PARALLEL IN TIME ALGORITHMS FOR MULTISCALE DYNAMICAL SYSTEMS USING INTERPOLATION AND NEURAL NETWORKS

Gopal R. Yalla

Institute for Computational Engineering and Sciences
University of Texas at Austin
201 East 24th Street
Austin, TX, USA
gopal@ices.utexas.edu

Bjorn Engquist

Institute for Computational Engineering and Sciences
University of Texas at Austin
201 East 24th Street
Austin, TX, USA
engquist@ices.utexas.edu

ABSTRACT

The parareal algorithm allows for efficient parallel in time computation of dynamical systems. We present a novel coarse scale solver to be used in the parareal framework. The coarse scale solver can be defined through interpolation or as the output of a neural network, and accounts for slow scale motion in the system. Through a parareal scheme, we pair this coarse solver with a fine scale solver that corrects for fast scale motion. By doing so we are able to achieve the accuracy of the fine solver at the efficiency of the coarse solver. Successful tests for smaller but challenging problems are presented, which cover both highly oscillatory solutions and problems with strong forces localized in time. The results suggest significant speed up can be gained for multiscale problems when using a parareal scheme with this new coarse solver as opposed to the traditional parareal setup.

Keywords: parareal, multiscale, dynamical systems, neural networks, interpolation

1 INTRODUCTION

Emerging multicore and many-core architectures will lead to a continued increase in computing power in the coming years. These complex architectures necessitate highly parallel and distributed computing for scientific simulations. While the capabilities of these machines benefit most scientific and engineering disciplines, they can also present challenges such as those for systems governed by time dependent dynamical systems. The effect of causality in time is more naturally handled sequentially. In (Lions, Maday, et al. 2001), the parareal algorithm for parallel in time computations is introduced, which is essential for highly parallel systems when distributed computation in space saturates. See also (Gander 2015) for a presenta-

tion of early parallel in time techniques including parareal algorithms. Initially the technique was applied to dissipative systems where the memory effect is more limited, but there are very important applications without dissipations as, for example, hyperbolic partial differential equations (PDE) and systems of ordinary differential equations (ODE) for molecular dynamics and celestial mechanics.

The parareal method is based on two solvers, one coarse scale solver, which is less accurate but fast enough to be applied sequentially, and one fine scale solver with full accuracy that is applied in parallel in time. At each iteration, the coarse solver is run sequentially over each subdomain similar to a shooting method, and then the fine solver is run in parallel as a correction. See section 2 for how the coarse and fine solvers interact. This coupling of the solvers is able to achieve the accuracy of the fine scale solver at the efficiency of the coarse scale solver. The overall technique can be seen as a domain decomposition method for the fine scale method; the coarse scale method and the iterative correction provide the coupling between the domains.

The coarse solver is a usually based on one or a few steps of a regular numerical solver. For instance, the coarse scale solver is typically taken to be an ODE solver with significantly larger time step compared to the fine solver or an approximation where the ODE itself has been modified to be less stiff or oscillatory. In opposition, the fine scale solver typically involves a very large number of local time steps for high accuracy. Gander and Hairer (2014) show that for Hamiltonian systems, very high accuracy is required already for the coarse solver for the process to work. This is not helpful in, for example, molecular dynamics where the fine step size is chosen to be at the limit of what gives a meaningful result. Since the computational efficiency of the parareal algorithm comes from the fact that the costly fine scale solver can be implemented in parallel in time, forcing the sequential coarse solver to be nearly as accurate as the fine solver succumbs the algorithm to Amdahl's law and ruins any speed up gained by parallelization.

The main innovation in this work is the new coarse solver, which relies on a pre-computed phase plane map, similar to the work in (Nievergelt 1964) and (Ying and Candès 2006), but applied in a parareal setting and optimized for highly oscillatory, autonomous dynamical systems. See (Barker 2014) for additional work in extending Nievergelt's method. The phase-plane map can be built with interpolation or the output of a neural network. A given dynamical system is accurately approximated with the fine solver for a single time interval and for a number of different initial values. The pre-computing for different initial values is embarrassingly parallel and the phase plane map does not suffer from the difficulties of highly oscillatory behavior, which is a main problem with standard parareal computations. Even a linear harmonic oscillator is a severe challenge for standard parareal simulations (Gander and Hairer 2014). For the interpolation based phase plane map linear ODEs require only linear interpolation for getting the accuracy of the fine scale solver at the efficiency of the coarse solver.

2 DESCRIPTION OF THE ALGORITHM

2.1 The Parareal Algorithm

Consider the dynamical system,

$$\begin{aligned} u'(t) &= f(u(t)), \qquad t \in (t_0, t_f) \\ u(t_0) &= u_0 \end{aligned}, \tag{1}$$

where $f : \mathbb{R}^d \to \mathbb{R}^d$ and $u : \mathbb{R} \to \mathbb{R}^d$. To obtain the parareal algorithm as in (Gander, Hairer, et al. 2008), first divide the time domain $\Omega \equiv (t_0, t_f)$ into N equal subdomains $\Omega_n = (T_n, T_{n+1})$, $n = 0, 1, \ldots, N-1$,

with $t_0 = T_0 < T_1 < \cdots < T_{N-1} < T_N = t_f$, and $\Delta T \equiv T_{n+1} - T_n$. Consider now the problem on each time subdomain:

$$\begin{aligned} u_n'(t) &= f(u_n(t)), \qquad t \in (T_n, T_{n+1}) \\ u(T_n) &= U_n \end{aligned}, \tag{2}$$

where the initial conditions U_n are the solution of (1) at time T_n. If we let $\phi_{\Delta T_n}(U_n)$ denote the solution of (1) with initial condition U_n after time ΔT_n, equation (2) is simply a shooting method in time and is equivalent to solving:

$$\begin{pmatrix} U_0 - u_0 \\ U_1 - \phi_{\Delta T_0}(U_0) \\ \vdots \\ U_{N-1} - \phi_{\Delta T_{N-2}}(U_{N-2})) \end{pmatrix} = 0 \tag{3}$$

Applying Newton's method to (3) gives,

$$\begin{aligned} U_0^{k+1} &= u_0 \\ U_n^{k+1} &= \phi_{\Delta T_{n-1}}(U_{n-1}^k) + \phi'_{\Delta T_{n-1}}(U_{n-1}^k)(U_{n-1}^{k+1} - U_{n-1}^k) \end{aligned}. \tag{4}$$

Now, introduce a fine solver $\mathcal{F}$ and coarse solver $\mathcal{G}$ as follows. Let $\mathcal{F}(U_n)$ be an accurate approximation to the solution $\phi_{\Delta T_n}(U_n)$ and $\mathcal{G}(U_n)$ be a less accurate approximation to $\phi_{\Delta T_n}(U_n)$. For example, $\mathcal{G}$ may be defined on a coarser grid than $\mathcal{F}$ or be a lower order method than $\mathcal{F}$. If we use the fine solver to approximate $\phi_{\Delta T_{n-1}}(U_{n-1}^k)$ in (4) and the coarse solver to approximate the Jacobian term in (4), the parareal algorithm to solve (1) is given by, (Gander and Hairer 2014),:

$$\begin{aligned} U_0^{k+1} &= u_0 \\ U_n^{k+1} &= \mathcal{G}(U_{n-1}^{k+1}) + \left[\mathcal{F}(U_{n-1}^k) - \mathcal{G}(U_{n-1}^k)\right] \end{aligned} \tag{5}$$

so that $U_n^k \approx u(T_n)$, where k represents the k^{th} iteration of the algorithm. See (Gander, Hairer, et al. 2008) for a general convergence theory. The term in brackets in (5) can also be seen as a correction term and highlights how the fine solver can be used in parallel at each iteration to correct the approximation made by the coarse solver.

2.2 Phase Plane Map

Suppose now that the time domain has been divided into N equal subdomains so that $\Delta T = (T_N - T_0)/N$. This will be the step size over which our new coarse solver operates. Consider a set of M initial conditions of (1) at time $t = t_0$, denoted by $\{u_0^i\}_{i=1}^M$, and a set of M corresponding target points $\{v^i\}_{i=1}^M$ defined by,

$$v^i = \phi_{\Delta T_0}(u_0^i) \qquad i = 1, \ldots, M\,. \tag{6}$$

As in the derivation of the parareal algorithm, we let $\mathcal{F}(u_0^i)$ approximate $\phi_{\Delta T_0}(u_0^i)$ so that, computationally,

$$v^i = \mathcal{F}(u_0^i) \qquad i = 1, \ldots, M\,. \tag{7}$$

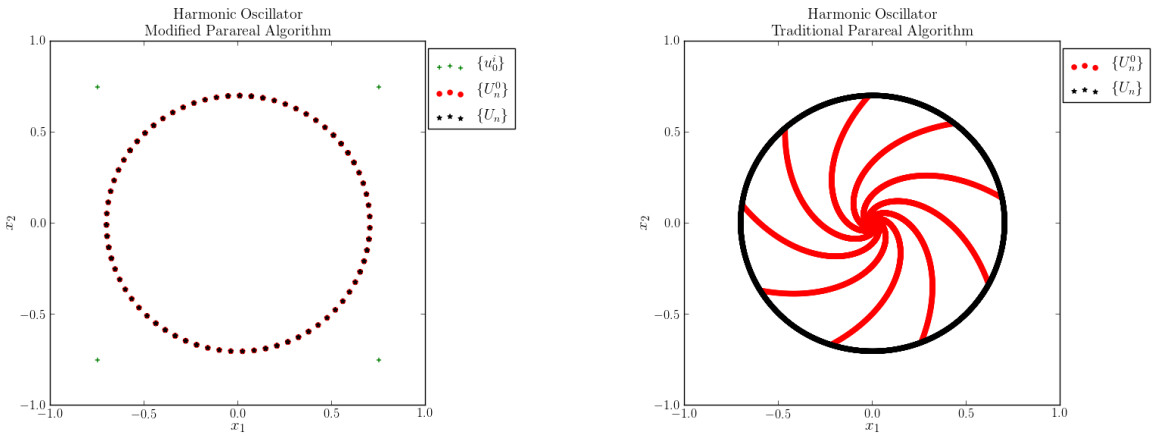
Together, the set $\left\{u_0^i \to v^i\right\}_{i=1}^M$ acts as a type of look-up table for future initial conditions; given a new initial condition and the mapping $\left\{u_0^i \to v^i\right\}_{i=1}^M$, it is possible to estimate a solution using, e.g., interpolation. Formally, we define a phase plane map $\mathcal{G}^{map}(\left\{u_0^i \to v^i\right\}_{i=1}^M, U_n)$ to be a coarse approximation of $\phi_{\Delta T}(U_n)$ that uses the information contained in $\left\{u_0^i \to v^i\right\}_{i=1}^M$ to determine the solution U_{n+1} at time T_{n+1}, given a point U_n at time T_n. $\mathcal{G}^{map}$ may be defined through interpolation on the set $\left\{u_0^i \to v^i\right\}_{i=1}^M$, or through a neural network where the network is trained using $\left\{u_0^i\right\}_{i=1}^M$ as the inputs and $\left\{v^i\right\}_{i=1}^M$ as the outputs. For autonomous dynamical systems like those commonly found in molecular dynamics or celestial mechanics, $\mathcal{G}^{map}$ can be applied over each subdomain Ω_n. For non-autonomous dynamical systems, one can easily extend this idea by generating a set $\left\{u_j^i \to v^i\right\}_{i=1}^M$ for each subdomain Ω_j, $j = 0, \ldots, N-1$, after some initial coarse approximation of the solution is obtained, as in (Nievergelt 1964). Moreover, the computation of $\left\{u_0^i \to v^i\right\}_{i=1}^M$ is embarrassingly parallel and can be done as a preprocessing step to the parareal algorithm, i.e., each v^i is computed independently using the fine solver over Ω_0.

There are several trade offs and design considerations that need to be addressed when building $\mathcal{G}^{map}$. The numerical examples presented in section 3 highlight initial findings for most of these considerations; however, many of them require further investigation. The first problem that needs to addressed is how to select the number of necessary initial conditions M used to build the set $\left\{u_0^i \to v^i\right\}_{i=1}^M$. Clearly, the higher M the more accurate $\mathcal{G}^{map}$ can be; however, it may be computationally expensive to generate a suitable mapping (even though the computation is highly parallel). Nievergelt (1964) suggests that a reasonable choice for M would be one such that the interpolation error is of the same order of magnitude as the truncation error of the fine scale method, $\mathcal{F}$. In a parareal setup, one might expect that M should be chosen so that the interpolation error is comparable to the truncation error of a suitable coarse solver in order for the scheme to provide meaningful results. Furthermore, one must discuss the bounds within which to place these initial conditions $\left\{u_0^i\right\}_{i=1}^M$. In practice, we have used two methods for determining appropriate constraints on the initial conditions. First, domain specific knowledge can often times provide meaningful bounds on the physical phenomenon be simulated. For example, when simulating two atoms bound by a molecular potential, the bounds can be determined from the actual initial condition of interest, U_0, which specifies the positions of the atoms and the total energy in the system. Secondly, for other problems where such information may be difficult to obtain, a coarse approximation to the solution's trajectory can be obtained. For example, if one is interested in simulating one hundred revolutions of a harmonic oscillator, a coarse solver may be used to estimate just one revolution of the oscillator, and appropriate bounds on $\left\{u_0^i\right\}_{i=1}^M$ can be determined from there. Lastly, the efficiency of the algorithm relies on choosing a suitable method for $\mathcal{G}^{map}$. For instance, interpolation may work well for problems in low dimensions, whereas a neural network approach may be better for problems in high dimensions where interpolation is more costly.

3 NUMERICAL EXAMPLES

3.1 Nomenclature

Throughout the numerical examples we will let T be the total simulation time, N be the number of time subdomains, and M be number of grid points in each dimension so that $\mathcal{G}^{map}$ is defined with $\{u_0^i \to v^i\}_{i=1}^{M^d}$. The fine solver used in each computation is SciPy's adaptive RK45 scheme (Jones, Oliphant, et al. 2001). We refer to the *traditional* parareal algorithm as a parareal scheme where RK4 is used as the coarse solver



(a) (b)

Figure 1: (a) Results of the modified parareal scheme applied to (8). (b) Results of the traditional parareal scheme applied to (8). Recall that $\{u_0^i\}$ are the points used to define $\mathcal{G}^{map}$, $\{U_n^0\}$ is the initial approximation made by the coarse solver, and $\{U_n\}$ is the final converged solution.

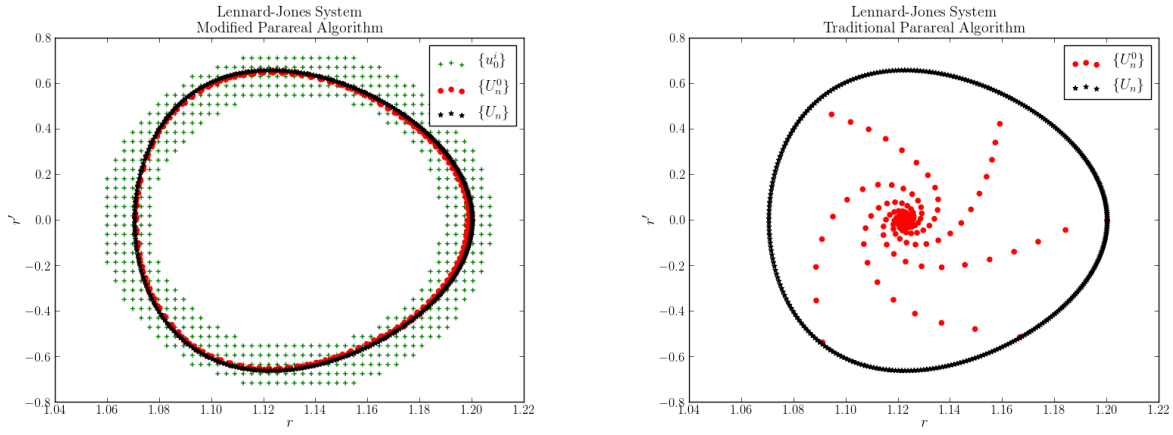
with time step T/N, and the *modified* parareal algorithm as that when the coarse solver is defined through $\mathcal{G}^{map}$.

3.2 Harmonic Oscillator

Following similar steps to those in (Nievergelt 1964), it is possible to prove the modified parareal algorithm with $\mathcal{G}^{map}$ defined through linear interpolation will converge in just one iteration for any linear dynamical system. Problems such as a harmonic oscillator that typically present a challenge for parareal schemes, can now be solved in only one iteration. To demonstrate this speedup, consider the following system:

$$\begin{aligned} \dot{x_1} &= \frac{1}{\varepsilon} x_2 \\ \dot{x_2} &= \frac{-1}{\varepsilon} x_1 \end{aligned} \tag{8}$$

with $\varepsilon = 0.01$ and the initial value $\mathbf{x}(0) = [-2/9, -2/3]^T$. We are interested in the long time solution of this problem, so we set $T = 70$ which corresponds to 1000 revolutions of the harmonic oscillator. Setting $M = 2$ and $N = 100$ and applying the modified parareal algorithm to (8) produces the results show in Figure 1a, and converges in just one iteration. In Figure 1a, the converged solution $\{U_n\}$ and the initial approximation made by the coarse solver, $\{U_n^0\}$, lie on top of each other, because the initial approximation made by $\mathcal{G}^{map}$ is exact. In fact, the results would be the same for any value of N. Recall that N sets the step size of the coarse solve, $\Delta T = T/N$. This is not the case if the traditional parareal scheme is applied to (8). For N smaller than around 10000, the algorithm diverges. For $N = 10000$, the algorithm converges in 42 iterations; however, the coarse step size is almost as small as the fine step size, so any speedup from the parareal algorithm is lost. The results of the traditional parareal algorithm are shown in Figure 1b; the initial approximation made by the coarse solver is far away from the true solution so many more parareal iterations are needed for convergence.



(a) (b)

Figure 2: (a) Results of the modified parareal scheme applied to (10). (b) Results of the traditional parareal scheme applied to (10).

These results for a simple harmonic oscillator motivate the study of physical systems that behave in a similar manner, such as those found in molecular dynamics or celestial mechanics.

3.3 Colinear Lennard-Jones System

Consider two colinear atoms bound by the Lennard-Jones Potential:

$$v(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right], \tag{9}$$

where r is the separation of the atoms, ε is the 'well depth' and σ is the distance at which the potential is zero. We set $\varepsilon = 1$ and $\sigma = 1$, which gives a minimum potential energy at $r^* = 1.12$. The Lennard-Jones potential is shown in Figure 3. The equation governing this system is given by:

$$\ddot{r} = \frac{2}{m} F(r) \tag{10}$$

where $F(r) = -\nabla v(r)$. Near the minimum potential energy, equation (10) behaves very similar to a harmonic oscillator. Further away, the system behaves nonlinearly.

First we set the initial condition to $\mathbf{r}_0 = [1.2, 0]^T$, $T = 50$, and $N = 400$. This corresponds to around 100 oscillations of the atoms. Applying the traditional parareal method to (10) gives the results shown in Figure 2b. As in the case of the harmonic oscillator, using a standard RK4 scheme as the coarse solver makes a poor initial approximation of the true solution, which results in a larger number of iterations needed for convergence; in this case, 53 iterations. The modified parareal algorithm with $\mathcal{G}^{map}$ defined through linear interpolation with $M = 50$ converges in just 10 iterations when applied to (10). Figure 2a shows the results. The phase plane map makes a very accurate initial approximation of the solution, so the fine solver only needs a few iterations of corrections to converge to the true solution.

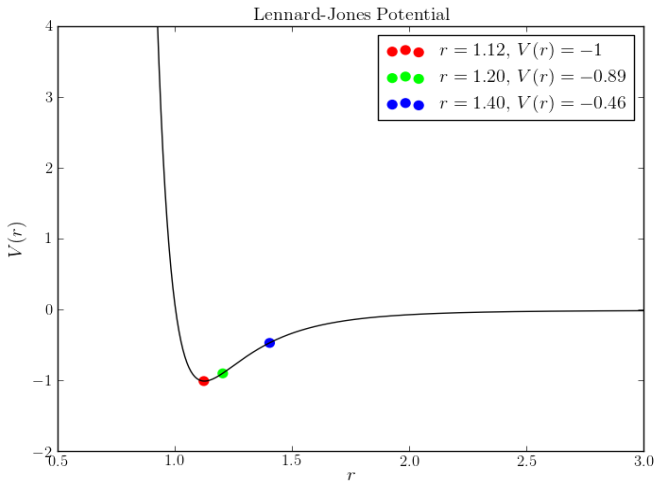


Figure 3: Lennard-Jones potential given by (9).

Unlike the linear harmonic oscillator, the efficiency of the modified parareal algorithm can be affected by defining the phase plane map with more or less training and target points $\{u_0^i \to v^i\}_{i=1}^{M^d}$. Figure 4 indicates how the number of parareal iterations needed for convergence decreases to one as the number of grid points gets arbitrarily large. Since the computation of $\{u_0^i \to v^i\}_{i=1}^{M^d}$ is embarrassingly parallel and done in a pre-processing step, the only adverse cost in increasing M comes from how one defines $\mathcal{G}^{map}$, e.g., interpolation will be more expensive as M increases.

The nonlinearity of the Lennard-Jones system also affects the performance of the modified parareal algorithm. Consider the three separation distances shown in Figure 3, $r = 1.12, r = 1.2$ and $r = 1.4$. As r increases, (10) becomes increasingly nonlinear. Table 1 shows how this nonlinearity affects the efficiency of the modified parareal algorithm. As r increases, so must M in order to maintain roughly the same number of parareal iterations for convergence.

Table 1: Tables showing how the nonlinearity of the Lennard-Jones system affects the efficiency of the modified parareal algorithm. Selected r values correspond to those found in Figure 3.

(●) $r = 1.12$

M	Iterations
3	7
5	3
10	2
50	1

(●) $r = 1.2$

M	Iterations
10	19
50	10
100	8
500	4

(●) $r = 1.4$

M	Iterations
50	207
100	39
500	14
1000	9

As in the simple harmonic oscillator case, one can improve the results of the traditional parareal algorithm by making the step size for the coarse solver smaller and smaller; however, as the coarse step size approaches the fine step size, any speed up of the parareal algorithm is lost. With the phase plane map used as the coarse solver, the coarse step size can stay significantly larger than the fine step size, while still achieving good results. For example, suppose we increase the total time in the Lennard-Jones simulation shown in Figure 2 to $T = 500$ and take $M = 1000$. If we use the phase plane map as the coarse solver and take $N = 400$, the parareal algorithm converges in 14 iterations. If we were to use the traditional parareal algorithm and also wanted convergence in 14 iterations, we must take $N \approx 50,000$. At this point, the coarse solver is

almost as expensive as the fine solver. Furthermore, N is typically chosen to be the number of cores used for the parareal algorithm, which implies less cores are needed for the modified parareal algorithm than the traditional parareal algorithm to achieve the same efficiency.

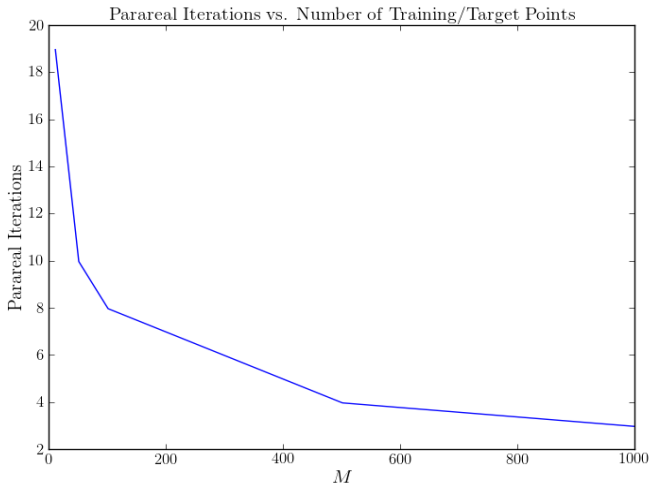


Figure 4: The number of the parareal iterations needed for convergence vs. the number of points used to define $\mathcal{G}^{map}$.

3.4 High Dimensional Harmonic Oscillator

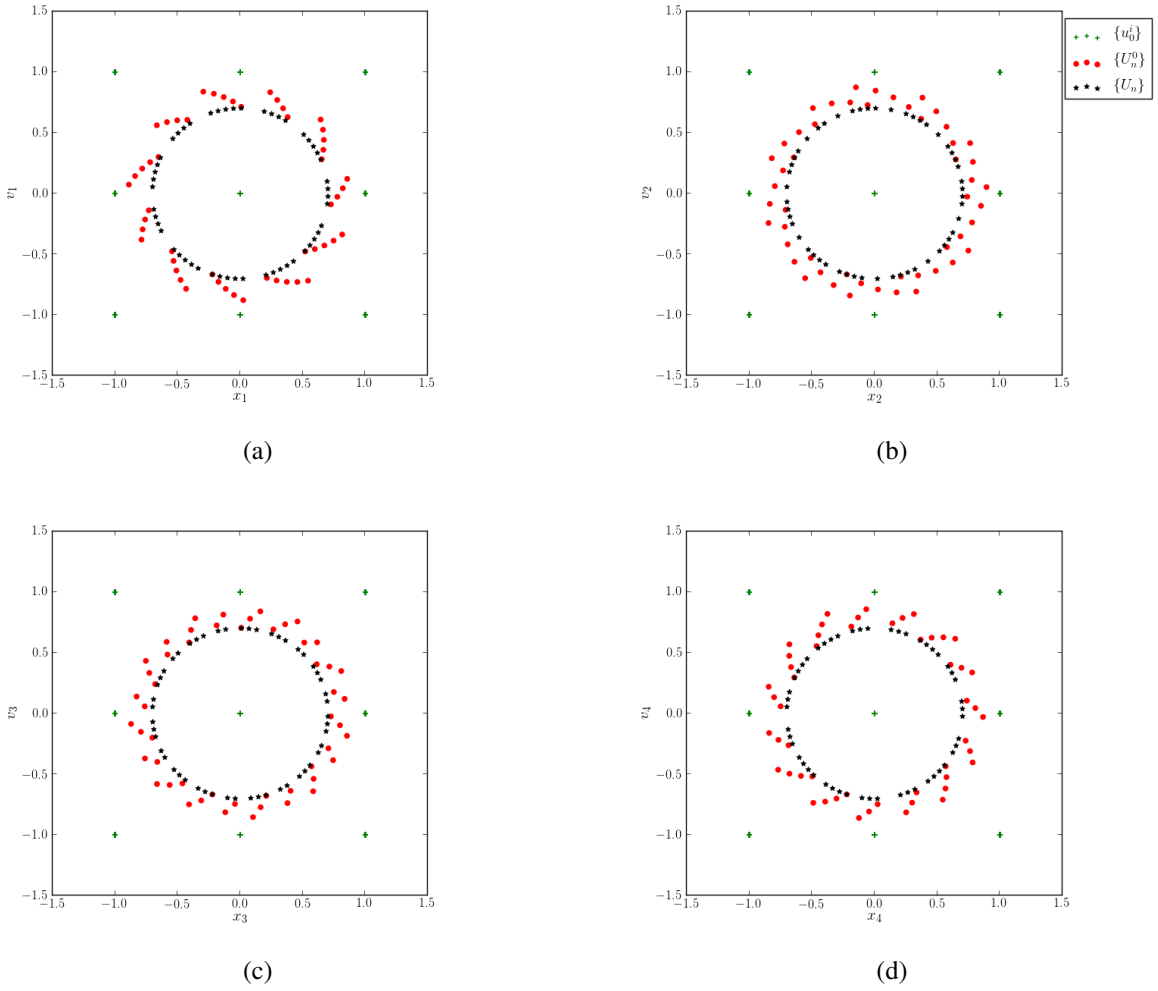
In high dimensions defining $\mathcal{G}^{map}$ through interpolation may be infeasible. This presents an interesting opportunity to make use of neural networks. As a testing case, consider a high dimensional system of harmonic oscillators:

$$\dot{\mathbf{x}} = \frac{1}{\varepsilon} A\mathbf{x} \tag{11}$$

where $x = [x_1, v_1, x_2, v_2, x_3, v_3, x_4, v_4]^T$, $\varepsilon = 0.01$, and $A \in \mathbb{R}^{8\times 8}$ is defined as,

$$A = \begin{bmatrix} 0 & a & 0 & 0 & 0 & 0 & 0 & 0 \\ -a & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b & 0 & 0 & 0 & 0 \\ 0 & 0 & -b & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & c & 0 & 0 \\ 0 & 0 & 0 & 0 & -c & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & d \\ 0 & 0 & 0 & 0 & 0 & 0 & -d & 0 \end{bmatrix} \tag{12}$$

for parameters a, b, c, d. Since this system is linear, the modified parareal algorithm with $\mathcal{G}^{map}$ defined through interpolation will converge in just one iteration for any N and $M \geq 2$. However, high dimensional interpolation is quite expensive, especially if we were to consider a nonlinear problem.



(a) (b)

(c) (d)

Figure 5: (a) Results of the modified parareal scheme applied to (11) with $\mathcal{G}^{map}$ defined with a neural network.

An alternative approach is to define $\mathcal{G}^{map}$ through a neural network. Below we test this approach on (11) setting $T = 10$, $N = 50$, $M = 3$, and take $a = 1, b = 2, c = 3$, and $d = 4$. The neural network is defined with a logistic activation function and one hidden layer of size 1000; a limited memory BFGS solver is used for optimization (Pedregosa, Varoquaux, et al. 2011). The results of this scheme applied to (11) are shown in Figure 5. The algorithm converges in 6 iterations, as opposed to one iteration when linear interpolation was used, however, the neural network approach converges in roughly a third of the time. The initial approximation of the neural network, while not exact, is still very close to the true solution, so only a few number of parareal iterations are needed for correction. Moreover, the neural network can be improved by increasing M, as in section 3.3, and has the additional advantage that the cost of applying the neural network does not increase with M, as opposed to the case when interpolation is used. This makes defining $\mathcal{G}^{map}$ through a neural network particularly advantageous for high dimensional problems, as all additional cost is made in the preprocessing step, which is embarrassingly parallel. Using the output of a neural network as a macroscopic model and coupling the result with a fine scale microscopic model for corrections is an interesting approach for solving physical systems in general, and may help bridge the gap between data science and computational science in the future.

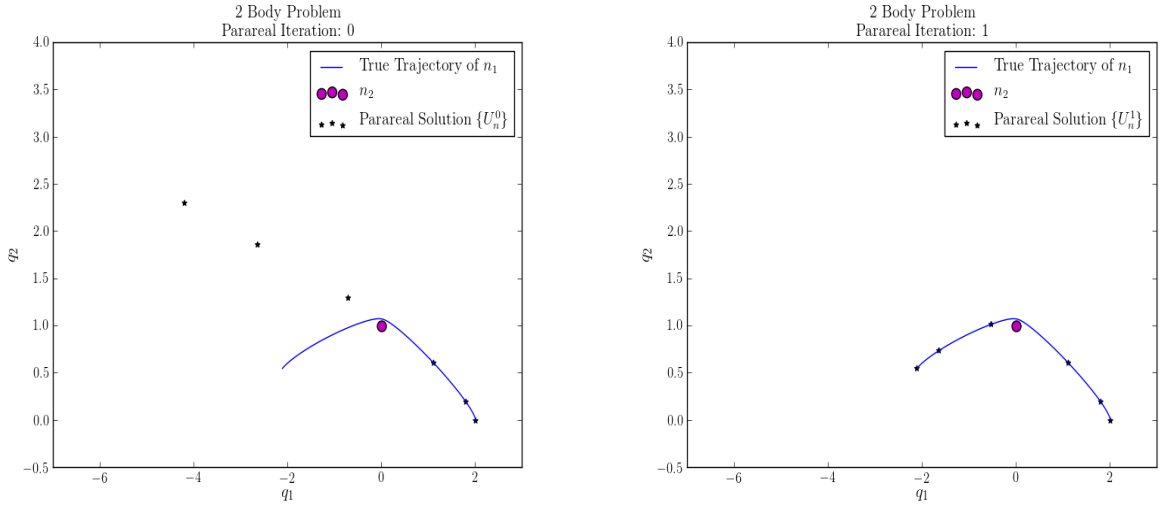
3.5 Localized Multiscale Problems

E, Engquist, et al. (2003) and E (2011) describe two types of multiscale problems. The first type are problems that contain local defects or singularities so that a macroscale model is sufficient for most of the physical domain, and a microscale model is needed only near the defects. These type of problems are known as *type A* multiscale problems. The second type of multiscale problems are those for which a microscale model is needed everywhere. These are known as *type B* multiscale problems. In the previous sections we have only consider type B multiscale problems, and in fact, this is the situation in which most parareal algorithms are applied, since the fine solver and coarse solver are coupled everywhere in the domain. The coarse solver acts as the macroscale solver and picks up on slow scale motion, while the fine solver acts as the microscale solver and corrects for fast scale motions. An important note can be made about the parareal algorithm in general applied to type A multiscale problems.

A good example of a type A multiscale problem can be found in celestial mechanics. In the context of an n-body problem, let m_i denote the mass of the i^{th} body, and $\mathbf{q}_i$ denote the position of the i^{th} body. The equation governing the motion of the i^{th} body is

$$m_i \ddot{\mathbf{q}}_i = \sum_{j \neq i}^{n} \frac{m_i m_j (\mathbf{q}_j - \mathbf{q}_i)}{\|\mathbf{q}_j - \mathbf{q}_i\|^3} \tag{13}$$

For illustrative purposes, let $n = 2$ in (13) and denote the two bodies as n_1 and n_2. Assume that $m_2 >> m_1$ so that $\mathbf{q}_2$ is essentially fixed. When the first body is far away from the second body, a macroscopic solver is accurate enough to predict its motion. However, as the two bodies get closer, a fine scale solver is needed to resolve their interactions. This situation is well-suited for the parareal algorithm. The coarse solver will be accurate enough in all regions far from n_2, so that the only corrections needed occur in regions where n_1 and n_2 are close. We apply a traditional parareal algorithm to (13) where the fine scale solver is an adaptive RK45 scheme and the coarse solver is a RK4 scheme with large step size. If we set $T = 0.5$ and $N = 5$, the parareal algorithm converges in just two iterations. The true solution and the parareal solution



(a) (b)

Figure 6: (a) Initial coarse approximation given by the parareal algorithm applied to (13). (b) Results after one correction step of the parareal algorithm applied to (13).

for the trajectory of n_1 after each iteration is shown in Figure 6. The initial approximation made by the coarse solver (shown in Figure 6a) accurately predicts the motion of n_1 during the first two time intervals Ω_0 and Ω_1. During Ω_3, n_1 interacts with n_2 and alters its trajectory. The coarse solver does not detect this interaction due to the large step size, and incorrectly predicts the motion of n_1 for Ω_3, Ω_4 and Ω_5. After one iteration of the parareal solution, the fine solver corrects for the interaction between the two bodies during Ω_3, and when the coarse solver is run again, it is able to accurately predict the motion of n_1 far from n_2 as before. Therefore, the solution converges in just one iteration of the parareal algorithm. The fine solver is only needed to detect the localized interaction of n_1 and n_2.

4 CONCLUSION

We have presented the methodology behind a novel coarse scale solver for the parareal computation of highly oscillatory dynamical systems. This paper can be seen as a proof of concept. Further research is required to improve this approach into a fully developed robust and successful technique. Potential improvements are using modern sparse grid and adaptive methods for the interpolation, or optimizing the approach for neural networks. We plan on performing numerical tests of large systems with more realistic examples in the realm of molecular dynamics as well as determining scaling results on modern supercomputing platforms, even if the embarrassingly parallel nature of the algorithms should generate predictable scaling.

REFERENCES

Barker, A. T. 2014. "A minimal communication approach to parallel time integration". *International Journal of Computer Mathematics* vol. 91 (3), pp. 601–615.

E, W. 2011. *Principles of multiscale modeling*. Cambridge University Press.

E, W., B. Engquist et al. 2003. "The heterognous multiscale methods". *Communications in Mathematical Sciences* vol. 1 (1), pp. 87–132.

Gander, M. J. 2015. "50 years of time parallel time integration". In *Multiple Shooting and Time Domain Decomposition Methods*, pp. 69–113. Springer.

Gander, M. J., and E. Hairer. 2014. "Analysis for parareal algorithms applied to Hamiltonian differential equations". *Journal of Computational and Applied Mathematics* vol. 259, pp. 2–13.

Gander, M. J., E. Hairer et al. 2008. "Nonlinear convergence analysis for the parareal algorithm". *Lecture Notes in Computational Science and Engineering* vol. 60, pp. 45.

Jones, E., T. Oliphant et al. 2001. "SciPy: Open source scientific tools for Python".

Lions, J.-L., Y. Maday et al. 2001. "A parareal method in time discretization of pde's". *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics* vol. 332 (7), pp. 661–668.

Nievergelt, J. 1964. "Parallel methods for integrating ordinary differential equations". *Communications of the ACM* vol. 7 (12), pp. 731–733.

Pedregosa, F., G. Varoquaux et al. 2011. "Scikit-learn: Machine Learning in Python". *Journal of Machine Learning Research* vol. 12, pp. 2825–2830.

Ying, L., and E. J. Candès. 2006. "The phase flow method". *Journal of Computational Physics* vol. 220 (1), pp. 184–215.

AUTHOR BIOGRAPHIES

GOPAL R. YALLA is a Graduate Student at the Institute for Computational Engineering and Sciences at the University of Texas at Austin. His research interests lie in multiscale modeling and simulation, turbulence modeling, high performance computing, and parallel algorithms. gopal@ices.utexas.edu.

BJORN ENGQUIST holds the Computational and Applied Mathematics Chair I at the Institute for Computational Engineering and Sciences (ICES) at the University of Texas at Austin, and is director of the ICES Center for Numerical Analysis. Before coming to ICES, Engquist received his Ph.D. in numerical analysis from Uppsala University, was a professor of mathematics at UCLA, and the Michael Henry Stater University Professor of Mathematics and Applied and Computational Mathematics at Princeton University. He was also the director of the Research Institute for Industrial Applications of Scientific Computing and of the Centre for Parallel Computers at the Royal Institute of Technology, Stockholm. Engquist's research focuses on the development and analysis of numerical methods for differential equations, computational multiscale methods, and fast algorithms for wave propagation with applications in seismology. engquist@ices.utexas.edu.