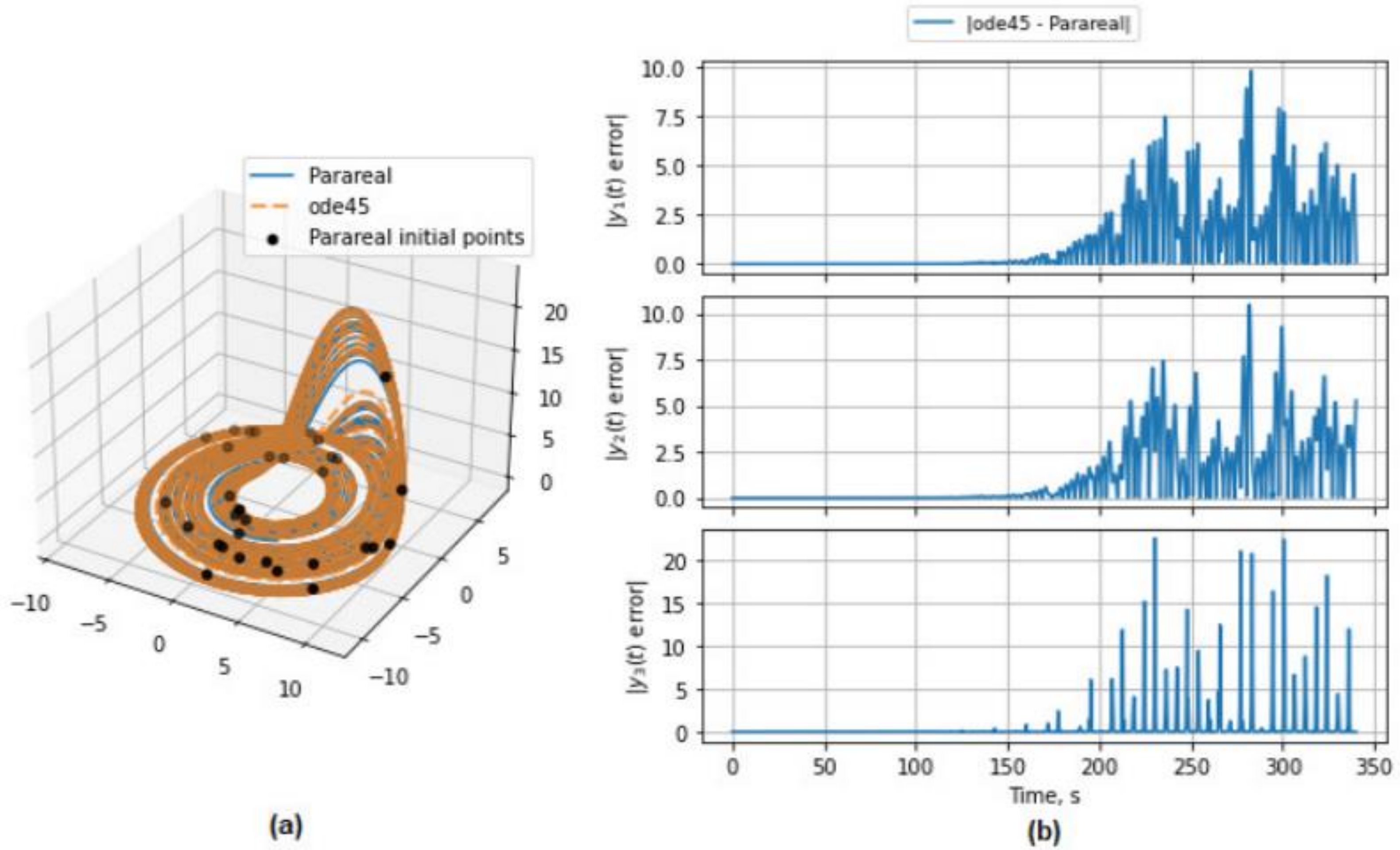
Results for FitzHugh-Nagumo model



(c) Average solution time for the serial ode45 solver and Parareal as a function of CPUs for 1e7 points;

(d) The speedup factor provided by Parareal as a function of points for 32 CPUs .

Solution of Rossler system (stiff ODE)



$$\frac{dy_1}{dt} = -y_2 - y_3,$$

$$\frac{dy_2}{dt} = y_1 + ay_2, \qquad (3)$$

$$\frac{dy_3}{dt} = b + y_3(y_1 - c),$$

$$a = 0.2, b = 0.2, c = 5.7.$$

(a) Solutions obtained with Parareal on 32 CPUs and serial ode45 solver;

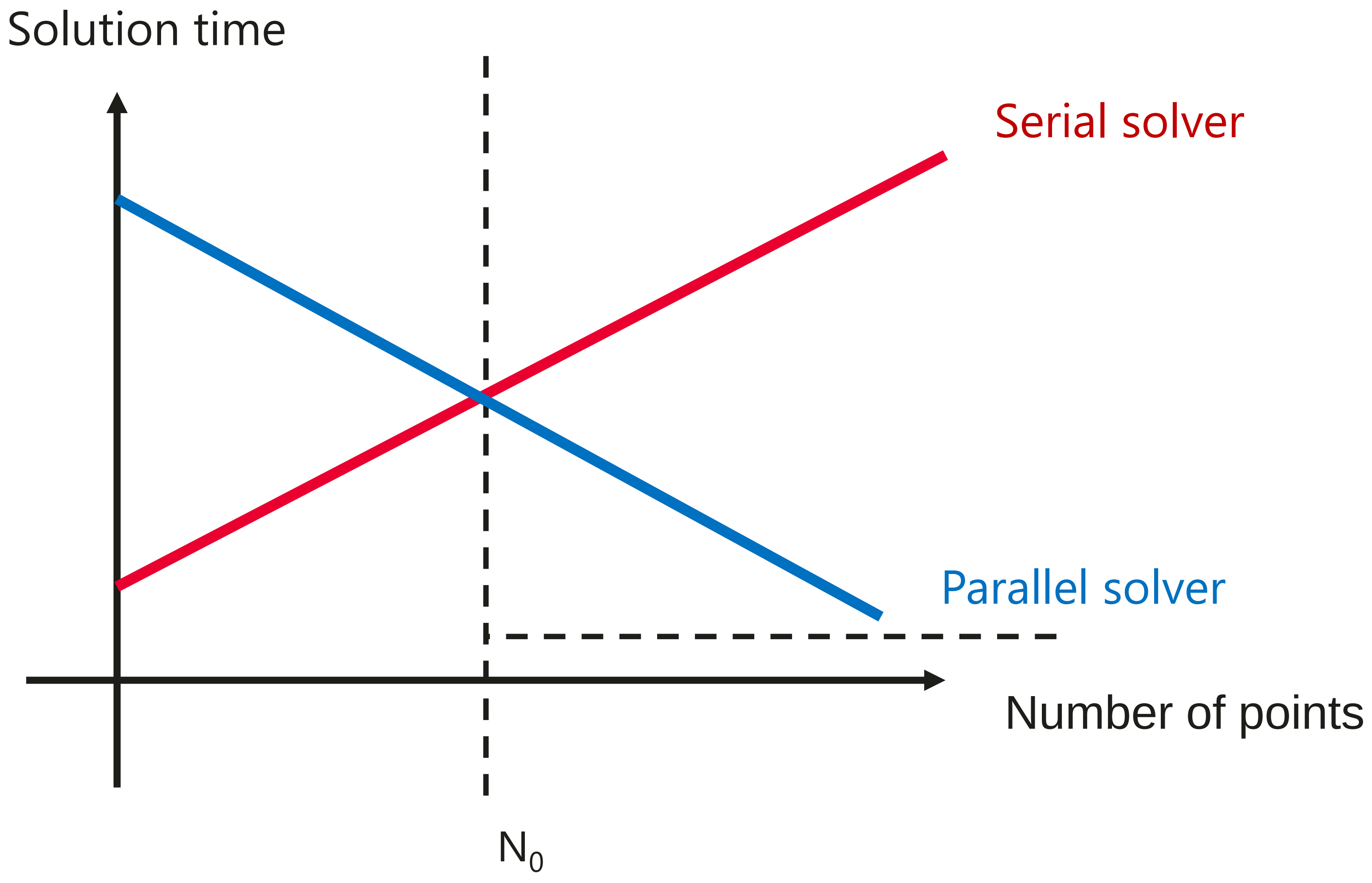
(b) Absolute errors of the Parareal solution.

Conclusions

- For our implementation of Parareal, two-fold speedup is achieved on 16 CPUs;
- A classical implementation of Parareal algorithm does not lead to a significant gain in performance (given an optimized ODE solver as a reference solver);
- Parareal algorithm with an adaptive step coarse solver is more robust and requires less iterations than the classical Parareal algorithm. Gain in speed is larger than the loss in performance due to the adaptive step;
- The gain in performance is pronounced for sufficiently large number of points -> a subject for fine tuning of data transfer between the processes;
- In the view of the overhead due to data transfer, the efficiency of the coarse solver is just as important as that of the fine solver.

Thank you.

Performance assessment



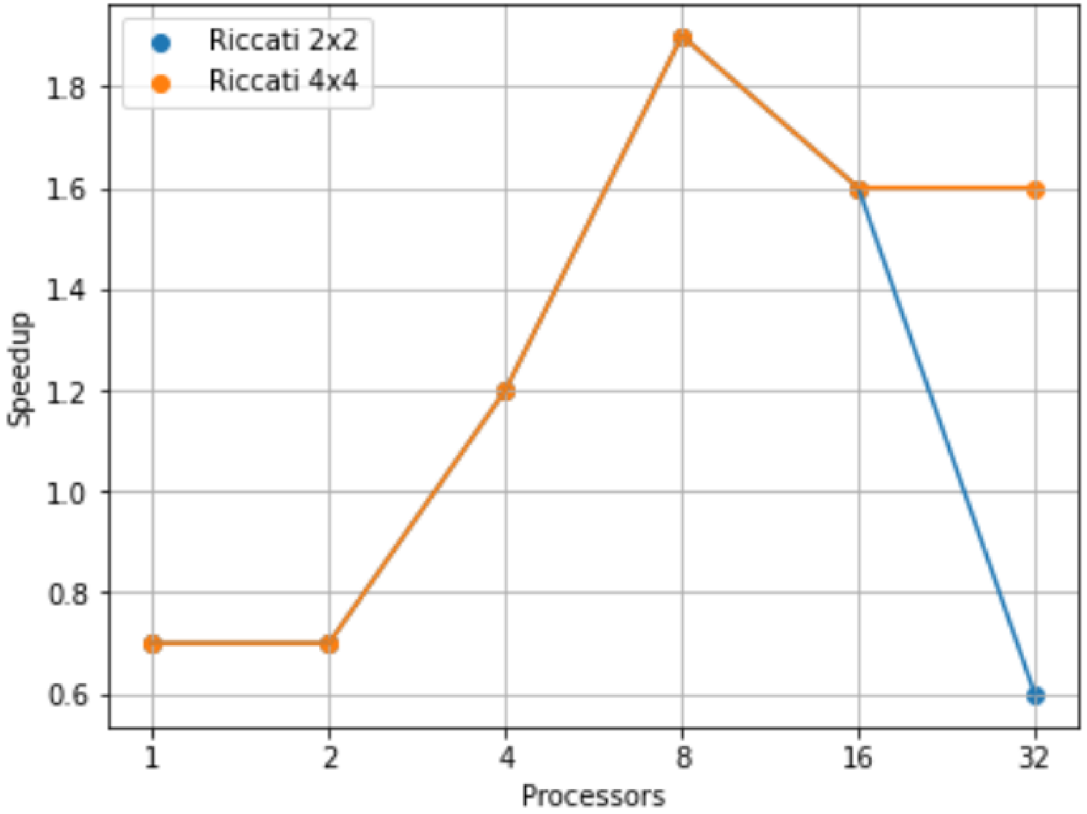
Schematic comparison solution time of computational load, number of points.

Problems

Solution scalable and matrix equations

Matrix Riccati equation

$$\frac{d}{dt}\mathbf{y}(t) = \mathbf{B} + \mathbf{A}\mathbf{y}(t) + \mathbf{y}(t)\mathbf{A} + \mathbf{y}(t)\mathbf{B}\mathbf{y}(t),$$


● Riccati 2x2
● Riccati 4x4
Speedup
Processors