Communication-aware adaptive parareal with application to a nonlinear hyperbolic system of partial differential equations

Allan S. Nielsen*, Gilles Brunner, Jan S. Hesthaven

Chair of Computational Mathematics and Simulation Science, Section de Mathématiques, École Polytechnique Fédérale de Lausanne

Abstract

In the strong scaling limit, the performance of conventional spatial domain decomposition techniques for the parallel solution of PDEs saturates. When sub-domains become small, halo-communication and other overheard come to dominate. A potential path beyond this scaling limit is to introduce domain-decomposition in time, with one such popular approach being the Parareal algorithm which has received a lot of attention due to its generality and potential scalability. Low efficiency, particularly on convection dominated problems, has however limited the adoption of the method. In this paper we introduce a new strategy, Communication Aware Adaptive Parareal (CAAP) to overcome some of the challenges. With CAAP, we choose time-subdomains short enough that convergence of the Parareal algorithm is quick, yet long enough that the overheard of communicating time-subdomain interfaces does not induce a new limit to parallel speed-up. Furthermore, we propose an adaptive work scheduling algorithm that overlaps consecutive Parareal cycles and decouples the number of time-subdomains and number of active node-groups in an efficient manner to allow for comparatively high parallel efficiency. We demonstrate the viability of CAAP trough the parallel-in-time integration of a hyperbolic system of PDEs in the form of the two-dimensional nonlinear shallow-water wave equation solved using a 3rd order accurate WENO-RK discretization. For the computational cheap approximate operator needed as a preconditioner in the Parareal corrections we use a lower order Roe type discretization.

Time-parallel integration of purely hyperbolic type evolution problems is traditionally considered impractical. Trough large-scale numerical experiments we demonstrate that with CAAP, it is possible not only to obtain time-parallel speedup on this class of evolution problems, but also that we may obtain parallel acceleration beyond what is possible using conventional spacial domain-decomposition techniques alone. The approach is widely applicable for parallel-in-time integration over long time domains, regardless of the class of evolution problem.

Keywords: PDE, Parallel-in-time, Parareal, Shallow Water Wave, Hyperbolic, HPC, Tsunami Simulation

1. Introduction

The ongoing and rapid evolution of computers used to model physical phenomenons in the computational sciences poses new challenges for algorithms. The growing number of cores, the increasingly convoluted cache hierarchies, and the use of accelerators all seek to boost the computational capacity of individual nodes. At the same time, the number of compute nodes in distributed memory machines has increased dramatically. The machine that currently crowns the top500 list of supercomputer Sunway TaihuLight at NSCC Wuxi, China, has more than 40.000 compute nodes, comprising a total of more than 10 millions cores. This ongoing development towards increasing hardware parallelism exposes algorithmic shortcomings and requires a rethinking of the fundamental algorithms to maintain scalability and enable efficient use of the computing platformDongarra et al. (2014). In this paper we show how the Parareal method may be modified in such a way as to allow for parallel-in-time acceleration of a tsunami simulation beyond what is possible using conventional spacial domain-decomposition techniques alone. The underlying PDE

*Corresponding Author

Email addresses: allan.nielsen@epfl.ch (Allan S. Nielsen), gilles.brunner@alumni.epfl.ch (Gilles Brunner), jan.hesthaven@epfl.ch (Jan S. Hesthaven)

governing the dynamics of the model is the shallow water wave equation. The equation is a purely hyperbolic system of coupled non-linear PDEs, the solutions of which typically contain both shocks and smooth regions interacting in a non-trivial manor. We thus conjecture that the positive result, presented in section 6, is an indication that it is possible for other similar systems to benefit from parallel-in-time acceleration.

1.1. Time-Domain Parallelism and Parareal

Solving time dependent PDEs is often done in a methods-of-line approach where the spatial components are discretized in some appropriate manner and a numerical integration technique is applied to advance in time. The approach extends to distributed memory machines by applying some form of domain decomposition, letting independent nodes communicate boundary information of their local sub-domains. The limitation to the approach lies in the strong scaling limit, i.e. increasing the number of nodes for a fixed problem size to achieve a reduction in time to compute.

One might naively suspect that one may "run out of parallelism" - i.e. as the combined number of cores become sufficiently high, there are simply not enough parallel degree's of freedom for all cores to work all the time. But consider this, solving a problem with upwards of a billion degree's of freedom in space may today be done on a potent workstation. Conversely, even the largest clusters available for researchers in the world has no more than a few million cores, everything included. This scaling limit is therefore somewhat theoretical, and not yet of much relevance for practitioners. So why does obtainable speed-up saturate in the strong-scaling limit? Consider what happens as spacial sub-domains decrease in size, given a three dimensional domain in space divided into a number of quadratic sub-domains with n elements spanning each dimension. The compute work in each sub-domain is proportional to n^3. The boundary information that needs to be exchanged with neighboring sub-domains is proportional to $\sim n^2$. As $n \to 1$, compute nodes will increasingly be spending time communicating boundary information rather than computing.

This particular limit is very much of practical concern. Moving a double between two individual compute nodes in a cluster is many orders of magnitude more expensive than a compute operation in terms of both wall-time and energy consumption. On large machines comprising thousands of nodes, this is a substantial bottleneck for scaling application efficiently and new algorithmic developments are required. A potential new path in obtaining scaling beyond what is possible with conventional methods, is to extract parallelism in the time integration procedure. Once a system of partial differential equation has been reduced to a large system of ordinary differential equations to be integrated over time, the problem is usually viewed as a sequential process. However, many attempts to extract parallelism do exists. For a complete overview of research in the direction we refer to a recent paper Gander (2015).

The focus of this paper will be on the Parareal method that has received a lot of attention over the past decade. The Parareal method, first proposed in Lions et al. (2001), borrows from ideas in spatial domain decomposition to construct an iterative approach for solving the temporal problem in a parallel global-in-time approach. To present the method, consider a problem on the form

$$\begin{cases} \frac{d\mathbf{u}}{dt} + \mathcal{A}(t, \mathbf{u}) = 0 \\ \mathbf{u}(T_0) = \mathbf{u}_0 \quad t \in [T_0, T] \end{cases} \tag{1}$$

where $\mathcal{A} : \mathbb{R} \times V \to V'$ is a general operator depending on $\mathbf{u} : \Omega \times \mathbb{R}^+ \to V$ with V being a Hilbert space and V' its dual. Now, assume there exists a unique solution $\mathbf{u}(t)$ to (1) and decompose the time domain into n_t individual time slices

$$T_0 < T_1 < \cdots < T_{n_t-1} < T_{n_t} = T. \tag{2}$$

Let $T_n = n\Delta T$. We now define a numerically accurate solution operator $\mathcal{F}_{\Delta T}$ that, for any $t > T_0$, advances the solution as

$$\mathcal{F}_{\Delta T}(T_n, \mathbf{u}(T_n)) = \mathbf{U}_{T_n+\Delta T} \approx \mathbf{u}(T_n + \Delta T) \tag{3}$$

To solve (1) on $[T_0, T_0 + n_t\Delta T]$, define the matrix of operators $M_{\mathcal{F}}$

$$M_{\mathcal{F}} = \begin{bmatrix} 1 & & & \\ -\mathcal{F}_{\Delta T}^{T_0} & \ddots & & \\ & \ddots & \ddots & \\ & & -\mathcal{F}_{\Delta T}^{T_{n_t-1}} & 1 \end{bmatrix} \tag{4}$$

with $\bar{\mathbf{U}} = [\mathbf{U}_0, \ldots, \mathbf{U}_{n_t}]$ and $\bar{\mathbf{U}}_0 = [u(T_0), 0, \ldots, 0]$. The sequential solution procedure is then equivalent to solving

$$M_{\mathcal{F}} \bar{\mathbf{U}} = \bar{\mathbf{U}}_0 \tag{5}$$

by forward substitution for $\bar{\mathbf{U}}$ so to recover $\mathbf{U}_0 \cdots \mathbf{U}_{n_t}$ as approximations to $\mathbf{u}(T_0) \cdots \mathbf{u}(T_{n_t})$. If we instead seek to solve the system using a point-iterative approach, i.e., we seek the solution $\bar{\mathbf{U}}^{k+1} = \bar{\mathbf{U}}^k + \left(\bar{\mathbf{U}}_0 - M_{\mathcal{F}} \bar{\mathbf{U}}^k\right)$, we observe that at the beginning of each iteration, $\bar{\mathbf{U}}^k$ is known. In each iteration we may thus compute $\mathcal{F}_{\Delta T}^{T_1} \cdots \mathcal{F}_{\Delta T}^{T_{n_t}}$ on all intervals in parallel. Note that the computational complexity of every iteration is strictly larger than that of the sequential solution procedure. Hence a reduced time to solution is possible only if the number of iterations k_{conv} needed for convergence is much smaller than the number of time sub-domains n_t. To achieve this, a preconditioner is needed. Assuming the existence of some $M_{\mathcal{G}} \approx M_{\mathcal{F}}$, where $M_{\mathcal{G}}$ is computationally cheap, we can solve a preconditioned system on the form

$$(M_{\mathcal{G}})^{-1} M_{\mathcal{F}} \bar{\mathbf{U}} = (M_{\mathcal{G}})^{-1} \bar{\mathbf{U}}_0 \tag{6}$$

instead. A natural approach to construct the above preconditioner $M_{\mathcal{G}}$ is to define a new operator $\mathcal{G}_{\Delta T}$ as with $\mathcal{F}_{\Delta T}$

$$\mathcal{G}_{\Delta T}(T_n, \mathbf{u}(T_n)) = \mathbf{U}_{T_n + \Delta T} \approx \mathbf{u}(T_n + \Delta T) \tag{7}$$

and relax the requirements on the accuracy of $\mathcal{G}_{\Delta T}$, by using a coarser grid or a different numerical model. Solving the system (6) iteratively by a standard preconditioned Richardson iterations we have

$$\bar{\mathbf{U}}^{k+1} = \bar{\mathbf{U}}^k + (M_{\mathcal{G}})^{-1} \left(\bar{\mathbf{U}}_0 - M_{\mathcal{F}} \bar{\mathbf{U}}^k\right) \tag{8}$$

which is equivalent to

$$\begin{bmatrix} 1 & & & \\ -\mathcal{G}_{\Delta T}^{T_0} & \ddots & & \\ & \ddots & \ddots & \\ & & -\mathcal{G}_{\Delta T}^{T_{n_t-1}} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{U}_0^{k+1} \\ \mathbf{U}_1^{k+1} \\ \vdots \\ \mathbf{U}_{n_t}^{k+1} \end{bmatrix} = \begin{bmatrix} 1 & & & \\ \mathcal{F}_{\Delta T}^{T_0} - \mathcal{G}_{\Delta T}^{T_0} & \ddots & & \\ & \ddots & \ddots & \\ & & \mathcal{F}_{\Delta T}^{T_{n_t-1}} - \mathcal{G}_{\Delta T}^{T_{n_t-1}} & 1 \end{bmatrix} \tag{9}$$

from this we recover the Parareal algorithm in the form that it is typically presented

$$\mathbf{U}_{n+1}^{k+1} = \mathcal{G}_{\Delta T}^{T_n} \mathbf{U}_n^{k+1} + \mathcal{F}_{\Delta T}^{T_n} \mathbf{U}_n^k - \mathcal{G}_{\Delta T}^{T_n} \mathbf{U}_n^k \quad \text{with} \quad \mathbf{U}_{n+1}^0 = \mathcal{G}_{\Delta T}^{T_n} \mathbf{U}_n^0 \quad \text{and} \quad \mathbf{U}_0^k = \mathbf{u}(T_0) \tag{10}$$

A comprehensive introduction to Parareal may be found in Nielsen (2012), whilst important early contributions on the analysis of the method can be found in Staff and Rønquist (2005); Bal (2005); Gander and Vandewalle (2007). The algorithm has been applied to a wide range of applications such as simulation of dynamic physical models in automotive industry by Loderer et al. (2014), and pricing of American Options by Pages et al. (2016). Notably it has also been applied to plasma simulation with some success as presented in Samaddar et al. (2010); Reynolds-Barredo et al. (2012). Using various coarse operators the authors achieve parallel-in-time speed-up, although with low parallel efficiency of single digit percent whilst using up to 400 processors in time. In Randles and Kaxiras (2014), convergence of the Parareal algorithm on lattice Boltzmann applied to a laminar flow problem is presented. It is demonstrated that parallel speed-up is possible, albeit again with low parallel efficiency, and no comparison with conventional domain decomposition in space is supplied. Numerical experiments of parallel-in-time integration using Parareal on the three-dimensional incompressible Navier-Stokes equations on a cavity problem is presented in Croce et al. (2014). The authors report that the space-time-parallel method can provide speedup beyond the saturation of a purely space-parallel approach. In the cavity test case, the performance saturates at a speedup of 18 with 32 cores in space. Using another 16 time-subdomains they report a combined space-time parallel speed-up of 27 using a total of 512 cores. Their results are in line with previous results, reporting low parallel efficiency on a limited time-domain, but nevertheless reaching higher speedup than what is possible with the purely space-parallel approach. Similar results are reported in Minion (2011); Speck et al. (2012), on a space-time parallel version of the Barnes-Hut tree code. The authors use PFASST for parallel-in-time integration, and report scaling up to 262,144 cores on the IBM Blue Gene/P installation JUGENE, demonstrating that the space-time parallel code provides speedup beyond the saturation

of the purely space-parallel approach. Low parallel efficiency due to slow convergence of the algorithm is a general trend in the results being reported. In Steiner et al. (2015) the authors present numerical experiments measuring the convergence of the Parareal algorithm applied to the two dimensional Navier-Stokes equations on a driven cavity benchmark for different Reynolds numbers. They report that the problems of instability and slow convergence increase with decreasing viscosity, i.e., when the flow becomes increasingly dominated by convection. The effect is found to strongly depend on the spatial resolution of the problem. In Dai and Maday (2013) the authors presents an analysis on the stability of Parareal applied to hyperbolic systems and convection dominated problems and they show that the instabilities are related to the regularity of the solution over time. They propose a stabilization scheme that modifies the iterative algorithm in such a way to avoid any transient phase of divergence before convergence. However convergence is observed to still be slow for long time-subdomains and the genereralization of this approach is unclear. Other stabilizing schemes has later been proposed Chen et al. (2014, 2015), but suffer from the same limitatins. An important contribution to the understanding of how convergence is affected by the length of the time-interval to be integrated in parallel was presented in Gander and Hairer (2014). The authors show that for Hamiltonian systems, for a given problem with some coarse integrator, there exists a "window" in which time parallel integration is possible, and outside of which the method does not convergence. The author similarly shows that convergence speed increase with smaller time-subdomains. In a recent paper Eghbal et al. (2016), it is conjectured that there exists an optimal time-subdomain length at which convergence is quick, yet the time-subdomain is still long enough that the communication of time-subdomain interfaces does not become a limiting factor to parallel speedup.

In this paper we explore this relationship, and in Section 4 we present theoretical considerations on how to a priori estimate the optimal time-subdomain length for the time-decomposition. To effectively decouple the time-subdomain length and the total time to be integrated with Parareal, we introduce an adaptive Parareal variant in Section 3 based on the scheduler introduced in Aubanel (2011). The approach allows consecutive Parareal cycles to overlap in time to balance processor utilization and convergence efficiently. Henceforth, we will denote the combination of the adaptive work scheduler and an informed choice on the time-subdomain length as CAAP, an abbreviation of Communication-aware Adaptive Parareal. The shallow water wave equation test case is introduced in Section 2 along with relevant notation, and the numerical experiments and scaling tests that demonstrate the performance of our approach are presented in Section 5.

2. Test case: 2D Shallow Water Wave Equation

The shallow water wave equation is used to model a wide array of wave phenomenons. Simulation of trans-ocean waves, flows in rivers and coastal areas, hydraulic engineering, and atmospheric modeling are among the many examples of application. The system of coupled partial differential equations that constitutes the shallow water wave equation is nonlinear and purely hyperbolic. The equation captures fundamental phenomena across different scales in space and time involving shocks that may form during the solutions procedure even for perfectly smooth initial conditions. The equation is therefore a challenging case for parallel-in-time integration and, as such, an excellent platform for measuring to what extend time-parallel integration is possible for hyperbolic problems. The two dimensional version of the equations may be written as

$$\begin{cases} h_t + (hu)_x + (hv)_y = 0 \\ (hu)_t + \left(hu^2 + \frac{1}{2}gh^2\right)_x + (huv)_y = -ghz_x \\ (hv)_t + (huv)_x + \left(hv^2 + \frac{1}{2}gh^2\right)_y = -ghz_y \end{cases} \tag{11}$$

where $h := h(x, y, t)$ denotes the water height, $u := u(x, y, t)$ and $v := v(x, y, t)$ the velocity in x and y direction respectively. $z := z(x, y)$ denotes overland topography and underwater bathymetry whilst g denotes the gravitational constant. The equation is often presented in conservation form as

$$q_t + f(q)_x + g(q)_y = s(h, z) \tag{12}$$

where $q = (h, hu, hv)^T$, and the two flux functions $f(q)$, $g(q)$ are given by

$$f(q) = \begin{pmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{pmatrix}, \; g(q) = \begin{pmatrix} hv \\ huv \\ hv^2 + \frac{1}{2}gh^2 \end{pmatrix} \tag{13}$$

The source term $s(h, z)$ is needed for non-flat bathymetrys. For inundation modeling, the source term is often also made to include Manning's law, an empirically derived friction term that is added to better capture the physics of land-overflow. In the code the be accelerated, a simple thin-layer mesh reduction technique is used for inundation modelling, but no friction terms are added as hydrological studies is not the primary concern of this paper. We refer to the works of Maidment and Mays (1988); Xing and Shu (2011) for an introduction to the shallow water wave equation and inundation modelling. Finite volume schemes are a popular approach for solving hyperbolic conservation laws as the underlying physics is represented in a natural way. Let $I_{i,j} = \left[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}\right] \times \left[y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}\right]$ define a structured rectangular uniform mesh. In a finite-volume scheme, we seek the cell average

$$Q_{ij}(t) = \frac{1}{\Delta x \Delta y} \int_{I_{i,j}} q(x, y, t)\, dxdy \tag{14}$$

that approximates $q(x, y, t)$ at every cell $I_{i,j}$ for a given time-step t. The spatial derivatives must therefore be approximated using cell-averages. In the section that follows we give a short introduction to the Weighted Essentially Non-Oscillatory (WENO) scheme for doing so. The scheme is used to discretize the equations in space, and the resulting system of coupled ODE's may then be integrated using a Strong-Stability-Preserving type, explicit Runge-Kutta scheme from some given initial condition, see Shu (1998) for a comprehensive introduction to the schemes.

2.1. The Test-Case

We present the test-case used to evaluate to what extend the original Parareal algorithm and CAAP can be used to accelerate the process of finding a numerical approximation to the solution of (11). The test-case uses the radially-symmetric elliptic paraboloid bathemetry as the classic test-case presented in Thacker (1981), but with a different initial condition so that shocks develop as time progresses. In the model, a simulation of inundation is included. The initial condition in the model proposed by Thacker is a standing half-wave in a radially-symmetric elliptic paraboloid bathymetry. For the test case, Thacker presents an analytical solution. The case is excellent for testing correctness of an implementation, but is insufficient in the context of time-parallel integration as the test-case contains no shocks. In the numerical solution of hyperbolic systems of partial differential equations, an important aspect is the computational challenges associated with handling shocks. It is not unreasonable to assume that the presence of shocks may have an effect on the convergence rate of the Parareal method. Furthermore the complexity of the solution is limited. Using the simple test-case of Thacker may thus lead to a false positive in the sense that observing a fair convergence rate on this particular problem may not say much about the case for hyperbolic problems in general. Due to the limitations of the classical test-case we instead propose a new shock-containing test-case which is more suitable to investigate the extend to which Parareal is applicable for such a problem. We maintain the usage of a radially-symmetric elliptic paraboloid to describe the bottom bathymetry of the basin, given by

$$z(x, y) = h_0 \frac{r^2}{a^2} \tag{15}$$

with $r = \sqrt{(x - L/2)^2 + (y - L/2)^2}$ for $(x, y) \in [0, L] \times [0, L]$. Here a is the radius of the basin and h_0 the basin depth at the center of the paraboloid. We define the perturbation h_p to the water surface at rest as

$$h_p(x, y) = A \cos^2(\omega\theta) \exp\left(-\frac{(r - R)^2}{2\sigma^2}\right) \tag{16}$$

with $\theta = \arctan\left(\frac{y-.5L}{x-.5L}\right)$. The initial water height may then be written as

$$h(x, y, 0) = \begin{cases} h_0 + h_p(x, y) - z(x, y) & \text{if } r(x, y) \le a \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

with the discharges $hu(x, y, 0) = 0$ and $hv(x, y, 0) = 0$ for all $(x, y) \in [0, L] \times [0, L]$. For the parallel integration tests in Section 5, we let $L = 1000km$ with a basin radius of $a = 400km$ and a depth $h_0 = 1km$. The waves peak H_p at radius $R = 300km$ from the center of the map, with an amplitude $A = 500m$, frequency $\omega = 4$ and width $\sigma = 10km$.

In Fig. 1 the initial condition is depicted on a 1600x1600 cells map. In Fig. 2 the solution, as computed by the scheme introduced in Section 2.2.2, is presented in 10 minute intervals. Clearly the solution contains rich interaction between smooth regions and shocks as well as wetting and drying. The complexity of the solution may be increased by increasing ω.

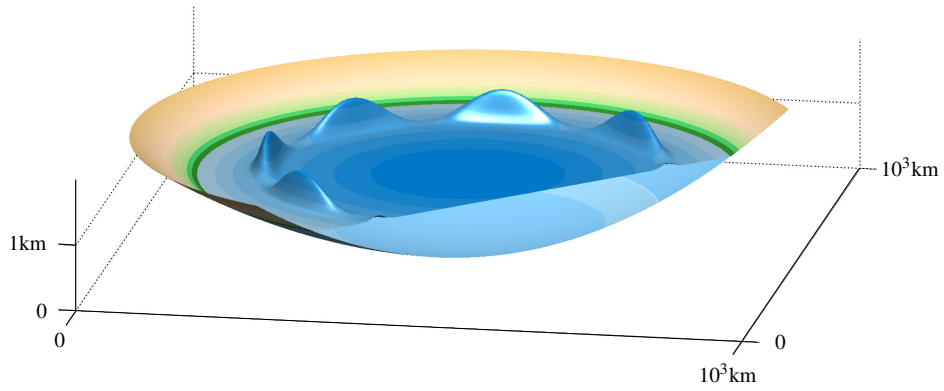


Figure 1: The initial condition, 17 for a wave amplitude of $A = 500m$, a frequency $\omega = 4$ and width a $\sigma = 10km$ with the bathymetry 15 using $L = 1000km$, a basin radius of $a = 400km$, and a depth $h_0 = 1km$. The solution of 11 with the initial condition depicted is given for 10 minute intervals in Fig. 2.

2.2. High order WENO for the shallow water wave equation

To solve the shallow water wave equation, a highly accurate and efficient WENO method to descritize derivatives in space along with a Strong-Stability-Preserving Explicit Runge-Kutta Scheme for integration in time was used. This scheme is 3rd order accurate in both space and time. To use Parareal, a "coarse" operator $\mathcal{G}_{\delta T}$ is required that acts as a preconditioner in the Parareal iteration matrix. To construct such a preconditioner, we use a simple finite volume scheme with an approximate Riemann solver for a first order in time and space method. This coarse operator is introduced in Section 2.2.1, followed by the introduction of the 3rd order WENO-SSPRK scheme in Section 2.2.2. In both cases, the methods are presented in their complete form, but without derivation and analysis.

2.2.1. Roe's Method

For the coarse operator $\mathcal{G}_{\Delta T}$ to solve Eq. (11), we use a standard finite volume method method with an approximate Riemann solver. The method is explicit, first order in time and space. The complete scheme is outlined in equations (18)-(28). Here we omit any details on derivation and analysis and instead refer to LeVeque (2002) for a comprehensive introduction to finite volume methods. Henceforth we will refer to the scheme as "Roe's Method" following the convention used in LeVeque (2002). To find Q_{ij}^{n+1} from Q_{ij}^{n+1}, one must evaluate

$$Q_{ij}^{n+1} = Q_{ij}^{n} - \frac{\Delta t}{\Delta x}\left(F_{i+\frac{1}{2},j}^{n} - F_{i-\frac{1}{2},j}^{n}\right) - \frac{\Delta t}{\Delta y}\left(G_{i,j+\frac{1}{2}}^{n} - G_{i,j-\frac{1}{2}}^{n}\right) + \frac{1}{\Delta t} s\left(Q_{i,j}^{n+1}\right) \tag{18}$$

where

$$F_{i-\frac{1}{2},j}^{n} = \frac{1}{2}\left(f\left(Q_{i-1,j}^{n}\right) - f\left(Q_{i,j}^{n}\right)\right) - \frac{1}{2}\left|\tilde{\mathbf{J}}_{i-\frac{1}{2},j}^{f}\right|\left(Q_{i,j}^{n} - Q_{i-1,j}^{n}\right) \tag{19}$$

$$G_{i,j-\frac{1}{2}}^{n} = \frac{1}{2}\left(g\left(Q_{i,j-1}^{n}\right) - g\left(Q_{i,j}^{n}\right)\right) - \frac{1}{2}\left|\tilde{\mathbf{J}}_{i,j-\frac{1}{2}}^{g}\right|\left(Q_{i,j}^{n} - Q_{i,j-1}^{n}\right) \tag{20}$$

Here s is the RHS function taking into account bathymetry. Note that due to the special structure of $s(Q)$, the scheme (18) may be evaluated explicitly, see (11). In the above, $\tilde{\mathbf{J}}_{i-\frac{1}{2},j}^{f}$ and $\tilde{\mathbf{J}}_{i,j-\frac{1}{2}}^{g}$ are derived from the Jacobian matrices of $f(q)$ and $g(q)$ respectively.

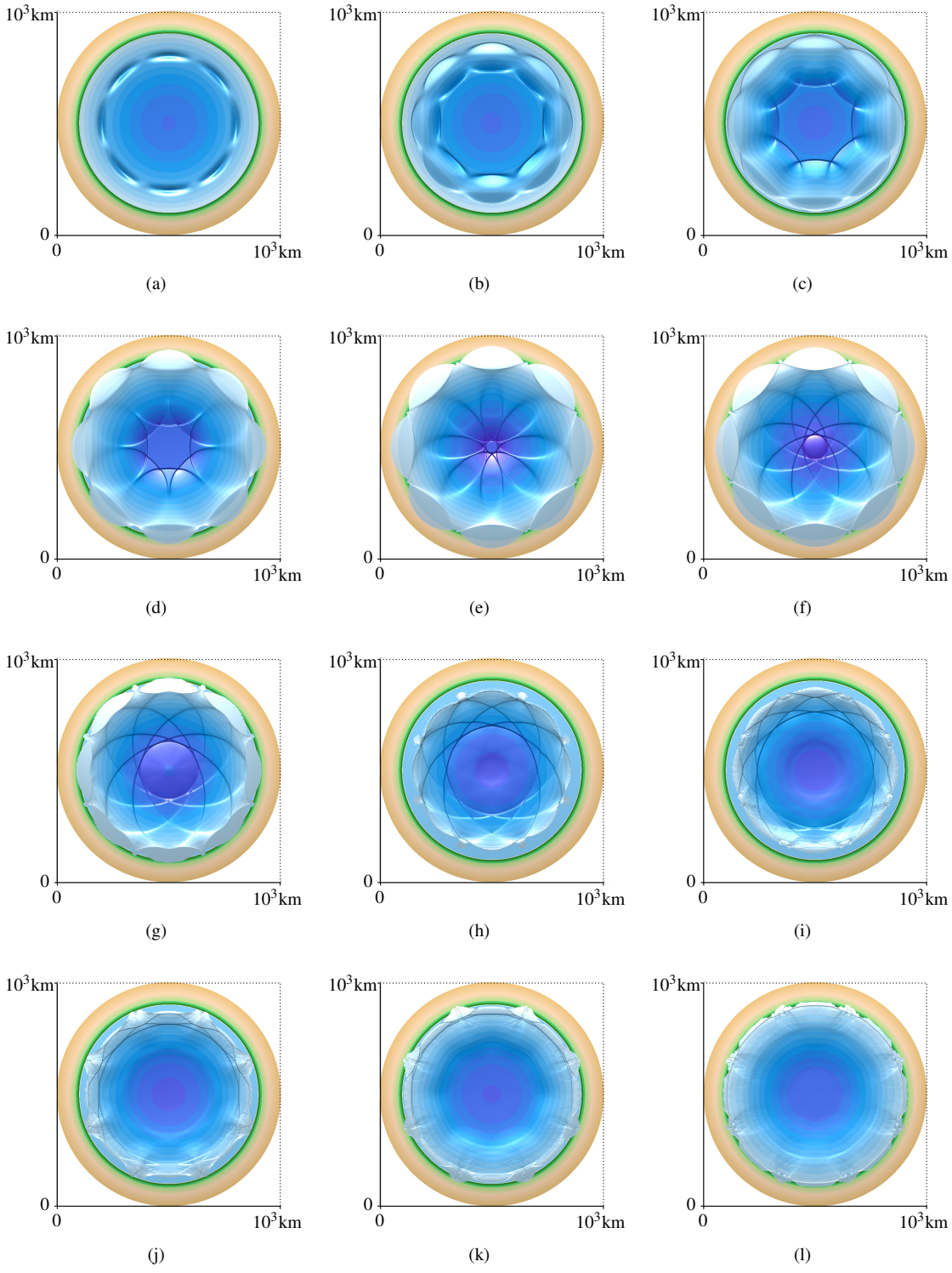


Figure 2: Water height as a function of time for the test case (17) at 10 minute intervals from $T_0 = 0$ to $T = 110$min. Beige and green colors are used to indicate land whilst shades of blue indicate water depth. Light effects are added to highlight location of shocks.

From (13) we see that

$$f'(q) = \begin{bmatrix} 0 & 1 & 0 \\ -u^2 + gh & 2u & 0 \\ -uv & v & u \end{bmatrix}, \quad g'(q) = \begin{bmatrix} 0 & 0 & 1 \\ -uv & v & u \\ -v^2 + gh & 0 & 2v \end{bmatrix} \tag{21}$$

The matrices $\tilde{\mathbf{J}}^f_{i-\frac{1}{2},j}$ and $\tilde{\mathbf{J}}^g_{i,j-\frac{1}{2}}$ are then defined as the Jacobian matrices (21) evaluated at the "Roe Averages" defined by

$$\tilde{u}_{i-\frac{1}{2},j} = \frac{\sqrt{h_{i-1,j}}u_{i-1,j} + \sqrt{h_{i,j}}u_{i,j}}{\sqrt{h_{i-1,j}} + \sqrt{h_{i,j}}}, \quad \tilde{v}_{i-\frac{1}{2},j} = \frac{\sqrt{h_{i-1,j}}v_{i-1,j} + \sqrt{h_{i,j}}v_{i,j}}{\sqrt{h_{i-1,j}} + \sqrt{h_{i,j}}}, \quad \tilde{h}_{i-\frac{1}{2},j} = \frac{1}{2}\left(h_{i-1,j} + h_{i,j}\right) \tag{22}$$

$$\tilde{u}_{i,j-\frac{1}{2}} = \frac{\sqrt{h_{i,j-1}}u_{i,j-1} + \sqrt{h_{i,j}}u_{i,j}}{\sqrt{h_{i,j-1}} + \sqrt{h_{i,j}}}, \quad \tilde{v}_{i,j-\frac{1}{2}} = \frac{\sqrt{h_{i,j-1}}v_{i,j-1} + \sqrt{h_{i,j}}v_{i,j}}{\sqrt{h_{i,j-1}} + \sqrt{h_{i,j}}}, \quad \tilde{h}_{i,j-\frac{1}{2}} = \frac{1}{2}\left(h_{i,j-1} + h_{i,j}\right) \tag{23}$$

where $\tilde{c}_{i-\frac{1}{2},j} = \sqrt{g\tilde{h}_{i-\frac{1}{2},j}}$ and $\tilde{c}_{i,j-\frac{1}{2}} = \sqrt{g\tilde{h}_{i,j-\frac{1}{2}}}$. The Jacobian matrices (21) have the following two eigensystem decompositions

$$\Lambda^f = \begin{bmatrix} u & & \\ & u - \sqrt{gh} & \\ & & u + \sqrt{gh} \end{bmatrix}, \quad R^f = \begin{bmatrix} 0 & 1 & 1 \\ 0 & u - \sqrt{gh} & u + \sqrt{gh} \\ 1 & v & v \end{bmatrix} \tag{24}$$

$$\Lambda^g = \begin{bmatrix} v & & \\ & v - \sqrt{gh} & \\ & & v + \sqrt{gh} \end{bmatrix}, \quad R^g = \begin{bmatrix} 0 & 1 & 1 \\ 1 & u & u \\ 0 & v - \sqrt{gh} & v + \sqrt{gh} \end{bmatrix} \tag{25}$$

from which we define

$$\tilde{\Lambda}^f_{i-\frac{1}{2},j} = \begin{bmatrix} \tilde{u}_{i-\frac{1}{2},j} & & \\ & \tilde{u}_{i-\frac{1}{2},j} - \tilde{c}_{i-\frac{1}{2},j} & \\ & & \tilde{u}_{i-\frac{1}{2},j} + \tilde{c}_{i-\frac{1}{2},j} \end{bmatrix}, \quad \tilde{R}^f_{i-\frac{1}{2},j} = \begin{bmatrix} 0 & 1 & 1 \\ 0 & \tilde{u}_{i-\frac{1}{2},j} - \tilde{c}_{i-\frac{1}{2},j} & \tilde{u}_{i-\frac{1}{2},j} + \tilde{c}_{i-\frac{1}{2},j} \\ 1 & \tilde{v}_{i-\frac{1}{2},j} & \tilde{v}_{i-\frac{1}{2},j} \end{bmatrix} \tag{26}$$

and

$$\tilde{\Lambda}^g_{i,j-\frac{1}{2}} = \begin{bmatrix} v_{i,j-\frac{1}{2}} & & \\ & \tilde{v}_{i,j-\frac{1}{2}} - \tilde{c}_{i,j-\frac{1}{2}} & \\ & & \tilde{v}_{i,j-\frac{1}{2}} + \tilde{c}_{i,j-\frac{1}{2}} \end{bmatrix}, \quad \tilde{R}^g_{i,j-\frac{1}{2}} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & \tilde{u}_{i,j-\frac{1}{2}} & \tilde{u}_{i,j-\frac{1}{2}} \\ 0 & \tilde{v}_{i,j-\frac{1}{2}} - \tilde{c}_{i,j-\frac{1}{2}} & \tilde{v}_{i,j-\frac{1}{2}} + \tilde{c}_{i,j-\frac{1}{2}} \end{bmatrix} \tag{27}$$

so we can express $\left|\tilde{\mathbf{J}}^f_{i-\frac{1}{2},j}\right|$ and $\left|\tilde{\mathbf{J}}^g_{i,j-\frac{1}{2}}\right|$ as

$$\left|\tilde{\mathbf{J}}^f_{i-\frac{1}{2},j}\right| = R^f_{i-\frac{1}{2},j}\left|\Lambda^f_{i-\frac{1}{2},j}\right|\left(R^f_{i-\frac{1}{2},j}\right)^{-1} \qquad \left|\tilde{\mathbf{J}}^g_{i,j-\frac{1}{2}}\right| = R^g_{i,j-\frac{1}{2}}\left|\Lambda^g_{i,j-\frac{1}{2}}\right|\left(R^g_{i,j-\frac{1}{2}}\right)^{-1} \tag{28}$$

which completes the scheme for computing Q^{n+1}_{ij} from Q^n_{ij} using only explicit evaluations.

2.2.2. WENO and Explicit SSP-RK

In simulation Fig. 2, a 3rd order accurate finite volume WENO discretization with a Strong Stability Preserving Runge-Kutta is used to compute the numerical solution. We here present a very short overview of the method. Integrating (12) over a cell I_{ij} one finds that

$$\begin{aligned} \frac{dQ_{ij}(t)}{dt} = -\frac{1}{\Delta x \Delta y}\Bigg(&\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} f\left(q\left(x_{i+\frac{1}{2}}, y, t\right)\right) dy - \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} f\left(q\left(x_{i-\frac{1}{2}}, y, t\right)\right) dy + \\ &\int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} g\left(q\left(x, y_{j+\frac{1}{2}}, t\right)\right) dx - \int_{x_{j-\frac{1}{2}}}^{x_{j+\frac{1}{2}}} g\left(q\left(x, y_{j-\frac{1}{2}}, t\right)\right) dx - \int_{I_{i,j}} s(q,z)\, dxd\Bigg) \end{aligned} \tag{29}$$

where $Q_{ij}(t)$ is the cell average as defined in (14). We introduce the operator L as an approximation to the RHS of (29) by the following conservative scheme

$$L\left(Q_{ij}\right) = -\frac{1}{\Delta x}\left(\hat{f}_{i+\frac{1}{2},j} - \hat{f}_{i-\frac{1}{2},j}\right) - \frac{1}{\Delta y}\left(\hat{g}_{i,j+\frac{1}{2}} - \hat{g}_{i,j-\frac{1}{2}}\right) + \frac{1}{\Delta x \Delta y}\int_{I_{i,j}} s\left(q,z\right)\, dxd \tag{30}$$

where the numerical flux $\hat{f}_{i+\frac{1}{2},j}$ is defined as

$$\hat{f}_{i+\frac{1}{2},j} = \sum_{\alpha} w_\alpha F\left(q^{-}_{i+\frac{1}{2},y_j+\beta_\alpha\Delta y}, q^{+}_{i+\frac{1}{2},y_j+\beta_\alpha\Delta y}\right) \tag{31}$$

Here β_α and w_α are Gaussian quadrature nodes and weights for approximating the integration in y as

$$\hat{f}_{i+\frac{1}{2},j} \approx \frac{1}{\Delta y}\int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} f\left(q\left(x_{i+\frac{1}{2}},y,t\right)\right) dy \tag{32}$$

and $q^{\pm}_{i+\frac{1}{2},y}$ are the WENO reconstructed values, computed as described in Shu (1998). F is the numerical flux as defined in (19). The approximation of $\hat{g}_{i+\frac{1}{2},j}$ is defined in the same way, but using (20). With $L\left(Q_{ij}\right)$ as defined in (30), we perform the integration using the optimal third order SSP Runge-Kutta scheme that reads

$$\begin{aligned} Q^{(1)} &= Q^n + \Delta t L\left(Q^n\right) \\ Q^{(2)} &= \frac{3}{4}Q^n + \frac{1}{4}Q^{(1)} + \frac{1}{4}\Delta t L\left(Q^{(1)}\right) \\ Q^{n+1} &= \frac{1}{3}Q^n + \frac{2}{3}Q^{(1)} + \frac{2}{3}\Delta t L\left(Q^{(2)}\right) \end{aligned} \tag{33}$$

A complete introduction to the numerical method is outside the scope of this paper, we refer to Shu (1998) for an introduction to ENO and WENO based methods, and Xing and Shu (2011) for higher order WENO applied to the shallow water wave equation with wetting and drying of cells.

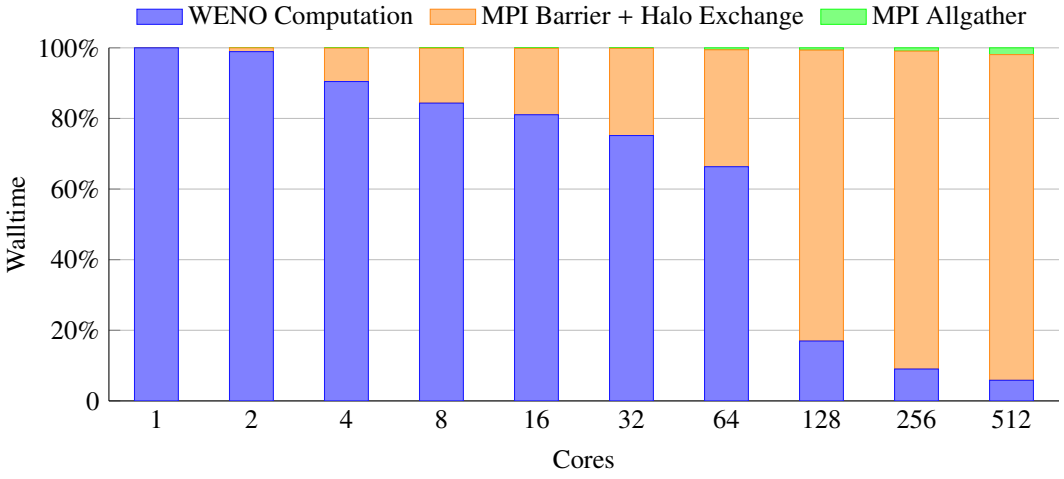


Figure 3: Profiling the parallel-in-space WENO based Tsunami simulation when solving the 1600x1600 cells test-case presented in Section 2.1. As the number of cores increase, the halo-exchange between subdomains comes to dominate. Due to the small size of the test-case, the code already effectively stops scaling with 128 cores. The code was profiled on the EPFL Bellatrix cluster. Each node in the cluster contains two 8-core Intel Xeon E5-2660 CPUs and Inifiniband QDR 2:1 connectivity between nodes. Using a single core on a single node, the computation takes 11817 seconds to complete. Using 4 nodes for a total of 64 cores, the computation needs 278 seconds to complete.

3. Space-Time Domain Decomposition

The simulation that we wish to accelerate uses an explicit numerical scheme as described in Section 2.2.2, no linear systems needs to be solved so its parallel-in-space implementation is straightforward. Following each time-step of the explicit Runge-Kutta scheme (33), a two-cell wide halo is exchanged between all adjacent spacial subdomains using MPI with one rank per subdomain. The parallel-in-space implementation scales well up to 5 nodes (80 cores) on the small 1600x1600 cell test-case using the EPFL Bellatrix cluser. With 6 nodes and above, the cost of communication between subdomains becomes too large for further parallel acceleration as illustrated in the profiling measurements in Fig. 3. For further parallel acceleration, we turn to time-parallel techniques. The efficient implementation of the standard Parareal algorithm for combined space and time parallelism is more involved. In Fig. 4, the space-time parallel implementation is conceptually depicted. White lines indicate division of spatial subdomains, and each image indicate a time-subdomain. For Parareal, a coarse operator acting as a preconditioner in the solution of the system (9) is needed, and this preconditioner must be parallel in space as well. Unlike the 3rd order WENO+SPP RK used to solve equation (11) for the simulation, the preconditioner $\mathcal{G}_{\delta T}$, based on Roe's method, only needs a single cell halo to be exchanged between the subdomains in space. Whilst the coarse operator $\mathcal{G}_{\delta T}$ is trivial to make parallel, efficiently solving the recursive Parareal formulation (10) is not so. A number of schedulers for dividing the computational work on clusters have been proposed in the literature. A direct approach for distributing the work is to apply the coarse and fine operators in strictly separate phases, i.e., in each iteration, a number of worker node-groups each compute the application of $\mathcal{F}_{\delta T}$ in parallel. When done, data is collected on a manager node that performs the sequential application of $\mathcal{G}_{\delta T}$, followed by the Parareal corrections, before distributing the data to the worker node-groups for a new iteration. If $\mathcal{G}_{\delta T}$ is computationally very cheap, the manager-worker approach may work sufficiently well. In practice however, it has been observed repeatedly that this approach is too restrictive for Parareal to achieve speed-up for anything but simple problems. In Aubanel (2011), a better scheduler was introduced, simple in design, yet near optimal in terms of exploiting the dependencies that exists in the recursive tree that defines Parareal. The algorithm introduced is equvilant to first executing Alg. 1 followed by a single execution of Alg. 2.

Parallel-in-time integration with long time-subdomains has certain advantages when using an appropriate scheduler. In that case the communication pattern becomes dominated by a few large time-subdomain interfaces that must be communicated between node-groups. The potential of this latency tolerant communication pattern was investigated in early papers by Srinivasan and Chandra (2005); Xiao and Aubanel (2012). Unfortunately convergence is slow for most problems when using parallel-in-time integration of long intervals with the standard Parareal algorithm as has been demonstrated in many early papers, see Nielsen (2012) for an overview, and later established rigorously in Gander and Hairer (2014). To achieve faster convergence one must therefore yet again divide the interval to be integrated into smaller intervals onto where we can apply multiple cycles of Parareal. This could be done as outlined in Fig. 5 with Alg. 3, using multiple consecutive cycles of Parareal implemented with the scheduler proposed by Aubanel (2011). In Berry et al. (2012), an "event-based" Parareal scheduler was proposed. Here a data-dependency driven approach is taken to scheduling such that when all dependencies are satisfied for a time-subdomain, the work is scheduled to an available node.

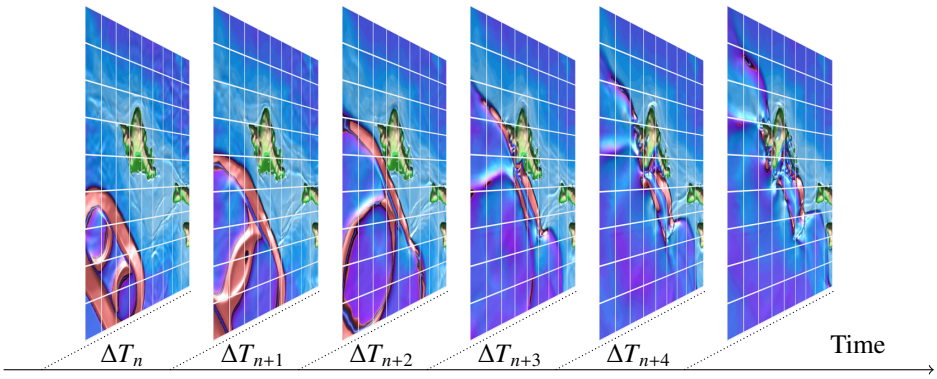


Figure 4: Domain decomposition in space and time of the WENO based solver introduced in Section 2. White lines indicate division of spatial subdomains, and each image indicate a subdomain spanning ΔT in time.

In the scheduler we present here, we combine the simplicity of the scheduler introduced in Aubanel (2011), with the scalability of Berry et al. (2012), whilst introducing a parameter to tune the trade-off between node-group occupancy and speed of convergence.

We denote this proposed work scheduler as "adaptive Parareal". The fundamental approach is to let adjacent cycles of parareal overlap in execution time. When a node-group completes its work on a time-subdomain in a given cycle, it will immediately begin working on the closest inactive time-subdomain in the next cycle. This may happen in one of two ways. The node-group may commence working on the time-subdomain on the next cycle using the most recent iteration available on the preceding time-subdomain, or it may wait for the next iteration on the preceding time-subdomain to become available. Which choice is better is not obvious and will be situation and problem dependent. If a recent iterate is soon to be available, it may be better waiting. If on the other hand the preceding time-subdomain has just commenced work on a new iteration, it may be better initiating work on the most recent iteration available rather than waiting for the preceding time-subdomain to complete it's current work. We introduce a parameter $\beta \in [0, 1]$ that controls how patient the node-groups shall be. When a node-group receives a signal that the next time-subdomain has become active, it will immediately return a solution state if the progress on the application of $\mathcal{F}_{\delta T}$ is less than β. The progress indicator on $\mathcal{F}_{\delta T}$ could for example be on the time that has been integrated relative to the total time-subdomain length. Thus, if β is small, the scheduler is patient and if β is large the scheduler is impatient, and will value minimizing idle nodes over convergence in a small number of iterations. The communication pattern in this model is asynchronous. The iteration and time at which the convergence criteria is satisfied is unknown for all time-subdomains, it is therefore not possible a priori to predict the communication pattern. To enable node-groups to signal their status, and potentially send a time-subdomain interface, whilst they are in the process of computing $\mathcal{F}_{\delta T}$, a separate signal thread is needed. Each signal thread will receive $n_c - 1$ signals, each time a signal is received it indicates that next time-subdomain has become active. When receiving a signal, it will set the status of the flag that indicates if the next time-subdomain is active to true, After doing so, the thread checks if the progress indicator on $\mathcal{F}_{\delta T}$ is smaller than β. If it is, it will send the most recent iterate of the time-subdomain interface that it is computing. In our implementation, Posix Threads was used to create the signal threads needed in the adaptive scheduler.

Schematic examples for $\beta = 0$ and $\beta = 1$ are presented in Fig. 6a and in 6b, respectively. In the figures, black dots and arrows indicate the sending and receiving of time-subdomain interfaces. The blue arrows indicate the message to signal that a time-subdomain has become active. Pseudo code for the proposed adaptive scheduler is given in Alg. 5 along with Alg. 4 for the corresponding signal thread. Separate communicators are created for each spatial-subdomain, and all communication of time-subdomain interfaces happens trough dedicated inter-communicators. Doing so provides encapsulation of application code already written, whilst providing a natural distinction between the parallelism in computing derivatives in space and parallelism in the time integration procedure.

Algorithm 1 Initialization procedure for a parareal cycle

1: $k \leftarrow 0$
2: **if** $FirstNodeGroup$ **then**
3: $\quad \tilde{\mathbf{U}}^0_{id_{\Delta T}} \leftarrow \mathcal{G}_{\Delta T}\mathbf{U}^k_{id_{\Delta T}-1}$
4: $\quad \mathbf{U}^0_{id_{\Delta T}} \leftarrow \tilde{\mathbf{U}}^0_{id_{\Delta T}}$
5: $\quad$ Send $Converge$ and $\mathbf{U}^0_{id_{\Delta T}}$ on forw_intercomm
6: $\quad tag\_send = tag\_send + 1$;
7: $\quad ConvergeNext \leftarrow$ TRUE
8: **else**
9: $\quad$ Recv $Converge$ and $\mathbf{U}^0_{id_{\Delta T}-1}$ on back_intercomm
10: $\quad \tilde{\mathbf{U}}^0_{id_{\Delta T}} \leftarrow \mathcal{G}_{\Delta T}\mathbf{U}^0_{id_{\Delta T}-1}$
11: $\quad \mathbf{U}^0_{id_{\Delta T}} \leftarrow \tilde{\mathbf{U}}^0_{id_{\Delta T}}$
12: $\quad$ **if** $!LastNodeGroup$ **then**
13: $\quad\quad$ Send $Converge$ and $\mathbf{U}^0_{id_{\Delta T}}$ on forw_intercomm
14: $\quad\quad tag\_send = tag\_send + 1$;
15: $\quad$ **end if**
16: **end if**

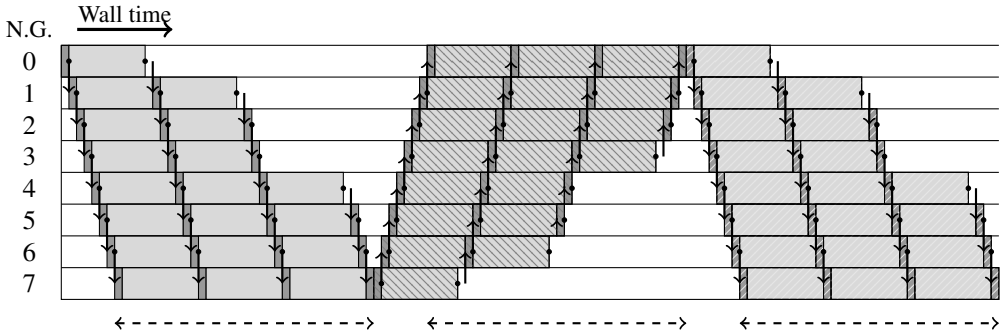
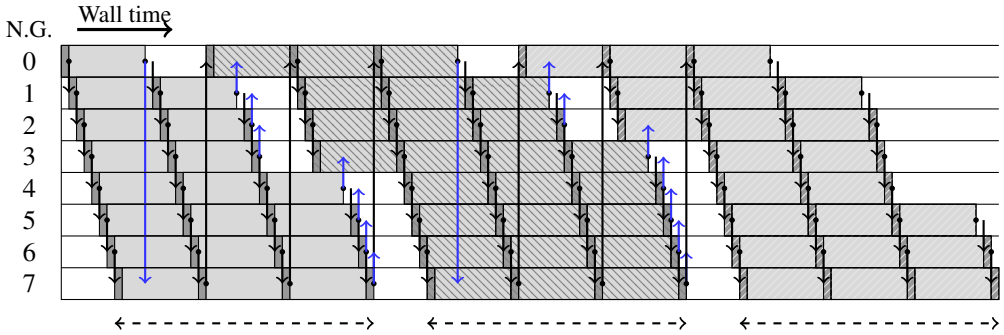
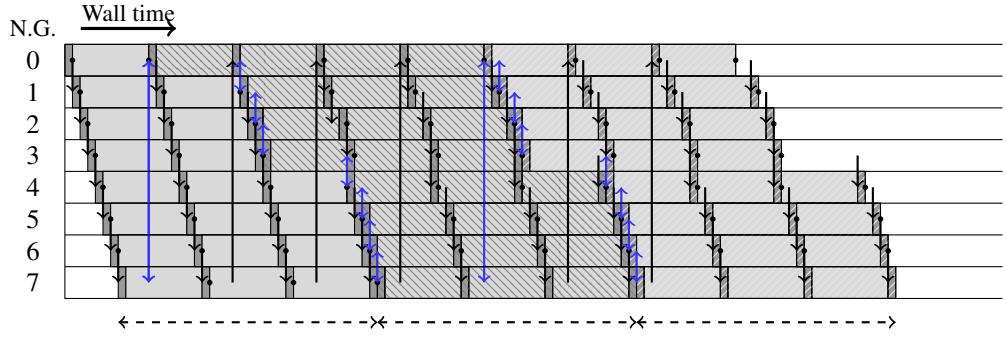


Figure 5: Multiple consecutive executions of the standard fully distributed Parareal by Aubanel (2011). Each time-subdomain is handled by a unique node-group, possibly parallel-in-space. Dark gray indicate that a node-group is computing the preconditioner $\mathcal{G}_{\delta T}$, light gray indicate that a node-group is computing $\mathcal{F}_{\delta T}$. Drawn for $n_t = 8$, $n_c = 3$. Shorter time-subdomains may lead to faster convergence, but this comes at the cost of more frequent communication of the solution-states at time-subdomain interfaces. In each cycle, the tolerance is satisfied by the 3rd correction. The direction of information flow shifts with each cycle to reduce the number of time-subdomain interface states sent.



(a) Adaptive scheduler with $\beta = 0$



(b) Adaptive scheduler with $\beta = 1$

Figure 6: Schematic representation of a proposed "Adaptive" Parareal scheduler. The scheduler lets multiple cycles of Parareal overlap in execution time. Drawn here for $n_t = 8$, $n_c = 3$. Dark gray indicate that a node-group is computing the preconditioner $\mathcal{G}_{\delta T}$, light gray indicate that a node-group is computing $\mathcal{F}_{\delta T}$. Black dots and arrows indicate the sending and receiving of time-subdomain interface solution state. Blue arrows indicate a signal being sent to inform a the node-group working on a time-subdomain that the next time-subdomain has become active. Each node-group has such a boolean flag that indicates if the next time-subdomain is active or not. (a) $\beta = 0$ for a fully patient model where node-groups that finished a time-subdomain in a cycle will wait for a correction to be made before receiving a new state to commence their work. (b) $\beta = 1$ for a fully impatient model where node-groups that finished a time-subdomain in a cycle will receive a new state to commence their work immediately.

Algorithm 2 A single parareal cycle.

while !$Converge$ **do**
  $k \leftarrow k + 1$
  $\hat{\mathbf{U}}^{k-1}_{id_{\Delta T}} \leftarrow \mathcal{F}_{\Delta T}\mathbf{U}^{k-1}_{id_{\Delta T}-1}$
  if $ConvergeNext$ **then**
    $Converge \leftarrow$ TRUE
    $\mathbf{U}^{k}_{id_{\Delta T}} \leftarrow \hat{\mathbf{U}}^{k-1}_{id_{\Delta T}}$
    if !$LastNodeGroup$ **then**
      Send $Converge$ and $\mathbf{U}^{k}_{id_{\Delta T}}$ on forw_intercomm
      $tag\_send = tag\_send + 1$;
    end if
    break
  end if
  Recv $Converge$ and $\mathbf{U}^{k}_{id_{\Delta T}-1}$ on back_intercomm
  $\tilde{\mathbf{U}}^{k}_{id_{\Delta T}} \leftarrow \mathcal{G}_{\Delta T}\mathbf{U}^{k}_{id_{\Delta T}-1}$
  $\mathbf{U}^{k}_{id_{\Delta T}} \leftarrow \tilde{\mathbf{U}}^{k}_{id_{\Delta T}} + \hat{\mathbf{U}}^{k-1}_{id_{\Delta T}} + \tilde{\mathbf{U}}^{k-1}_{id_{\Delta T}}$
  if $Converge$ & $|\mathbf{U}^{k}_{id_{\Delta T}} - \mathbf{U}^{k-1}_{id_{\Delta T}}| > \epsilon$ **then**
    $Converge \leftarrow$ FALSE
    $ConvergeNext \leftarrow$ TRUE
  end if
  if !$LastNodeGroup$ **then**
    Send $Converge$ and $\mathbf{U}^{k}_{id_{\Delta T}}$ on forw_intercomm
    $tag\_send = tag\_send + 1$;
  end if
  if $Converge$ **then**
    break
  end if
end while

Algorithm 3 Pseudocode for a parareal implementation running n_c consecutive cycles of parareal, each with n_t time-subdomains. The direction of information flow shifts with each cycle to reduce the number of time-subdomain interfaces solution states to be sent. Schematical example in Fig. 5.

$id_{\Delta T} \leftarrow id_{ng}$, $\mathbf{U}^0_0 \leftarrow u_0$
back_intercomm $\leftarrow$ intercomm between node-groups id_{ng} and $id_{ng} - 1$
forw_intercomm $\leftarrow$ intercomm between node-groups id_{ng} and $id_{ng} + 1$
for $i = 1$ to n_c **do**
  $tag\_send = tag\_send + 1$;
  $id_{\Delta T} \leftarrow (i-1)n_t + id_{ng}$
  $converge, convergeNext, FirstNodeGroup, LastNodeGroup \leftarrow$ FALSE
  if ($id_{ng} = 1$ **and** $\bmod i = 1$) **or** ($id_{ng} = n_t$ **and** $\bmod i = 0$) **then**
    $FirstNodeGroup \leftarrow$ TRUE
  end if
  if ($id_{ng} = 1$ **and** $\bmod i = 0$) **or** ($id_{ng} = n_t$ **and** $\bmod i = 1$) **then**
    $LastNodeGroup \leftarrow$ TRUE
  end if
  procedure Algorithm 1
  procedure Algorithm 2
  procedure Swap back_intercomm and forw_intercomm
end for

Algorithm 4 Signal thread for asynchronous communication in the adaptive parareal Alg. 5. The values k, *Converge*, $\mathbf{U}$, *LastNodeGroup* and *tag_send* are shared with the main work thread.

for $i = 1$ to $n_c - 1$ **do**
  Recv *LastNodeGroup* on forw_intercomm
  if $k > 0$ **and** $status\,(\mathcal{F}_{\Delta T}) < \beta$ **and** $tag\_send = 0$ **then**
    $Converge \leftarrow$ FALSE
    Send *Converge* and $\mathbf{U}^{k-1}_{id_{\Delta T}}$ on forw_intercomm.
    $tag\_send = tag\_send + 1$;
  end if
end for

Algorithm 5 Pseudocode for an adaptive parareal implementation with n_c cycles each with n_t simultaneously active time-subdomains. Schematical examples of the scheduler is given in figure 6a for $\beta = 0$ and 6b for $\beta = 1$.

procedure: Initiate algorithm 4 on separate thread
back_intercomm $\leftarrow$ intercomm between node-groups id_{ng} and $id_{ng} - 1$
forw_intercomm $\leftarrow$ intercomm between node-groups id_{ng} and $id_{ng} + 1$
$Converge, ConvergeNext, FirstNodeGroup, LastNodeGroup \leftarrow$ FALSE
$id_{\Delta T} \leftarrow id_{ng}$, $\mathbf{U}^0_0 \leftarrow u_0$, $i \leftarrow 0$
if $id_{ng} = 1$ **then**
  $FirstNodeGroup \leftarrow$ TRUE
end if
if $id_{ng} = n_t$ **then**
  $LastNodeGroup \leftarrow$ TRUE
end if
procedure Algorithm 1
while $i < n_c$ **do**
  $id_{\Delta T} \leftarrow i \cdot n_t + id_{ng}$
  procedure Algorithm 2
  $i \leftarrow i + 1$, $k \leftarrow 0$, $tag\_send \leftarrow 0$
  if $i < n_c$ **then**
    Send *LastNodeGroup* on back_intercomm
    $LastNodeGroup \leftarrow$ TRUE
    $ConvergeNext \leftarrow$ FALSE
    Recv *Converge* and $\mathbf{U}^k_{id_{\Delta T}-1}$ on back_intercomm
    if *Converge* **then**
      $Converge \leftarrow$ FALSE
      $ConvergeNext \leftarrow$ TRUE
    end if
    $\tilde{\mathbf{U}}^k_{id_{\Delta T}} \leftarrow \mathcal{G}_{\Delta T}\mathbf{U}^k_{id_{\Delta T}-1}$
    $\mathbf{U}^k_{id_{\Delta T}} \leftarrow \tilde{\mathbf{U}}^k_{id_{\Delta T}}$
    $k \leftarrow 1$
    if !*LastNodeGroup* and $tag\_send = 0$ **then**
      Send *Converge* and $\mathbf{U}^{k-1}_{id_{\Delta T}}$ on forw_intercomm
      $tag\_send = tag\_send + 1$;
    end if
  end if
end while

4. Choosing the time-subdomain length

In the previous section we introduced a scheduler for efficiently executing multiple consecutive Parareal cycles. It does so by letting adjacent cycles overlap in execution time, and by adaptively choosing what previous iteration to initiate a new cycle from. This means that that we are effectively free to choose whatever time-subdomain length we wish, regardless of the number of active subdomains in time to use. Given some IBVP problem, and a fixed number of nodes n_t at our disposal to do parallel-in-time integration of some fixed (long) time-interval $[0, T_{end}]$. The question arises of what time-subdomain length δT should one choose?

Since the question is posed for some fixed n_t and T_{end}, it is equivalent to asking: How many cycles of Parareal should we split our time domain of length T_{end} into? The purpose of this section is to develop an approach that allows for an informed choice on δT before running the code. The original Parareal algorithm was presented as parallel-in-time integration of a fixed time interval with the time-subdomain $\delta T = \frac{T_{end}}{n_t}$. In practice, with this approach, there is a limit to how long T_{end} may be. Therefore, for integration over a long time interval, one must decouple the number of independent time-subdomains from the total time-domain to be integrated. This can be done either trough a simple stop-start strategy of multiple consecutive "cycles" of plain Parareal, as depicted in Fig. 5 for $n_c = 3$ cycles, or with a more advanced approach were consecutive cycles are allowed to overlap in execution time across nodes, as introduced with the adaptive scheduler Alg. 5 and visualized schematically in Fig. 6. Decoupling the time-subdomain length δT from n_t and T_{end}, in addition to allowing for integration of long time-domains, also induces the freedom to choose the time-subdomain length as one deem appropriate for achieving high parallel efficiency. When choosing a time-subdomain length, one makes a fundamental trade-off between how often to run the coarse operator and communicate the full solution-state sequentially across all active time subdomains, and the speed at which the algorithm will converge. If the time-subdomain δT is chosen to be very short, one can expect fast convergence in few iterations k. But, the algorithm may fail to provide any parallel acceleration because the preconditioner $\mathcal{G}_{\delta T}$ will be applied more frequently, and because all the intermediate solution-states $\mathbf{U}_k^n$ also has to be communicated across nodes more often. Conversely, if one choose a "very long" time-subdomain δT, up to $\delta T \leq \frac{T_{end}}{n_t}$, although we may not be limited by the sequential execution of $\mathcal{G}_{\delta T}$ and communication of solution states $\mathbf{U}_k^n$, but the algorithm may instead need many iterations to convergence, limiting the extent to which parallel acceleration is possible. Ideally, we want to choose δT so that these effects are balanced in such a way that we achieve the highest possible speed-up. Finding such a δT is made complicated by the fact that we do not know in general how k depends on n_t and δT. As we shall see, this does however not mean that we must make an uninformed random choice on δT.

4.1. Single Cycle Analysis

Before embarking on an analysis of the multi-cycle Parareal algorithm presented in Section 3, we consider the standard "single cycle" Parareal algorithm. Henceforth we will refer to the collection of CPU's and possible co-processors that compute the application of $\mathcal{F}_{\delta T}$ and $\mathcal{G}_{\delta T}$ to $\mathbf{U}_k^n$ as a node-group. With this notation we abstract away whatever (spacial) domain-decomposition that may have been applied in constructing $\mathcal{F}_{\delta T}$ and $\mathcal{G}_{\delta T}$. The parallel speed-up of a single cycle of length $\Delta T = n_t \delta T = T_{end}$ as presented in Fig. 5 can be written as

$$\psi = \frac{n_t C_{\mathcal{F}} \delta T}{n_t \left(C_{\mathcal{G}} \delta T + T_C^w \right) + \kappa(n_t, \delta T) \left(C_{\mathcal{F}} \delta T + C_{\mathcal{G}} \delta T + T_C^w \right)} \tag{34}$$

Here n_t denotes the number of time-subdomains, i.e. the number of node groups that may be used. $T_{\mathcal{F}}^w$ and $T_{\mathcal{G}}^w$ denotes the wall-time it takes for the two operators $\mathcal{F}_{\delta T}$ and $\mathcal{G}_{\delta T}$ to be applied to a state $\mathbf{U}_k^n$ when computed on a node-group. Both $T_{\mathcal{F}}^w$ and $T_{\mathcal{G}}^w$ are assumed proportional to δT with some constants $C_{\mathcal{F}}, C_{\mathcal{G}}$ so that

$$T_{\mathcal{F}}^w = C_{\mathcal{F}} \delta T, \quad T_{\mathcal{G}}^w = C_{\mathcal{G}} \delta T + T_C^w \tag{35}$$

Here T_C^w denotes the wall-time it takes to communicate a solution state from one node-group to another. T_C^w is included in the complete execution time of the preconditioner $\mathcal{G}_{\delta T}$ since for every application of $\mathcal{G}_{\delta T}$ there will be one state $\mathbf{U}_k^n$ that must be communicated from one node-group to another. The function $\kappa : \mathbb{N}^+ \times \mathbb{R}^+ \to \mathbb{N}^+$ is the number of iterations needed before the convergence criteria is satisfied. It is important to note here that the only unknown is $\kappa(n_t, \delta T)$. δT and n_t are known, and the rest are constants that can be measured for a specific cluster

on a given problem. Let us first explore the limit as the time-subdomain length δT becomes small. Assuming that $\lim_{\delta T \to 0} \kappa(n_t, \delta T) = 1$, from (34) one finds that

$$\lim_{\delta T \to 0} \psi = \frac{n_t}{(n_t + 1)} \frac{T^w_{\mathcal{F}}}{T^w_C} \delta T \tag{36}$$

becomes an upper bound for achievable speed-up. This is not surprising, as by Amdahls law, $T^w_{\mathcal{F}}/T^w_{\mathcal{G}}$ must be an upper limit to speed-up for the Parareal algorithm. The above simply states that for small δT, the cost T^w_C of communicating the solution states $\mathbf{U}^n_k$ sequentially across nodes becomes the dominating term that limits parallel speedup. Whilst (36) may be descriptive, it does not let us choose δT in any meaningful way. Our goal is to find the δT that maximize equation (34), so let us frame the problem in the form of an optimization problem

$$\delta T_{opt} = \arg\max_{\delta T \in \mathbb{R}^+} \psi(\delta T) \tag{37}$$

For the optimization, the derivative of $\psi(\delta T)$ w.r.t. δT is given as

$$\begin{aligned} \frac{\partial}{\partial(\delta T)} \psi(\delta T) &= \frac{\partial}{\partial(\delta T)} n_t C_{\mathcal{F}} \delta T \left(n_t \left(C_{\mathcal{G}} \delta T + T^w_C \right) + \kappa(n_t, \delta T) \left(C_{\mathcal{F}} \delta T + C_{\mathcal{G}} \delta T + T^w_C \right) \right)^{-1} \\ &= n_t C_{\mathcal{F}} \frac{(\kappa(n_t, \delta T) + n_t) T^w_C - (C_{\mathcal{F}} + C_{\mathcal{G}}) \kappa'(n_t, \delta T) \delta T^2}{\left(n_t C_{\mathcal{G}} \delta T + (\kappa(n_t, \delta T) + n_t) T^w_C + (C_{\mathcal{F}} + C_{\mathcal{G}}) \kappa(n_t, \delta T) \delta T \right)^2} \end{aligned} \tag{38}$$

The optimization problem is not solvable as we do not know $\kappa(n_t, \delta T)$, nor do we know it's derivative $\kappa'(n, \delta T)$. We can, however, still gain insight from the above. We assume that $\lim_{\delta T \to 0} \kappa(n_t, \delta T) = 1$ and that $\lim_{\delta T \to \infty} \kappa(n_t, \delta T) = n_t$. In addition we know that $\kappa'(n_t, \delta T) = 0$ for all but a select few δT when $\kappa(n_t, \delta T)$ changes abruptly as the convergence criteria is accepted. For some fixed $\kappa^* \in [1, n_t]$ with $\kappa'(n_t, \delta T) = 0$, as $\delta T \to 0$, $\frac{\partial}{\partial(\delta T)} \psi$ increases and the speed-up $\psi \to 0$. Conversely, as $\delta T \to \infty$ then $\frac{\partial}{\partial(\delta T)} \psi \to 0$ asymptotically. So the maximum exists in the limit $\delta T \to \infty$. With $\kappa'(n_t, \delta T) = 0$ it follows that $(C_{\mathcal{F}} + C_{\mathcal{G}}) \kappa'(n_t, \delta T) \delta T^2 = 0$. The denominator does not change sign, so for any fixed κ^*, there are no inflection points $\delta T \in \mathbb{R}^+$, i.e., we can not hope to find any point where the speedup decreases particularly fast as we decrease δT, even for some fixed κ^*. Because of these limitations, we take another approach at finding some approximate solution $\delta \hat{T}_{opt} \approx \delta T_{opt}$ to (37). While we might not be able to quantify the gain in speed-up from $\kappa(n_t, \delta T)$ becoming smaller as we let $\delta T \to 0$, we can quantify the associated cost of doing so in terms of added wall-time spent communicating $\mathbf{U}^n_k$ across node-groups. Let $\epsilon_c \in (0., 1)$ be a parameter that denotes the fraction of the decrease, due to communication, in speedup that we are willing to accept. The optimization problem is then recast in the following form

$$\delta \hat{T}_{opt} = \min_{\delta T \in \mathbb{R}^+} \delta T \text{ s.t. } \psi(\delta T) \geq \epsilon_c \psi_{max} \tag{39}$$

with some prior guess $\kappa^* \approx \kappa(n_t, \delta T)$. The above stated problem is now to find the smallest δT for which the reduced speed-up due to communication is less than ϵ_c fraction of the maximal speedup ψ_{max}. The above problem is much simpler to solve. Writing out the inequality (39) using (34) one arrives at the following criteria

$$\frac{n_t C_{\mathcal{F}} \delta T}{n_t \left(C_{\mathcal{G}} \delta T + T^w_C \right) + \kappa^* \left(C_{\mathcal{F}} \delta T + C_{\mathcal{G}} \delta T + T^w_C \right)} \geq \epsilon_c \lim_{\delta T \to \infty} \frac{n_t C_{\mathcal{F}} \delta T}{n_t \left(C_{\mathcal{G}} \delta T + T^w_C \right) + \kappa^* \left(C_{\mathcal{F}} \delta T + C_{\mathcal{G}} \delta T + T^w_C \right)} \tag{40}$$

to be satisfied. We may derive an explicit expression for $\delta \hat{T}_{opt}$ by evaluating the r.h.s limit and manipulating the inequality to find

$$\delta \hat{T}_{opt} = \frac{\epsilon_c}{1 - \epsilon_c} \frac{(\kappa^* + n_t) T^w_C}{\kappa^* (C_{\mathcal{G}} + C_{\mathcal{F}}) + n_t C_{\mathcal{G}}} \tag{41}$$

as an approximate solution to (39). In what sense does the problem (39) with solution (41) relate to the original optimization problem (37)? Equation (41) is not a solution to the optimization problem (37), rather it is a intended as a rough estimate of $\delta \hat{T}_{opt}$, that could possibly also be used over several iterations of running the algorithm to improve the estimate. The above analysis, however interesting, is somewhat artificial. Solving (37), we seek to optimize

parallel speed-up over a domain $T_{end} = n_t \delta T$ that varies in size with δT. For parallel-in-time speedup when solving (11) on some (long) fixed time-domains T_{end}, we are interested in the multi-cycle Parareal introduced in the previous section. In that case $T_{end} = n_c \Delta T = n_c n_t \delta T$. We consider the number of simultaneously active time-subdomains n_t to be a fixed parameter and wish to find the optimal time-subdomain length δT to use. In what follows we build on the insight gained.

4.2. Multi Cycle Analysis

We seek an approach to estimate δT_{opt} for Parareal with multiple cycles, as introduced in Section 4.1. For the stop-restart Parareal as drawn in Fig. 5, the parallel speed-up may be written as

$$\begin{aligned} \psi(\delta T) &= \frac{n_c n_t T_{\mathcal{F}}}{n_c n_t T_{\mathcal{G}} + \sum_{i=1}^{n_c} \kappa_i(n_t, \delta T)(T_{\mathcal{F}} + T_{\mathcal{G}})} \\ &= \frac{n_c n_t C_{\mathcal{F}} \delta T}{n_c n_t \left(C_{\mathcal{G}} \delta T + T_C^w\right) + \sum_{i=1}^{n_c} \kappa_i(n_t, \delta T)\left(C_{\mathcal{F}} \delta T + C_{\mathcal{G}} \delta T + T_C^w\right)} \end{aligned} \tag{42}$$

Note that the above is technically an upper bound estimate as the Parareal correction and other overhead is ignored. As in the previous section, $\kappa_i(n_t, \delta T)$ is unknown, and it is therefore not possible to solve the optimization problem (37) to find δT_{opt} a priori. As before we take the approach of solving an optimization problem on the form (39) instead. We first let $\langle \kappa^* \rangle = \frac{1}{n_c} \sum_{i=1}^{n_c} \kappa_i$ and assume $\langle \kappa^* \rangle \approx \kappa\left(\frac{T_{end}}{n_t}, n_t\right)$ to recover

$$\begin{aligned} \frac{n_c n_t C_{\mathcal{F}} \delta T}{n_c n_t \left(C_{\mathcal{G}} \delta T + T_C^w\right) + n_c \langle \kappa^* \rangle \left(C_{\mathcal{F}} \delta T + C_{\mathcal{G}} \delta T + T_C^w\right)} &\geq \epsilon_c \lim_{\delta T \to \frac{T_{end}}{n_t}} \psi(\delta T) \\ &= \frac{\epsilon_c n_t C_{\mathcal{F}} T_{end}}{n_t \left(C_{\mathcal{G}} T_{end} + n_t T_C^w\right) + \langle \kappa^* \rangle \left(C_{\mathcal{F}} T_{end} + C_{\mathcal{G}} T + n_t T_C^w\right)} \end{aligned} \tag{43}$$

With $n_c = \frac{T_{end}}{n_t \delta T}$, the solution to (39) becomes

$$\hat{\delta T}_{opt}^{\beta=0} = \frac{\epsilon_c \left(n_t + \langle \kappa^* \rangle\right) T_C^w T_{end}}{\left(n_t + \langle \kappa^* \rangle\right) n_t T_C^w + (1 - \epsilon_c)\left(n_t C_{\mathcal{G}} + \left(C_{\mathcal{F}} + C_{\mathcal{G}}\right) \langle \kappa^* \rangle\right) T_{end}} \tag{44}$$

In the case of the Adaptive Parareal as drawn in Fig. 6, consecutive cycles are allowed to overlap in execution time across nodes. For the case of the "impatient" algorithm, $\beta = 1$, this will ideally hide the initialization cost of the zeroth iteration for all but the first cycle. In this case we may approximate the speed-up as

$$\psi(\delta T) = \frac{n_c n_t C_{\mathcal{F}} \delta T}{n_t \left(C_{\mathcal{G}} \delta T + T_C^w\right) + \sum_{i=1}^{n_c} \kappa_i(n_t, \delta T)\left(C_{\mathcal{F}} \delta T + C_{\mathcal{G}} \delta T + T_C^w\right)} \tag{45}$$

Note that (45) is no longer equivalent to (34) for the single cycle case. We take another look at the derivative for ideas on how to solve (37).

$$\frac{\partial}{\partial(\delta T)} \psi(\delta T) = -T_{end} C_{\mathcal{F}} \frac{n_t C_{\mathcal{G}} + \frac{T}{n_t}\left(\langle \kappa^* \rangle' \left(C_{\mathcal{F}} + C_{\mathcal{G}} + \frac{T_C^w}{\delta T}\right) - \langle \kappa^* \rangle \frac{T_C^w}{\delta T^2}\right)}{\left(n_t \left(C_{\mathcal{G}} \delta T + T_C^w\right) + \frac{T}{n_t} \langle \kappa.^* \rangle \left(C_{\mathcal{F}} + C_{\mathcal{G}} + \frac{T_C^w}{\delta T}\right)\right)^2} \tag{46}$$

As before we assume $\langle \kappa^* \rangle' = 0$, from the above it is clear that if $\sqrt{\langle \kappa^* \rangle T_{end} T_C^w n_t^{-2} C_{\mathcal{G}}^{-1}} = \delta T \leq \frac{T_{end}}{n_t}$ then the derivative $\psi(\delta T)$ changes sign somewhere in $\left(0, \frac{T_{end}}{n_t}\right]$, e.g., the largest value of $\psi(\delta T)$ is not necessarily in the limit $\delta T \to \frac{T_{end}}{n_t}$. With $\frac{\partial}{\partial(\delta T)} \psi(\delta T) = 0$, it is easy to see that only a single maxima exists for δT in $\mathbb{R}^+$

$$\hat{\delta T}_{max} = \sqrt{\frac{\langle \kappa^* \rangle T_{end} C_C}{n_t^2 C_{\mathcal{G}}}} \tag{47}$$

Inserting (47) into (45) and using that $n_c = \frac{T_{end}}{n_t \delta T}$, the associated speedup is

$$\begin{aligned}\psi_{max} &= \frac{n_c n_t C_{\mathcal{F}} \delta T_{max}}{n_t \left(C_{\mathcal{G}} \delta T_{max} + T_C^w\right) + n_c \langle \kappa^* \rangle \left(C_{\mathcal{F}} \delta T_{max} + C_{\mathcal{G}} \delta T_{max} + T_C^w\right)} \\ &= \frac{n_t C_{\mathcal{F}} T_{end}}{T_{end} \langle \kappa^* \rangle \left(C_{\mathcal{F}} + C_{\mathcal{G}}\right) + n_t^2 T_C^w + 2\sqrt{n_t^2 C_C \langle \kappa^* \rangle C_{\mathcal{G}} T_{end}}}\end{aligned} \tag{48}$$

With ψ_{max} as above, the constraint (39) to be satisfied may be written as

$$\frac{n_c n_t C_{\mathcal{F}} \delta T}{n_t \left(C_{\mathcal{G}} \delta T + T_C^w\right) + n_c \langle \kappa^* \rangle \left(C_{\mathcal{F}} \delta T + C_{\mathcal{G}} \delta T + T_C^w\right)} \geq \epsilon_c \psi_{max} \tag{49}$$

Inserting (48) in (49) and manipulating the inequality we recover

$$\epsilon_c n_t^2 C_{\mathcal{G}} \delta T^2 + \epsilon_c \langle \kappa^* \rangle T_C^w T_{end} \leq \left(2\sqrt{n_t^2 \langle \kappa^* \rangle T_C^w C_{\mathcal{G}} T_{end}} + (1-\epsilon_c)\left(n_t^2 C_C + \langle \kappa^* \rangle \left(C_{\mathcal{F}} + C_{\mathcal{G}}\right) T_{end}\right)\right) \delta T \tag{50}$$

from which one may show that the smallest δT satisfying the above quadratic inequality is

$$\delta \hat{T}_{opt}^{\beta=1} = \delta \hat{T}_\epsilon - \sqrt{(\delta T_\epsilon)^2 - \delta \hat{T}_{max}^2} \tag{51}$$

where

$$\delta \hat{T}_\epsilon = \frac{1}{\epsilon_c} \delta \hat{T}_{max} + \frac{1-\epsilon_c}{2\epsilon_c}\left(\frac{C_C}{C_{\mathcal{G}}} + \frac{1}{C_C} \delta \hat{T}_{max}^2 \left(C_{\mathcal{F}} + C_{\mathcal{G}}\right)\right) \tag{52}$$

and $\delta \hat{T}_{max}$ as defined in (47). This yields a rough estimate of $\delta \hat{T}_{opt}^{\beta=1}$ that ensures that the choice of δT is not too long so that communication bandwidth is "wasted", while also making sure that moving data does not become a new significant limitation to parallel acceleration. In deriving $\delta \hat{T}_{opt}^{\beta}$ for the adaptive scheduler with $\beta \to 1$, i.e. impatient, it was assumed that the initialization procedure of the zeroth iteration was perfectly hidden by the adaptive scheduler for all but the first cycle as depicted in Fig. 6b. This is unlikely to be the case for all cycles, and we therefore suspect that the optimal δT will lie somewhere in the interval $\left[\delta \hat{T}_{opt}^{\beta=0}, \delta \hat{T}_{opt}^{\beta=1}\right]$ spanned by (51) and (51). In Section 5, parallel scaling is measured and presented for the test-case using each scheduler with a time-subdomain length as estimated from (44) and (51).

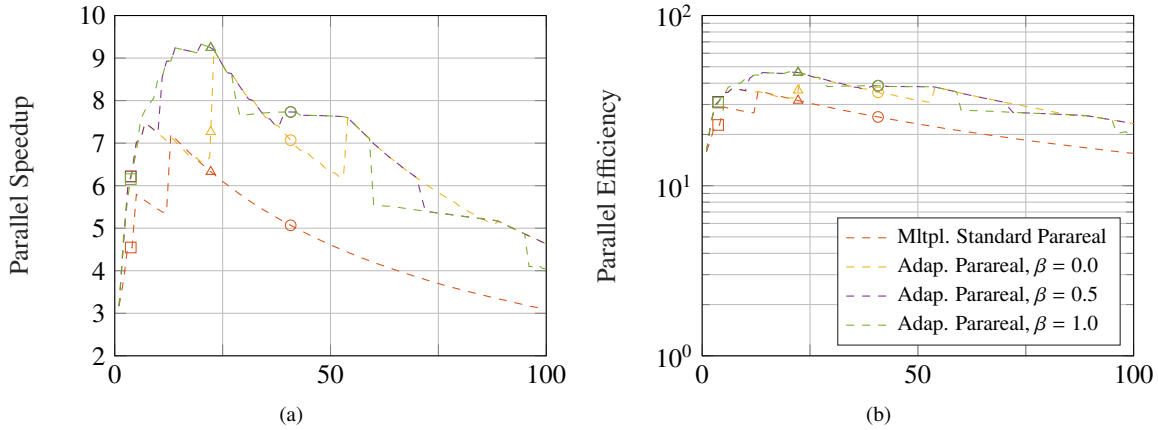


Figure 7: Parallel speedup and efficiency measured as a function of number of consecutive Parareal cycles n_c used when solving the complex wave ODE (53) on $n_t = 20$ processors parallel-in-time and letting $C_{\mathcal{F}} = 10000$ms, $C_{\mathcal{G}} = 100$ms, $T_c^w = 100$ms. The square, circle and triangle markers indicate the estimates for the optimal number of cycles to use as computed using (44), (51) and the average of the two respectively.

Measuring the speedup as a function of time-subdomain length for all possible time-subdomain lengths for the test-case introduced in Section 2.1 is computationally intractable. Instead we perform a test on a smaller numerical experiment by computing an approximation to the scalar complex initial value problem

$$\frac{d}{dt}u(t) = \lambda u(t), \quad u_0 = 1, \lambda = i \tag{53}$$

with $\mathcal{F}_{\delta T}$, and preconditioner $\mathcal{G}_{\delta T}$ given by

$$\mathcal{F}_{\delta T} U_n^k = (1 - \lambda dt)^{-\frac{\delta T}{dt}} U_n^k, \quad \mathcal{G}_{\delta T} U_n^k = (1 - \lambda dT)^{-\frac{\delta T}{dT}} U_n^k \tag{54}$$

using $dt = 10^{-5}$ and $dT = 10^{-3}$ in $\mathcal{F}_{\delta T}$ and $\mathcal{G}_{\delta T}$ respectively. Parallel-in-time integration until $T_{end} = 100$ with $n_t = 20$ simultaneously active time-subdomains on 20 processors. The test is performed using from $n_c = 1$ to $n_c = 100$ cycles, corresponding to time-subdomain lengths from $\delta T = 10^{-3}$ to $\delta T = 10^{-1}$. The tolerance on the residual is set to $\epsilon = 0.01$ for convergence. Since here the application of both $\mathcal{F}_{\delta T}$ and $\mathcal{G}_{\delta T}$ only require 6 floating point operations, the wall-time used by a processors is set manually to $C_{\mathcal{F}} = 10000$ms, $C_{\mathcal{G}} = 100$ms, and communication of a time-subdomain interval to $T_c^w = 100$ms. The numerical experiments are presented in Fig. 7. The estimates for the number of cycles that will result in the highest are indicated by the square, triangle and circle markers. Computed with the a priori guess $\langle \kappa^* \rangle = 2$ and limit $\epsilon_c = 95\%$. We observe good agreement between the predictions and the actual optimal time-subdomain lengths for large speedup.

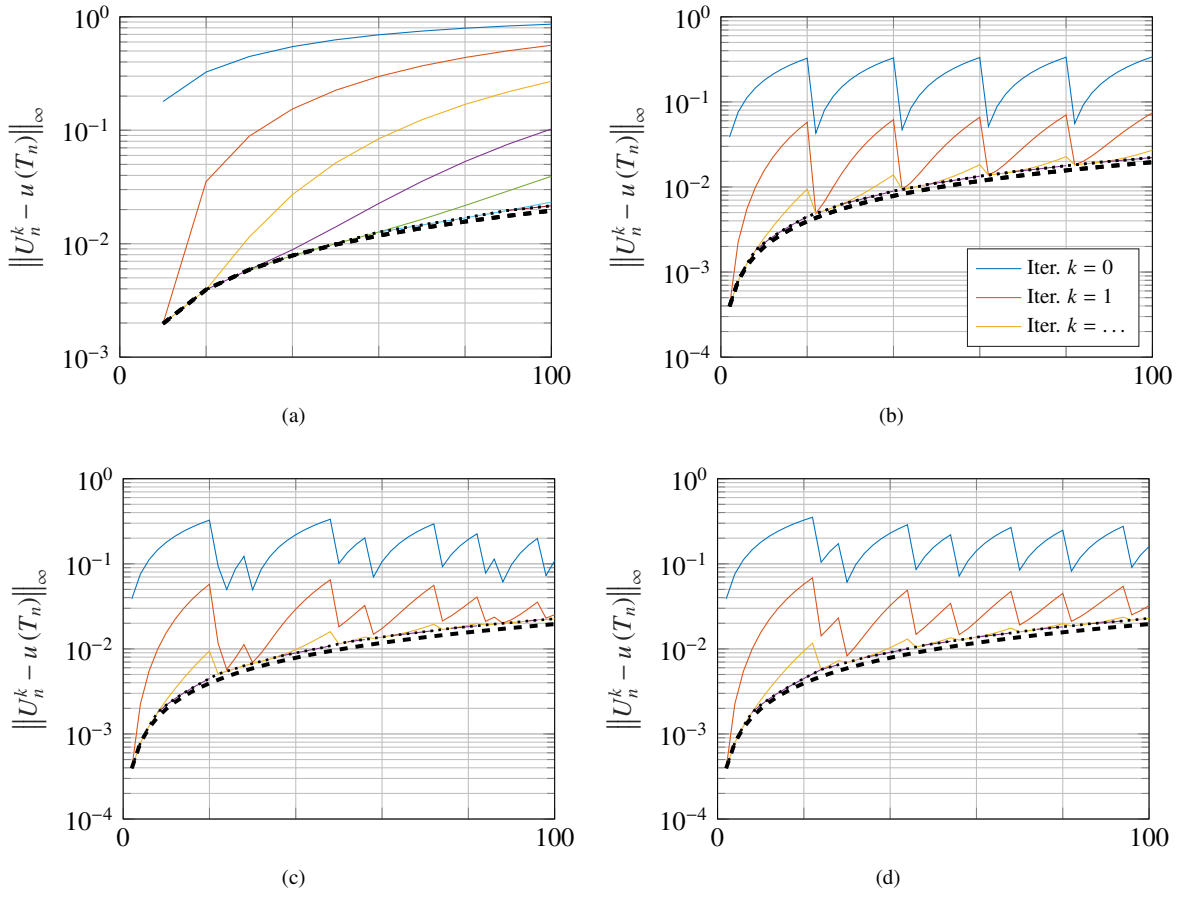


Figure 8: Error of U_n^k with respect to the analytical solution to (53) as a function of t for parallel-in-time integration of (11) with time-steps and tolerance $\epsilon = 0.01$ as used for the measurements in Fig. 7. (a) Standard Parareal, $n_c = 1$. (b) Standard Parareal, $n_c = 5$. (c) Adap. Parareal, $n_c = 5$, $\beta = 0$. (d) Adap. Parareal, $n_c = 5, \beta = 0.8$. The black dashed line indicate accuracy of the fine operator with respect to the analytical solution, and the black dotted line indicate the accuracy of $U_n^{k_{conv}}$ obtained at the iteration for which the convergence criteria was satisfied.

In particular we note how the δT resulting in the highest speedup for the adaptive scheduler was measured to be very close to the arithmetic average of n_c from the estimates (44) and (51) which indicate that the estimates may indeed be useful for approximating the optimal choice of time-subdomain length a priori.

4.3. Convergence Measurements

The convergence properties of a single cycle of the Parareal algorithm have been well studied both theoretically Bal (2005); Gander and Vandewalle (2007); Gander and Hairer (2008); Wu (2015) and experimentally by Ruprecht (2014); Steiner et al. (2015) among many others. The convergence properties of multiple consecutive cycles of Parareal and, in particular, when consecutive cycles are allowed to overlap, as in the proposed adaptive scheduler, might be very different. In Fig. 8 convergence for the test equation (53) is presented with $n_c = 5$ cycles. Since an analytical solution to (53) exists for all times t, we can measure the error as a function of t for each iteration so to gain insight into how the algorithm converges. One notes that convergence happens in substantially fewer iterations when using $n_c = 5$ time-subdomains in Fig. 8b than when only using a single cycle as in Fig. 8a. The convergence of the adaptive scheduler with $\beta = 0$ and $\beta = 0.8$ is presented in Fig. 8c and Fig. 8d respectively.

5. Numerical Experiments: Parallel Scaling

Numerical experiments using time-parallel integration on the test case introduced in section 2.1 is presented in the following. For all numerical experiments, the EPFL Bellatrix general purpose cluster consisting of 424 compute nodes, each with two 8-core Intel Xeon Sandy Bridge processors and infiniband QDR 2:1 network has been used. The adaptive work scheduler combined with the approach of estimating the time-subdomain length that optimally balances communication cost and convergence speed, is collectively denoted as CAAP. In all cases, an approximation to (11) is being computed on an interval of length $T_{end} = 60$min. A tolerance of $\epsilon = 10^{-4}$ on the residual between consecutive iterations is used as a convergence criteria. To evaluate if the solution found iteratively trough Parareal or CAAP is as accurate as the sequential WENO-RK solution, we define two errors measurements.

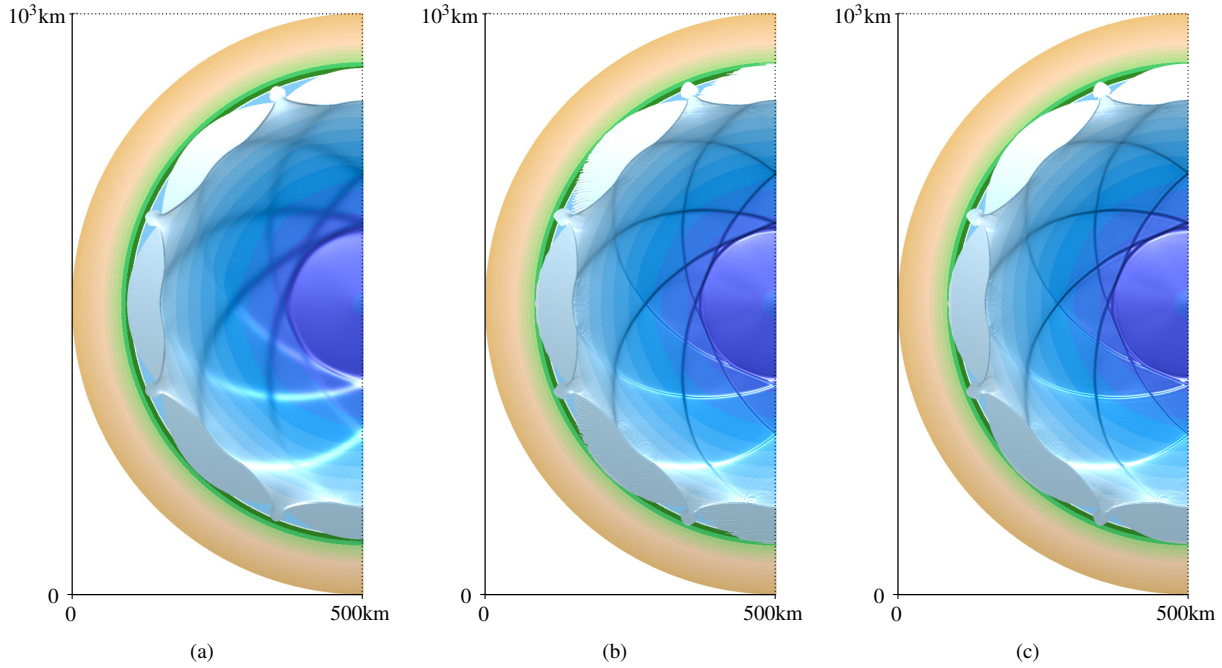


Figure 9: Water surface at $T = 60$min starting with the initial condition described in Section 2.1 at $T = 0$min as computed when using (a) Roe's method (b) WENO-RK (c) CAAP with tolerance $\epsilon = 10^{-4}$ on the residual.

The relative error with respect to the sequential WENO-RK solution

$$\tilde{\varepsilon}\left(U^{k_{conv}}_{n_t n_c}\right) = \frac{\sqrt{\int_\Omega \left|U^{k_{conv}}_{n_t n_c} - \mathcal{F}^{n_t n_c}_{\delta T} u\left(\cdot, T_0\right)\right|^2 d\Omega}}{\int_\Omega \mathcal{F}^{n_t n_c}_{\delta T} u\left(\cdot, T_0\right) d\Omega} \tag{55}$$

and the relative error with respect to the true solution

$$\hat{\varepsilon}\left(U^{k_{conv}}_{n_t n_c}\right) = \frac{\sqrt{\int_\Omega \left|U^{k_{conv}}_{n_t n_c} - u\left(\cdot, T_{end}\right)\right|^2 d\Omega}}{\int_\Omega u\left(\cdot, T_{end}\right) d\Omega} \tag{56}$$

No analytical solution is known to exists for our test-case. We therefore approximate the "true" solution using a very fine mesh with 14400x14400 cells. The relative error $\hat{\varepsilon}$ between the WENO-RK approximation and the true solution is $3.7 \cdot 10^{-3}$. For Roe's method, used here as a preconditioner in Parareal and CAAP, the relative accuracy $\hat{\varepsilon}$ is $1.2 \cdot 10^{-2}$. Ideally the error of the Parareal solution and the CAAP solution with respect to the true solution should be very close to that of the sequential WENO-RK solution. In the numerical experiments presented in Table 1, we measure $\hat{\varepsilon}$ to be between $3.7 \cdot 10^{-3}$ and $3.9 \cdot 10^{-3}$ for CAAP, and between $4.0 \cdot 10^{-3}$ and $4.1 \cdot 10^{-3}$ for the standard Parareal algorithm.

To estimate the number of Parareal cycles to use in CAAP, i.e. the time-subdomain length δT, we need to measure the constants $C_{\mathcal{F}}$, $C_{\mathcal{G}}$ and $T^w_{\mathcal{F}}$ on the EPFL Bellatrix cluster on which we will run our numerical experiments. We run the code over the full interval of 60 minutes so to estimate the two ratios $C_{\mathcal{F}}$ and $C_{\mathcal{G}}$. Doing so we find that we need roughly 4000ms to compute 1 minute of simulation time when using 5 nodes in space. For the coarse operator, we need roughly 300ms.

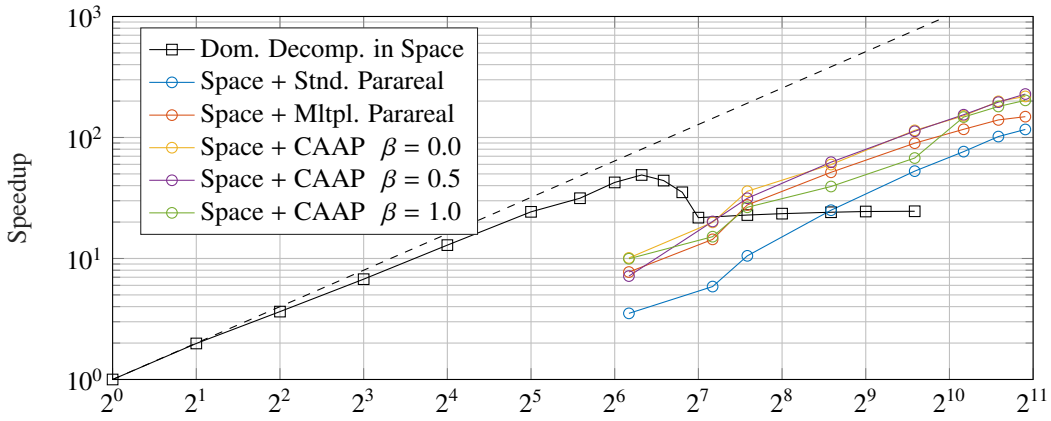
$$C_{\mathcal{F}} = 4018.3\text{ms/min}, \qquad C_{\mathcal{G}} = 309.4\text{ms/min}, \qquad T^w_{\mathcal{F}} = 75\text{ms} \tag{57}$$

Measuring the time-consumption of transferring a complete solution state across a time-subdomain interface from one group of nodes to another, denoted $T^w_{\mathcal{F}}$, proved quite difficult as the number fluctuates substantially depending on the load of the cluster and of the location of the nodes. Doing multiple runs on different node locations, we settled on 75ms on average when using 5 nodes in each group. In comparison it takes around 15ms to exchange the halo on the 1600x1600 cell test-case mesh. We let $\epsilon_c = 0.95$ and $\langle k^* \rangle = 2$, and use these measured quantities with (44) and (51) to estimate the optimal number of cycles n_c to split the time domain into. The values are listed in Table 1, the high estimates computed using (51) and the low using (44) , the estimates are rounded. The parallel speed-up of the average number of cycles between the two estimates is also tested. As can be seen from the numbers in the table, it appears to generally give the highest speed-up to use the average of the two estimates. In Fig. 10, the equivalent scaling measurements of the space+time parallel code is presented. The maximum attainable speed-up for the conventional parallel-in-space WENO-RK implementation is 49.0 using 5 nodes (80 cores). Using 6 nodes and above, no further speedup is possible. Adding CAAP with $\beta = 0.5$ and $n_t = 24$, we measure a speedup of 228.6 using 1920 cores. The factor 4.5 reduction in time to solution of the test-case is substantial and may be critical in cases where a simulation is expected to take days. It is also worth noticing that for CAAP with $\beta = 0.5$ and $n_t = 8$, we measure a parallel-in-time efficiency of almost 35%. To the knowledge of the author, this is the highest parallel efficiency that has been demonstrated for parallel-in-time integration of a purely hyperbolic PDE system using domain decomposition in time. We conjecture that for less challenging problems, e.g. diffusion dominated, still higher efficiency may be possible using the proposed method.

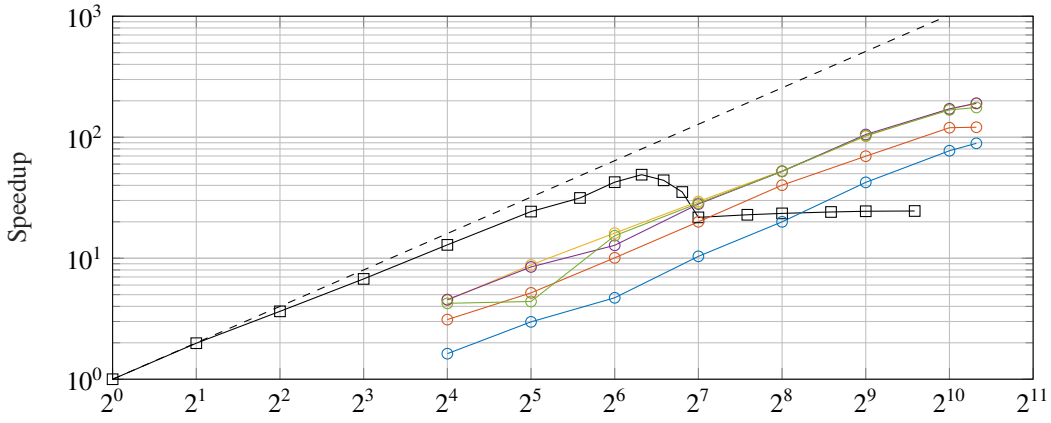
In our application the Roe's method that we use as a coarse operator is roughly 13 times faster than the WENO-RK solver that we wish to accelerate. This means that no matter how many simultaneously active time-subdomains we use, we will never gain more than a factor 13 reduction in time to solution with respect to the purely space parallel WENO-RK implementation. We did attempt using a cheaper coarse operator, the Lax-Friedrichs method with a diffusion term for stability. With this operator the ratio in question was roughly 40:1, potentially allowing for much create parallel-in-time speed-up. Unfortunately we observed that in order for the algorithm to convergence, the length of the time-subdomain had to be as short as a single time-step, otherwise the algorithm would diverge for a couple of iterations before eventually converging, leading to no, or very little, speedup. With a time-subdomain length equivalent to a single time-step, little parallel-speed-up was possible because the full spatial solution state had to be transfered from one group of nodes to another in between every timestep.

Type, $n_t = 8$	n_c	δT $[s]$	$\tilde{\varepsilon}$	$\hat{\varepsilon}$	P. Eff.[%]	P. Eff.[%]	P. Speedup
Stnd. Parareal	1	450	$1.4 \cdot 10^{-3}$	$4.1 \cdot 10^{-3}$	14.9	9.1	58.4
Mltpl. Parareal	5	90	$1.4 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	18.3	11.2	71.8
CAAP, $\beta = 0.0$	5	90	$1.4 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	25.5	15.6	100.1
CAAP, $\beta = 0.5$	5	90	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	22.2	13.6	87.2
CAAP, $\beta = 1.0$	5	90	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	22.0	13.5	86.4
Mltpl. Parareal	20	22.5	$1.4 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	27.1	16.6	106.4
CAAP, $\beta = 0.0$	20	22.5	$1.4 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	27.9	17.1	109.5
CAAP, $\beta = 0.5$	20	22.5	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	34.1	20.9	133.8
CAAP, $\beta = 1.0$	20	22.5	$1.5 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	31.7	19.4	124.4
Mltpl. Parareal	40	11.25	$1.4 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	24.8	15.2	97.0
CAAP, $\beta = 0.0$	40	11.25	$1.4 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	25.8	15.8	101.2
CAAP, $\beta = 0.5$	40	11.25	$1.4 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	31.8	19.5	125.1
CAAP, $\beta = 1.0$	40	11.25	$1.5 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	29.4	18.0	115.1
Type, $n_t = 16$	n_c	δT $[s]$	$\tilde{\varepsilon}$	$\hat{\varepsilon}$	P. Eff.[%]	P. Eff.[%]	P. Speedup
Stnd. Parareal	1	225	$1.4 \cdot 10^{-3}$	$4.1 \cdot 10^{-3}$	11.1	6.8	86.8
Mltpl. Parareal	3	75	$1.5 \cdot 10^{-3}$	$4.0 \cdot 10^{-3}$	8.7	5.3	67.3
CAAP, $\beta = 0.0$	3	75	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	11.9	7.3	93.7
CAAP, $\beta = 0.5$	3	75	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	12.1	7.4	95.0
CAAP, $\beta = 1.0$	3	75	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	11.6	7.1	90.3
Mltpl. Parareal	15	15	$1.5 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	15.5	9.5	120.9
CAAP, $\beta = 0.0$	15	15	$1.6 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	23.8	14.6	187.4
CAAP, $\beta = 0.5$	15	15	$1.6 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	23.8	14.6	186.7
CAAP, $\beta = 1.0$	15	15	$1.7 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	21.2	13.0	166.9
Mltpl. Parareal	15	7.5	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	17.1	10.5	134.0
CAAP, $\beta = 0.0$	30	7.5	$1.7 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	22.4	13.7	175.8
CAAP, $\beta = 0.5$	30	7.5	$1.6 \cdot 10^{-3}$	$3.7 \cdot 10^{-3}$	23.3	14.3	182.5
CAAP, $\beta = 1.0$	30	7.5	$1.7 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	22.5	13.8	176.9
Type, $n_t = 24$	n_c	δT $[s]$	$\tilde{\varepsilon}$	$\hat{\varepsilon}$	P. Eff.[%]	P. Eff.[%]	P. Speedup
Stnd. Parareal	1	150	$1.5 \cdot 10^{-3}$	$4.0 \cdot 10^{-3}$	10.1	6.2	118.6
Mltpl. Parareal	3	50	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	7.3	4.5	85.6
CAAP, $\beta = 0.0$	3	50	$1.6 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	9.0	5.5	106.0
CAAP, $\beta = 0.5$	3	50	$1.6 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	8.5	5.2	98.8
CAAP, $\beta = 1.0$	3	50	$1.6 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	8.5	5.2	99.3
Mltpl. Parareal	15	10	$1.5 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	13.1	8.0	153.7
CAAP, $\beta = 0.0$	15	10	$1.7 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	18.6	11.4	218.0
CAAP, $\beta = 0.5$	15	10	$1.7 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	19.4	11.9	228.6
CAAP, $\beta = 1.0$	15	10	$1.7 \cdot 10^{-3}$	$3.9 \cdot 10^{-3}$	17.1	10.5	201.0
Mltpl. Parareal	25	6	$1.6 \cdot 10^{-3}$	$3.7 \cdot 10^{-3}$	11.4	7.9	151.3
CAAP, $\beta = 0.0$	25	6	$1.8 \cdot 10^{-3}$	$3.7 \cdot 10^{-3}$	17.1	10.5	201.7
CAAP, $\beta = 0.5$	25	6	$1.7 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	18.6	11.4	218.4
CAAP, $\beta = 1.0$	25	6	$1.9 \cdot 10^{-3}$	$3.8 \cdot 10^{-3}$	14.7	9.9	189.4

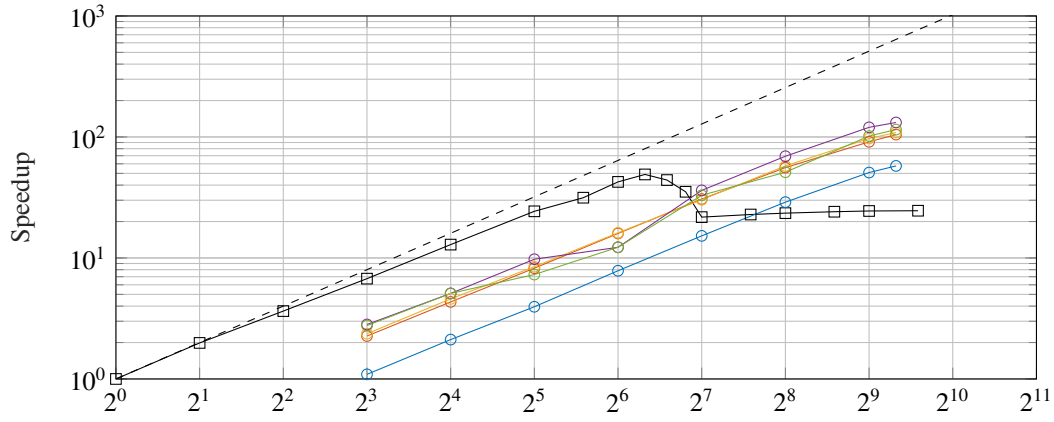
Table 1: Parallel acceleration as measured using $n_t = \{8, 16, 24\}$ simultaneously active time-subdomains. Within each time-subdomain, 5 nodes (80 cores) are used parallel in space, for a total of $\{640, 1280, 1920\}$ cores, respectively. The first parallel efficiency column indicate the efficiency of the parallel-in-time integration procedure, the second column indicates the combined space-time parallel efficiency. The relative error $\hat{\varepsilon}$ between the WENO-RK approximation and the correct solution is $3.7 \cdot 10^{-3}$. The maximum attainable speed-up for the conventional parallel-in-space WENO-RK implementation is 49.0 using 5 nodes (80 cores). Adding CAAP with $\beta = 0.5$ and $n_t = 24$, we measure a speedup of 228.6 using 1920 cores.



(a) Using $n_t = 24$ simultaneously active time-subdomains



(b) Using $n_t = 16$ simultaneously active time-subdomains



(c) Using $n_t = 8$ simultaneously active time-subdomains

Figure 10: Speedup as a function of number of cores used in the space-time parallel code for solving the test case introduced in Section 2.1 as measured on the EPFL Bellatrix cluster. The cluster consists of 424 compute nodes, each with two 8-core Intel E5-2660 CPUs and an Infiniband QDR 2:1 network. $T_{end} = 60$min, and $n_c = \{20, 15, 15\}$ for $n_t = \{8, 16, 24\}$ respectively. Tolerance $\epsilon = 10^{-4}$ on the residual was used as the convergence criteria.

6. Summary

The Parareal algorithm is one among several new algorithms that seek to introduce parallelism in time. Unlike other proposed models, Parareal has the distinct advantages of being potentially highly scalable and minimally invasive. The algorithm may to a large extent simply be wrapped around existing simulation code in combination with the introduction of some coarse operator. The parallel efficiency of the method, particularly for hyperbolic problems and for integration of long time domains has however been a major concern. In this paper we have demonstrated parallel-in-time efficiency of upwards of 35% for long time integration of a purely hyperbolic problem using CAAP. This is more than double of what we could achieve using the standard Parareal approach using a scheduler presented in Aubanel (2011). With CAAP, we achieved a speedup of 228 in our space-time parallel tests, compared to a maximum speedup of 49 for the original code. The factor 4.5 reduction in time to solution of the test-case is substantial and may be critical in cases where a simulation is expected to take days.

For the coarse operator $\mathcal{G}_{\delta T}$ we used a lower order discretization that operates on the same mesh as the original WENO-RK solver. The choice was motived by experience gathered testing multiple different coarse operators on a simpler 1D problem. In our experience, for fast convergence, one should avoid if possible to introduce a coarsened mesh as the interpolation or projection between grids appear to adversely affect convergence speed. In addition, we observed during these preliminary tests that designing a coarse operator in such a way that it is as accurate as possible under the constraint that $C_{\mathcal{F}}/C_{\mathcal{G}} > n_t$ is satisfied seems to lead to the highest overall speedup.

While we consider our findings to be substantial progress in the understanding of how to move past the strong scaling saturation limit of classical spatial domain-decomposition methods, there are potential improvements to be made. Using CAAP we have observed small load imbalances arise due to the nature of the test-case used. In the nonlinear shallow water wave equation solver for which we implemented time-parallel integration, time-steps are adaptive. In between each integration step of (33), the longest possible next timestep is computed and therefore some time-subdomains end up using a different number of time-steps than others. This in turn creates small load imbalances which may only be partially hidden by the adaptive scheduler. We speculate that a better approach may be to use the proposed optimal time-subdomain estimates derived in Section 4 to compute an optimal wall-time and from it set a fixed number of time-steps to use in a time-subdomain during the zeroth iteration by the preconditioner, i.e., the decomposition in time is not fixed, but computed dynamically and set during the zeroth iteration for each time-subdomain as the parallel-in-time integration procedure progresses. In this way one may achieve significantly better load-balancing for problems with an adaptive timestep integrator, and in doing so the proposed scheduler would be adaptive not only in balancing utilization and convergence, but also in setting the length of each time-subdomain dynamically during the integration procedure.

Finally, we note that Parareal and CAAP as methods for domain-decomposition in time are different from classical (spatial) domain decomposition in the sense that several copies of the solution states at time-subdomain interfaces must be stored. This extra use of memory is not of primary concern since Parareal is of interest in the strong scaling limit. In Nielsen and Hesthaven (2016) it was, however, demonstrated how these extra states may be used to make the algorithm in Aubanel (2011) tolerant to node failures, even when the underlying operators $\mathcal{F}_{\delta T}$ and $\mathcal{G}_{\delta T}$ themselves are not. Since the adaptive scheduler proposed in Section 3 is based on overlapping cycles of the same scheduler, we expect that it too may be made tolerant towards node failures in a similar manner.

Acknowledgements

The authors were supported by a European Commission Horizon 2020 project grant entitled ExaFLOW: Enabling Exascale Fluid Dynamics Simulations (grant reference 671571). The authors acknowledge the use of the EPFL SCITAS Bellatrix Compute Cluster for numerical experiments.

Aubanel, E., 2011. Scheduling of tasks in the parareal algorithm. Parallel Computing 37 (3), 172–182.
Bal, G., 2005. On the convergence and the stability of the parareal algorithm to solve partial differential equations. In: Domain decomposition methods in science and engineering. Springer, pp. 425–432.
Berry, L. A., Elwasif, W., Reynolds-Barredo, J. M., Samaddar, D., Sanchez, R., Newman, D. E., 2012. Event-based parareal: A data-flow based implementation of parareal. Journal of Computational Physics 231 (17), 5945–5954.
Chen, F., Hesthaven, J. S., Maday, Y., Nielsen, A. S., 2015. An adjoint approach for stabilizing the parareal method. Tech. rep.
Chen, F., Hesthaven, J. S., Zhu, X., 2014. On the use of reduced basis methods to accelerate and stabilize the parareal method. In: Reduced Order Methods for modeling and computational reduction. Springer, pp. 187–214.
Croce, R., Ruprecht, D., Krause, R., 2014. Parallel-in-space-and-time simulation of the three-dimensional, unsteady navier-stokes equations for incompressible flow. In: Modeling, Simulation and Optimization of Complex Processes-HPSC 2012. Springer, pp. 13–23.
Dai, X., Maday, Y., 2013. Stable parareal in time method for first-and second-order hyperbolic systems. SIAM Journal on Scientific Computing 35 (1), A52–A78.
Dongarra, J., Hittinger, J., Bell, J., Chacon, L., Falgout, R., Heroux, M., Hovland, P., Ng, E., Webster, C., Wild, S., 2014. Applied mathematics research for exascale computing. Tech. rep., Lawrence Livermore National Laboratory (LLNL), Livermore, CA.
Eghbal, A., Gerber, A. G., Aubanel, E., 2016. Acceleration of unsteady hydrodynamic simulations using the parareal algorithm. Journal of Computational Science.
Gander, M. J., 2015. 50 years of time parallel time integration. In: Multiple Shooting and Time Domain Decomposition Methods. Springer, pp. 69–113.
Gander, M. J., Hairer, E., 2008. Nonlinear convergence analysis for the parareal algorithm. In: Domain decomposition methods in science and engineering XVII. Springer, pp. 45–56.
Gander, M. J., Hairer, E., 2014. Analysis for parareal algorithms applied to hamiltonian differential equations. Journal of Computational and Applied Mathematics 259, 2–13.
Gander, M. J., Vandewalle, S., 2007. Analysis of the parareal time-parallel time-integration method. SIAM Journal on Scientific Computing 29 (2), 556–578.
LeVeque, R. J., 2002. Finite volume methods for hyperbolic problems. Vol. 31. Cambridge university press.
Lions, J.-L., Maday, Y., Turinici, G., 2001. Résolution d'edp par un schéma en temps pararéel. Comptes Rendus de l'Académie des Sciences-Series I-Mathematics 332 (7), 661–668.
Loderer, T., Heuveline, V., Lohner, R., 2014. The parareal algorithm as a new approach for numerical integration of odes in real-time simulations in automotive industry. PAMM 14 (1), 1027–1030.
Maidment, D., Mays, L., 1988. Applied Hydrology. McGraw-Hill series in water resources and environmental engineering. Tata McGraw-Hill Education.
Minion, M., 2011. A hybrid parareal spectral deferred corrections method. Communications in Applied Mathematics and Computational Science 5 (2), 265–301.
Nielsen, A. S., 2012. Feasibility study of the parareal algorithm. Master's thesis, MSc thesis, Technical University of Denmark.
Nielsen, A. S., Hesthaven, J. S., 2016. Fault tolerance in the parareal method. In: Proceedings of the ACM Workshop on Fault-Tolerance for HPC at Extreme Scale. FTXS '16. ACM, New York, NY, USA, pp. 1–8.
Pages, G., Pironneau, O., Sall, G., 2016. The parareal algorithm for american options. Comptes Rendus Mathematique 354 (11), 1132–1138.
Randles, A., Kaxiras, E., 2014. Parallel in time approximation of the lattice boltzmann method for laminar flows. Journal of Computational Physics 270, 577–586.
Reynolds-Barredo, J. M., Newman, D. E., Sanchez, R., Samaddar, D., Berry, L. A., Elwasif, W. R., 2012. Mechanisms for the convergence of time-parallelized, parareal turbulent plasma simulations. Journal of Computational Physics 231 (23), 7851–7867.
Ruprecht, D., 2014. Convergence of parareal with spatial coarsening. PAMM 14 (1), 1031–1034.
Samaddar, D., Newman, D. E., Sánchez, R., 2010. Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm. Journal of Computational Physics 229 (18), 6558–6573.
Shu, C.-W., 1998. Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws. In: Advanced numerical approximation of nonlinear hyperbolic equations. Springer, pp. 325–432.
Speck, R., Ruprecht, D., Krause, R., Emmett, M., Minion, M., Winkel, M., Gibbon, P., 2012. A massively space-time parallel n-body solver. In: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, p. 92.
Srinivasan, A., Chandra, N., 2005. Latency tolerance through parallelization of time in scientific applications. Parallel Computing 31 (7), 777–796.
Staff, G. A., Rønquist, E. M., 2005. Stability of the parareal algorithm. In: Domain decomposition methods in science and engineering. Springer, pp. 449–456.
Steiner, J., Ruprecht, D., Speck, R., Krause, R., 2015. Convergence of parareal for the navier-stokes equations depending on the reynolds number. In: Numerical Mathematics and Advanced Applications-ENUMATH 2013. Springer, pp. 195–202.
Thacker, W. C., 1981. Some exact solutions to the nonlinear shallow-water wave equations. Journal of Fluid Mechanics 107, 499–508.
Wu, S.-L., 2015. Convergence analysis of some second-order parareal algorithms. IMA Journal of Numerical Analysis 35 (3), 1315–1341.
Xiao, H., Aubanel, E., 2012. Scheduling of tasks in the parareal algorithm for heterogeneous cloud platforms. In: Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International. IEEE, pp. 1440–1448.
Xing, Y., Shu, C.-W., 2011. High-order finite volume weno schemes for the shallow water equations with dry states. Advances in Water Resources 34 (8), 1026–1038.