# Time parallelization of plasma simulations using the parareal algorithm

D. Samaddar[1], W.A. Houlberg[1], L.A. Berry [2], W.R. Elwasif [2], G. Huysmans [1], D. Batchelor [2]

[1] *ITER Organization, Route de Vinon sur Verdon, 13115 Saint Paul Lez Durance, France*

[2] *Oak Ridge National Laboratory, Oak Ridge, USA*

## Introduction

Simulation of fusion plasmas involve a broad range of timescales. In magnetically confined plasmas, such as in ITER, the timescale associated with the microturbulence responsible for transport and confinement timescales vary by an order of $10^6 - 10^9$. Simulating this entire range of timescales is currently impossible, even on the most powerful supercomputers available [1].

Space parallelization has so far been the most common approach to solve partial differential equations. Space parallelization alone has led to computational saturation for fluid codes, which means that the walltime for computaion does not linearly decrease with the increasing number of processors used. The application of the parareal algorithm [2] to simulations of fusion plasmas ushers in a new avenue of parallelization, namely temporal parallelization.

The algorithm has been successfully applied to plasma turbulence simulations [3], prior to which it has been applied to other relatively simpler problems [4, 5, 6, 7, 8, 9]. This work explores the extension of the applicability of the parareal algorithm to ITER relevant problems, starting with a diffusion-convection model.

## Parareal Algorithm

The parareal algorithm involves a coarse (G) and a fine solver (G). The coarse solver requires a shorter computation time than F, but generates a less accurate solution. G runs as a serial process across all processors while F is executed in parallel on individual processors, thus allowing temporal parallelization. G and F are alternated through successive iterations until the error reaches a certain tolerance. The error per processor is typically estimated by computing the relative error between 2 successive F solutions. Details of the algorithm are available in [2] and [3]. In this work, the parareal framework developed as part of the SWIM IPS project has been utilized [10]. Prior to this, parareal implementation was typically carried out using MPI and OpenMP Parallelization techniques. That method, though often generating optimum computational gain, is considerably more tedious. Moreover, such implementation is problem specific - which means each separate problem had to be treated individually. Using a Python based framework, such as the one used here, allows incorporating different problems in a relatively simpler fashion.

The framework also addresses another problem commonly faced with the parareal algorithm. Detecting the optimum coarse solver, G for the best gain and efficiency is a challenge. This task is relatively simpler when using a framework. However the framework depends on a I/O file based system which can often contribute to a computational overhead.

**Diffusion-Convection Model**

The parareal algorithm is applied to a diffusion-convection system. The system of equations used in this 1D - transport model is given as follows (Private communication G. Huijsmans):

$$\frac{\delta T}{\delta t} = \nabla \cdot \kappa \nabla T - v \cdot \nabla T + S_T \quad (1)$$

$$\frac{\delta n}{\delta t} = \nabla \cdot D \nabla n - v \cdot \nabla n + S_n \quad (2)$$

where T is the temperature and n is the density. $S_T$ and $S_n$ are source terms. D is the diffusivity and $\kappa$ is the thermal conductivity.

**Results**

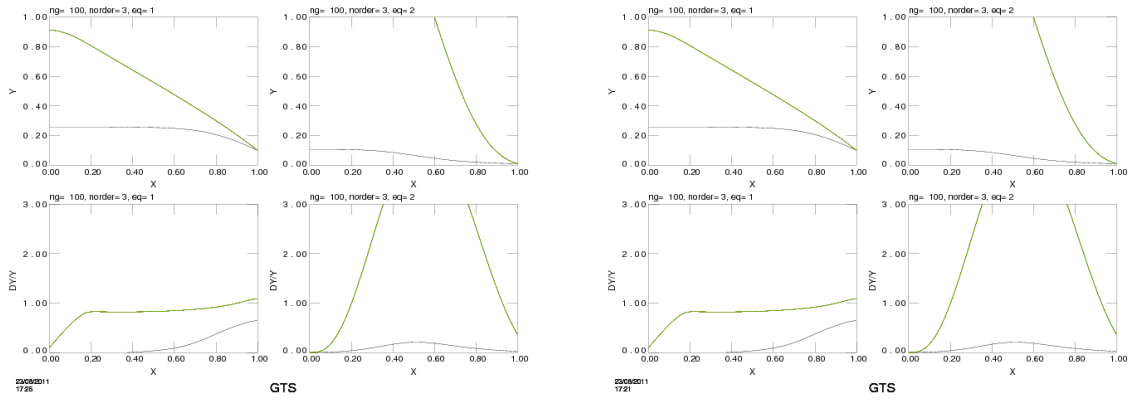
The application of the parareal algorithm to the 1D transport model yielded interesting results. Fig.(1(a)) is the solution obtained from a serial run on 1 processor, while Fig.(1(b)) is the same case obtained using the parareal algorithm on multiple processors. With an error tolerance of $1E-7$, the solutions are identical proving the success of the parareal algorithm.

For these simulations, the coarse solver was chosen to be the same as the fine, except with timesteps bigger by a factor of 100. Using bigger values than this yielded unstable solutions.

The number of iterations required for convergence depends quite expectedly on the value of tolerance (tol). For a simulation of 80000 timesteps across 16 processors, convergence was achieved in 3 iterations for tol$= 1e-7$ and in 2 iterations for tol$= 1e-4$. Fig.(2) shows the decline of error across processors with increasing iterations.

The time for a typical serial run for 80000 timesteps requires 160 seconds. It was observed while running the parareal simulations, that each fine run took 10 seconds for 5000 steps (which is the time slice solved per processor in parallel), and the coarse solver took 1.6 seconds for 80000 timesteps. Hence, for 3 iterations with tol$= 1e-7$, the total time was expected to be $\sim 32$ seconds, which in reality was 75.29 seconds. With tol$= 1e-4$, convergence is achieved in 2 iterations with computational walltime 49.7 seconds. Time spent on I/O and some interprocessor communication was primarily responsible for this discrepancy. However this is inherently a hardware issue on the hpc machine involved. This problem has been less significant in the

performance of the parareal framework on many other computers [10]. So, these simulations generated a modest gain of 3.24 with 16 processors, but there is room for improvement.




(a) Serial solution (b) Parareal solution

Figure 1: Solution after 30000 timesteps. The first column of plots represents T and the second column represents n.

## Conclusions

The parareal algorithm has been successfully applied to a 1D transport system. It may be extended to other problems which are characterized by events resulting from pellet injection, MHD activity or an introduction of ELMs. In case of event driven systems, a more careful estimation of the coarse solver will be required. A successful implementation of time parallelization to these simulations will allow multiple levels of concurrency, thus allowing simulations of burning plasmas more tractable.

## Disclaimer

The views and opinions expressed here do not necessarily represent those of the ITER Organization.

## References

[1] J. Dahlburg, J. Corones, D. Batchelor, et al. J. Fusion Energ. 20 (4) 135.19.(2001)

[2] J. Lions, Y. Maday and G. Turinici, CR Acad. Sci. I - Math. 332 (7), pp. 661-668 (2001)

[3] D. Samaddar,D. E. Newman and R. Sanchez, Journal of Computational Physics 229 (18), pp. 6558-6573(2010)

[4] L. Baffico, S. Bernard, et. al, Phys. Rev. E 66 (5), 057706 (2002)

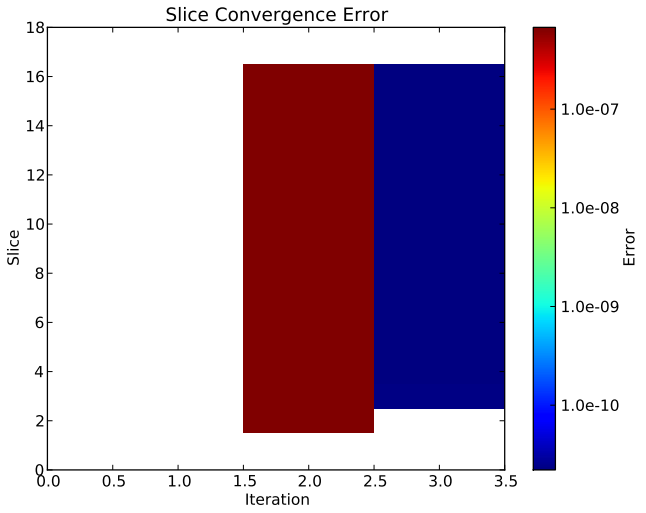



Figure 2: Error across processors for 3 iterations. The tolerance in this case was $1e-7$.

[5] G. Bal, Y. Maday, Lecture Notes in Computer Science and Engineering, vol. 23, Springer, Berlin, 2002.

[6] G. Staff, G. Ronquist, Proceedings of Fifteen International Conference on Domain Decomposition Methods, Springer Verlag, pp. 449-456, (2003)

[7] I. Garrido, M.S. Espedal, G.E. Fladmark, Proceedings of Fifteen International Conference on Domain Decomposition Methods, vol. 40, Springer, Berlin, pp. 469-476.

[8] P.F. Fischer, F. Hecht, Y. Maday, Proceedings of Fifteen International Conference on Domain Decomposition Methods, vol. 40, Springer Verlag, 2004, pp. 433-440, (2004)

[9] M.J. Gander, E. Hairer, Domain Decomposition Methods in Science and Engineering XVII,vol. 60, Springer, pp. 45-56, (2008)

[10] L. A. Berry, W. Elwasif, J. Reynolds, D. Samaddar, R. Sanchez and D. E. Newman, Journal of Computational Physics (Submitted)