Geochemistry Geophysics Geosystems G3

Technical Brief
Volume 13, Number 1
18 January 2012
Q01003, doi:10.1029/2011GC003905
ISSN: 1525-2027

Published by AGU and the Geochemical Society

Time domain parallelization for computational geodynamics

Henri Samuel

Bayerisches Geoinstitut, Universität Bayreuth, D-95447 Bayreuth, Germany (henri.samuel@uni-bayreuth.de)

[1] I present a time domain parallelization approach for geodynamic modeling. This algorithm, named *parareal*, is based on the use of coarse sequential and fine parallel propagators to predict and to iteratively correct the solution of the governing equations over a given time interal. Although the method has been successfully used to solve differential equations, in various scientific areas, it has not been applied to model solid-state convective motions relevant to the Earth and other planetary mantles. In that case, the time-dependence of the velocity is only implicit, which requires modifications to the original algorithm. The performances of this adapted version of the *parareal* algorithm were investigated using theoretical model predictions in good agreement with numerical experiments. I show that under optimum conditions, the parallel speedup increases linearly with the number of processors, and speedups close to 10 were measured, using only few tens of CPUs. This *parareal* approach can be used alone or combined with any spatial parallel algorithm, allowing significant additional increase in speedup with increasing number of processors.

Components: 8500 words, 10 figures.

Keywords: computational geodynamics; parallel computing; parallel speedup; parareal.

Index Terms: 1213 Geodesy and Gravity: Earth's interior: dynamics (1507, 7207, 7208, 8115, 8120); 1956 Informatics: Numerical algorithms; 2753 Magnetospheric Physics: Numerical modeling.

Received 4 October 2011; **Revised** 29 November 2011; **Accepted** 30 November 2011; **Published** 18 January 2012.

Samuel, H. (2012), Time domain parallelization for computational geodynamics, *Geochem. Geophys. Geosyst.*, *13*, Q01003, doi:10.1029/2011GC003905.

1. Introduction

[2] Modern computational geodynamics heavily relies on parallel algorithms to speed up calculations. Such a tendency is continuously growing over time as the available parallel resources increase, in particular with the development of multi-core architectures and Graphical Processing Unit computations [*Schmidt et al.*, 2010]. One of the most widely used approaches in parallel geodynamic codes is spatial decomposition [*Bunge and Baumgardner*, 1995; *Zhong et al.*, 2000; *Schmalzl and Hansen*, 2000; *Kageyama and Sato*, 2004; *Choblet et al.*, 2007; *Zhong et al.*, 2008; *Tackley*, 2008; *Hütigg and Stemmer*, 2008; *Aleksandrov and Samuel*, 2011], where the physical computational space is subdivided into smaller domains that are attributed to one processor or to a set of processors. Each sub-domain carries out its own calculation in parallel and exchanges information periodically with other sub-domains. Such approaches are efficient as long as the size of the sub-domains is large enough so that computational time remains larger than communication time. However, when the size of the sub-domains becomes too small, the speedup stagnates, which puts bounds on the maximum performances of the algorithm (Figure 1).

[3] Other approaches consist in solving at each time step the global set of discretized governing equations using parallel direct libraries or parallel iterative solvers [e.g., *Katz et al.*, 2007; *Braun et al.*,

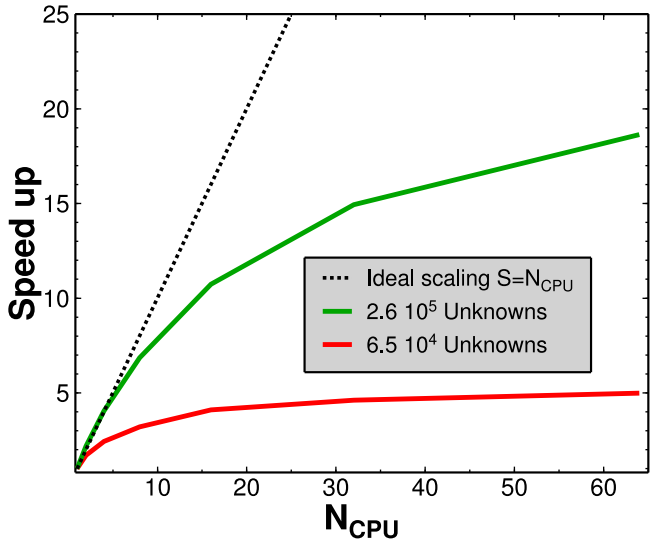

Figure 1. Parallel speedup associated to the resolution of a Poisson Equation in a square domain, as a function of the total number of processors N_{CPU}. Parallelization is achieved through spatial domain decomposition, using an iterative geometric multigrid solver. The two solid curves represent cases with different resolution, resulting in different number of unknowns.

2008; *Tosi et al.*, 2010; *Suckale et al.*, 2010; *Samuel and Evonuk*, 2010; *Thieulot*, 2011]. Similar to the domain-decomposition approach, such a parallelization only concerns the spatial domain and the performances of parallel solvers also saturate and sometimes decrease when the number of processors becomes too large.

[4] I present here an alternative approach named *parareal* [*Lions et al.*, 2001], which is based on time domain decomposition. This method has been successfully applied to solve ordinary differential equations and time-dependent systems of partial differential equations in various scientific areas, including molecular dynamic simulations [*Baffico et al.*, 2002], wave propagation [*Mercerat et al.*, 2009] and finite Prandtl number fluid dynamics [*Fisher et al.*, 2003; *Trindade and Pereira*, 2004; *Liu and Hu*, 2008; *Samaddar et al.*, 2010]. However, to my knowledge, the *parareal* algorithm has not been applied in geodynamic studies where motions relevant to the Earth and other planetary mantles are that of a convective fluid at infinite Prandtl number. In that case, the time dependence of the mass and momentum equations is only implicit, due to thermal and/or viscous couplings with the explicitly time-dependent energy equation. This requires a number of modifications to the original algorithm.

[5] This *parareal* approach can be combined to spatial domain decomposition or to any other parallel algorithm, allowing an additional increase in speedup with increasing the number of processors.

[6] The main objective of this study is to adapt the original version of the *parareal* algorithm to the governing equations for solid state convection, to test its robustness, and to evaluate its parallel performances theoretically and experimentally, using test cases representative of typical geodynamic scenarios.

[7] The paper is organized as follows: section 2 introduces the set of governing equations to be solved with the *parareal* approach. Section 3 describes the algorithm. Section 4 presents the theoretical performances of the *parareal* algorithm, which are compared with those measured experimentally in the context of two geodynamic scenarios, presented in section 5, preceding the discussion.

2. Governing Conservation Equations

[8] Mantle solid-state flow may be reasonably described by the motion of a Boussinesq, highly viscous fluid in the limit of infinite Prandtl number (*i.e.*, inertia is negligible). In that case, the dimensionless governing equations are the conservation of mass:

$$\nabla \cdot \mathbf{u} = 0, \tag{1}$$

the conservation of momentum:

$$-\nabla p + \nabla \cdot \eta\left(\partial_j u_i + \partial_i u_j\right) - Ra\ T\vec{e}_z = 0, \quad (2)$$

and the conservation of energy:

$$\frac{\partial T}{\partial t} + \mathbf{u}.\nabla T = \nabla^2 T. \quad (3)$$

[9] In these equations, $\mathbf{u}$ is the dimensionless velocity vector, p is the dynamic pressure, T is the temperature, t is the time, $\eta = \exp(-\gamma T)$, is the dimensionless viscosity, $\vec{e}_z$ is a unit vector pointing upward. The Rayleigh number Ra and the sensitivity of the viscosity to temperature, γ, are the two governing parameters. Additional complexities relevant to the Earth's and other planetary mantles could be added to the above equations, such as compressibility, variable thermal conductivity and expansion coefficients, or multiple phase changes. By simplicity these were ignored here, but this should not affect significantly the application of the *parareal* algorithm.

[10] As mentioned previously, the approach has been applied to finite Prandtl number Navier-Stokes Equations [*Fisher et al.*, 2003; *Trindade and Pereira*, 2004; *Samaddar et al.*, 2010], where the solutions of the conservation equations (temperature, velocity and pressure) are explicitly time-dependent. However, in the case of infinite Prandtl number convection, the explicit time dependence only appears in the conservation of energy (equation (3)). In this case, the mass and momentum equations do not explicitly depend on time and the temporal dependence of the velocity field is only due to the coupling of the Navier-Stokes and the energy equations via the buoyancy term $RaT(t)$ and the viscosity $\eta(T(t))$. We shall see in the following sections that this requires modifications to the original *parareal* algorithm.

3. The Algorithm

[11] Consider the solution vector of equations (1)–(3): $\mathcal{U}(t) = (T(t), \mathbf{U}(t))$, where $\mathbf{U}(t)$ is the velocity field that satisfies the momentum equation subject to the incompressibility constraint.

[12] In the serial case, $\mathcal{U}(t_i)$ is obtained by propagating the solution until the desired time is reached, starting from the previous time step $t_i - \delta t$, and using an operator $\mathcal{F}_{\delta t}$ such that:

$$\mathcal{U}(t_i) = \mathcal{F}_{\delta t}(\mathcal{U}(t_i - \delta t)). \quad (4)$$

[13] In most numerical geodynamic studies, the time step size δt, varies and is subject to stability constraints, which are often (but not systematically) based on a Courant-Friedrichs-Lewy (CFL) criterion. Consequently, for a given spatial resolution, δt varies with time as it is a function of the time-dependent solution $\mathcal{U}(t)$.

[14] To speed up the calculations, a way around this CFL time step restriction would be to solve equation (3) using implicit schemes that are unconditionally stable, and therefore not subject to a CFL criterion. This would allow propagating the solution using a coarse operator $\mathcal{C}_{\Delta t}$ based on a larger time step Δt such that:

$$\mathcal{U}(t_i) = \mathcal{C}_{\Delta t}(\mathcal{U}(t_i - \Delta t)). \quad (5)$$

[15] While this approach allows one to reach the solution at a desired time more rapidly, the use larger time steps would by definition yield poorer time resolution.

[16] In order to accelerate the calculations without affecting the accuracy of the solution *Lions et al.* [2001] have proposed a time domain decomposition algorithm named *parareal*. Compared to spatial decomposition methods, the idea of time domain decomposition is much more recent, because the evolution of time-dependent systems is serial by nature, which complicates the parallelization. However, time domain parallelization can be achieved by considering a predictor-corrector procedure embedded in an iterative approach. This constitutes the basis of the *parareal* algorithm [*Lions et al.*, 2001]. This approach is based on the use of coarse and fine operators to predict and to correct the solution over a given time interval.

[17] Consider a time dependent problem whose initial condition is $\mathcal{U}(t_0) = \mathcal{U}_0$ (Figure 2). One seeks the solution $\mathcal{U}(t_i) = \mathcal{U}_i$ over a time interval of size $\Delta t_{\text{parareal}} = t_N - t_0$, which is divided into N sub-intervals of equal size $\Delta t = \Delta t_{\text{parareal}}/N$. This time interval can contain several CFL time steps and we are not only interested in the solution at the end of the time interval but also the solution at a given time step "i" within the interval. To solve this problem in parallel, each time sub-domain can be assigned to a processor. Applying the *parareal* approach over $\Delta t_{\text{parareal}}$ consists in two steps, monitored by an index k:

[18] 1. The initialization ($k = 0$), where a first guess is obtained over $\Delta t_{\text{parareal}}$ by propagating the

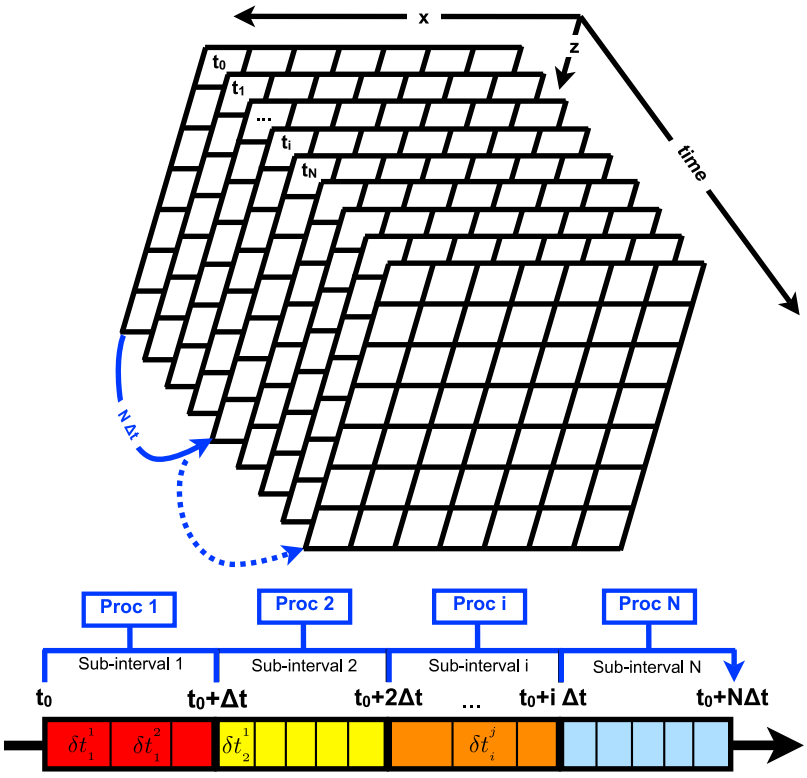

Figure 2. Schematic representation of the *parareal* algorithm. (top) Representation of the discrete grid in a 3D space-time domain (x, z, t). (bottom) Representation of the discrete domain along the time axis (*i.e.*, cross section in time of the 3D domain above) for one *parareal* time interval $N\Delta t$, distributed among N slave processors.

solution with the coarse operator (according to equation (5)):

$$\mathcal{U}_i^0 = \mathcal{C}_{\Delta t}(\mathcal{U}_{i-1}^0). \tag{6}$$

As the knowledge of the solution at the previous time step is required, this step needs to be performed in serial.

[19] 2. The iterative (k-indexed) improvement of the solution according to:

$$\mathcal{U}_i^k = \mathcal{C}_{\Delta t}(\mathcal{U}_{i-1}^k) + (\mathcal{F}_{\delta t}(\mathcal{U}_{i-1}^{k-1}) - \mathcal{C}_{\Delta t}(\mathcal{U}_{i-1}^{k-1})). \tag{7}$$

The first term on the right hand side of the above equation can be seen as a predictor step, while the second represents a correction, which is simply the jump between the fine and coarse solutions at a given time step i and at the previous *parareal* iteration $k-1$. Since the coarse predictor term requires the knowledge of the coarse solution from the previous time step $(i-1)$ at the present *parareal* iteration k, it must be computed serially. However, this is not the case of the jump, which can be computed in parallel over each of the N time sub-intervals. This iterative procedure continues until the solution has reached the desirable level of accuracy, which in all cases is bounded by the truncation error of the fine operator [*Lions et al.*, 2001].

[20] Of course, if all executed in serial, the iterative nature of the above algorithm makes it less efficient than a simple use of a fine operator applied over $\Delta t_{\text{parareal}}$. However, as illustrated in Figure 3, this algorithm can be efficiently parallelized since all the operations involving the use of the fine propagators can be carried out in parallel, over each corresponding time sub-interval. It is also evident that in order for the *parareal* algorithm to be efficient, the computational time associated with the coarse operator must be much smaller than the one associated with the use of the fine operator.

[21] To minimize synchronization among processors, I present here a *parareal* version that uses a master-slave configuration [*Farhat and Chandesris*, 2003]: Among the $N_{\text{CPU}} = N + 1$ processors used, one "master" CPU is systematically assigned to serial calculations, and each of the remaining N "slave" processors are assigned to a time sub-domain to perform calculations in parallel. The master node therefore distributes information to, and gathers information from the slave

```
if (proc = master) then
  for i = 1 to N do
    U_i^0 = C_Δt(U_{i-1}^0) {Initial guess: serial coarse propagation}
    send U_i^0 to slave i
  end for
end if

k = 1
while (converged = false) do
  if (proc = slave i) then
    U_f^i = F_δt(U_{i-1}^{k-1}) {Parallel fine propagation}
    send U_f^i to master
  else if (proc = master) then
    for i = 1 to N do
      Δ_fc = U_f^i − C_Δt(U_{i-1}^{k-1}) {Compute the correction}
      U_i^k = C_Δt(U_{i-1}^k) + Δ_fc {Predictor corrector}
      send U_{i-1}^k to slave i
    end for
    check for convergence
  end if
  k = k + 1
end while
```

Figure 3. Pseudo-code for the *parareal* algorithm, using a master-slave configuration.

processors. The pseudo-code for the algorithm with such a configuration is shown in Figure 3.

[22] An important remark is that convergence of the *parareal* algorithm is reached within at most $k = N$ iterations. This is illustrated in Figure 4 showing the evolution of the solution $\mathcal{U}$ (using the average temperature T_{mean} as a proxy) as a function of the *parareal* iteration k, with $N = 4$ sub-intervals. At $t = t_0$ (*i.e.*, $i = 0$) all the solutions $\mathcal{U}_{i=0}^{k} = \mathcal{U}_0$ are identical since this represents the initial condition for the *parareal* process applied over the time interval $[t_0, t_0 + N\,\Delta t]$. Even at $t = t_0 + \Delta t$ ($i = 1$), all the solutions remain identical to each other. Such a match at $i = 1$ is not fortuitous: for instance,

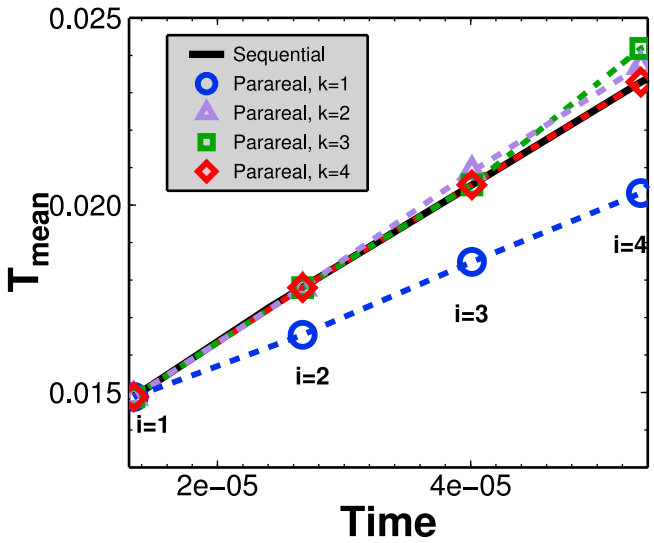

Figure 4. Result of the *parareal* algorithm with $N = 4$ during one time interval $N\Delta t$. Time evolution of the average dimensionless temperature over one time interval containing 4 time steps ($i = 1,4$). Each colored symbol corresponds to the solution at a different iteration k of the *parareal* algorithm. Complete convergence with the sequential solution (black line) is reached within at most $k = N = 4$ iterations.

according to equation (7), the new *parareal* solution at $k = 1$ is:

$$\mathcal{U}_{i=1}^{k=1} = \mathcal{C}_{\Delta t}(\mathcal{U}_{i=0}^{k=1}) + \mathcal{F}_{\delta t}(\mathcal{U}_{i=0}^{k=0}) - \mathcal{C}_{\Delta t}(\mathcal{U}_{i=0}^{k=0}). \quad (8)$$

[23] In addition, since the initial condition ($i = 0$) is the same for all k, $\mathcal{C}_{\Delta t}(\mathcal{U}_{i=0}^{k=1})$ and $\mathcal{C}_{\Delta t}(\mathcal{U}_{i=0}^{k=0})$ are identical and therefore cancel each other. This yields $\mathcal{U}_{i=1}^{k=1} = \mathcal{F}_{\delta t}(\mathcal{U}_{i=0}^{k=0}) = \mathcal{F}_{\delta t}(\mathcal{U}_0)$, which would have been obtained in the sequential case. One can generalize this result for all $k \geq 1$, and therefore:

$$\mathcal{U}_{i=1}^{k} = \mathcal{F}_{\delta t}(\mathcal{U}_0). \quad (9)$$

[24] Differences between the solutions at different *parareal* iterations only appear at the next time step $t = t_0 + 2\Delta t$ ($i = 2$): the *parareal* solution at the first iteration $k = 1$ significantly differs from the sequential solution while the next *parareal* solutions $\mathcal{U}_{i=2}^{k\geq 2}$ are identical to the sequential solution. Using the same previous reasoning, one can show that the *parareal* solution at $k = 2$ is:

$$\mathcal{U}_{i=2}^{k=2} = \mathcal{C}_{\Delta t}(\mathcal{U}_{i=1}^{k=2}) + \mathcal{F}_{\delta t}(\mathcal{U}_{i=1}^{k=1}) - \mathcal{C}_{\Delta t}(\mathcal{U}_{i=1}^{k=1}) \quad (10)$$

[25] In addition, using equation (9), $\mathcal{C}_{\Delta t}(\mathcal{U}_{i=1}^{k=2}) = \mathcal{C}_{\Delta t}(\mathcal{U}_{i=1}^{k=1})$, yielding $\mathcal{U}_{i=2}^{k=2} = \mathcal{F}_{\delta t}(\mathcal{U}_{i=1}^{k=1}) = \mathcal{F}_{\delta t}(\mathcal{U}_{i=1})$, which again would be obtained with a serial propagation using the fine operator.

[26] It is easy to see that by the same mechanism at $t \leq t_0 + k\Delta t$ both *parareal* and sequential solutions are identical, as illustrated in Figure 4. This is the reason why convergence of the *parareal* algorithm is guaranteed within at most $k = N$ iterations. Although this example is interesting for educational purposes, in practice such a situation must be avoided. Indeed, convergence reached after $k = N$ iterations yields worse computational performances than the sequential case, because more operations are involved compared to a serial execution. Therefore, one must reduce the total number of *parareal* iterations, K, to a minimum, ideally 1.

3.1. Coarse and Fine Operators

[27] The efficiency of the *parareal* algorithm depends heavily on the choice of the coarse and fine operators. In addition to the use of a larger time step, Δt, for $\mathcal{C}_{\Delta t}$ than a CFL time step, δt, for $\mathcal{F}_{\Delta t}$, several choices are possible.

[28] The fact that the size of Δt is by construction larger than a CFL time step would naturally suggest to use an implicit scheme to solve the time dependent heat equation (3). Although this is a popular choice, including for fluid dynamics applications of the *parareal* algorithm [*Fisher et al.*, 2003; *Trindade and Pereira*, 2004, 2006; *Samaddar et al.*, 2010], we will see that at least for the cases presented in this paper this is not the best strategy. The main reason is that the use of implicit schemes yields significantly larger numerical diffusion compared to the best explicit schemes (e.g., WENO schemes [*Jiang and Shu*, 1996] or the use of TVD schemes combined with flux limiters [*Sweby*, 1984; *Roe*, 1986]). This numerical artifact further deteriorates the coarse solution, which yields a larger number of *parareal* iterations K in order to reach convergence. In addition, the computational cost associated with implicit solvers is generally larger than the one associated with explicit solvers, which are optimum.

[29] On the other hand, using an explicit scheme for the coarse operator can be problematic since Δt may not satisfy the CFL stability criteria. To circumvent this problem, one can split the resolution of the governing equations into two groups: one elliptic group corresponding to the Navier-Stokes equations (1) and (2), and a second parabolic group which corresponds to the conservation of energy (equation (3)), where the time dependence of the temperature is explicit. The solution for the parabolic group is propagated in time using an explicit scheme with a time step δt subject to a CFL criterion (therefore smaller than Δt). However, the solution of the elliptic group, which does not explicitly depend on time is determined only at every Δt (*i.e.*, at the end of each time sub-domain). The error associated with such a decoupling of the Navier-Stokes equations and the conservation of energy was found to be smaller than the numerical diffusion associated with the use of an implicit solver for the coarse operator. Indeed, this elliptic-parabolic splitting associated with explicit coarse propagation was found to be the best choice for the coarse operator in terms of convergence of the *parareal* algorithm (see section 5). In addition, the use of a larger time step for the elliptic group reduces the computational cost of the coarse operator, which further improves the efficiency of the *parareal* algorithm.

[30] There is much less freedom in the choice of the fine operator. It simply consists in solving the set of governing equations over a time interval Δt, with a series of smaller time steps δt, each satisfying a CFL criteria, exactly as one would proceed for the serial case (equation (4)).

3.2. Size of the Time Interval

[31] At each initialization step of the *parareal* procedure ($k = 0$), a time step, $\delta t^0_{\mathrm{CFL}}$, satisfying the CFL criteria is determined using the corresponding initial velocity field. The size of the time interval over which the *parareal* algorithm is applied is then obtained by extrapolation of this time step (Figure 2):

$$\Delta t_{\mathrm{parareal}} = N \delta t^0_{\mathrm{CFL}} n_{\mathrm{CFL}} \tag{11}$$

[32] If the velocity field was constant throughout $\Delta t_{\mathrm{parareal}}$, the user-defined parameter n_{CFL} would corresponds to the number of CFL time steps per sub-domain over which each fine operator is applied. However, as the velocity field changes with time, the assumption that $\delta t^0_{\mathrm{CFL}}$ is constant is no longer exact. In this general case, n_{CFL} represents the approximate/average number of CFL time steps per sub-domain. As mentioned previously, each time sub-domain has the same size, which is then: $\Delta t = \delta t^0_{\mathrm{CFL}} n_{\mathrm{CFL}}$.

[33] Ideally, one would choose the largest possible value for n_{CFL} in order to minimize the importance of the *parareal* initialization step. However, as shown in the following sections, in general for problems with strong time variations of the solution when n_{CFL} exceeds ~20 the number of *parareal* iterations exceeds 1, which is not desirable. There is therefore an optimum size of the interval corresponding to the maximum value of n_{CFL} for which $K = 1$.

3.3. Convergence Criteria

[34] It has been shown in section 3 that the accuracy of the *parareal* method converges to that of the fine operator within at most N iterations. Clearly, for efficiency purposes it is desirable to reach convergence as quickly as possible ($K = 1$) and in any cases for a total number of *parareal* iterations K lower than N. Therefore, as for any iterative method, the choice of a good measure of convergence is crucial, and several choices are possible.

[35] A simple, but blind criterion would be to stop the iterations whenever the Root Mean Squared of the changes between two consecutive *parareal* iterations:

$$\Delta^k_i = \sqrt{\frac{1}{n}\int_V \left(\mathcal{T}^{k-1}_i - \mathcal{T}^k_i\right)^2 dV}, \tag{12}$$

falls below some prescribed threshold [*Samaddar et al.*, 2010]. In the above equation, $\mathcal{T}^k_i$ is the solution vector of equation (3) at the end of the time domain i and at the kth *parareal* iteration, n is the size of the solution vector, and V represents the computational domain.

[36] Another criterion based on the value of the maximum of the jump between the coarse and fine solutions has also been proposed [*Trindade and Pereira*, 2006].

[37] I choose instead a more general criterion based on the comparison between Δ^k_i and the value of the Local Truncation Error (LTE) [*Lepsa and Sandu*, 2010]. In this case, convergence is reached according to the following condition:

$$\max\left(\Delta^k_i\right) < tol \ \max\left(LTE^k_i\right), \tag{13}$$

where *tol* is an empirical parameter adjusted to 10^{-1}. For each *parareal* iteration k, the Local Truncation Error vector on the temperature LTE^k_i, is estimated at the end of each *parareal* time sub-interval i, by subtracting the solution $\mathcal{T}^k_{i,\delta t}$ to equation (3) over one CFL time step, δt, to the solution $\mathcal{T}^k_{i,\delta t/2}$, obtained at the same time but using a time step twice smaller *i.e.*, $LTE^k_i = |\mathcal{T}^k_{i,\delta t} - \mathcal{T}^k_{i,\delta t/2}|$.

4. Theoretical Performances of the *Parareal* Algorithm

[38] As mentioned previously, a misuse of the *parareal* algorithm can yield slow convergence and could result in a slower execution time, τ_{parareal}, compared to that of a serial execution, τ_{serial}. It is therefore important to identify the conditions for which the performances of the *parareal* algorithm are optimum, as well as those where the performances are reduced. This can easily be revealed by a theoretical performance model. Since the time to compute the solution $\mathcal{U}$ over a single time step is much larger than the time to communicate the solution from one processor to the other, the communication time is negligible and will not be considered. The validity of this approximation is shown *a posteriori*.

[39] The parallel performances of the algorithm can be measured by the speedup $S = \tau_{\mathrm{serial}}/\tau_{\mathrm{parareal}}$ and the efficiency $E = S/N$. The overall execution time of the parareal algorithm is:

$$\tau_{\mathrm{parareal}} \cong N\tau_c + K\left(\tau_f + N\tau_c\right) \tag{14}$$

where τ_c and τ_f represent the computational time associated to the use of the coarse and the fine

operators, respectively. The serial execution time for the same problem over the same time interval is simply:

$$\tau_{\text{serial}} \cong N\tau_f. \tag{15}$$

Therefore, the speedup of the parareal algorithm is:

$$S = \frac{N}{N\,\beta + K(1 + N\,\beta)}, \tag{16}$$

with $\beta = \tau_c/\tau_f$. Optimal speedup is achieved for $K = 1$ and $\beta \sim 0$. Conversely, the speedup degrades very quickly with the total number of parareal iterations K, and can even reach values lower than 1.

[40] Since the presented version of the *parareal* algorithm uses an elliptic-parabolic splitting (*i.e.*, the energy equation is solved more frequently than the Navier-Stokes equations when applying the coarse operator, see section 3.1) it is more revealing to express β more explicitly, by considering the contribution in solving the time-dependent energy equation, τ_{advdiff}, and the time required to solve the elliptic Navier-Stokes equations, τ_{NS}:

$$\beta = \frac{\tau_c}{\tau_f} = \frac{\tau_{NS} + n_{\text{CFL}}\tau_{\text{advdiff}}}{n_{\text{CFL}}(\tau_{\text{advdiff}} + \tau_{NS})} = \frac{1 + n_{\text{CFL}}\alpha}{n_{\text{CFL}}(1 + \alpha)}, \tag{17}$$

where $\alpha = \tau_{\text{advdiff}}/\tau_{\text{NS}}$ and the parameter n_{CFL} was defined in section 3.2.

[41] Using equations (16) and (17), one can predict the parallel performance of the algorithm as a function of the number of *parareal* iterations K, the number of slave processors N (*i.e.*, the total number of processors minus one, see section 3 and Figure 3), the size of the time sub-intervals, determined by the value of n_{CFL}, and α. The latter depends on the size of the problem n and the geometry considered (*i.e.*, 2D vs. 3D) but is always smaller than one. For instance, using the MUMPS library [*Amestoy et al.*, 1998] to solve the 2D Navier-Stokes equations recast as a biharmonic equation and an explicit scheme for the energy equation with $n = 10^4$ grid cells $\alpha \sim 0.03$ and decreases with decreasing n. This behavior could be even more pronounced in 3D geometries, since for large problems the computational time associated with the Navier-Stokes equations solved with a direct method tends to increase as $\sim n^2$, while the time to solve for the energy equation with an explicit method goes as $\sim n$.

[42] Further reduction of the computational cost associated to $\mathcal{C}_{\Delta t}$ can be obtained by decreasing the spatial resolution during the coarse time propagation, and remapping the results onto the original grid with finer spatial resolution. This can be performed using restriction and prolongation operations as it is done in geometric multigrid methods [*Brandt*, 1982]. In this case, one can introduce an additional parameter f, which expresses the spatial coarsening ratio used during the coarse propagation (*i.e.*, when using $\mathcal{C}_{\Delta t}$). For instance, one can decide to propagate the coarse solution on a grid that is twice coarser than the original one in all spatial directions (*i.e.*, $f = \Delta x_{\text{coarse}}/\Delta x = \bar{\Delta} y_{\text{coarse}}/\Delta y = \bar{\Delta} z_{\text{coarse}}/\Delta z = 2$). Then, in the case of a 2D (in space) problem the computational cost associated with the coarse propagation over one time sub-interval becomes:

$$\tau_c \cong \frac{1}{f^2}(\tau_{NS} + n_{\text{CFL}}\tau_{\text{advdiff}}), \tag{18}$$

where I have assumed that the dependence of both τ_{NS} and τ_{advdiff} with the problem size n is linear, which is reasonable for relatively small 2D problems.

[43] In that case, the theoretical expression of the parallel speedup becomes:

$$S = \frac{N}{N\,\beta\,f_0^{-2} + K\left(1 + N\,\beta\,f_k^{-2}\right)}, \tag{19}$$

where f_0 and f_k are the spatial coarsening ratios used during the initialization and the iterative predictor-corrector steps of the *parareal* algorithm, respectively. Tests were performed with f_0 and f_k equal to unity (*i.e.*, no spatial coarsening), or 2. While spatial coarsening inevitably decreases the computation time associated with $\mathcal{C}_{\Delta t}$, it may also degrade the accuracy of the coarse solution too severely, and consequently yield poor convergence of the *parareal* algorithm (*i.e.*, larger K values).

[44] Figure 5 shows the influence of the total number of iterations K on the speedup of the *parareal* algorithm for different values of N. The parallel performances of the algorithm decrease dramatically with increasing K, and speedups lower than 1 can even be obtained (grey areas in Figure 5). The strategy of the present paper is to determine the optimum set of adjustable parameters that yield convergence within the minimum number of *parareal* iterations: the size of the *parareal* interval n_{CFL}, the number of time sub-domains, N, and to some extent α, f_0 and f_k. Therefore in the following I will only focus on cases where $K = 1$.

[45] Figures 6a and 6b display the speedup and efficiency as a function of n_{CFL} and N, assuming

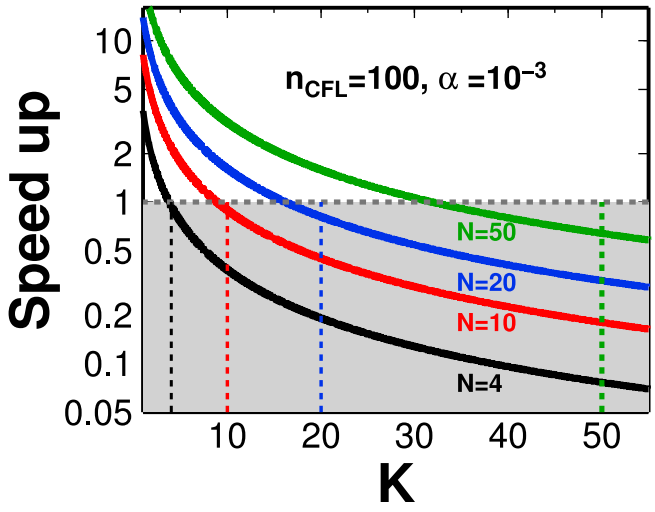

Figure 5. Result of the theoretical performance model (equation (16)). Speedup as a function of the *parareal* total iteration number K for various values of N. The speedup degrades very rapidly with increasing K, possibly reaching values lower than 1 (gray area) for which the *parareal* algorithm is slower than the sequential solution. This occurs when $K >\sim N$. See text for further details.

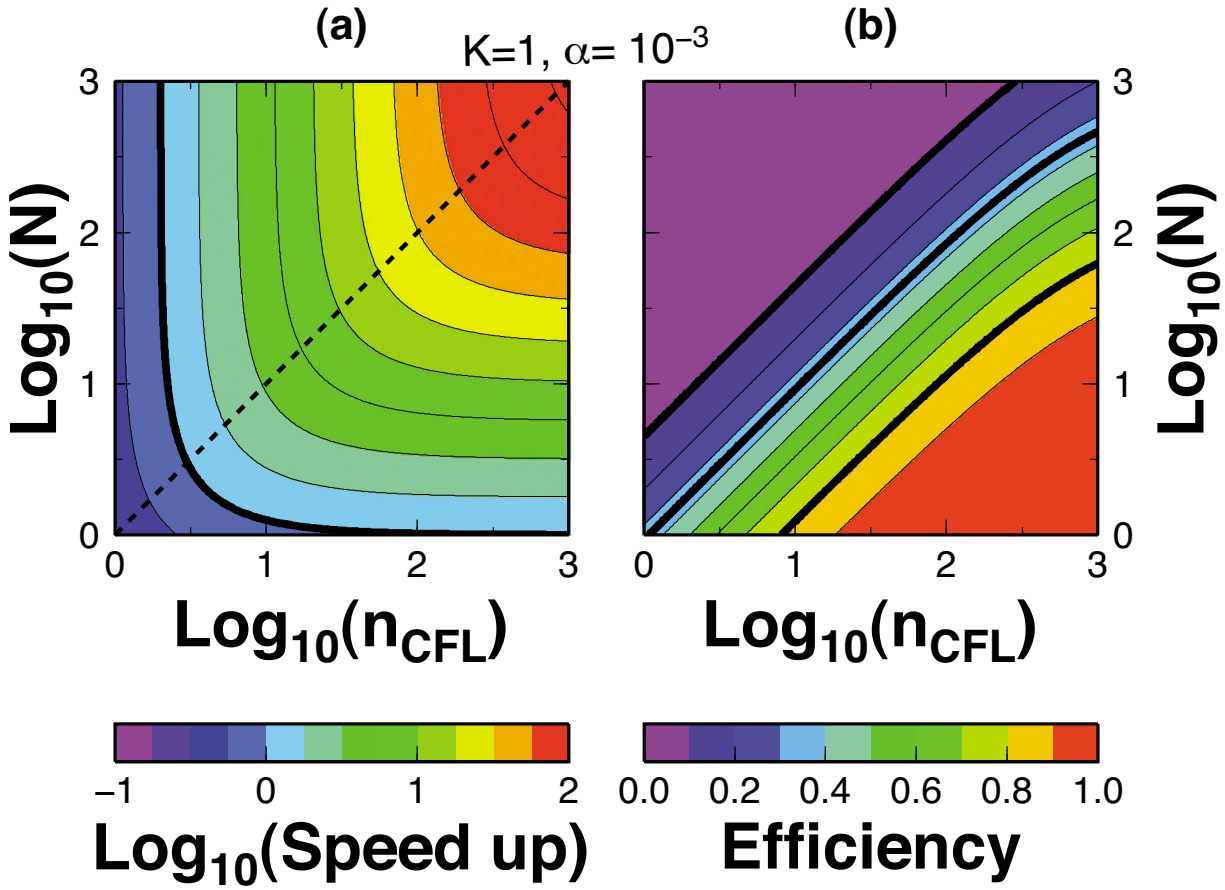

Figure 6. Results of the theoretical performance model, equation (16), with ($K = 1$ and $\alpha = 10^{-3}$). (a) Speedup S and (b) efficiency of the *parareal* algorithm as a function of the number of slave processors N and the size of the parareal interval n_{CFL}. The black thick curve in Figure 6a corresponds to $S = 1$, and the dashed line corresponds to the optimum setting $n_{CFL} = N$. The black thick curves in Figure 6b correspond to $E = 0.1$, $E = 0.35$ and $E = 0.8$. See text for further details.

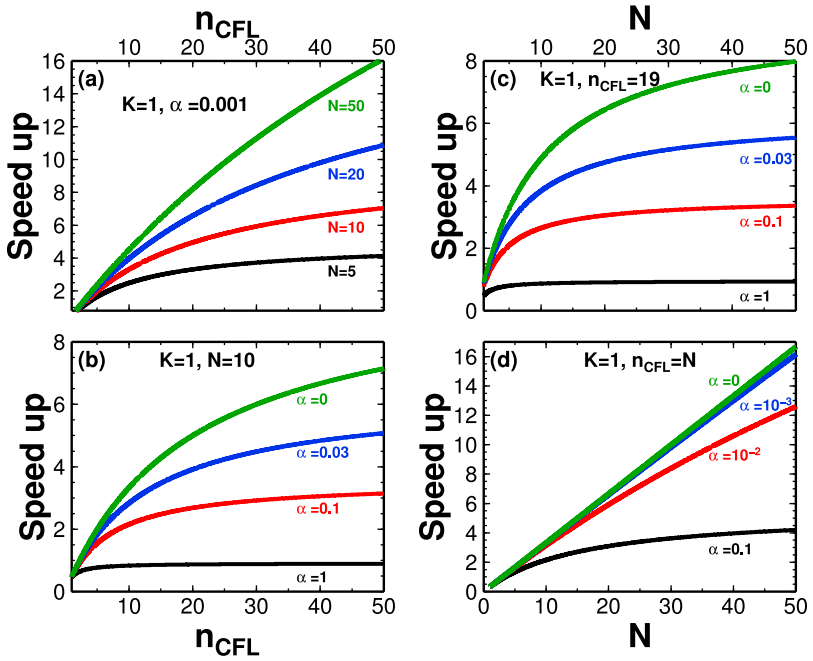

Figure 7. Results of the theoretical performance model, equation (16), with $K = 1$. Speedup S (a and b) as a function of the size of the parareal interval n_{CFL}, or (c and d) as a function of the number of slave processors N. Different values of α, N and n_{CFL} are shown.

$K = 1$ and $\alpha = 10^{-3}$. One clearly sees in Figure 6a that S increases with increasing either n_{CFL} or N. However, this increase in S rapidly stagnates if either n_{CFL} or N is held constant. This is shown more clearly in Figures 7a–7c. Therefore, in order to make the best use of the *parareal* algorithm the optimum setup is:

$$n_{CFL} \sim N. \tag{20}$$

[46] In this case, for small values of α, S increases with N (or n_{CFL}) almost linearly (dashed line in Figures 6a and 7d). In theory linear speedup of 100 can be reached with $N = n_{CFL} = 1000$ (Figure 6a). Figure 6b shows that even for the optimum setting (equation (20)), the *parareal* efficiency is far from 1 and increases with increasing the size of the *parareal* time interval (or n_{CFL}). This can be understood as increasing n_{CFL} tends to minimize the computational time spent during the initialization step, which is purely sequential (see section 3 and Figure 3). Although the speedup is linear along the optimum setting line, the parallel efficiency is close to 0.35 instead of 1 because the slope N/n_{CFL} is smaller than one.

[47] It is also important to measure the influence of α on the parallel speedup, which is not directly adjustable as α depends essentially on the spatial resolution of the computational domain. The later is displayed in Figures 7b–7d and shows that smaller values of α yield higher speedup and more linear speedup increase with increasing N, especially for cases with large number of processors. As mentioned previously, α should decrease with increasing the size and the spatial dimension of the computational domain. Therefore, one can expect an improvement of the *parareal* performances for large size problems with high dimensionality.

5. Performance Tests

[48] To measure the performances of the *parareal* algorithm and to compare them with the theoretical predictions (equations (16), (19), and (17)), two test cases relevant to typical geodynamic situations are considered. For both cases, $Ra = 10^7$, $\gamma = \ln(100)$ and equations (1) and (2) were formulated in terms of a stream function. The whole system is discretized with a finite volume method, using the *StreamV* code [*Samuel*, 2009; *Samuel and Evonuk*, 2010] that was benchmarked against various analytic and numerical solutions. The code is written in FORTRAN 95, using the MPI library for the communications. The overall implementation of the *parareal* algorithm (Figure 3) is rather short and is greatly facilitated by the use of object-oriented like programming, allowing more flexibility to manipulate the coarse and fine operators.

Using different MPI communicators, simultaneous space and time parallelization is possible. In that case, a group of processors belonging to same communicator are assigned to master tasks or to a specific slave time sub-interval.

[49] Equation (3) was discretized using a first order in time and second order in space Eulerian TVD scheme with a Sweby flux limiter. Further details about the implementation are given by *Samuel and Evonuk* [2010]. Unless specified otherwise, the results shown here were obtained using the MUMPS direct solver [*Amestoy et al.*, 1998].

5.1. Axisymmetric Plume Test

[50] I consider here the rise of a thermal plume through an axisymmetric domain discretized using 50 × 200 square cells. Fixed temperature and free-slip boundary conditions are applied on all external boundaries. The plume is generated by imposing a heating patch at the bottom of the domain. The test starts when the advective (CFL) time step becomes smaller than the diffusive time step, and ends at a dimensionless time $t = 3.510^{-4}$, which corresponds to 100 Myr if dimensionalized using a thermal diffusivity of 10^{-6} m^2/s, and a mantle thickness of 3000 km. Throughout the plume rise shown in Figure 8a, the average temperature and the Root Mean Squared velocity are monitored as proxies for the solution $\mathcal{U}(t)$ and displayed in Figures 8b and 8c. The *parareal* solution closely matches the sequential computation, both for temperature (Figure 8b) and velocities (Figure 8c).

[51] Figure 8d displays the number of *parareal* iterations required to reach convergence, K, as a function of the size of the time sub-interval n_{CFL}, with the optimum setup given by equation (20). Different values of spatial coarsening (f_0 and f_k) during the coarse propagation were considered, along with either an explicit or an implicit scheme for solving equation (3). For small sub-interval sizes $n_{\mathrm{CFL}} \leq 5$, all configurations converge within the minimum possible value of $K = 1$. The best convergence is observed when no spatial coarsening is used (black circles), with $K = 1$ up to $n_{\mathrm{CFL}} \leq 20$, followed by an increase for larger sizes of time interval. Spatial coarsening applied only during the iterative step of the *parareal* algorithm ($f_0 = 1$ and $f_k > 1$, red squares) also yields optimum convergence up to $n_{\mathrm{CFL}} \leq 20$, with however an improvement in parallel speedup as predicted by equation (19). Beyond this threshold, the convergence deteriorates dramatically with $K = 18$ for $n_{\mathrm{CFL}} = 30$. Further spatial coarsening ($f_0 > 1$ and $f_k > 1$, blue triangles) significantly degrades the *parareal* convergence for even smaller values of n_{CFL}, therefore reducing the parallel speedup (Figure 5). This is a logical consequence of the fact that a poor initial guess requires more corrections, as it is often the case with iterative methods. This suggests to avoid the use of spatial coarsening during the initialization step (*i.e.*, $f_0 > 1$). For similar reasons, the use of an implicit scheme to propagate the coarse solution is not recommended as it introduces larger amounts of numerical diffusion compared to the explicit scheme used here. Consequently, the accuracy of the initial guess decreases, leading to poor convergence (green diamonds). In summary, the optimum use of the *parareal* algorithm is obtained using the following setup: $n_{\mathrm{CFL}} = N = 20$, $f_0 = 1$, and $f_k = 2$.

[52] Figure 8e shows the speedup of the *parareal* calculations measured experimentally (squares) for different values of the number of slave processors N and compared with the theoretical predictions (curves). Two cases are considered: one where no spatial coarsening was considered ($f_0 = f_k = 1$) for $\mathcal{C}_{\delta t}$ (blue), and another where spatial coarsening was applied only during the predictor-corrector iterations ($f_0 = 1$, $f_k = 2$), yielding better performances, as predicted by equation (19). In this case, the *parareal* approach has reduced the computational execution by almost one order of magnitude, using 40 CPUs.

5.2. Rayleigh Bénard Convection Test

[53] This test consists in computing the evolution of a convective mantle of aspect ratio 1:3, starting from the initial condition displayed in Figure 9a, for a total time period of $t = 1.410^{-3}$, corresponding to 40 Myr using the characteristic scales listed in section 5.1. The spatial domain was discretized using 225 × 150 identical cells. Temperature is 0 at the top and 1 at the bottom boundaries. Free-slip boundary conditions are applied on the horizontal boundaries. The vertical side-walls are reflective. As for the plume case, the average temperature and the Root Mean Squared velocity are monitored as proxies for the solution $\mathcal{U}(t)$ and displayed in Figures 9b and 9c. As previously, the *parareal* solution closely matches the sequential computation. This is remarkable since the time dependence of the solution is strong as illustrated by the "roller-coaster" shape of the V_{RMS} time evolution shown in Figure 9c. This demonstrates the robustness of the *parareal* algorithm even after only one iteration $K = 1$.

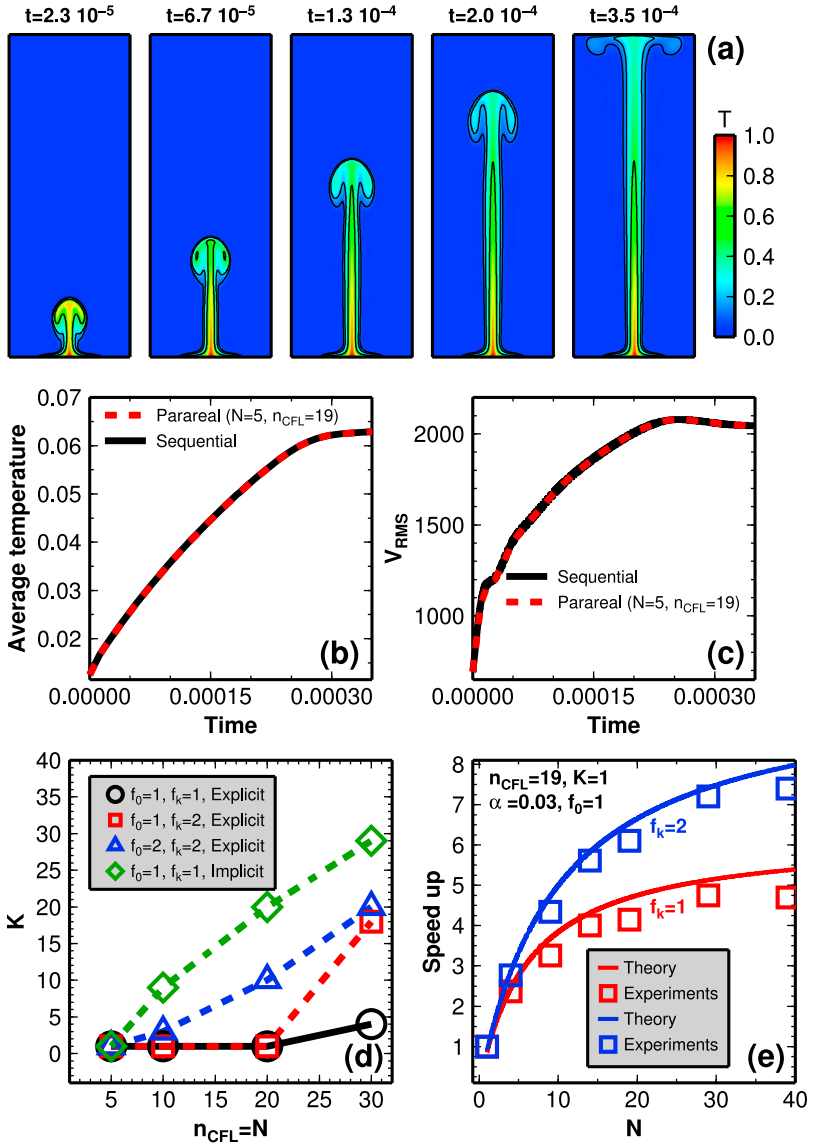

Figure 8. Result of the axisymmetric plume test ($Ra = 10^7$, $\gamma = \ln(100)$). (a) Snapshots in time showing the plume dimensionless temperature field. Comparison between the sequential (black) and *parareal* (red) solutions displayed with the (b) dimensionless average temperature and (c) RMS velocities. (d) Total number of *parareal* iterations K required to reach convergence as a function of the number of slave processors N, for the optimum setup ($N = n_{\mathrm{CFL}}$), for different values of spatial coarsening (f_0 and f_k), and using either explicit or implicit schemes for the coarse propagation. (e) Comparison of the speedup predicted with the theoretical model, equation (19), (solid curves) and the one measured experimentally (squares) for two configurations: $f_k = 1$ in red (*i.e.*, no spatial coarsening) and $f_k = 2$ in blue (*i.e.*, using spatial grid coarsening during the coarse propagation).

[54] Figure 9d displays the number of *parareal* iterations required to reach convergence, as a function of the size of the time sub-interval, with the optimum setup given by equation (20). Spatial coarsening (not shown here) was also tested, but contrary to the plume test case (Figure 8) it yielded systematically very poor convergence, with values of K close to N. The reason why spatial coarsening severely degrades the convergence here may be explained by the fact that further reduction of the spatial resolution may lead to stronger under-resolution of important features (such as thermal boundary layers) than for the plume case. In addition, as for the plume test, the use of an implicit scheme to propagate the coarse solution severely deteriorates the *parareal* convergence for all cases with $n_{\mathrm{CFL}} > 5$.

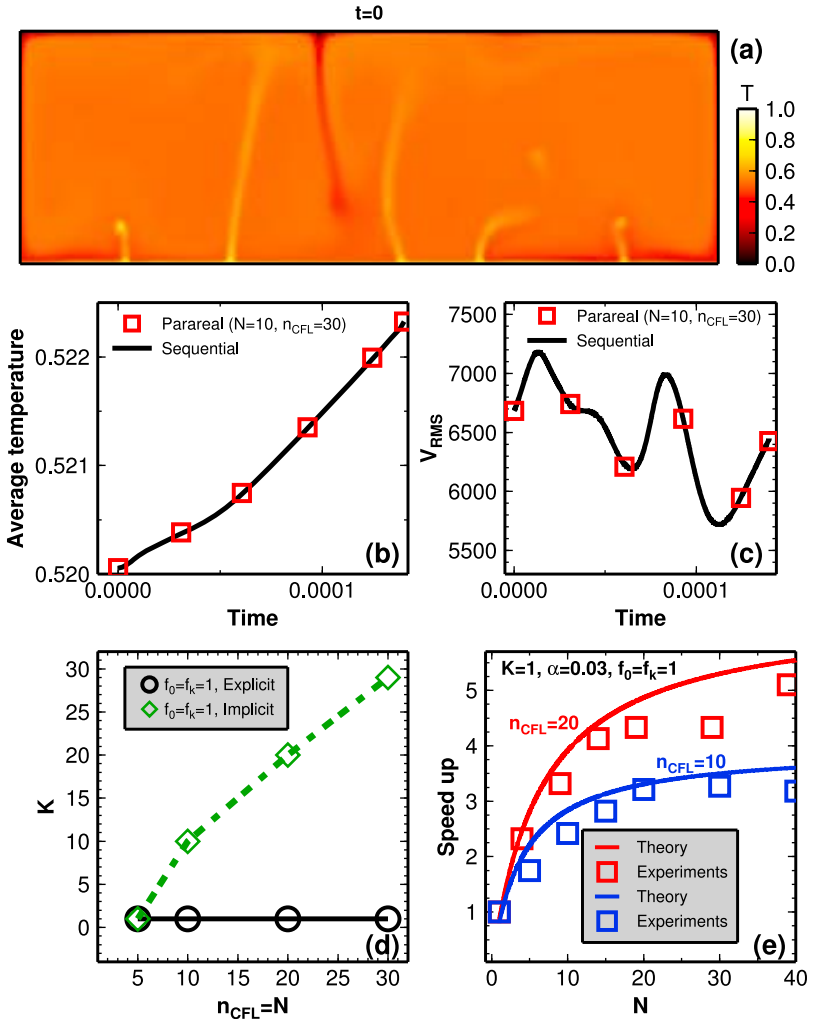

Figure 9. Result of the Rayleigh-Bénard convection test ($Ra = 10^7$, $\gamma = \ln(100)$). (a) Initial dimensionless temperature field. Comparison between the sequential (black) and *parareal* (red) solutions displayed with the (b) dimensionless average temperature and (c) RMS velocities. (d) Total number of *parareal* iterations K required to reach convergence as a function of the number of slave processors N, for the optimum setup ($N = n_{CFL}$), using either explicit or implicit schemes for the coarse propagation. (e) Comparison of the speedup predicted with the theoretical model (solid curves) and the speedup measured experimentally (squares) for two configurations: $n_{CFL} = 20$ in red and $n_{CFL} = 10$ in blue (*i.e.*, using a twice smaller *parareal* time interval).

[55] The measured *parareal* speedup displayed in Figure 9e for different values of N compares well with the theoretical predictions (curves) and shows that the computational cost is reduced by up to a factor 5, using 40 CPUs. While the obtained speedup is relatively modest, for a problem of such small size, spatial parallelization using the MUMPS direct solver would yield an even smaller speedup value (~1.8, using 40 CPUs).

5.3. Space and Time Parallelization

[56] One can further extend the *parareal* approach by combining it with spatial parallelization. To do so, I have considered the same Rayleigh Bénard convection test described previously, with a grid composed of 512 × 256 rectangular cells. Despite the use of a larger grid, the use of a parallel direct solver was found to be rather inefficient, with maximum speedups of 2 using up to 64 CPUs. Parallel direct solvers on distributed memory machines generally scale better for much larger 2D problems or for 3D problems. Therefore, instead of using a direct solver, the Navier-Stokes equations are solved here using a parallel geometric multigrid solver performing V-cycles with a simple domain decomposition. Figure 10 shows the parallel speed up obtained for pure spatial parallelization (red curve), pure time parallelization (blue curve) and a hybrid space-time domain parallelization (green

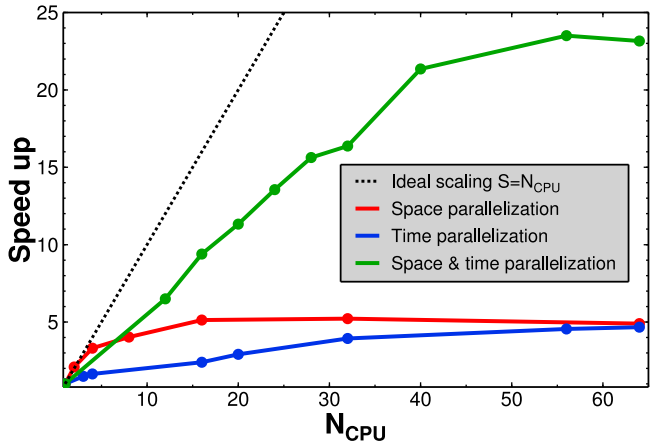

Figure 10. Result of the Rayleigh-Bénard convection test on a 512 × 256 grid. Parallel speedup as a function of the total number of processors used N_{CPU}. Case with pure spatial parallelization, pure time parallelization and hybrid space and time parallelization are shown with the red, blue and green solid lines, respectively. For the hybrid case, the spatial domain is decomposed into four identical sub-domains, each assigned to one processor. This results in four processors assigned to each (slave) time sub-domains and another group of four CPUs to perform the master tasks. The dashed line corresponds to an ideal linear scaling with a slope of one. The parameters used for the time parallelization are: $n_{CFL} = 20$, $f_0 = f_k = 1$.

curve) as a function of N_{CPU}, the total number of CPUs. Note that N_{CPU} used in Figure 10 is different from the total number of slave processes N; in the case of purely time parallelization $N = N_{CPU} = N - 1$. However, for space-time parallelization, more than one processor are assigned to the master tasks or to a given slave time sub-interval. Denoting this number by N_{space}, the number of time sub-interval is written $N = N_{CPU}/N_{space} - 1$.

[57] As for the previous experiments (e.g., Figures 8e and 9e), the speedup slope for the *parareal* case (blue curve) progressively decreases with increasing the number of CPUs.

[58] Similar to the behavior displayed in Figure 1, the speedup for the case with purely spatial parallelization first increases linearly with N_{CPU}. However, for N_{CPU} above 4, the communications start to dominate the execution time and the slope of the speedup curve progressively decreases to zero. In the case of both space and time parallelization, four processors are used for the spatial decomposition, as this number corresponds to the maximum value that yields optimum (linear) speedup scaling observed for the purely spatial parallelization (Figure 10, red line). This results in $N_{space} = 4$ processors assigned to each (slave) time sub-domain and another group of four CPUs to perform the master tasks (Figure 3). While the speedup for cases with either purely spatial or time parallelization saturates at $S \cong 5$, the combination of space and time domain decomposition yields a speedup close to 25, using up to 64 CPUs.

[59] This test shows that the *parareal* approach applied in addition to spatial decomposition drastically enhances the speedup by a factor ~5, even though saturation is reached for the spatial parallelization.

[60] Overall, the optimal use of space-time parallelization is obtained with applying the *parareal* algorithm when the following condition is met:

$$\left(\frac{dS}{dN}\right)_{parareal} > \left(\frac{dS}{dN_{CPU}}\right)_{spatial}. \tag{21}$$

[61] Using equation (16), equation (21) is more explicitly written:

$$\frac{K}{[N\ \beta + K(1 + N\ \beta)]^2} > \left(\frac{dS}{dN_{CPU}}\right)_{spatial}. \tag{22}$$

The above expression can be used to determine the number of processors used for the spatial parallelization, N_{space}.

6. Discussion

[62] The theoretical predictions and experimental tests presented in the previous sections have allowed to define the optimum conditions for the use of the parareal algorithm, which are summarized below:

[63] 1. A maximum size of time sub-interval Δt of about 20 CFL time steps (*i.e.*, $n_{CFL} = 20$). Larger

sizes of Δt were found to decrease the convergence rate of the algorithm for the problems considered. This value, however, is probably model-dependent, and may in particular be very sensitive to the time-dependency of the solution.

[64] 2. A number of time sub-intervals N equal to n_{CFL}.

[65] 3. The use of explicit schemes minimizing the numerical diffusion during the coarse propagation, combined with a parabolic-elliptic splitting (*i.e.*, the energy equation is solved more frequently than the Navier-Stokes equations). Coarse propagation using implicit schemes introduces larger amounts of numerical diffusion and results in poorer convergence of the algorithm.

[66] 4. Spatial coarsening during the coarse (time) propagation can improve the performance in some cases, but only if it is used during the iterative stage of the algorithm. Spatial coarsening during the initialization systematically yields poorer convergence and should be avoided.

[67] The performances of the present version of the *parareal* algorithm may be optimized in the future, for instance by considering different sizes of time sub-intervals Δt assigned to each processor and/or by adjusting the frequency of the elliptic-parabolic splitting according to the time dependence of the velocity field. In addition, a better spatial coarsening could be constructed using adaptive mesh refinements [*Alam et al.*, 2006]. Another possible way to improve convergence with large time interval sizes could be to use a higher order time integrator to compute the initial guess. These are few ongoing areas of investigations.

[68] Although this paper primarily focused on the use of the *parareal* algorithm to solid-state convective flows, the approach can certainly be applied to the modeling of finite Prandtl number fluid motions related to Earth and planetary dynamics: magma ocean and magma chamber dynamics [*Hoïnk et al.*, 2006; *Samuel*, 2012; *Verhoeven and Schmalzl*, 2009], geodynamo studies [*Christensen et al.*, 1999], or giant planets dynamics [*Evonuk and Glatzmaier*, 2004].

7. Conclusion

[69] I have presented a time domain parallel algorithm [*Lions et al.*, 2001] adapted to the resolution of infinite Prandtl number convection relevant to geodynamic problems. This *parareal* approach is based on the use of coarse and fine operators to predict and to iteratively correct the solution over a given time interval. The coarse operator, applied serially, propagates the solution in time using a time step larger than a CFL time step, while the fine operator propagates the solution using a CFL time step and can be applied in parallel, over N time sub-intervals, distributed among at least N slave processors. Although the algorithm must converge to the accuracy of the fine operator within at most N iterations, I have verified experimentally that convergence can be achieved within the minimum of one iteration, even for cases with large values of N (~10–100). Using a simple performance model I have shown that under optimum conditions, the parallel execution time scales linearly with the number of CPUs used. These theoretical predictions are in good agreement with numerical experiments (axisymmetric plume and 2D Rayleigh-Bénard convection), for which speedup close to 10 were measured, using up to 40 CPUs.

[70] Another attractive feature of the *parareal* algorithm is that it can be combined to other parallel spatial domain decomposition methods, which alone tend to saturate when the number of CPUs is too large and the problem size is too small. In that case, speedups close to 25 were obtained with a space-time parallelization, using up to 64 CPUs.

[71] As present parallel codes used for geodynamic modeling only use spatial decomposition, the addition of parallel time domain decomposition should allow a significant (10-fold and more) increase in speedup, even for the most computationally demanding models.

Acknowledgments

[72] This work was inspired from a conversation with Laurent Guillot during a memorable flight connection. I thank two anonymous reviewers and the Editor for their constructive comments. All the calculations were performed on a Linux cluster funded by the Stifterverband für die Deutsche Wissenschaft. All plots were made with the Generic Mapping Tools [*Wessel and Smith*, 1995].

References

Alam, J. M., N. Kevlahan, and O. V. Vasilyev (2006), Simultaneous space-time adaptive wavelet solution of nonlinear parabolic differential equations, *J. Comput. Phys.*, *214*, 829–857.

Aleksandrov, V., and H. Samuel (2011), The schur complement method solution of large-scale problems in geophysics, to be sumitted to *J. Comput. Phys*.

Amestoy, P. R., I. S. Duff, and J.-Y. L'Excellent (1998), Multifrontal parallel distributed symmetric and unsymmetric solvers, *Comput. Methods Appl. Mech. Eng.*, *184*, 501–520.

Baffico, L., S. Bernard, Y. Maday, G. Turinici, and G. Zérah (2002), Parallel in time molecular dynamics simulations, *Phys. Rev. E*, *66*, 057701.

Brandt, A. (1982), Guide to multigrid development, *Lect. Notes Math.*, *960*, 220–312.

Braun, J., C. Thieulot, P. Fullsack, M. Dekool, C. Beaumont, and R. Huismans (2008), DOUAR: A new three-dimensional creeping flow numerical model for the solution of geological problems, *Phys. Earth Planet. Inter.*, *171*, 76–91.

Bunge, H.-P., and J. Baumgardner (1995), Mantle convection modeling on parallel virtual machines, *Comput. Phys.*, *9*, 207–215.

Choblet, G., O. Čadek, F. Couturier, and C. Dumoulin (2007), Oediupus: A new tool to study the dynamics of planetary interiors, *Geophys. J. Int.*, *170*, 9–30.

Christensen, U. R., P. Olson, and G. A. Glatzmaier (1999), Numerical modeling of the geodynamo: A systematic parameter study, *138*, 393–409.

Evonuk, M., and G. Glatzmaier (2004), 2D studies of various approximations used for modeling convection in giant planets, *Geophys. Astrophys. Fluid Dyn.*, *98*, 241–255.

Farhat, C., and M. Chandesris (2003), Time-decomposed parallel time-integrators: Theory and feasibility studies for fluid, structure and fluid-structure applications, *58*, 1397–1434.

Fisher, P., F. Hecht, and Y. Maday (2003), A parareal in time semi-implicit approximation of the navier-stokes equations, paper presented at the 15th International Conference on Domain Decomposition Methods, Frei Univ. Berlin, Berlin.

Hoïnk, T., J. Schmalzl, and U. Hansen (2006), Dynamics of metal-silicate separation in a terrestrial magma ocean, *Geochem. Geophys. Geosyst.*, *7*, Q09008, doi:10.1029/2006GC001268.

Hütigg, C., and K. Stemmer (2008), The spiral grid: A new approach to discretize the sphere and its application to mantle convection, *Geochem. Geophys. Geosyst.*, *9*, Q02018, doi:10.1029/2007GC001581.

Jiang, G. S., and C. W. Shu (1996), Efficient implementation of weighted ENO schemes, *J. Comput. Phys.*, *126*, 202–228.

Kageyama, A., and T. Sato (2004), "Yin-Yang grid": An overset grid in spherical geometry, *Geochem. Geophys. Geosyst.*, *5*, Q09005, doi:10.1029/2004GC000734.

Katz, R. F., M. Knepley, B. Smith, M. Spiegelman, and E. Coon (2007), Numerical simulation of geodynamic processes with the portable extensible toolkit for scientific computation, *Phys. Earth Planet. Inter.*, *163*, 52–68.

Lepsa, B., and A. Sandu (2010), An effcient error control mechanism for the adaptive parareal time discretization algorithm, paper presented at Spring Simulation Multiconference, Soc. for Model. and Simul. Int., New York.

Lions, J.-L., Y. Maday, and G. Turicini (2001), Résolution d'EDP par un schéma en temps "pararéel," *C. R. Acad. Sci.*, *332*, 661–668.

Liu, Y., and J. Hu (2008), Modified propagators of parareal in time algorithm and application to Princeton Ocean model, *Int. J. Numer. Methods Fluids*, *57*, 1793–1804.

Mercerat, E. D., L. Guillot, and J.-P. Villotte (2009), Application of the parareal algorithm for acoustic wave propagation, paper presented at 7th International Conference of Numerical Analysis and Applied Mathematics, Eur. Soc. of Comput. Methods in Sci. and Eng., Rethymno, Greece.

Roe, P. L. (1986), Characteristic-based schemes for the Euler equations, *Annu. Rev. Fluid Mech.*, *18*, 337–365.

Samaddar, D., D. E. Newman, and R. Sànchez (2010), Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm, *J. Comput. Phys.*, *229*, 6558–6573.

Samuel, H. (2009), STREAMV: A fast, robust and modular numerical code for modeling various 2D geodynamic scenarios, annual report, Bayerisches Geoinst., Bayreuth, Germany.

Samuel, H. (2012), A re-evaluation of metal diapir breakup and equilibration in terrestrial magma oceans, *Earth Planet. Sci. Lett.*, *313–314*, 105–114 doi:10.1016/j.epsl.2011.11.001.

Samuel, H., and M. Evonuk (2010), Modeling advection in geophysical flows with particle level sets, *Geochem. Geophys. Geosyst.*, *11*, Q08020, doi:10.1029/2010GC003081.

Schmalzl, J., and U. Hansen (2000), A fully implicit model for simulating dynamo action in a Cartesian domain, *Phys. Earth Planet. Inter.*, *120*, 339–349.

Schmidt, J., C. Piret, N. Zhang, B. J. Kadlec, D. A. Yuen, Y. Liu, G. B. Wright, and E. O. D. Sevre (2010), Modeling of tsunami waves and atmospheric swirling flows with graphics processing unit (GPU) and radial basis functions (RBF), *Concurrency Comput. Practice Exper.*, *22*, 1813–1835.

Suckale, J., J.-C. Nave, and B. H. Hager (2010), It takes three to tango: 1. Simulating buoyancy-driven flow in the presence of large viscosity contrasts, *J. Geophys. Res.*, *115*, B07409, doi:10.1029/2009JB006916.

Sweby, P. K. (1984), High resolution schemes using flux limiters for hyperbolic conservation laws, *SIAM J. Numer. Anal.*, *21*, 995–1011.

Tackley, P. J. (2008), Modelling compressible mantle convection with large viscosity contrasts in a three-dimensional spherical shell using the yin-yang grid, *Phys. Earth Planet. Inter.*, *171*, 7–18.

Thieulot, C. (2011), FANTOM: Two- and three-dimensional numerical modelling of creeping flows for the solution of geological problems, *Phys. Earth Planet. Inter.*, *188*, 47–68.

Tosi, N., D. A. Yuen, and O. Čadek (2010), Dynamical consequences in the lower mantle with the post-perovskite phase change and strongly depth-dependent thermodynamic and transport properties, *Phys. Earth Planet. Inter.*, *298*, 229–243.

Trindade, J. M., and J. C. Pereira (2004), Parallel-in-time simulation of the unsteady Navier-Stokes equations for incompressible flow, *Int. J. Numer. Methods Fluids*, *45*, 1123–1136.

Trindade, J. M., and J. C. Pereira (2006), Parallel-in-time simulation of two-dimensional, unsteady, incompressible laminar flows, *Int. J. Numer. Methods Fluids*, *50*, 25–40.

Verhoeven, J., and J. Schmalzl (2009), A numerical method for investigating crystal settling in convecting magma chambers, *Geochem. Geophys. Geosyst.*, *10*, Q12007, doi:10.1029/2009GC002509.

Wessel, P., and W. H. F. Smith (1995), New version of the generic mapping tools, *Eos Trans. AGU*, *76*, 33, doi:10.1029/95EO00198.

Zhong, S., M. T. Zuber, L. N. Moresi, and M. Gurnis (2000), The role of temperature-dependent viscosity and surface plates in spherical shell models of mantle convection, *J. Geophys. Res.*, *105*, 11,063–11,082.

Zhong, S., A. McNamara, E. Tan, L. Moresi, and M. Gurnis (2008), A benchmark study on mantle convection in a 3-D spherical shell using CitcomS, *Geochem. Geophys. Geosyst.*, *9*, Q10017, doi:10.1029/2008GC002048.