Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm

D. Samaddar [a,*], D.E. Newman [a], R. Sánchez [b]

[a] Department of Physics, University of Alaska Fairbanks, Fairbanks, AK 99775-5920, USA
[b] Fusion Energy Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-8070, USA

A B S T R A C T

It is shown that numerical simulations of fully-developed plasma turbulence can be successfully parallelized in time using the parareal algorithm. The result is far from trivial, and even unexpected, since the exponential divergence of Lagrangian trajectories as well as the extreme sensitivity to initial conditions characteristic of turbulence set these type of simulations apart from the much simpler systems to which the parareal algorithm has been applied to this day. It is also shown that the parallel gain obtainable with this method is very promising (close to an order of magnitude for the cases and implementations described), even when it scales with the number of processors quite differently to what is typical for spatial parallelization.

1. Introduction

In magnetically confined hot plasmas with interest for fusion research, such as those confined in a tokamak or a stellarator, the disparity in timescales between the microturbulence responsible for the transport processes and the plasma confinement times is of the order of 10^6–10^9. Thus, in order to properly capture the transport dynamics of these strongly turbulent plasmas, they would need to be simulated for thousands or even tens of thousands of turbulent decorrelation times. Such a task is currently well beyond the reach of even the most powerful supercomputers. As a result, simplified approaches are used in which the microturbulence is only evolved for a few tens of decorrelation times under the assumption that the dynamics become decorrelated after this timescale, together with the assumption that plasma profiles can be assumed 'frozen' during these timescales. These approximations enable a description via effective transport coefficients derived from these restricted simulations, whose validity should however be checked *a posteriori*, whenever possible. But even with these simplifying approximations, these simulations are still extremely challenging from a computational point of view [1]. On the one hand, because parallelization along the space domain reaches saturation (typically, at a few thousand CPUs in most codes, as a result of Amdahl's law and inter-processor communication overflow), for which a further increase of the number of processors beyond a certain point does not contribute to enhanced speed-up. On the other hand, higher spatial resolutions imply the use of smaller time steps for numerical stability reasons. This makes that reaching the needed simulation times takes much longer, due to the serial nature of the temporal coordinate. The situation becomes even more dramatic in cases in which the aforementioned simplifications may not be justified, as might be the case when turbulence is near-marginal [2], or in the presence of strong sheared flows [3].

* Corresponding author. Tel.: +1 907 474 1938; fax: +1 907 474 6130.
E-mail address: dsamaddar@alaska.edu (D. Samaddar).

It thus seems self-evident that if these turbulent simulations could also be parallelized in time (in addition to their spatial parallelization), it would enable a more efficient utilization of the many processors currently available in supercomputers while, at the same time, allowing the running of much longer simulations in considerably shorter wallclock times. This would also open the path to including a more complete set of the physics in the simulations in the near future, maybe even the full transport dynamics. Various approaches have been proposed over the years to decompose the time direction when solving a partial differential equation [4–13], although with varying degrees of success. Of these, the parareal (parallel in time) algorithm, which we explore in this paper, was first proposed by Lions et al. [14] and has received an increasing amount of attention in recent years. It has been successfully applied to a number of relatively simple problems, like molecular dynamics simulations [15], linear and nonlinear parabolic ordinary differential equations [16,17], stochastic ordinary differential equations [18], reservoir simulations [19] and even, the laminar regime of the Navier–Stokes equation [20]. The scheme has also been applied very recently to the Princeton ocean model, dominated by convection, although with a rather modest success [21].

In this paper, we report the first (and very promising) results regarding the parallelization of the temporal direction of numerical simulations of turbulent plasmas using the parareal algorithm. To the best of our knowledge, the parareal technique has never been applied to a fully-developed turbulent problem, although it has been used successfully with low-dimensional chaotic systems, such as the Lorenz system [22]. Turbulent systems represent a very challenging case of study, and there are in fact many reasons to expect failure. Indeed, the parareal algorithm parallelizes along time, despite the sequential nature of the time domain, using a predictor–corrector approach. Since the corrector step carried out at each cycle of the algorithm feeds on the results of the uncoupled predictor runs, it might be expected that the strong sensitivity to initial conditions and exponential divergent growth of uncertainties characteristic of turbulence should deteriorate or even impede the convergence of the algorithm, in contrast to what happens in a more laminar regime. In this paper we show that this is not the case, and that the parareal method, when properly tuned, can be applied successfully to fully-developed turbulent simulations and yields considerable parallel speed-ups (an order of magnitude, for the cases studied here).

In order to avoid the complexities associated with the toroidal geometries characteristic of fusion plasmas, we have chosen to apply the parareal method to a simpler dissipative-trapped electron mode (DTEM) turbulence model in a doubly-periodic slab geometry. Some kind of drift wave turbulence is the most probable candidate for governing transport in these plasmas and thus, this model has been studied extensively [23–26]. For its numerical implementation we use the BETA code [23], which uses a pseudo-spectral approach and advances the system in time using an implicit, preconditioned integrator. BETA routinely provides with fully-developed turbulent states, with large positive Lyapunov exponents. It thus provides an excellent platform to explore and test the merits of the parareal method in this context, before embarking on its implementation in any of the state-of-the-art codes used by the fusion community. The paper is thus organized as follows. Section 2 briefly reviews the parareal algorithm. The physics of the DTEM model are then described in Section 3. Section 4 introduces an analysis of the parallel performance to be expected from the algorithm, which will clarify its strong and weak points as well as guide us through its tuning. Next, Section 5 comprises the numerical results obtained in this study. Finally, some conclusions are drawn in Section 6.

2. The parareal algorithm

In this section, we provide a review of the basic algorithm, including some modifications of our own devising that are appropriate to the turbulent context and yield a significant boost in its performance.

2.1. Review of the algorithm

The parareal algorithm is based on a predictor–corrector iterative approach. It is best understood by describing its application to a single ordinary differential equation of the type:

$$\frac{d\lambda}{dt} = A(\lambda, t), \quad \lambda(0) = \lambda_0, \tag{1}$$

where A is an arbitrary, possibly nonlinear, function. Let us assume that we are interested in finding the value of λ at some later time $T > 0$. Let us also assume that we can numerically advance this equation from an arbitary time $t \in [0,T]$ to time $t + dt$ by means of several discretization schemes. We will formally write this advance using;

$$\lambda(t + dt) = \mathbf{F}^t_{dt} \cdot \lambda(t), \tag{2}$$

where $\mathbf{F}$ represents our advancing operator, acting on the appropriate space to which λ belongs, and dependent on the discretization scheme chosen (the superscript t is used here to denote that the operator may depend explicitly on t, although we will drop it in what follows). Clearly, to go from the initial time to T, we will need to apply $\mathbf{F}$ as many times as required given the value of dt.

The parareal algorithm assumes that there are two different advancing operators (or solvers) at our disposal, that we will denote as $\mathbf{F}$ and $\mathbf{G}$. The distinction between these two solvers is that $\mathbf{G}$ is much faster (usually at the price of being coarser and more inaccurate) than $\mathbf{F}$, the one we are really interested in using for our problem but which is computationally too expen-

sive to be run serially between the initial time and T. In the parareal algorithm, **G** is run serially between $t = 0$ and T, whilst **F** is always run in parallel. Let us see how this works.

Let there be N processors, denoted by $P_0, P_1, P_2, P_3, \ldots, P_{N-1}$. Let the total simulation time, T, be divided into N smaller *chunks* of size $\Delta T = T/N$. In what follows, the i index is used to represent the ith instant of time defined as $t_i = i \cdot \Delta t$ for $i = 0, 1, 2, \ldots, N$. The index $k = 0, 1, 2, \ldots$ represents the iteration number in the parareal cycle. Then, λ_i^k represents the solution at time t_i at the kth iteration of the parareal cycle. The initial value, that is already given, is then denoted by λ_0^0.

The steps involved in the parareal scheme are as follows:

- Iteration $k = 0$:
  P_0 uses **G** serially to calculate initial values λ_i^0 for the start time of every time chunk, t_i.
- Iteration $k > 0$:
  Step 1 : Each processor (i.e., P_j) then separately applies **F** to propagate the solution, starting with the initial values provided by **G** (i.e., λ_j^{k-1}), between the initial (t_j) and final time (t_{j+1}) of its respective time chunk. This process is of course carried out in parallel. The result of this propagation is forwarded to the next processor in line (P_{j+1}).
  Step 2 : **G** is now applied as a sequential (but not continuous) process, using the parareal prescription to update the initial value at each time chunk:

$$\lambda_{i+1}^{k+1} = \mathbf{G}_{\Delta T}(\lambda_i^{k+1}) + \mathbf{F}_{\Delta T}(\lambda_i^k) - \mathbf{G}_{\Delta T}(\lambda_i^k). \tag{3}$$

  Note that this part of the algorithm cannot be done in parallel, because of the first term on the right hand side. Note also that the second and third terms have already been obtained in previous steps and/or iterations.
  Step 3 : Check for convergence. The measure of convergence is discussed in the next subsection. If the solution is converged for all chunks, the cycle is exited. Otherwise, another iteration of the parareal cycle is done which, in the standard implementation, involves all chunks. This will not be the case in our simulations, as discussed in the next subsection.

Some remarks are useful at this point. First, in order for the parareal cycle to converge, certain mathematical conditions must be satisfied by **G** and **F**, which were made explicit in Lions et al. seminal paper and appear as specific conditions on the boundedness of the norm of the difference between the two solvers in an appropriate mathematical space [14]. Regretfully, it is very difficult (if not impossible) to translate these conditions into a practical prescription for any particular problem. Trial and error, combined with experience, seems to be the most reliable guide to choose **G**. However, note that in practice, the parareal algorithm will always converge in at most N interactions, independently of how 'badly' **G** is chosen. This happens because, at the end of iteration $k = 1$, both P_0 and P_1 already have the correct value of the solution, since **F** has been used in the first processor to propagate the exact initial condition at $t = 0$. For the same reason, at $k = j$, all processors $P_0, \ldots, P_j$ already have the correct value. And so on. However, note also that if N cycles are finally needed to converge to the exact solution, we would have used the same (or more, including communications and time for **G**) wallclock time that if we had run the simulation serially with **F**. So no parallel speed-up is gained, and we would have done N times more computing work! Thus, parareal works only if convergence is achieved for a number of cycles K much smaller than N, implying that at each iteration more than one chunk needs to converge on average. Whether this is the case or not will depend on our ability to choose **G** for a given **F**.

2.2. Metric for convergence

Convergence is achieved whenever some convergence measure reduces below a certain prescribed tolerance. Following with our previous example, in which $\lambda^k(t)$ represents the solution at time t in the kth iteration of the parareal algorithm, we define the local convergence measure as,

$$\sigma_i^k = \int_{t_{i-1}}^{t_i} \frac{\left|\lambda^k(s) - \lambda^{k-1}(s)\right|}{\left|\lambda^{k-1}(s)\right|}\, ds. \tag{4}$$

That is, the average relative error between the solution at the kth and the $(k-1)$th parareal cycles integrated over each chunk i. It should be noted that we have purposely avoided using the exact serial solution that **F** would provide, since it is not useful in practice due to the fact that, in most cases of interest it will not be known, since the main aim of the parareal scheme is to replace calculations involving serial processors! The solution is then converged up to time chunk I if,

$$\sigma_i^k < \text{tol}, \quad \forall i \leqslant I. \tag{5}$$

Typically, convergence occurs in an orderly fashion, starting with the first chunk and propagating to later times due to the fact that errors are propagated (and enhanced) in that direction by the parareal cycle. This fact has prompted us to modify the standard parareal algorithm in the following way: instead of 'correcting' the solution at each parareal cycle starting with the first chunk, *we perform the 'correction' only for those chunks corresponding to times larger than the last converged chunk time*. In this way, one still obtains a reasonably converged solution, and avoids at the same time resonances that may deteriorate

the solution at already converged chunk times as observed in some problems [12]. With this modification, the cases that will be discussed in Section 5 have been made to converge using up to four times less parareal cycles than with the original prescription.

2.3. Particulars of the application of the parareal method to fully-developed turbulent simulations

We now discuss some crucial issues pertaining to turbulent systems which are relevant to the application of the parareal algorithm to the BETA simulations. First, it should be noted that for any numerical simulation of a very high dimensional chaotic system with positive Lyapunov exponents, it is impossible to define *a unique solution*. The reason is that these systems exhibit exponential Lagrangian (i.e., along turbulent trajectories) divergence and as a result the final solution depends very sensitively on the initial values. So sensitively that, in fact, a serial run done with the same parameters and numerical scheme, but compiled using two different compilers or run on two different machines unavoidably leads to different results when the simulation time is long enough. But note that this fact does not make the solutions worthless because even when different, they are *statistically identical*. What we mean by that is that the fluctuations exhibit the same statistical and correlation properties at saturation, even if the fluctuations are not identical when compared point-by-point at every time. That is, if one compares the probability distribution function (pdf) of the fluctuations at any point, or the temporal or spatial correlation functions, all these solutions will give the same results. The reason is ultimately the fact that the Eulerian (i.e., fixed point) fluctuations are bounded by the finite saturated fluctuation levels, which are themselves limited by the finite amount of free energy available to the system. This fact translates into the simulations having well-defined statistical properties. So, although the Lagrangian trajectories diverge due to their high sensitivity to initial conditions, the solutions are statistically indistinguishable. And this is ultimately what matters, specially when what one cares about is the mean transport or dynamics of these simulations. In the case of the parareal scheme just sketched, this observation is essential, because the scheme matches together solutions obtained at different chunks. If the mismatch at those connecting times could grow exponentially without bound, it might prevent any convergence of the method (in fact, this is why it has been suggested that turbulence and parareal would be a deadly mix). But on the contrary, the same boundedness that exists for Eulerian fluctuations holds also for these mismatches. As we will show in what follows, a solution can be found by the parareal method. And it is very easy to see that this parareal solution, although not identical to the serial one obtained starting from the same initial values, is statistically identical in the sense previously described.

A second important aspect regarding the parareal parallelization of turbulent simulation has to do with the convergence measure. As discussed in the previous subsection, the traditional parareal method computes the relative error between the solution of the problem at two successive parareal cycles. In the case of a turbulent simulation, the number of degrees of freedom is enormous (for instance, $\sim 6 \times 10^5$ for the BETA simulations used in this paper), which would make the construction of the convergence measure rather cumbersome. However, it turns out that this is not needed. In fact, we will show that it is enough to require convergence in one global quantity. In our case, the time history of the energy in the system, defined as:

$$E_\phi(t) = \sum_k \phi_k(t) \cdot \phi^*_{\mathbf{k}}(t), \tag{6}$$

where $\phi_k(t)$ is the (complex) amplitude of the Fourier harmonic with wave number k (see Section 3 for details). Then we will show that, if the relative error between successive cycles of the total energy is less than a prescribed tolerance, all the individual modes containing enough energy to affect that tolerance value are also converged. This is a remarkable result, which simplifies the calculation enormously. It is ultimately due to the fact that in a fully-developed turbulent system, the relative energy fluxes between any pair of modes are set by the non-linearities, which endows the system with a very strong and resilient coupling [27]. Thus, convergence in one quantity is only possible if all others converge simultaneously, so that the relations imposed by this coupling are preserved. We indeed observed this in our simulations, as will be shown later.

3. The model

The parareal algorithm is applied in what follows to numerical simulations of a reduced model, which is a paradigm of plasma drift waves derived in the limit of long wavelengths and a two-dimensional $(x - y)$ slab geometry. This limit of the DTEM model has been extensively studied in the plasma physics literature [23,25]. It assumes that a uniform magnetic field that confines the plasma exists perpendicular to the slab (i.e., along $\hat{z}$), and that all profile gradients are directed along the $\hat{x}$ direction. These gradients are responsible for the existence of drift waves that propagate along the $\hat{y}$ direction. The underlying instability is assumed to be the dissipative-trapped electron mode (DTEM). Plasma ions are treated as a cold fluid, while electrons are assumed to satisfy the Boltzmann relation except for those trapped in the magnetic ripple, which contribute to the dynamics in a way that allows the drift waves to grow unstable. The model can be reduced to a single equation for the potential fluctuations by assuming quasi-neutrality and using the long-wavelength limit to retain only the so-called $\mathbf{E} \times \mathbf{B}$ non-linearity:

$$\frac{\partial}{\partial t}(1 - \rho_s^2 \nabla_\perp^2)\tilde{\phi} + D\frac{\partial^2 \tilde{\phi}}{\partial y^2} + \frac{V_D}{2}\frac{\partial \tilde{\phi}}{\partial y} - \frac{4L_n D}{\epsilon^{1/2}}\left[\nabla_\perp\left(\frac{\partial \tilde{\phi}}{\partial y}\right) \times z\right] \cdot \nabla_\perp \tilde{\phi} = 0. \tag{7}$$

Here, $\tilde{\phi}$ is the fluctuating potential. $V_D = \epsilon^{1/2}(cT_i/eB)L_n^{-1}$ represents the effective diamagnetic drift velocity, T_i being the ion temperature, $\epsilon^{-1/2}$ the fraction of trapped electrons, B the magnetic field strength and L_n the density gradient scale length. $D = V_D^2/4\nu_{\mathrm{eff,e}}$, where $\nu_{\mathrm{eff,e}}$ is the effective collision frequency of ion–electron collisions. The last term on the left hand side of Eq. (7) represents the $\mathbf{E} \times \mathbf{B}$ non-linearity, the second one is the instability drive and the third one is the one which causes the drift waves to propagate along the y direction.

The numerical solution of this model is done using the spectral code BETA [23]. The Fourier transform of Eq. (7) has the form:

$$\frac{d\tilde{\phi}_k}{dt} - \frac{Dk_y^2}{1+\rho_s^2 k_\perp^2}\tilde{\phi}_k + \frac{iV_D k_y}{2(1+\rho_s^2 k_\perp^2)}\tilde{\phi}_k - \frac{4iL_n D}{\epsilon^{1/2}(1+\rho_s^2 k_\perp^2)}\sum_{\mathbf{k}'} k'_y(k \times k' \bullet z)\tilde{\phi}_k\tilde{\phi}_{k-k'} = S_k - T_k, \tag{8}$$

where $\tilde{\phi}_k$ are the (complex) Fourier harmonics of the potential. S_k are sources and T_k are sinks in k-space that may also be added at this point for convenience. This set of ordinary differential equations (ODEs) are solved using a pseudo-spectral method to evaluate the non-linearities. Time is advanced by the stiff solver within the VODPK package [28], which uses a (diagonally, right-) preconditioned, Krylov implicit solver based on backward differentiation formulas of variable order and timestep, that are tuned as the integration proceeds in order to keep the errors below a prescribed tolerance.

An example of a typical simulation is shown in Figs. 1 and 2. The first one shows a snapshot of the vorticity distribution in the system at saturation (the local turbulent velocity is obtained from the potential via $\mathbf{v} = \nabla_\perp\phi \times \hat{z}$, so that the vorticity is given by $w = \nabla_\perp^2\phi$). It illustrates a salient feature of a turbulent system – namely, the existence of eddies at multiple scales, which non-linearly interact with each other. Also, the anisotropy introduced by the explicit dependence on k_y of the model can be noticed. Fig. 2 is a plot of the power spectrum that shows that the system is in a state of fully-developed turbulence with most of the energy concentrated at low k values. If one computes the Lyapunov exponent of the system, it is large and positive as expected. This is illustrated in Fig. 3, that shows the log-linear plot of the separation of two initially close trajectories along the x and y directions with time, the slope of which gives the Lyapunov exponent.

4. Expected performance of the parareal algorithm

In order to gain some insight on what could be expected from the parareal algorithm (and, perhaps, also allow us to design an optimal coarse solver), we carry out in this section a small study on the performance of the model. Two regimes are examined: the so-called *strong* and *weak* scaling regimes. In the strong one, the performance as a function of the number of processors (denoted by N) is examined for a problem of fixed size in time. In the weak regime, the performance as a function of N is considered as the problem is made longer, but keeping the part of the simulation done by a single processor constant. In perfect parallelism, one should find that the speed-up in the strong scaling improves as N whilst, in the weak regime, one finds that the work-per-processor remains constant as N is increased. Of course, in spatial parallelization these ideal scalings are only maintained up to a certain number of processors, above which the serial part of the calculation (i.e. Amdahl's law) or the inter-processor communication dominate the calculation time. Due to the iterative nature of the parareal cycle, its strong

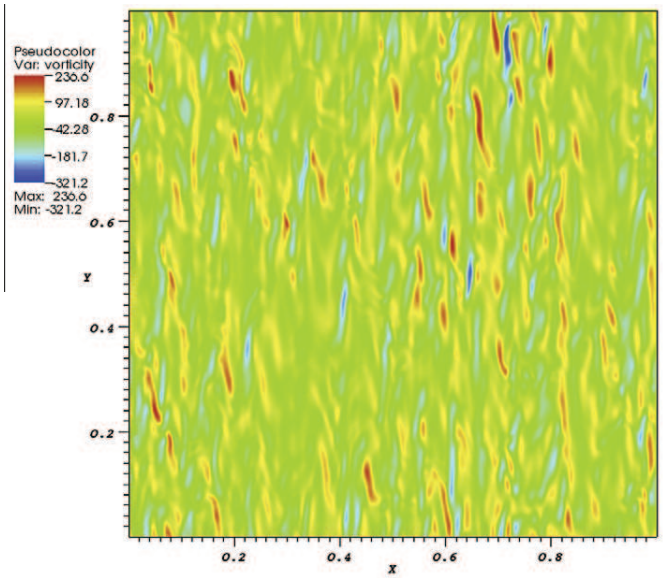

Fig. 1. Snapshot of vorticity for a typical BETA long-wavelength DTEM simulation.

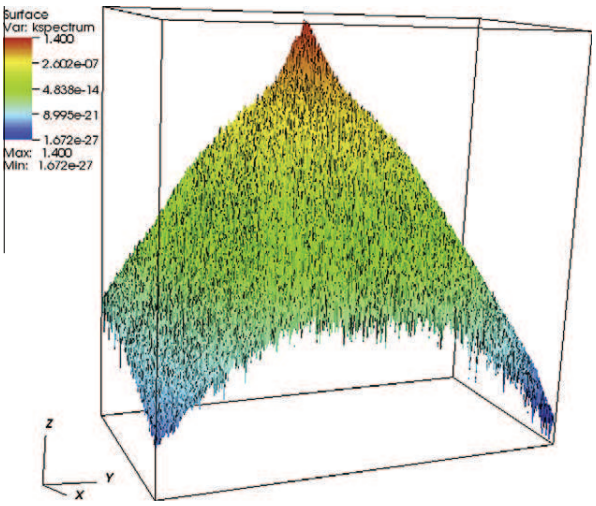

Fig. 2. Power spectrum (in lin–log scale) for the same BETA simulation shown in Fig. 1.

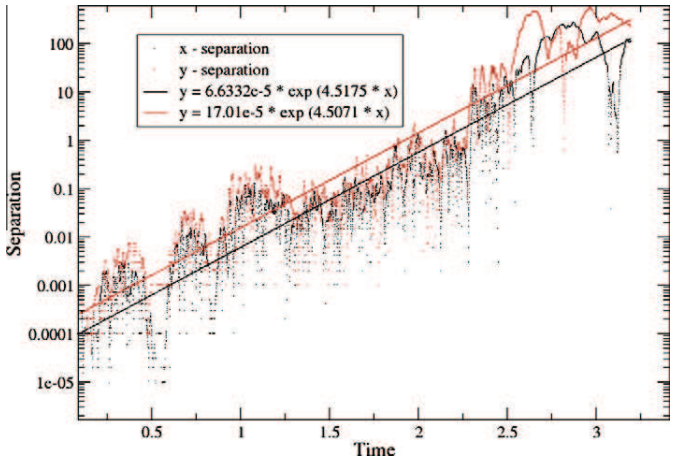

Fig. 3. Separation of two initially close trajectories as a function of time for the same BETA simulation shown in Fig. 1. Both x (black) and y (red) directions are included. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

and weak scalings are somewhat different, which sets this type of parallelization apart from spatial parallelization strategies. We briefly discuss them in what follows.

4.1. Strong scaling

We first consider the case in which the problem to be solved is fixed in size or, in our case, the final time T that must be reached at the end of the simulation is fixed. In the parareal algorithm, two propagators are used – **G** and **F**. Let the wallclock time to solve the problem (even if much more inaccurately) serially using **G** be $T_G^{ser}(T)$, and to solve it serially with **F**, be $T_F^{ser}(T)$. We then define the parameter β as the ratio of these two times:

$$\beta = \frac{T_F^{ser}(T)}{T_G^{ser}(T)}. \tag{9}$$

That is, β measures how much faster **G** is. Clearly, $\beta > 1$, as **G** is the cheap solver. In each cycle of the parareal algorithm, **F** is applied in parallel with N processors for a period of the simulation $\Delta T = T/N$, and **G** is run in serial for the whole length of the simulation T. Thus, the total time to solve the problem can be estimated (ignoring overheads) as:

$$T_{PA} = k_s(N)\left(T_G^{ser}(T) + \frac{T_F^{ser}(T)}{N}\right), \tag{10}$$

where $k_s(N)$ is the number of iterations of the parareal cycle required to achieve convergence, which is an unknown function of N. Thus, the parallel speed-up factor (or gain) for the parareal algorithm is given by:

$$H_{PA}(N) = \frac{T_F^{ser}(T)}{T_{PA}} = \left[k_s(N)\left(\frac{1}{\beta}+\frac{1}{N}\right)\right]^{-1}. \tag{11}$$

Note that the typical strong scaling for spatial parallelization, $H(N) = N$, is only recovered when $\beta \to \infty$ and $k_s(N) = 1$. But in the parareal case, $k_s(N)$ will be a function of N and β will be finite. Success or failure of the parareal will depend on the value for $k_s(N)$, which will depend itself on the choice of the coarse solver **G**. But even without knowing $k_s(N)$ at this point, several things can be learnt from this model. First, β seems to roughly set the maximum number of processors for which the parareal method yields any net parallel gain. For N much larger than this value, the serial part of the code dominates the calculation and, as predicted by Amdahl's law, performance deteriorates quickly (although our numerical simulations will show that this statement needs to be somewhat revised!). Secondly, a net parallel gain is obtained only for as long as $k_s(N) < N$. In the next section we will construct a phenomenological model for this function based on the BETA simulations.

4.2. Weak scaling

We next consider the case in which the problem to be solved has a length $T = N{\cdot}\Delta T$, being ΔT fixed. That is, the problem length increases linearly with the number of processors. As we said previously, in perfect parallelism one would expect that the wallclock time required to do the simulation is independent of N, since its processor would be doing exactly the same amount of work. That is,

$$W \equiv \frac{T_N(N * \Delta T)}{T_1(\Delta T)} = 1, \tag{12}$$

where $T_n(t)$ denotes the wallclock time needed to solve a problem of length t using n processors. In the parareal case, the time needed to solve a problem of size $N{\cdot}\Delta T$ using N processors is:

$$T_{PA}^{w} = k_w(N)\left(N \cdot T_G^{\text{ser}}(\Delta T) + T_F^{\text{ser}}(\Delta T)\right), \tag{13}$$

from which the work for processor becomes:

$$W_{PA}(N) \equiv \frac{T_{PA}^{w}}{T_F^{\text{ser}}(\Delta T)} = k_w(N)\left(1+\frac{N}{\beta}\right) \tag{14}$$

Note that the function $k_w(N)$ is different from $k_s(N)$, since now T is not kept fixed. As before, the classical weak scaling for the spatial case, $W_{PA} = 1$, is only recovered if $\beta \to \infty$ and if $k_w(N) = 1$. This will certainly not be the case. Again, it seems clear that β roughly sets the maximum number of processors for which a favorable scaling for the work-per-processor should be expected, although how good the scaling would be ultimately depends on the form of $k_w(N)$. We will also use BETA simulations to try to get its phenomenological form in the next section.

5. Results

In what follows, we describe the results of applying the parareal method to parallelize in time a typical BETA simulation that implements the DTEM model as described in Section 3. The run used has a resolution of 385×385 (complex) modes in Fourier space [i.e., (k_x, k_y), with k_x and k_y running between $-k_{\max}$ and $+k_{\max}$, with $k_{\max} = 192$], and corresponds to a restart from an already non-linearly saturated run, in which the initial values of the mode amplitudes are read from a pre-existing file. Turbulence is fully developed at this stage. All linearly unstable modes in the simulation have $|k| < 70$, value that roughly sets the starting point of the dissipation range. Stability at higher k's is ensured by using a hyperviscosity. The (modified) parareal cycle has been implemented in BETA by making extensive use of subroutines from the Message Passing Interface (MPI) libraries. The different solvers are added as separate subroutines (one for **F**, and as many as desired for all the **G**'s that will be tested), and are called as required within the parareal cycle. The section is structured as follows: first, we will simply describe a typical simulation to show that the method works. Secondly, we will use all the simulations we have done to construct a phenomenological model for $k(N,T)$, that gives the number of parareal cycles needed for convergence once N and T are prescribed. We will use this function in the final two subsections, that describe the results of two scaling studies: one for the strong scaling regime, and a second for the weak scaling one.

5.1. Parallelization in time via the parareal scheme of BETA simulations

As already mentioned in the previous section, the success of the parareal scheme depends on making a sound choice of **G**. In fact, arguments could be made, based on the sensitivity to initial conditions and the exponential divergence of Lagrangian trajectories present in turbulent situations, to the effect that such a choice should not exist in a turbulent system, and that the parareal algorithm will inevitably need $K = N$ iterations to converge when using N processors. The results presented and

discussed in this section simply intend to show that this is not the case, and that convergence can be indeed achieved for $K \ll N$, which in itself is a quite remarkable result.

After exploring multiple options for **G**, we have found that one that seems to work pretty well is solving the problem using a smaller resolution in k-space (that is, using a smaller $k_{\max}$ than **F**) and, in addition, changing the time-advance scheme to something simpler with respect to the fine solver **F**. The smaller resolution allows us to use a much larger time step, dt_G, for the time advance (in comparison to that used by the fine solver, dt_F). This, in combination with the simpler implementation (for instance, we have tried 2nd and 4th Runge–Kutta (RK) schemes, in addition to the original VODPK scheme), allows us to reach values of β as large as $\beta \sim 1000$. However, the best results (i.e., maximum parallel gain) are not obtained for the largest β. The reason is that the strategies used to speed up **G** (and thus increase β) may also increase the number of cycles required for convergence for a given number of processors N (i.e., the functions $k_s(N)$ and $k_w(N)$ from the previous section). For instance, we have observed that if the region of k-space solved by **G** is too restricted (if $k_{\max}$ is too small), the parareal performance deteriorates quickly. The minimum acceptable size seems to be imposed by the physics of the underlying turbulence: the reduced k-space region must contain a sufficient number of dissipative scales!

We conclude this subsection by showing an example of a successful parallelization in time of a BETA simulation using the parareal method. In Fig. 4, the time history of the total energy of a saturated BETA simulation is shown as a function of the parareal cycle, k. The coarse solver used for **G** is a 4th-order RK scheme with $dt_G = 40dt_F$, and including only a reduced k-space of size 201×201 harmonics (i.e., $k_{\max} = 100$). The serial time of the simulation (that has a total length of $T = 7040dt_F$) using the fine solver is 2.62 h on the *pingo* supercomputer at the Arctic Research Supercomputing Center (ARSC) at Fairbanks. The coarse solver **G** is about 200 times faster than **F** (in fact, $\beta = 210.3$). Using $N = 88$ processors (which corresponds to time chunks of size $\Delta T = 80dt_F$), the parareal calculation converges in only $K = 5$ cycles (see inset in Fig. 4), requiring only 0.29 h. That is, the parareal scheme yields a speed-up $H_{PA} = 8.80$. In Fig. 5, we also show the time history of the energy contained in one of the low-k modes (in particular, the $(k_x, k_y) = (0, -1)$ mode), to show that convergence of the total energy also implies convergence of the individual harmonics as previously stated. Similar behavior is also observed at modes with higher k's, at least until the contribution to the total energy of the mode gets of the order or smaller than the required tolerance. Thus, there is no doubt that the parareal method can be applied successfully to fully-developed turbulent simulations.

5.2. Phenomenological model for k(N,T) for the BETA model

We now carry out the first step in the determination of two unknowns of the performance model from BETA simulations: $k_s(N)$ (strong scaling) and $k_w(N)$ (weak scaling). That step is the building of a model for $k(N,T)$, the number of parareal cycles needed to converge a simulation of length T using N processors (the chunk size is then $\Delta T = T/N$).

We proceed as follows. First, we use the simulations to 'measure' the convergence rate function, $\Delta n(k;N,T)$, defined as the number of new, successive time chunks converged during the parareal cycle k. We have collected this information for many runs with various values of length T and number of processors N, and using various implementations for the coarse solver **G**. Interestingly, all of them seem to follow the same pattern, that is sketched in Fig. 6: at first only one chunk (the minimum possible, since that is the rate at which the **F** solution advances!) is converged per cycle for the first k_1 parareal cycles. Then, after $k > k_2$ cycles ($k_2 > k_1$), the convergence rate is again roughly constant, but equal to some value $B > 1$. In the intermediate region, i.e. for cycles $k_1 < k < k_2$, we assume that the convergence rate is described by a linear model. That is:

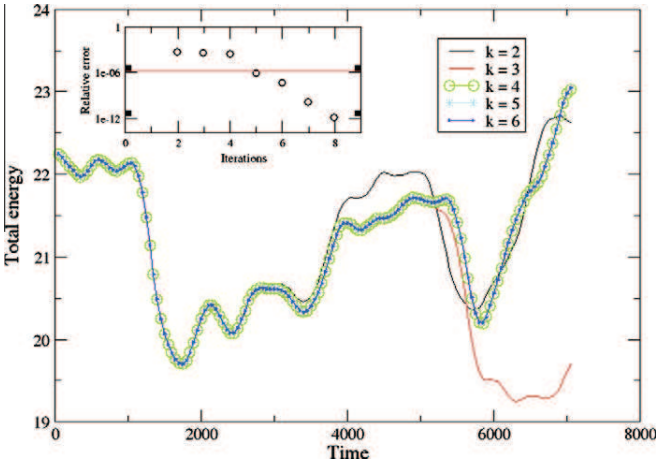

Fig. 4. Total energy as a function of time for a BETA run using $N = 88$ processors and chunk size $\Delta T = 80$ as a function of the parareal cycle index k. Coarse solver is a 4th order Runge–Kutta described in Section 5. Convergence is observed at $k = 5$. Inset: Relative error averaged over all chunks as a function of k.

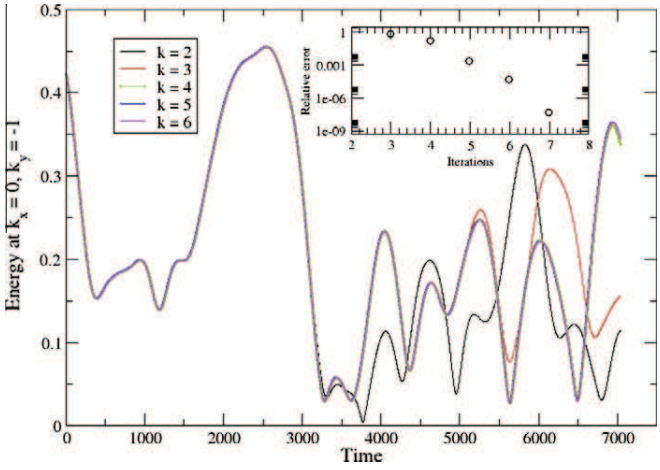

Fig. 5. Convergence history of the energy contained in the low-k mode $(k_x, k_y) = (0, -1)$ as a function of the parareal cycle index k for the same run as Fig. 4. Inset: Relative error averaged over all chunks as a function of k.

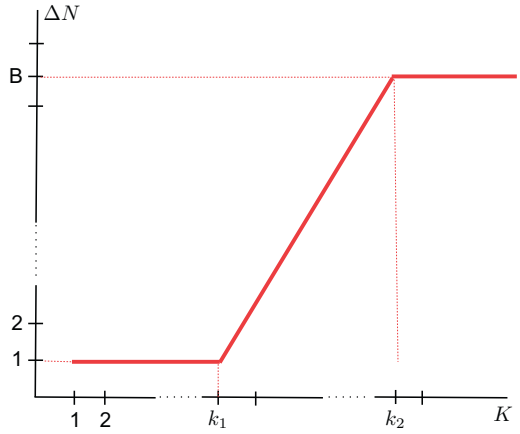

Fig. 6. Sketch shows the typical features of the convergence rate function $\Delta n(k;T,N)$ observed in the BETA simulations discussed in Section 5.

$$\Delta n(k;N,T) \simeq \begin{cases} 1, & k \leqslant k_1, \\ 1 + \left(\frac{B-1}{k_2-k_1}\right)(k-k_1), & k_1 < k < k_2, \\ B, & k \geqslant k_2, \end{cases} \tag{15}$$

where the slope of the linear part is $m = (B-1)/(k_2 - k_1)$. To find $k(N,T)$ from Eq. (15), one first computes $n(k;N,T)$, the accumulated number of converged time chunks via integration of $\Delta n(k;N,T)$:

$$n(k;N,T) = \begin{cases} k, & k \leqslant k_1, \\ k + \frac{1}{2}\left(\frac{B-1}{k_2-k_1}\right)(k-k_1)^2, & k_1 < k < k_2, \\ Bk - \frac{B-1}{2}(k_2+k_1), & k \geqslant k_2, \end{cases} \tag{16}$$

Finally, one inverts $n(k;N,T) = N$ to get:

$$k(N,T) = \begin{cases} N, & N \leqslant N_1, \\ k_1 + \frac{k_2-k_1}{B-1}\left\{\sqrt{1 + 2(B-1)\frac{N-k_1}{k_2-k_1}} - 1\right\}, & N_1 < N < N_2, \\ \frac{N}{B} + \frac{B-1}{2B}(k_2+k_1), & N \geqslant N_2, \end{cases} \tag{17}$$

where $N_1 = k_1$ and $N_2 = [(B+1)k_2 - (B-1)k_1]/2$.

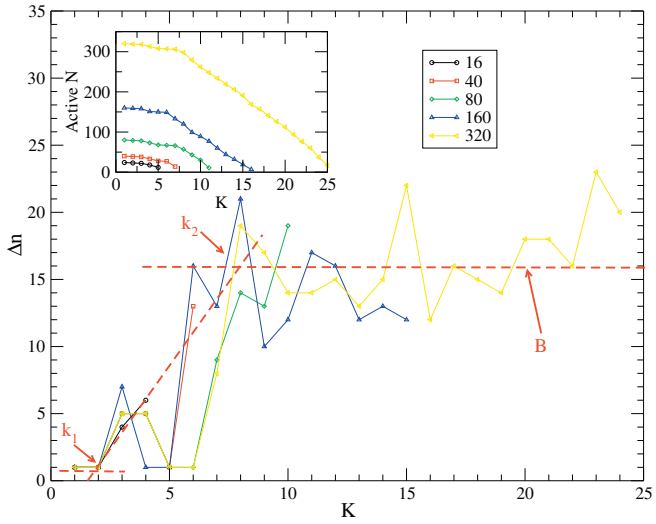

Fig. 7. Example of convergence rate function $\Delta n(k;T,N)$ obtained with the VODPK coarse solver described in Section 5 and fixed chunk size $\Delta T = 80 dt_F$. Number of processors used is shown in the legend. Inset: Number of unconverged chunks as a function of parareal cycle index k for the same cases.

However, Eq. (17) is still not very useful at this point because the values of k_1, k_2 and B change with T and N, *even when the coarse solver* **G** *remains unchanged*. An important observation at this point is that these values seem to be roughly constant when the simulations keep ΔT fixed and use the same coarse solver. For example, Fig. 7 shows the convergence rate functions $\Delta n(k;T,N)$ for various runs done using as **G** the same VODPK stiff solver as for the fine solver **F**, but including only 145×145 modes (i.e., $k_{\max} = 72$) and $dt_g = 4dt_F$ (i.e., $\beta = 16.3$) [of course, there are important fluctuations around the mean values, but that is also a consequence of the fact that one must take discrete derivatives to compute them]. This observation suggests that maybe one can find more universal quantities if k_1, k_2 and B are expressed in physical time units. And indeed, we have found that for each fixed solver **G**, $t_1 \equiv N(k_1)\cdot\Delta T$, $t_2 \equiv N(k_2)\cdot\Delta T$ and $b = B\cdot\Delta T$ (that is, expressed in units of dt_F, our physical timescale) are roughly the same for all the simulations. An example is shown in Fig. 8. The inset shows the convergence rate functions $\Delta n(k)$ obtained again with the VODPK stiff solver as **G** for simulations with various chunk sizes and number of processors. They have a general shape similar to that sketched in Fig. 6, but the parameter values are all different. However, when expressed using physical units, all of them collapse to the same universal curve. What these observations suggest is that there appears to be two timescales (t_1 and t_2) that must be resolved before the scheme can transition from

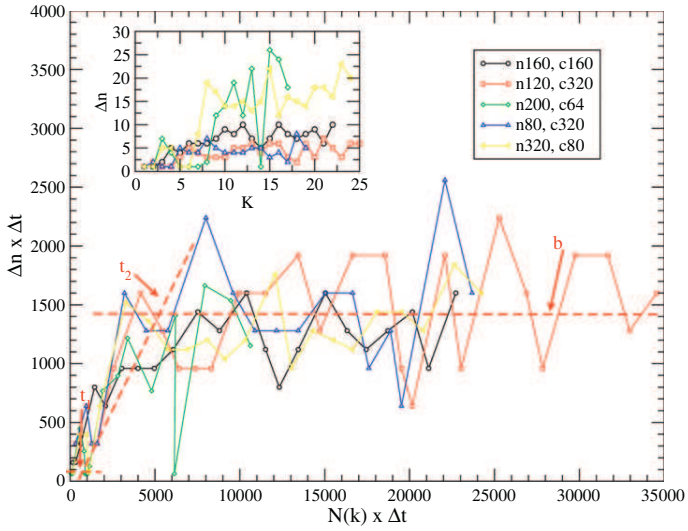

Fig. 8. Collapse of convergence rate curves when expressed in physical time for parareal BETA runs using the same coarse solver (VODPK, described in Section 5) and varying values of processors (n in legend) and chunk size ΔT (c in legend). Inset: Same convergence rate curves when expressed in parareal cycles as in Fig. 7.

Table 1
Values of parameters defining $k(N)$ model in physical units (see Eq. (17)) for the three different coarse solvers discussed in Section 5.

G	$k_{\max}$	dt_F/dt_G	β	t_1	t_2	b
VODPK	72	4	16.3	290 ± 60	4120 ± 500	1320 ± 280
RK2	100	8	84.3	32 ± 32	3100 ± 1600	1340 ± 480
RK4	100	40	210.3	80 ± 80	2560 ± 1000	1280 ± 480

a slow (i.e., 1 chunk per cycle) to a fast convergence rate b which is independent of ΔT, or T, or N, once the coarse solver is chosen.

Thus, the values of $\{t_1, t_2, b, \beta\}$ seem to be a good practical way to characterize each **G** in the parareal context. We have collected in Table 1 the values of these quantities for the three coarse solvers discussed in this paper; (1) the VODPK solver with $k_{\max} = 72$ and $dt_G = 4dt_F$ just discussed; (2) a 4th-order RK with $k_{\max} = 100$ and $dt_G = 40dt_F$; and (3) a 2nd-order RK with $k_{\max} = 100$ and $dt_G = 8dt_F$. The last two coarse solvers will be used in the strong and weak scaling sections. Once t_1, t_2 and b are known (for the coarse solver **G** chosen), the model for $k(N,T)$ given by Eq. (17) can be completed via the inverse relations:

$$k_1(N,T) = \frac{t_1 N}{T}, B(N,T) = \frac{bN}{T}, \tag{18}$$

$$k_2(N,T) = \left[2\frac{t_2 N}{T} + \left(\frac{bN}{T} - 1\right)\frac{t_1 N}{T}\right] \Big/ \left(\frac{bN}{T} + 1\right). \tag{19}$$

5.3. Strong scaling study

We are now in a position to discuss the strong scaling properties of the parareal method when applied to the BETA runs. We start by deriving an expression for $k_s(N)$ from Eq. (17). To do that, it is sufficient to require that T, *the length of the simulation, be fixed*. The result depends on how T compares with the other two timescales present, t_1 and t_2:

$$k_s(N) = \begin{cases} N, & T < t_1, \\ \frac{t_1}{T}N + \Sigma(t_1, t_2, b, T, N), & t_1 < T < t_2, \\ \frac{T}{b} + \frac{\frac{bN}{T}-1}{\frac{bN}{T}+1}\left(\frac{t_2}{b} + \frac{t_1}{T}N\right), & T \geqslant t_2, \end{cases} \tag{20}$$

where the complicated function for the transition range of T's is given by:

$$\Sigma(t_1, t_2, b, T, N) \equiv \frac{2(t_2 - t_1)\frac{N}{T}}{\left(\frac{bN}{T}\right)^2 - 1}\left\{\sqrt{1 + \left(\left(\frac{bN}{T}\right)^2 - 1\right)\frac{(T - t_1)}{t_2 - t_1}} - 1\right\}. \tag{21}$$

Although we will use the full Eq. (20) to compute the theoretical gains to compare with the simulation results, it is useful to examine this equation in several limits. First, one easily notes that if the simulation is so short that $T < t_1$ then parareal fails completely since it will converge at a rate of one chunk per cycle, thus needing $K = N$ to converge. Indeed, the speed-up factor (Eq. (11)) becomes,

$$H_{PA}^{T<t_1}(N) = \left(\frac{\beta}{N + \beta}\right) < 1. \tag{22}$$

Luckily, most actual applications will be in the opposite limit, $T \gg t_2$ (since the interest is in doing very long simulations of transport dynamics). Note that the imprint of having to resolve t_1 one chunk at a time is still present in Eq. (11), (appears as $t_1 N/T$), but it is not always deleterious for performance. In fact, its effect is small as long as $N \ll N_s \equiv T^2/bt_1$, which separates one regime in which the parareal cycle converges roughly in a constant number of cycles from another where the number of cycles required increases with N. Clearly, the transition occurs when N gets large enough so that the resolution of t_1 dominates the convergence process. But luckily, note that this transition occurs at a rather large number of processors in practical cases, since $N_s \propto T^2$. Assuming $bN/T \gg 1$, $\beta \ll N_s$ and $T \gg t_2$, conditions that will be almost always satisfied in very long simulations, three asymptotic regimes can be identified:

$$H_{PA}^{T\gg t_2}(N) \sim \begin{cases} \left(\frac{b}{T}\right)N, & N \ll \beta, \\ \frac{b\beta}{T}, & \beta \ll N \ll N_s, \\ \left(\frac{\beta T}{t_1}\right)\frac{1}{N}, & N \gg N_s. \end{cases} \tag{23}$$

That is, the speed-up of the strong scaling increases first linearly with N as in the spatial parallelization ideal scaling, but note that the slope is in general smaller than unity, b/T. The other aspect that is quite different from the spatial scaling laws is that, for a wide range of processors (between β and N_s), the speed-up saturates at a roughly constant maximum value, $\sim b\beta/T$. Thus, $N \sim \beta$ sets the value of N at which the speed-up no longer improves as we predicted, but it may take a much larger

number of processors to observe deterioration of the wallclock time. The reason is that, in this regime, the wallclock time of the calculation is completely dominated by the number of times that the coarse solution with **G** must be applied serially! The efficiency of the parallel part of the algorithm is of no importance at this stage. Finally, when the resolution of t_1 at one chunk at a time becomes the dominant process, the speed-up deteriorates as $\sim 1/N$.

The analytical considerations compare well with actual numerical calculations of a strong scaling sequence, although the asymptotic formulas tend to overestimate actual results. As an illustration we present two such numerical strong scaling studies in Figs. 9 and 10. The first study corresponds to a BETA simulation with length $T = 6400 dt_F$, using as coarse solver the 2nd-order RK previously mentioned (see Table 1). The aforementioned linear phase is observed up to $N \sim 100$. The maximum speed-up obtained in the simulations is slightly below 11 using $N = 400$ processors, and it should be noted that the speed-up varies very little above $N \simeq 100$. For this case, the theoretical values obtained are $\beta \sim 84.29$ and $N_s \sim 800$, again in agreement with the simulations. However, note that the saturated speed-up predicted by the asymptotic formula Eq. (23) is $H_{PA}^{\max} \sim 17.6$, which is larger than the one observed. The reason for the discrepancy must be sought in the fact that the conditions to derive Eq. (23) are not satisfied in this example, since the simulation is too short ant T is only slightly larger than t_2 (see Table 1). In fact, when using the full model for $k_s(N)$, the agreement is much better, as shown in Fig. 9.

The second example corresponds to a BETA simulation with length $T = 25{,}600 dt_F$, that has been done using as coarse solver the 4th order RK previously mentioned (see Table 1). The maximum speed-up observed in this case is slightly above 5, using $N \simeq 300$ processors, but it should be noted that the speed-up value does not seem to be yet close to saturation. And

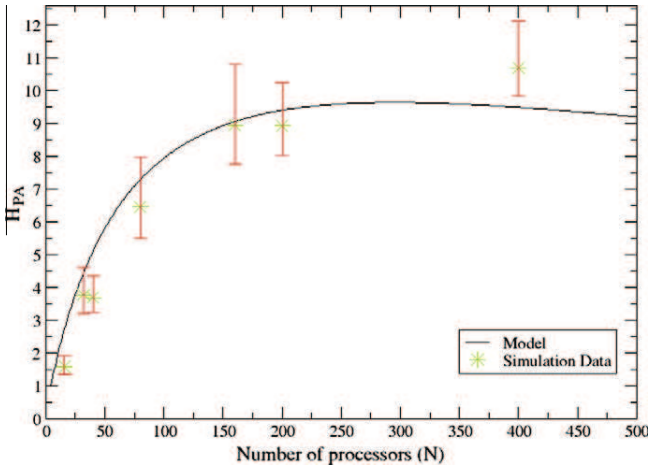

Fig. 9. Results of strong scaling study using as coarse solver **G** the 2nd-order Runge–Kutta solver described in Section 5. Model curve for gain corresponds to values from Table 1.

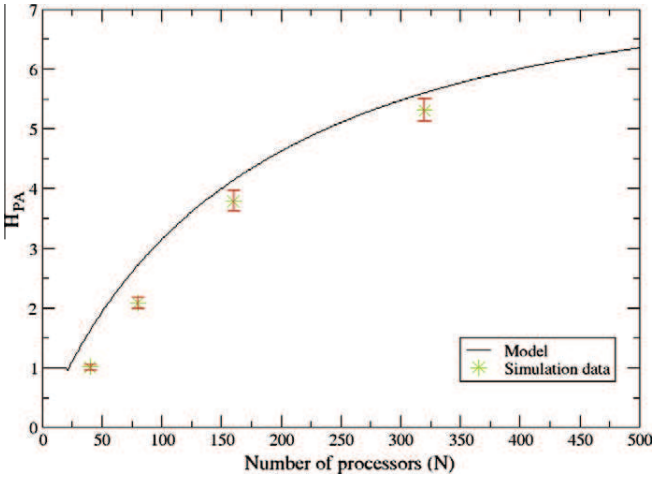

Fig. 10. Results of strong scaling study using as coarse solver **G** the 4th-order Runge–Kutta solver described in Section 5. Model curve for gain corresponds to values from Table 1.

indeed, one finds from the asymptotic formulas that $\beta \sim 210.3$ and $N_s \sim 3000$ in this case, with a maximum speed-up of roughly $H_{PA}^{\max} \sim 10$. We thus expect that we could do somewhat better with more processors in this second case.

5.4. Weak scaling

We now proceed to analyze the weak-scaling properties of the parareal scheme implemented in BETA. The figure-of-merit here is, as previously described, the amount of work per processor W_{PA} defined in Eq. (12). To obtain $k_w(N)$ it is sufficient to assume that ΔT is fixed, and that the length of the simulation is $T = N{\cdot}\Delta T$. In this case, the process is trivial because all the relevant quantities,

$$k_1 = \frac{t_1}{\Delta T}, \quad B = \frac{b}{\Delta T}, \tag{24}$$

$$k_2 = \left[2\frac{t_2}{\Delta T} + \left(\frac{b}{\Delta T} - 1\right)\frac{t_1}{\Delta T}\right] \Big/ \left(\frac{b}{\Delta T} + 1\right) \tag{25}$$

are all constants that do not change when the number of processors. Thus, $k_w(N)$ is given exactly by Eq. (17).

As in the case of the strong scaling, asymptotic formulas can be derived to better interpret the results. Two regimes can be distinguished depending on the value of T. First, we consider the case when N is not large enough to resolve t_1,$N < t_1/\Delta T$. Then, $k_W(N) = N$, and the work-per-processor is:

$$W_{PA}^{T<t_1}(N) = N\left(1 + \frac{N}{\beta}\right) > N, \quad N < t_1/\Delta T. \tag{26}$$

which is larger than N, signaling a complete failure of parallelism, as expected.

But again the limit of interest for applications is when N is large enough so that $T \gg t_2$. Then, assuming $T \gg t_2$ and $b > \Delta T$, the resulting work-per-processor becomes:

$$W^{T\gg t_2}(N) \sim \left\{\left(\frac{\Delta T}{b}\right)N + \frac{t_1}{\Delta T}\right\}\left(1 + \frac{N}{\beta}\right). \tag{27}$$

As in the strong limit case, the imprint of having to resolve t_1 still appears here (the $t_1/\Delta T$ term). But now, since ΔT is prescribed, this term will be unimportant once $N \gg N_w \equiv (bt_1)/(\Delta T)^2$. Thus, assuming also that $\beta \gg N_w$, one can easily distinguish three asymptotic regimes:

$$W_{PA}^{T\gg t_2}(N) \sim \begin{cases} \frac{t_1}{\Delta T}, & \frac{t_1}{\Delta T} \ll N \ll N_w, \\ \left(\frac{\Delta T}{b}\right)N, & N_w \ll N \ll \beta, \\ \left(\frac{\Delta T}{b\beta}\right)N^2, & N \gg \beta. \end{cases} \tag{28}$$

That is, the work-per-processor is first constant as in the ideal spatial parallelism case, although the value is usually greater than unity. And it does not imply good parallelism: only that the fixed amount of work needed to resolve t_1 dominates the calculation. But once $N > N_w$, the work-per-processor begins to increase linearly, in contrast to the spatial case. The slope is simply given by $\Delta T/b$, as expected, since $(\Delta T/b)N$ gives roughly the number of parareal cycles needed to converge in this

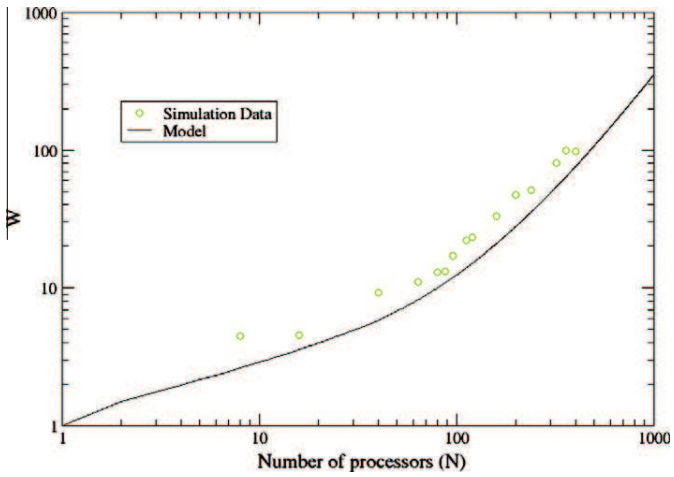

Fig. 11. Weak scaling study for series of runs done using the 4th-order Runge–Kutta solver described in Section 5. Model curve corresponds to values from Table 1.

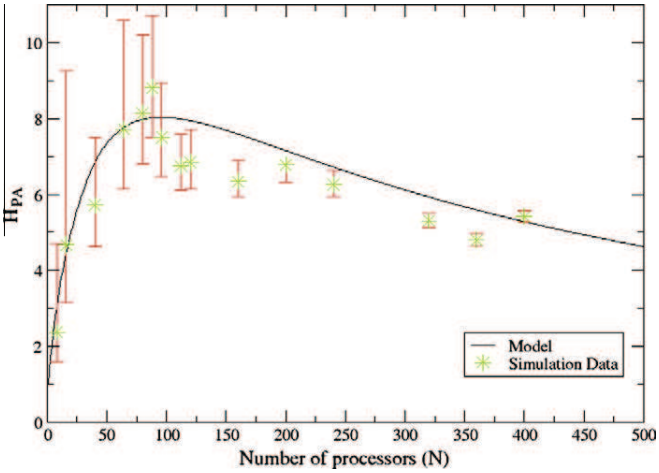

Fig. 12. Parallel gain for weak scaling series of runs using the 4th-order Runge–Kutta solver described in Section 5. Model curve plotted corresponds to values from Table 1.

regime, and the number of times each processor has to repeat the serial work. Finally, for $N \gg \beta$, things deteriorate and the work-per-processor increases quadratically with N.

To illustrate this behavior, we have also carried out a numerical weak scaling study using as coarse solver the 4th order RK previously mentioned and with a prescribed chunk size $\Delta T = 80 dt_F$. The work-per-processor as a function of the number of processors is shown in Fig. 11. The first two regimes of Eq. (28) can be clearly seen in the figure. First, the work-per-processor stays roughly constant at about $\sim$4 up to $N \sim 40$. Then, it increases close to linearly at least until $N \sim 300$–400 with a slope $\sim 0.10 < 1$, indicating that weak parallelism is different from the ideal one but not that bad. It is interesting to compare these numbers with the predictions of the asymptotic formulas. For these runs, the asymptotic expressions just derived predict that $N_w \sim 17$ and $\beta \sim 210.3$, consistent with the extent of the linear scaling in W seen in the numerical observations. The slope predicted by the model would be $\sim$0.06. On the other hand, the constant value at the smallest N would be $\sim$1 or 2. Again, as in the strong scaling case, these formulas tend to overestimate performance slightly. For completeness, we have also included a figure (Fig. 12) with the speed-up of the calculations, although note that the length of the simulation is increasing with N. The largest speed-up observed in this series is $\sim$9 using $N = 88$ processors, corresponding to the case cited as example at the beginning of this section.

6. Conclusion

The two main conclusions of this work are that turbulent simulations can be parallelized in time using the parareal algorithm, and that rather sizeable gains are achievable with this method. Indeed, we have shown speed-ups close to an order of magnitude, and even these could be further increased by further optimization of the coarse solver. As we mentioned several times, this is a remarkable result in view of the many existing concerns regarding the effect of the sensitivity to initial conditions and exponential Lagrangian divergence on the convergence of the method. In fact, it has turned out that parareal may probably work better here than in non-turbulent systems, since convergence only needs to be enforced on a single scalar quantity (here, the total energy), and the whole spectrum (containing hundreds of thousands of degrees of freedom) also converges even when unforced (at least, up to modes with an energy larger than the required convergence tolerance) due to the existence of strong nonlinear couplings between any pair of modes. This is indeed a pretty remarkable result.

We have also applied the technique to the DTEM model including only the polarization non-linearity instead of the $\mathbf{E} \times \mathbf{B}$ non-linearity which is to be discussed in a future paper. In that case, a direct parallel can be drawn between that model and the quasi-geostrophic equation for neutral fluids. This makes it likely that the technique will be readily applicable to neutral fluid turbulence as well.

In this work, we have also shown that it is possible to select a coarse solver that works for this type of problems based on using a reduced grid size in k-space, accompanied by a coarser timestep and a simpler time-advance scheme. We have also characterized the performance of the method using a model function for $k(N,T)$, the number of parareal cycles required to converge that, for the BETA simulations examined here, depends on four quantities: β, t_1, t_2 and b. Clearly, performance depends more sensitively on β and b. The larger they are, the larger the parallel speed-up will be. A matter of investigation for the future will be to investigate the origin and meaning of all these parameters. That is, to clarify whether their values are imposed by the physics of the underlying turbulence or by the choice of the coarse solver and its relation with the fine one. This knowledge would allow us to further optimize the parareal implementation inside BETA to obtain further gains. More

importantly for other applications, we also plan to repeat the current exercise in other turbulent models, in order to see if the model function found phenomenologically for $k(N,T)$ has a universal character, or whether the shape used here is only particular to the DTEM case and the $\mathbf{E} \times \mathbf{B}$-nonlinearity.

Other possibilities can and should also be, and in fact is being, explored in the future for further improvement in the gain and efficiency. For instance, a better $\mathbf{G}$ (i.e., larger β and b) than the ones used here may be an option. It may be noted here, that a faster $\mathbf{G}$ (i.e., larger β) will not necessarily mean a better one, unless the number of iterations required for convergence remains small enough (i.e., larger b as well) so that the total time for simulation is reduced. A combination of both space and time parallelization, thus leading to hybridization of the parallel scheme, should certainly be explored, since it may help to better utilize already existing resources. Time parallelization adds an additional domain for parallelizing.

A small load imbalance was observed when running the part with the fine solver in the parareal algorithm, but it did not appear to significantly affect the results. However, improving the algorithm in this regard may also be considered as an important future work. The load imbalance probably appeared due to the particular solver (VODPK) used, which uses adaptive time stepping and will therefore depend on the dynamics in the individual time chunks. A solver with fixed time steps is likely to improve this issue. But the price of having to solve all chunks with constant time steps might then be to use the timestep (dt) which is needed to solve the chunk in which the fine structures appear (smallest dt), which is not necessarily a good thing. So, a "fixed timestep solver" will avoid load imbalancing, but it is not likely to make it more efficient.

Also, it is worth noting that the current implementation of the parareal algorithm requires processors where convergence has been observed to remain idle until the solution along the entire time domain is converged. Reuse of idle processors may thus enhance the efficiency of the parareal scheme.

Acknowledgements

Part of the research was carried out at the University of Alaska Fairbanks, funded by the DOE Office of Science Grant No. DE-FG02-04ER54741. Research was also carried out in part at Oak Ridge National Laboratory, managed by UT-Battelle LLC, for US DOE under Contract No. DE-AC05–00OR22725, and funded via the Seed Money Initiative Program. For simulation runs, the authors are thankful for grants for the use of supercomputing resources at the University of Alaska's Arctic Region Supercomputing Center (ARSC) in Fairbanks. We also acknowledge valuable discussion with L. Chacon, L.A. Berry and W. Elwasif at ORNL.

References

[1] J. Dahlburg, J. Corones, D. Batchelor, R. Bramley, M. Greenwald, S. Jardin, S. Krasheninnikov, A. Laub, J.-N. Leboeuf, J. Lindl, W. Lokke, M. Rosenbluth, D. Ross, D. Schnack, Fusion simulation project: integrated simulation and optimization of magnetic fusion systems, J. Fusion Energ. 20 (4) (2001) 135.19.
[2] B.A. Carreras, D.E. Newman, V.E. Lynch, P.H. Diamond, A model realization of self-organized criticality for plasma confinement, Phys. Plasmas 3 (8) (1996) 2903–2912.
[3] R. Sanchez, D.E. Newman, J.N. Leboeuf, V.K. Decyk, B.A. Carreras, Nature of transport across sheared flows in electrostatic, ion-temperature-gradient driven gyrokinetic plasma turbulence, Phys. Rev. Lett. 101 (2008) 205002.
[4] J. Nivergelt, Parallel methods for integrating ordinary differential equations, Commun. ACM 7 (12) (1964) 731–733.
[5] A. Bellen, M. Zennaro, Parallel algorithms for initial value problems for difference and differential equations, Phys. Rev. E 25 (3) (1989) 341–350.
[6] P. Chartier, B. Philippe, A parallel shooting technique for solving dissipative odes, Computing 51 (3–4) (1993) 209–236.
[7] M. Kiehl, Parallel multiple shooting for the solution of initial value problems, Parallel Comput. 20 (3) (1994) 275–295.
[8] W.L. Miranker, W. Liniger, Parallel methods for the numerical integration of ordinary differential equations, Math. Comput. 21 (99) (1967) 303–320.
[9] J.H. Saltz, V.K. Naik, Towards developing robust algorithms for solving partial differential equations on MIMD machines, Parallel Comput. 6 (1) (1988) 19–44.
[10] D.E. Womble, A time-stepping algorithm for parallel computers, SIAM J. Sci. Stat. Comput. 11 (5) (1990) 824–837.
[11] C. Farhat, M. Chandersis, Time-decomposed parallel time-integrators – Part I: Theory and feasibility studies for fluid, structure, and fluid-structure applications, Int. J. Numer. Methods Eng. 58 (9) (2003) 1397–1434.
[12] C. Farhat, J. Cortial, C. Dastillung, H. Bavestrello, Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamic responses, Int. J. Numer. Methods Eng. 67 (5) (2006) 697–724.
[13] J. Cortial, C. Farhat, A time-parallel implicit method for accelerating the solution of non-linear structural dynamics problems, Int. J. Numer. Methods Eng. 77 (4) (2008) 451–470.
[14] J. Lions, Y. Maday, G. Turinici, A parareal in time discretization of pde's, CR Acad. Sci. I – Math. 332 (7) (2001) 661–668.
[15] L. Baffico, S. Bernard, Y. Maday, G. Turinici, G. Zérah, Parallel in time molecular dynamics simulations, Phys. Rev. E 66 (5) (2002) 057706.
[16] G. Bal, Y. Maday, A Parareal Time Discretization for Non-linear PDEs with Application to the Pricing of an American Put, Lecture Notes in Computer Science and Engineering, vol. 23, Springer, Berlin, 2002.
[17] G. Staff, G. Rønquist, Stability of the parareal algorithm, in: Proceedings of Fifteen International Conference on Domain Decomposition Methods, Springer Verlag, 2003, pp. 449–456.
[18] G. Bal, Parallelization in time of (stochastic) ordinary differential equations, 2003. <http://www.columbia.edu/gb2030/PAPERS/paralleltime.pdf>.
[19] I. Garrido, M.S. Espedal, G.E. Fladmark, A convergent algorithm for time parallelization applied to reservoir simulation, Proceedings of Fifteen International Conference on Domain Decomposition Methods, vol. 40, Springer, Berlin, 2004, pp. 469–476.
[20] P.F. Fischer, F. Hecht, Y. Maday, A parareal in time semi-implicit approximation of the Navier–Stokes equations, Proceedings of Fifteen International Conference on Domain Decomposition Methods, vol. 40, Springer Verlag, 2004, pp. 433–440.
[21] Y. Liu, J. Hu, Modified propagators of parareal in time algorithm and application to Princeton Ocean model, Int. J. Numer. Methods Fluids 57 (12) (2008) 1793–1804.
[22] M.J. Gander, E. Hairer, Nonlinear convergence analysis for the parareal algorithm, Domain Decomposition Methods in Science and Engineering XVII, vol. 60, Springer, 2008, pp. 45–56.
[23] D.E. Newman, P.W. Terry, P.H. Diamond, A two-nonlinearity model of dissipative drift wave turbulence, Phys. Fluids B 4 (3) (1992) 599–610.
[24] Y.M. Liang, P.H. Diamond, X.H. Wang, D.E. Newman, P.W. Terry, A two-nonlinearity model of dissipative drift wave turbulence, Phys. Fluids B 5 (4) (1993) 1128–1139.

[25] B.A. Carreras, K. Sidikman, P.H. Diamond, P.W. Terry, L. Garcia, Theory of shear flow effects on long-wavelength drift wave turbulence, Phys. Fluids B 4 (10) (1992) 3115–3131.
[26] J.A. Mier, R. Sanchez, L. Garcia, D.E. Newman, B.A. Carreras, On the nature of transport in near-critical dissipative-trapped-electron-mode turbulence: Effect of a subdominant diffusive channel, Phys. Plasmas 15 (11) (2008) 112301.
[27] F. Takens, Detecting Strange Attractors in Turbulence, Lecture Notes in Mathematics, vol. 898, Springer, Berlin, 1981.
[28] A. Hindsmarsh, Gear: Ordinary Equation System Solver, in: Report UCID-30001, Rev. 3, LLNL, 1974.