Feasibility study of the parareal algorithm

Allan S. Nielsen

Kongens Lyngby 2012
IMM-MSc-2012-nr.134

Technical University of Denmark
Informatics and Mathematical Modelling
Building 321, DK-2800 Kongens Lyngby, Denmark
Phone +45 45253351, Fax +45 45882673
reception@imm.dtu.dk
www.imm.dtu.dk IMM-MSc-2012-nr.134

Summary (English)

A general introduction to the topic of time-domain parallelism with motivation is given in Chapter 1. The 'parareal' algorithm itself along with pseudo-code for a simple implementation and all the basic properties are presented in chapter 2. Chapter 3 contains a comprehensive review of the research literature available as well as current state-of-the-art implementations of parareal. Investigations on the convergence rate of the algorithm applied to various types of differential equations using various numerical schemes as operators in the algorithm is presented in Chapter 4, followed by an investigation in Chapter 5 on optimal coarse propagator parameter choice when considering the distribution of parallel work. In this context a heuristic for the optimal choice of coarse propagator accuracy is proposed. In Chapter 6 the parareal algorithm is applied to a non-linear free surface water wave model. The convergence rate is analysed in a Matlab implementations of the algorithm, simulating the parallelism for a range of different wave types. As will be shown, the non-linearity of the wave and the water depth influence the parallel efficiency that can be obtained using the parareal algorithm. In Chapter 7 a large scale implementation of parareal is tested using a fully distributed task scheduling model to distribute the work that is to be computed concurrently on GPUs. The GPU implementation constitute a novel contribution to the literature on parareal. The findings in the report are summarized with a discussion on the feasibility and future outlook of the parareal algorithm for the parallel solution of initial value problems.

Summary (Danish)

En general introduktion til tids domæne parallisme samt grunde og motivation for at indføre dette er givet i kapitel 1, mens basale egenskaber ved algorithm præsenteres i kapitel 2. Kapitel 3 indeholder et omfattende litteraturstudie samt oversigt over nuværende state-of-the-art forskning indenfor parareal algoritmen. Undersøgelser af algoritmens konvergens rate for forskellige differential systemer med anvendelser af en række numeriske operatorer præsenteres i kapitel 4 efterfulgt af en undersøgelse i kapitel 5 omkring optimal parameter valg af den grove numeriske tids-integrator når der tages højde for distribution af parallelt arbejde. I denne sammenhæng forslås en heuristik til optimal valg af præcision af denne integrator. I kapitel 6 anvendes parareal algorithm til parallel acceleration af en ikke-lineær fri overflade vandbølge model. Her undersøges konvergens raten først grundigt ved hjælp af en rent sekventiel matlab implementering for forskellige bølge typer da det viser sig at ikke-lineariteten af modellen samt vanddybden influere den opnåelige parallelle effektivitet. I kapitel 7 introduceres en stor skala model af parareal ved anvendelse af en fuldt distribueret arbejdsdelings model og mange GPUer, hvilket udgør et nyt bidrag til litteraturen. Afslutningsvis opsummeres rapportens konklusioner i kapitel 7.4 med en diskussion af de anvendelsemæssige perspektiver af parareal til parallel acceleration af løsningen af begyndelsesværdiproblemer.

Preface

This thesis was prepared at the department of Informatics and Mathematical Modelling at the Technical University of Denmark in partial fulfilment of the requirements for acquiring the Master of Science in Mathematical Modelling and Computation.

The topic of the thesis is the parareal algorithm, a method for extracting parallelism in the solution of evolution problems. The potential for extracting parallelism in an otherwise seemingly sequential integration makes the parareal method interesting in the light of modern many-core architecture developments. The idea and purpose of the thesis is to investigate the applicability and potential usefulness of the algorithm as much work at DTU Informatics is somehow affiliated with the numerical solution of differential equations.

Lyngby, 5th of September 2012

Allan S. Nielsen

Acknowledgements

I would like acknowledge my gratitude towards my thesis supervisor Allan P. Engsig Karup for advice and guidance as well as many thorough discussions throughout the project. I would also like to thank my external supervisor Jan S. Hesthaven for supplying valuable literature contributions and feedback as well as access to the Oscar cluster at Brown University for tests using many GPUs.

In addition i have had the pleasure to work with Stefan L. Glimberg and his GPUlab library, supplying numerical solvers for the solution of the non-linear free surface water wave model to be tested with parareal.

Throughout the project i have made use of various compute resources and Bernd Dammann at IMM has been very kind to assist with questions arising in the usage of these resources, including the IMM GPUlab workstations, the General Access Central DTU HPC Cluster as well as the Oscar Compute Cluster at the Center for Computation & Visualization at Brown University.

Contents

Chapter 1

Introduction

In 2001, a new algorithm for the solution of evolution problems in parallel was proposed by Lions, Maday and Turinici [41]. The algorithm was named parareal on the prospects of being able to perform real time simulations by introducing large-scale parallelism in the time domain.

In this introductory chapter, the motivation for increased parallelism in numerical algorithms for the solution of ordinary and partial differential equations is clarified with respect to current hardware trends and modern compute architectures. As well as serving as a motivation, the chapter includes a short review of previous attempts at introducing parallelism in the time domain such as the time parallel multigrid methods and the waveform relaxation techniques. The methods are briefly illustrated and discussed with respect to the parareal algorithm proposed by Lion et. al. (2001) The current predictor-corrector form of parareal algorithm was first proposed in 2002 by Baffico et. al. [4] and several variants of the method has been proposed [23, 29]. The parareal algorithm has received a lot of attention over the past few years, particularly from the domain decomposition literature. In chapter 2, the algorithm is presented as in [27] where it was shown that the parareal algorithm can be seen as either a variant of the multiple shooting method, or a two level multigrid method with aggressive time coarsening. The chapter also includes a simple implementation example and an overview of essential algorithmic properties. Chapter 3 contains an extensive survey of the present state of research as well as an overview of

which problems the algorithm has been tested on.

In chapter 4 the implementation of the parareal algorithm applied to a range of equations from ODE systems to PDEs are presented along with convergence measurements and parallel speedup estimates. Chapter 5 extend the demonstrations of chapter 4 with a theoretical analysis of convergence in conjunction with an efficiency analysis, followed by a discussion on heuristics for optimal parameter choice. The final two chapters of the report are on the application of parareal to a fully nonlinear free surface wave model with the goal of using GPUs as workers in the algorithm. In chapter 6, stability and convergence of parareal on the wave model for long time integration is investigated and in addition, convergence speed over a wave parameter space is measured for various coarse propagators. In chapter 7 a CUDA/MPI/C++ implementation is presented with the parallel work being distributed following a model proposed by [2]. Results are discussed and compared with speedup obtained by classical domain decomposition.

1.1 Motivation for parallelism in numerical algorithms

As the usage of computer simulations within engineering and natural sciences increase, so does the demand for computational throughput and speed from engineers and scientists alike. In the context of numerical solvers to differential equations, these are often formulated as sequential algorithms, but unfortunately many problems faced in real-life are much to computationally expensive to be computed on a single processor. An abundance of such problems exists, e.g. weather forecasts, molecular dynamics, High energy and Quantum physics, semiconductor modelling and so on. Such large-scale problems have long been solved on large-scale clusters consisting of individual processors and as such, parallelism as an approach of accelerating the simulation physical models is by no means a new thing. Parallel numerical algorithms has been an area of research for a long time, and in this context the first papers on the specific topic of time-parallel time integrators where published as early as the nineteen sixties by J. Nievergelt [53] and Miranker Et. Al. [51] on parallel methods for the integration of ordinary differential equations.

For the past many decades, improvements within manufacturing and processor technologies has been the main driver of performance increases, allowing new generations of microprocessors to shrink dimensions and thereby reduce power consumption, add transistors and increase operating frequency. All of this to add to the sequential throughput generated by general purpose CPUs. This

ride, however wonderful, is over. Since mid 2000s, chip manufactures have had to seek out new ways of improving the performance of their hardware. While the number of transistors that can be fitted on a chip continues to increase, the ability to increase clock speed has hit a practical barrier often referred to as the power wall. The exponential increase in power required to increase the clock frequency has hit practical cost and design limits. [1]

The classic approach by chip-manufactures in the past has been to increase the processor operating frequency whenever new fabrication technologies would allow to do so, while at the same time adding transistors for more cache, branch prediction and control logic so to keep a single sequential thread running ever faster. However, with fundamental physical laws limiting the practical scalability of processor frequency, this approach is no longer viable. The ever increasing amount of transistors per chip coming available with new fabrication technologies is still projected to continue for another decade though, and the new strategy by chip-manufactures, wanting to provide competitive silicon for the end-users, has been to use the added transistors to supply more individual compute cores to each processor die [11]. Thus, end-users can no longer rely on their legacy code, per default, to run faster on new hardware generations. As of now, with added compute comes the need for added parallelism, and this forces much software to be changed, algorithms to be revised and new ones to be developed [65].

It is hard to understate the disruptive change that has happened within high performance computing for the last couple of years and the years to come - in addition to the challenges presented above by not being able to scale the clock frequency, HPC vendors and users have experienced other problems. The scale out strategy of increasing performance by racking and stacking an ever increasing amount of compute nodes has hit physical limitations of space("The wall wall"), as well as cooling and electricity supply limitations. The industry response to these challenges seem to converge towards what has become known as heterogeneous computing. A general purpose processor will by default include a wide range of functional units, to be able to respond to any computational demand. This is what makes it a general purpose processor. Accelerators are specialised to handle only specific functions or tasks, i.e. fewer transistors are wasted during compute because only those functional units required by the compute purpose are included. See [52] for a discussion of current hardware trends. The trends in compute hardware towards more parallelism and heterogeneity, unfortunately leads to an increase in the complexity of code and algorithms which adds to the burden of the application developers forcing these to search for new methods and opportunities to exploit these compute resources.

A now classical approach for the parallel solution of partial differential equations is the domain decomposition methods. Domain decomposition is used to

solve boundary value problems, and therefore typically parallel in space. The domain decomposition method adds parallelism to the problem by splitting the dimension on which the boundary values are defined into a number of smaller boundary value problems on sub-domains and iterating over these to coordinate the solution between adjacent sub-domains. The problems on each sub-domain is independent, and thus suitable for parallel computing. The efficiency of the best domain decomposition algorithms based on an update of internal boundary conditions is such that it leads to the expected convergence with a number of iterations that is only slightly dependent on the number of sub-domains that are involved and the solution of large scale problems over P processors using the domain decomposition algorithms is often close to P times faster than a single processor. The domain decomposition methods is also applicable on problems with mixed boundary and initial conditions, say solving parallel in space with boundary conditions while iterating sequentially forward in the time-domain from an initial state.

Even though domain decomposition is very efficient, at present, many scientific and industrial problems reach their speed-up limit for a limited number of processors due to a degradation of the scalability when the number of processors become large. For problems involving long time integration, a method to add parallelism to the temporal direction is certainly of great interest. The issue is even more critical for systems of ordinary differential equations where classical domain decomposition methods are not applicable [43]. Contrary to space however, time is, by its very nature, sequential, and this precludes a straightforward implementation of a parallel approach. In the following section, various approaches of introducing parallelism in the temporal domain is presented.

1.2 Parallelism in the time domain

In order to exploit the massive throughput capabilities of modern and emerging hardware architectures, increased parallelism in numericals algorithms are of great importance. For the most part, time has long not been considered competitive as a viable direction for parallelization in evolution problems. As detailed in the previous section, modern compute processors as a simple rule of thumb is now to be expected to contain more compute units rather than faster compute units, compared to that of older generation hardware. This development have lead to an increased focus on temporal parallelization in research. Parareal is by no means the first algorithm to propose the solution of evolution problems in a time-parallel fashion. Already in 1964, J. Nievergelt [53] proposed a parallel time-integration scheme where the interval of integration is subdivided and a number of different problems are solved on each interval concurrently followed

by a parallel interpolation. The approach is now considered to be impractical, but both [37] and [36] considered extensions of this approach trough the use of parallel multiple shooting techniques, and as such, Nievergelt work eventually evolved into the multiple shooting methods applied to the time domain

Since the paper by Nievergelt in 1964 there have been many contributions in the literature dealing with parallelization of time dependent problems. For an in-depth synthetic approach to previous attempts at time-domain parallism, the book [12] by K. Burrage published in 1995 presents a, by that time, up-to-date exposition of "state of the art" within numerical methods for solving ordinary differential equations in a parallel computing environment. The various techniques for solving evolution problems are by Burrage classified into three categories, parallelism across the system, parallelism across the method and parallelism across the time. Below they are here presented as in [43].

- **Parallelism across the system.** The methods in this category consists those where the right hand side is partitioned over its various components and the computations of the components spread across different processors. Parallelism across the system is the most common and efficient approach when the dimension of the system is large, such as in gravitational or molecular simulations.
- **Parallelism across the method** As the name implies, this approach is directly linked to the numerical scheme used to solve the equation, and as such, the efficiency highly depends on the characteristics of the scheme. Research in this direction has led to the design intrinsically new parallel schemes. Runge-Kutta methods and partitioned Runge-Kutta methods offer the possibility of parallelism across the method.
- **Parallelism across the time.** This approach consists in breaking up the integration interval into sub-intervals, solving these concurrently over each sub-interval. The obvious issue with this approach is to provide the correct seed values at the beginning of each integration sub-interval. The techniques in this category are mostly of the multishooting class methods, stemming from the precursor work by A. Bellen et M. Zennaro [8] and [13] on the parallelism across the steps, leading to the waveform relaxation methods introduced by E. Lelarasmee, A. E. Ruehli and A. L. Sangiovanni-Vincentelli [38] and the multigrid approaches introduced by W. Hackbush [31].

The focus of this thesis is on the recently proposed parareal in time approach that was first presented in [41] and enters into the third category mentioned above. In [27] it was shown that the parareal scheme can be interpreted as

a multishooting technique or as a two level multigrid in time approach. The leading idea however came from techniques used in spacial domain decomposition. Other attempts to parallelize in time such as the parallel runge-kutta schemes [34,35] are very limited in the type of schemes to select from and they have yet to prove effective for stiff problems. The waveform relaxation [59] is a more general method which works on a wider-range of problems, but in general the various approaches to time parallel time integration suffer from a number of issues. Many are only applicable to certain classes of problems or suffer from issues with non-linear equations, are limited to small-scale parallelism only, or even limited to certain hardware architectures or numerical solvers. Classical domain decomposition methods have proved very effective as a mean of introducing parallelism and as such, 3rd category time-parallelism has yet to find broad usage. With projected trends in hardware development this may very well change and in this context the parareal algorithm has shown a number of promising results. In addition, the algorithm has several advantages over other parallel-in-time algorithms as listed below

- Has already been tested successfully on a wide range of problems.
- Successfully applied on non-linear equations.
- Not limited to small-scale parallelism.
- Can be used in conjunction with any combination of integrators.
- Relatively easily implemented in it's most basic form.
- Fault tolerant algorithm.

The parareal in time algorithm is still very much an active area of research with many recent publications as challenges and unknowns still exists. A number of these issues will be identified and discussed in the chapters 3 and 4. Due to the paramount importance of finding new ways of extracting parallelism going together with the ever increasing number of compute units present in the new generations of compute architectures, algorithms such as parareal in time and other time-parallel integrators have become an active research topic showing promising applicability for large-scale computing and perhaps even amounting to a small step towards the exascale computations challenge [58]. In the chapter to follow the parareal algorithm is introduced.

Chapter 2

The Parareal Method

The parareal algorithm was first introduced in 2001 in a paper authored by J.-L. Lions, Y. Maday and G. Turinici. In this chapter the currently used predictor-corrector form first proposed in [4] is introduced to the reader. The predictor-corrector formulation is completely equivalent to the one presented originally in [41] when the algorithm is applied to linear differential equations, but the approach presented in [4] has been shown to be better at handling nonlinear equations than the original formulation. In the section to follow the algorithmic idea is clarified along with the introduction of basic definitions and conventions to be used throughout the report. An algebraic interpretation as presented in [46] is given along with a visual presentation of the iterative solution procedure for a simple ordinary differential equation. To round of the presentation, a walk-trough of important properties of the algorithm that may otherwise not seem obvious at a first encounter is given along with an introductory discussion on computational complexity and parallel efficiency of the algorithm.

2.1 The Algorithmic Idea

It seems intuitive to consider time to be inherently sequential. And considering the research that has gone into developing parallel time integrators, it must be

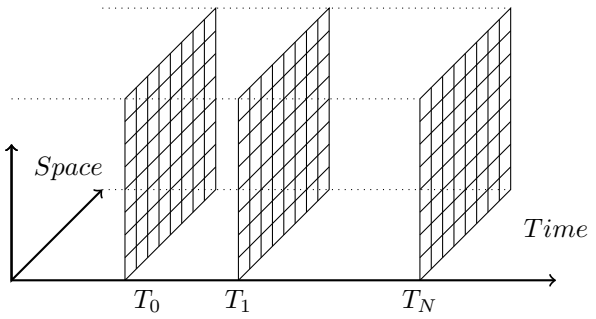


Figure 2.1: Visual presentation of time decomposition.

fair to state that it is indeed difficult to simulate the far future without knowing the close future in great detail.

The leading idea for the parareal algorithm came from spacial domain decomposition. The parareal in time approach proposes to break the global problem of time evolution into a series of independent evolution problems on smaller intervals. Initial states for these problems are needed, and supplied by a simple, much less accurate, but fast sequential time integrator. The smaller independent evolution problems can then be solved in parallel with the use of more expensive and more accurate integrators.

The information generated during the concurrent solution of the independent evolution problems with accurate propagators and inaccurate initial states is then used in a predictor-corrector fashion in conjunction with a simple integrator to propagate the solution faster, now using the information previously generated. The strategy is to do a time decomposition in the spirit of domain decomposition. We define the decomposition into N intervals as depicted in figure 2.2, that is

$$T_0 < T_1 < \cdots < T_n = n\Delta T < T_{n+1} < T_N$$

Now, let us define the general problem on the above decomposed time domain,

$$\begin{cases} \dfrac{\partial u}{dt} + \mathcal{A}u = 0 & \text{(2.1a)} \\ u(T_0) = u^0\,, t \in [T_0, T_N] & \text{(2.1b)} \end{cases}$$

where A is an operator from one Hilbert space to another. To solve the differential problem 2.1 we define a numerical operator $\mathcal{F}_{\Delta T}$ that operates on some initial state $u(T_{n-1}) = U_{n-1}$ and approximate the solution to 2.1 at time $T_{n-1} + \Delta T$, using some small time step $\delta t \ll \Delta T$. The operator $\mathcal{F}$ will throughout the report as of now be referred to as the fine propagator or fine integrator. A numerical solution to 2.1 can be obtained by applying the fine propagator 2.2

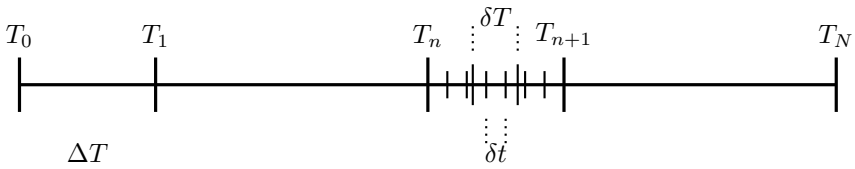


Figure 2.2: Decomposition of time into smaller domains.

sequentially for $n = 1, 2, \ldots, N$.

$$\hat{U}_n = \mathcal{F}_{\Delta T}\left(T_{n-1}, \hat{U}_{n-1}\right), \quad \hat{U}_0 = u^0 \tag{2.2}$$

In addition to the fine propagator, let us define a coarse propagator $\mathcal{G}_{\mathcal{T}}$. Again $\mathcal{G}_{\Delta T}$ operates on some initial state $u(T_{n-1}) = U_{n-1}$ and propagate the solution over a time ΔT, but now using a time-step δT. Typically $\delta t < \delta T < \Delta T$. For the parareal algorithm to be effective, the coarse propagator $\mathcal{G}_{\Delta T}$ has to be substantially faster to evaluate numerically than the fine propagator $\mathcal{F}_{\Delta T}$. Many ways of constructing the coarse propagator $\mathcal{G}_{\Delta T}$ has been proposed such as using a different type numerical solver, simplified physics or simply longer steps making the propagator less accurate but faster. Various types of coarse propagators are presented in chapter 3. The coarse operator reads

$$\tilde{U}_n = \mathcal{G}_{\Delta T}\left(T_{n-1}, \tilde{U}_{n-1}\right), \quad \tilde{U}_0 = u^0 \tag{2.3}$$

A less accurate numerical solution to 2.1 can then be obtained by applying the coarse operator 2.3 sequentially for $n = 1, 2, \ldots, N$. The introduction of the notation $\hat{U}$ for a state returned by the fine propagator and the notation $\tilde{U}$ for a state returned the coarse operator is adopted from [2] as it is very useful in pseudo code descriptions of what data can be discarded and what is needed for further iterations, the notation is used throughout the report. The predictor-corrector form of the parareal algorithm, hereafter simply referred to as parareal or PA is stated in 2.4. When the algorithm initiates, a strictly sequential application of the coarse propagator $\mathcal{G}_{\Delta T}$ for $n = 1, 2, \ldots N$ is applied to provide initial states $\tilde{U}_n^0$. This purely sequential propagation is to provide the first predictions, and we define this propagation as iteration $k = 0$. Now, processors $n = 1, 2, \ldots N$ can solve $\hat{U}_n^0 = \mathcal{F}_{\Delta T}\left(\tilde{U}_{n-1}^0\right)$ concurrently. Following the concurrent solution of $n = 1, 2, \ldots N$ initial value problems, a sequential coarse propagator will generate predictions to be corrected with the information generated from the concurrent fine propagation, at each subinterval in time the initial state for the fine propagation in the next iteration is computed. As such, the PA algorithm reads

$$U_n^k = \mathcal{G}_{\Delta T}\left(U_{n-1}^k\right) + \mathcal{F}_{\Delta T}\left(U_{n-1}^{k-1}\right) - \mathcal{G}_{\Delta T}\left(U_{n-1}^{k-1}\right), \quad U_0^k = u^0, \quad n = 1, \ldots, N \tag{2.4}$$

where $k = 1, 2, \ldots k_{max}$. The number of iterations k, is here defined as the number of times the fine propagator $\mathcal{F}_{\Delta T}$ is used to concurrently solve N initial value problems, followed by sequential propagation $\mathcal{G}_{\Delta T}\left(U_{n-1}^{k}\right)$ corrected by the last two terms in 2.4 The iterations are repeated until the solutions has converged to the solution one would have obtained using the operator $\mathcal{F}_{\Delta T}$ in a strictly sequential fashion. In algorithm 1 pseudo code for the implementation of parareal in its most simple form is presented as in [2]. The pseudo code

Algorithm 1 Pseudo code for a simple implementation of parareal

$U_0^0 \leftarrow \tilde{U}_0^0 \leftarrow y_0$
\\iteration 0
for $n = 1$ to N **do**
  $\tilde{U}_n^0 \leftarrow G_{\Delta T}\left(\tilde{U}_{n-1}^0\right)$ \\Initial prediction
  $U_n^0 \leftarrow \tilde{U}_n^0$
end for
\\First parareal iteration
for $k = 1$ to K_{max} **do**
  $U_0^1 \leftarrow y_0$
  for $n = 1$ to N **do**
    $\hat{U}_n^{k-1} \leftarrow F_{\Delta T}\left(\tilde{U}_{n-1}^{k-1}\right)$ \\Parallel step
  end for
  for $n = 1$ to N **do**
    $\tilde{U}_n^k \leftarrow G_{\Delta T}\left(U_{n-1}^k\right)$ \\Predict
    $U_n^k \leftarrow \tilde{U}_n^k + \hat{U}_n^{k-1} + \tilde{U}_n^{k-1}$ \\Correct
  end for
  if $|U_n^k - U_n^{k-1}| < \epsilon \, \forall \, n$ **then**
    BREAK \\Terminate loop if converged
  end if
end for

presented in algorithm 1 calls for a discussion of conventions. In some parts of the literature, iterations are defined as the number of sequential propagations, not the number of fine propagations. The difference is particularly important to take note of when evaluating analytical convergence results as presented in chapter 3. In addition to this difference, a number of variations on the parareal algorithm in its basic form as presented in 1 can be found in the literature. The code presented terminates when measuring convergence after using the fine propagation information to generate a new solution to the problem 2.1. This is favourable if one is only looking for the solution at the final time T_N. However, if an accurate solution is needed over the entire interior interval at the fine time grid points, then one would need to do an additional fine propagation after the final predictor-corrector to "update" the interior. If the later is the

case it would be favourable to define the loop in a different manner so that the first fine propagation happen before the loop initiates and instead the loop begins with the predictor-correction procedure 2.4 upon evaluating convergence followed by the parallel fine propagation to update the interior $[T_0, T_N]$. This fairly basic difference in conventions has lead to some confusion with regards to the theoretical obtainable efficiency and speed-up of algorithm.

For the reader who is not familiar with the term speed-up and efficiency in the context of parallel computations, a simple definition is given. The speed-up is the ratio between the wall-clock time of the serial and the parallel computation. The efficiency is the ratio of the speed-up to the number of compute units used in the acceleration.

Parareal is an iterative algorithm and therefore needs a stopping criteria, let us for the moment just note that a sufficiently small value can be used as criteria with respect to the norm of the difference between the solution after two consecutive parareal iterations. A more involved discussion on this topic and present results in literature is given in chapter 3 and in chapter 4 it will be shown how it is possible to use such a simple stopping criteria, but at the same time it is also likely to decrease the obtainable efficiency to a, for practical purposes, unacceptable level. Before further discussions on parareal, an algebraic interpretation of the algorithm is presented in the following section.

2.2 An algebraic interpretation

Given the decomposition into N time domains,

$$T_0 < T_1 < \cdots < T_n = n\Delta T < T_{n+1} < T_N$$

the differential problem 2.1 can then be rewritten as N initial value problems on the form

$$\begin{cases} \dfrac{\partial u_n}{dt} + \mathcal{A}u_n = 0 & \text{(2.5a)} \\ u_n(T_n^+) = U_n \;\; t \in [T_n, T_{n+1}] & \text{(2.5b)} \end{cases}$$

for $n = 0, \ldots, N-1$. The collection of solutions $\{u_0, u_1, \ldots, u_{N-1}\}$ is connected to the solution u of the original problem if and only if, for all $n = 0, \ldots, N-1$, we have that $U_n = u_{n-1}(T_n)$ with $u_{-1}(T_0) = u^0$. Recall our fine propagator $\mathcal{F}_{\Delta T}(U_{n-1})$, using the propagator we can write the numerical solution of

problem 2.5 in a matrix form as

$$\begin{pmatrix} I & 0 & 0 & \cdots & 0 \\ -\mathcal{F}_{\Delta T} & I & 0 & \dots & 0 \\ 0 & -\mathcal{F}_{\Delta T} & I & 0 & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & -\mathcal{F}_{\Delta T} & I \end{pmatrix} \begin{pmatrix} U_0 \\ U_1 \\ U_2 \\ \vdots \\ U_{N-1} \end{pmatrix} = \begin{pmatrix} u^0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \tag{2.6}$$

or with matrix notation as

$$M\Lambda = F \tag{2.7}$$

where $\Lambda = \{U_0, U_1, \dots, U_{N-1}\}$. The sequential nature of the scheme clearly appears. A lower triangular system is solved by a standard inversion involving $\mathcal{O}(N)$ propagations. However, we are not interested in solving the matrix problem 2.6 in a sequential fashion, but rather we are interested in somehow introducing parallelism. Notice that the conditions $U_n = u_{n-1}(T_n)$ for all $n = 0, \dots, N-1$ can be seen as a a cost functional to be minimized in some norm

$$\mathcal{J}(\Lambda) = \sum_{n=1}^{N-1} \left\| u_{n-1}(T_n^-) - U_n \right\|^2$$

When the conditions on the time interval boundaries are completely satisfied, $\mathcal{J}$ is zero. To introduce parallelism in the inversion of 2.6, we recall the definition of the coarse operator $\mathcal{G}_{\Delta T}$ and use it to define the matrix

$$\widetilde{M} = \begin{pmatrix} I & 0 & 0 & \cdots & 0 \\ -\mathcal{G}_{\Delta T} & I & 0 & \cdots & 0 \\ 0 & -\mathcal{G}_{\Delta T} & I & \cdots & \cdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \mathcal{G}_{\Delta T} & I \end{pmatrix}$$

from which it is possible to deduce the following algebraic formulation of the parareal in time algorithm as by [46]

$$\Lambda^{k+1} = \Lambda^k + \widetilde{M}^{-1} Res^k \tag{2.8}$$

where the residual is defined by $Res^k = F - M\Lambda^k$. To some extend $\widetilde{M}^{-1}$ can then be considered as a pre-conditioner for 2.7, in the sense that the matrix $\widetilde{M}^{-1}M$ is close to identity.

2.3 A Visual Presentation

To ease the qualitative interpretation of the solution procedure, a visual presentation of the iterative approach is presented in this section. We define the

ordinary differential equation initial value problem

$$\frac{\partial y}{dt} = \sin(t)\, y(t) + t \tag{2.9a}$$

$$y(0) = 1 \quad , t \in [0, 14] \tag{2.9b}$$

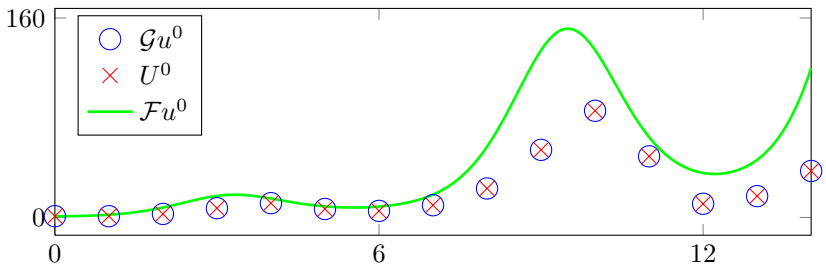
and in addition we define a time-domain grid as in 2.1. In order to apply the parareal algorithm, we need to define a fine propagator $\mathcal{F}_{\Delta T}$. For this example we will use a simple forward euler approach with a fine time-step δt. The method reads

$$Y_{i+1} = (1 + \delta t \sin(t_i))\, Y_i + t_i \delta t \tag{2.10}$$

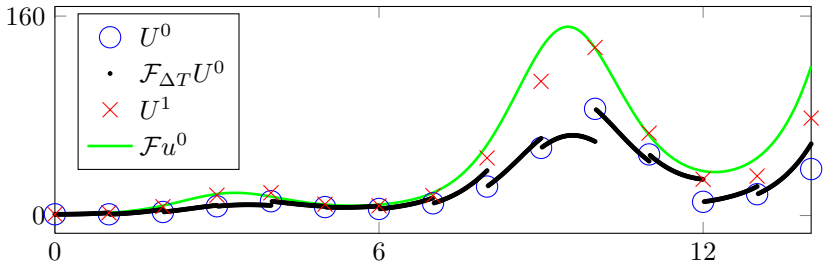
To propagate a distance ΔT, we would need to do $\frac{\Delta T}{\delta t}$ steps of the above discretization, this constitutes the fine operator $\mathcal{F}_{\Delta T}$. Similarly we define the coarse operator to also use the forward euler discretization 2.10, but now with a time-step δT, and $\frac{\Delta T}{\delta T}$ steps to integrate a distance in time ΔT.

For the sole purpose of demonstration, the solution is implemented in Matlab with simulated parallelism, using the most basic application approach as given in algorithm 1. The solution time domain $t \in [0, 14]$ is split into 14 intervals of length $\Delta T = 1$. The time-step of the coarse propagator is set at $\delta T = \Delta T = 1$ and the time-step of the fine operator is chosen at $\delta t = 0.02$. Notice the ratio between computations spend applying the coarse propagator to a state, and applying the fine propagator. Both propagating the solution a time-interval $\Delta T = 1$. In this case the ratio is $R = 50 : 1$. This ratio is of great interest in estimating speedup of the method and throughout the report we refer to it as simply $R = 50$, occasionally using r, with $r = R^{-1}$.

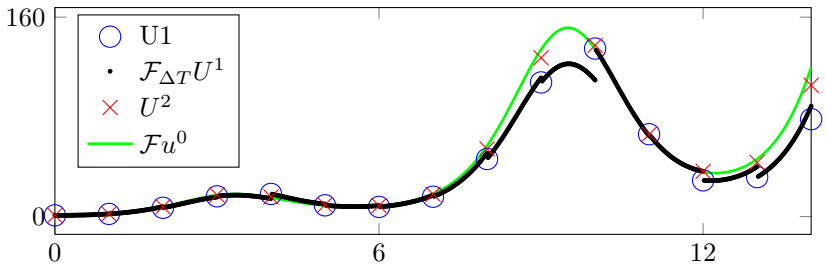
In figure 2.3 the results of the first four iterations of the parareal algorithm implemented as described above is visualised. For clarity, only the previous iteration corrected predictions U^{k-1} as well as the fine propagation $\mathcal{F}_{\Delta T}\left(U^{k-1}\right)$ on the later and the new corrected prediction U^k is visualised. Figure 2.3a show iteration 0, figure 2.3b iteration 1 and so forth. Notice how in iteration zero, the algorithm is initialised and the corrected prediction U^0 is simply the coarse propagation applied to the initial condition and propagated trough the entire domain.



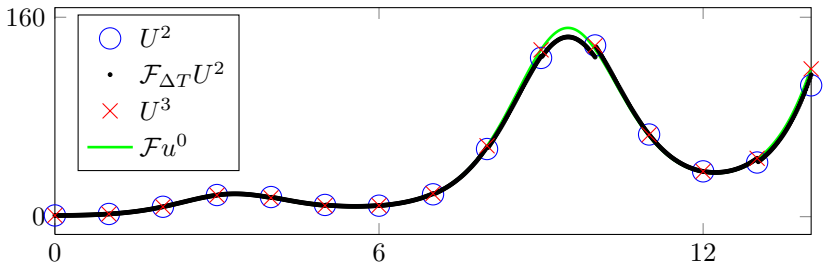
(a) Iteration 0



(b) Iteration 1



(c) Iteration 2



(d) Iteration 3

Figure 2.3: Parareal iterations in the solution of the ordinary differential equation initial value problem 2.9

The green line appearing in all figures is the solution obtained by applying the fine solver in a purely sequential manner throughout the time domain. The corrected predictions, and the fine propagations on these, are seen to converge towards the fine propagator reference solution.

This is essential, and can be deducted from the algebraic presentation of parareal in section 2.2. Parareal allows to iteratively converge towards the solution one would have obtained using a purely sequential fine propagation over the entire time-domain interval. The iterative method is actually more computational expensive than the pure sequential method, and therefore it has no interest in a purely sequential implementation like other iterative methods. The advantage of the iterative method is that some parts of the algorithm can be solved concurrently, enabling potential speedup despite the increased computational load by the addition of more compute units.

2.4 Complexity and parallel efficiency

In this section we depart on a basic analysis of the computational complexity of the parareal algorithm. As mentioned previously, the iterative algorithm is not useful as a tool to accelerate the sequential propagation trough fast convergence. As will be shown, the computational complexity of the iterative approach is strictly larger than that of a plain sequential approach using the fine propagator. The advantage of the algorithm is that the parareal iterative approach allows some parts of the computation to be performed in parallel.

In the analysis of the computational complexity, we first recognize that the computational complexity of both the coarse and the fine propagator, regardless of the type of discretization scheme used, involves a complexity that is proportional to the number of time-steps being used. Let us define two scalar sizes $\mathcal{C}_\mathcal{F}$ and $\mathcal{C}_\mathcal{G}$ as the computational cost of performing a single step within the fine and coarse propagators. The computational complexity of a propagator integrating over an interval $\triangle T$ is then given by $\mathcal{C}_\mathcal{F}\frac{\triangle T}{\delta t}$ and $\mathcal{C}_\mathcal{G}\frac{\triangle T}{\delta T}$ respectively. But how does this relate to the total complexity of the pararela algorithm? Let us define the number of iterations k as the basic algorithm presented in algorithm 1. If we disregard the cost of correction (line 15), the total complexity with k iterations can be written as

$$(k+1)\,N\mathcal{C}_\mathcal{G}\frac{\Delta T}{\delta T} + kN\mathcal{C}_\mathcal{F}\frac{\Delta T}{\delta t}$$

But the second term can be distributed over N processors, so

$$(k+1)\,N\mathcal{C}_\mathcal{G}\frac{\Delta T}{\delta T} + k\mathcal{C}_\mathcal{F}\frac{\Delta T}{\delta t} \tag{2.11}$$

The above should be compared to the computational complexity of a purely sequential propagation using the fine operator, which would lead to a computational complexity $\frac{T_N - T_0}{\delta t}\mathcal{C}_\mathcal{F} = N\frac{\Delta T}{\delta t}\mathcal{C}_\mathcal{F}$. Assuming that communication between compute units is instant, and therefore negligible. We can estimate the speed-up, denoted ψ, as the ratio between the computational complexity of the purely sequential solution using the fine operator only, and the complexity 2.11 of the parareal algorithm. The estimated speed-up ψ then reads

$$\psi = \frac{N\frac{\Delta T}{\delta t}\mathcal{C}_\mathcal{F}}{(k+1)N\mathcal{C}_\mathcal{G}\frac{\Delta T}{\delta T} + k\mathcal{C}_\mathcal{F}\frac{\Delta T}{\delta t}} = \frac{N}{(k+1)N\frac{\mathcal{C}_\mathcal{G}}{\mathcal{C}_\mathcal{F}}\frac{\delta t}{\delta T} + k} \tag{2.12}$$

If we in addition assume that the time spend on coarse propagation is negligible compared to time spend on the fine propagation, so that we are in the limit $\frac{\mathcal{C}_\mathcal{G}}{\mathcal{C}_\mathcal{F}}\frac{\delta t}{\delta T} \to 0$, the above expression reduces to

$$\psi = \frac{N}{k} \tag{2.13}$$

Under these ideal circumstances, with convergence in $k = 1$ iterations, one would obtain perfect speed-up. Now what kind of assumptions did we make in the above derivation? First of, it was assumed that the correction time could be neglected. Second, we also assumed that communication time between compute units was negligible. Third, we assumed the computational complexity of the coarse propagator to be much smaller than that of the fine propagator. These assumptions are important to keep in mind and, as will be shown, the later is intimately coupled with the number of iterations needed for convergence. Using a similar deduction, it is readily seen that the iterative method has no use in sequential computation. Using a single compute unit, the estimated speed-up would then be

$$\psi_{N=1} = \frac{1}{(k+1)\frac{\mathcal{C}_\mathcal{G}}{\mathcal{C}_\mathcal{F}}\frac{\delta t}{\delta T} + k}$$

which is strictly less than one, since $k \geq 1$. From 2.13 we concluded that the number of iterations to convergence k pose an upper bound $\frac{1}{k}$ on the efficiency. The obvious next step is to investigate the convergence properties of the algorithm. It is a straight forward exercise to prove by induction that $U_n^n = \mathcal{F}_{\Delta T}^n u^0$, which means that the method is exact in a finite number of iterations. With $k = N$ we obtain the exact solution at the final time T_N one would obtain by using a fine propagation only. This would however not yield any speed-up since we would then have the estimate

$$\psi_{k=N} = \lim_{\frac{\mathcal{C}_\mathcal{F}}{\mathcal{C}_\mathcal{G}}\frac{\delta t}{\delta T} \to 0} \frac{N}{(k+1)N\frac{\mathcal{C}_\mathcal{G}}{\mathcal{C}_\mathcal{F}}\frac{\delta t}{\delta T} + N} = 1$$

But in practice it would lead to a slow down rather than a speed-up of the algorithm due to the time spend on correction and coarse propagation. Fortunately, the algorithm convergence much faster, and in chapter 3, a review of analytical

convergence results from the literature is presented. The iterations needed for convergence is intermediately coupled with the ratio between the speed of the coarse and the speed of the fine propagator $\frac{\mathcal{C}_\mathcal{G}}{\mathcal{C}_\mathcal{F}} \frac{\delta t}{\delta T}$. Using a fast and accurate coarse propagator will lead to convergence in fewer iterations k, but at the same time make $\frac{\mathcal{C}_\mathcal{G}}{\mathcal{C}_\mathcal{F}} \frac{\delta t}{\delta T}$ larger, thereby degrading the speed-up as can be deduced from 2.12. The convergence speed is thus coupled to the assumption on the relation between speed of coarse and speed of fine solver. The ratio $\frac{\mathcal{C}_\mathcal{F}}{\mathcal{C}_\mathcal{G}} \frac{\delta t}{\delta T}$ *can not* be made arbitrarily small since the relation is inversely proportional to the iterations k needed for convergence. This poses a challenge in obtaining speed-up and is a problem of trade-off between time spend on a fundamentally sequential part of the algorithm and number of iterations to convergence. The problem is of optimization type and a much more involved discussion on this topic is presented in chapter 4 and 5.

2.5 Summary

In this chapter the parareal algorithm was introduced along with an algebraic interpretation followed by a visual presentation of the method to give the reader an intuitive feel of the iterative procedure. From a fairly simple analysis of the computational complexity, it was proven that the upper bound on parallel efficiency scales as $1/k$. A number of fundamental results was presented in this introductory chapter, and for the purpose of clarity, the most notable conclusions are stated in bullet form below.

- The parareal algorithm is an iterative approach at speeding up the solution of an otherwise strictly sequential propagation. The algorithm is computationally more expensive and therefore has no value in a serial computation.
- The algorithm convergence towards the solution one would have obtained using the fine propagator $\mathcal{F}$ in a sequential propagation throughout the time domain.
- Parareal is an iterative algorithm, so a stopping criteria is needed. It is possible to use a sufficiently small value, such as the machine precision, comparing it to the norm of the difference between two consecutive iterations. This approach may lead to superfluous iterations as will be shown in chapter 4, and other better methods exists.
- Like the iterative conjugated gradient method, parareal is exact at a maximum number of iterations, equal to the number of intervals N. However,

the upper bound on efficiency scales as $\frac{1}{k}$, so convergence must be reached in far fewer iteration.

- The coarse propagator $\mathcal{G}_{\triangle T}$ does not need to be of same type as the fine propagator $\mathcal{F}_{\triangle T}$. We are free to choose the coarse propagator in any way we like. The choice of $\mathcal{G}_{\triangle T}$ will only effect the stability and convergence rate of the algorithm, not the accuracy of the end result.

- The algorithm favours single step methods. During each iteration, N individual initial value problems are solved. Due to the start-up issues of multi step schemes, such solvers are less suitable in this context.

- The number of parareal iterations k should be small. Ideally $k = 1$, which leads to potential perfect parallel efficiency.

- The choice of coarse operator $\mathcal{G}$ is critical in the effort to obtain high efficiency. The coarse operator $\mathcal{G}$ should be close enough in terms of accuracy to $\mathcal{F}$ so to allow convergence in few iterations, but still cheap enough so that $(k+1)\, N\mathcal{C}_{\mathcal{G}} \frac{\Delta T}{\delta T} \ll k\mathcal{C}_{\mathcal{F}} \frac{\Delta T}{\delta t}$. Throughout the report it will become clear that balancing this trade-off is the biggest challenge in obtaining speed-up using the parareal algorithm.

Chapter 3

A Survey of Present Work

The very first paper introducing the predictor-corrector form of the parareal algorithm was proposed by Baffico et. al. [4] and published in 2002. During the decade that has passed, much research have gone into establishing the properties of the algorithm. In [27] it was shown that the parareal method is type of multiple shooting method with a Jacobian matrix computed by finite difference on a coarse mesh. In addition it was shown the method could also be rewritten as a two-level multi-grid in time method of "full approximation storage type" with an unusual smoother corresponding to a "single phase in a two-color" relaxation scheme. The parareal algorithm is still very much an active field of research, with many recent contributions [9, 18, 30, 33, 54, 56]. The challenges posed by hardware trends towards increased parallelism and heterogeneity as introduced in chapter 1 have sparked a recent rise in the search for algorithms to exploit the new compute architectures. The parareal algorithm is a method of extracting parallelism in the numerical solution of initial value differential equations, and as such fits nicely into this overall trend.

During the course of the work going into the thesis at hand, much material on the topic has been read and in this chapter highlights and notable results from the literature is presented. Initially, results on the topic of convergence and stability is covered. If a method is not convergent, it is not of much use and therefore the most important results and theorems are presented. In addition, the stability criteria on the choice of coarse propagator $\mathcal{G}$ as derived in [6] and [63]

is presented. From the theorem it can be shown that under certain conditions the parareal algorithm may have stability issues.

A number of results from various papers on the scaling properties of the algorithm is presented and two sections are included on the topic of parallel efficiency of the algorithm. The first section involves a discussion on the choice of stopping criteria while in the second section we show that much efficiency can be gained by distributing the algorithm as information become available rather than taking a strictly sequential-parallel-sequential approach as presented in the basic implementation in 1. The chapter ends with a discussion on potential gaps in the theory, challenges that need to be addressed and perspectives for the parareal algorithm.

3.1 Stability and Convergence

Most of the early work published on the parareal method is either on the topic of stability and convergence, or on speed-up results of the application of parareal to various problems. Much of the initial work on stability and convergence analysis began as contributions to the 15th international conference on Domain Decomposition Methods held in Berlin 2003, participants including C. Farhat, Y. Maday, G. Bal, M. Gander among others. Here we present the most important results on the topic available in the literature. For proofs and references, the papers [6] [26] [63] [27] constitutes the bulk work on stability and convergence analysis. In addition to these papers, the stability issues that may arise when parareal is applied to hyperbolic problems is further analysed and various ways of fixing these issues are proposed in the recent work [17] and [18]. A short literature review on the topic of parareal on hyperbolic problems is given in section 6.3 before investigating the stability of the non-linear wave model, a hyperbolic type PDE, that is presented in the chapter 6.

In this section the most notable results on stability of the parareal algorithm available in the literature are presented followed by a review of convergence results. The theoretical work is typically performed on constant coefficient linear ordinary differential equations.

Stability

The stability analysis is presented as in [44] followed by a review of the results in [6]. The analysis to be presented is based on the application of parareal to a

linear system of coupled ordinary differential equations with constant coefficient. First, let us consider the linear system of M coupled equations

$$\frac{\partial}{\partial t}\bar{y} = \mathbf{A}\bar{y},\ \bar{y}(0) = \bar{y}_0,\ \ \mathbf{A} \in \mathbb{R}^{M\times M} \tag{3.1}$$

If we assume that a spectral factorization is possible, we may write the coefficient matrix as $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$ with $\mathbf{D}$ being a diagonal matrix containing the eigenvalues $\{\lambda_1, \ldots \lambda_M\}$ and $\mathbf{V}$ a matrix containing the corresponding eigenvectors of $\mathbf{A}$. The numerical approximation of the system (3.1) using a simple forward euler can then be written as

$$\bar{y}_n = \mathbf{V}\left(\mathbf{I} + \Delta T\mathbf{D}\right)^n \mathbf{V}^{-1}\bar{y}_0$$

If $|1 + \Delta T\lambda_i| = |R(z_i)| \leq 1, i = 1, \ldots, M$ with $z_i = \lambda_i \Delta T$, then the numerical scheme will be non increasing for any choice of n. A numerical scheme which results in a non-increasing approximation for a given time-step is called stable. As such, the stability analysis for the case of a simple scalar equation (3.2) is easily extended to the stability of systems of linear coupled equations, and so we move our attention to the scalar equation as as stated below

$$\frac{\partial y}{\partial t} = \lambda y,\ y(0) = y_0,\ \ \lambda \in \mathbb{C} \tag{3.2}$$

The fundamental idea in the analysis is to first construct a stability function on the form

$$U_n^k = H\left(n, k, \Delta T, r\left(\lambda\delta t\right), R\left(\lambda\delta T\right)\right) U_0^0 \tag{3.3}$$

and then derive an upper bound to H expressed only by the stability functions of the coarse and fine propagator $r\left(\lambda\delta t\right)^{\frac{\Delta T}{\delta t}}$ and $R\left(\lambda\delta T\right)^{\frac{\Delta T}{\delta T}}$, the expression is then again to be bounded by some constant for the parareal solution to be non increasing for a fixed k. As a first step in constructing the stability function H, we apply the predictor-correcter scheme (2.4) to the problem (3.2), essentially replacing the operators with stability functions to arrive at

$$U_n^k = \bar{R}U_{n-1}^k + \bar{r}U_{n-1}^{k-1} - \bar{R}U_{n-1}^{k-1} \tag{3.4}$$

Where we have adopted the short form notation $\bar{R} = R\left(\lambda\delta T\right)^{\frac{\Delta T}{\delta T}}$ and $\bar{r} = r\left(\lambda\delta t\right)^{\frac{\Delta T}{\delta t}}$ for simplicity. The recursion is solved by writing the above expression as $U_n^k = \bar{R}U_{n-1}^k + \left(\bar{r} - \bar{R}\right)U_{n-1}^{k-1}$ from which the Pascal tree is recognised so that (3.4) may be written as

$$U_n^k = \left(\sum_{i=0}^{k} \begin{pmatrix} n \\ i \end{pmatrix} \left(\bar{r} - \bar{R}\right)^i \bar{R}^{n-1}\right) U_0^0$$

Thus, stability function H for parareal applied to a linear autonomous equation can be written as

$$H = \sum_{i=0}^{k} \left(\begin{array}{c} n \\ i \end{array} \right) \left(\bar{r} - \bar{R}\right)^i \bar{R}^{n-1} \tag{3.5}$$

We wish to somehow find a criteria that will guarantee H to be bounded by a constant so that the parareal solution is non increasing given a fixed iteration number k. To derive such a bound based on the stability function of the coarse and fine propagator, the absolute value of (3.5) is expanded using basic rules of absolute value of complex numbers

$$\begin{aligned} |H| &= \left| \sum_{i=0}^{k} \left(\begin{array}{c} n \\ i \end{array} \right) \left(\bar{r} - \bar{R}\right)^i \bar{R}^{n-1} \right| \\ &\leq \sum_{i=0}^{k} \left(\begin{array}{c} n \\ i \end{array} \right) \left|\left(\bar{r} - \bar{R}\right)\right|^i \left|\bar{R}\right|^{n-1} \\ &\leq \sum_{i=0}^{n} \left(\begin{array}{c} n \\ i \end{array} \right) \left|\left(\bar{r} - \bar{R}\right)\right|^i \left|\bar{R}\right|^{n-1} \\ &= \left(\left|\bar{r} - \bar{R}\right| + \left|\bar{R}\right|\right)^n \leq C \end{aligned}$$

The extension of the above criteria to systems of the form (3.1) is straight forward. Stability is achieved if there exists a constant C so that

$$\sup_{1 \leq n \leq N} \sup_{1 \leq k \leq N} \left| H\left(n, k, \Delta T, r\left(\lambda_j \delta t\right), R\left(\lambda_j \delta T\right)\right)\right| \leq C, \ \forall \lambda_j \, , \, j = 1, \ldots, M. \tag{3.6}$$

By (3.6), theorem 1 is proposed in [44]. The proof is omitted. The theorem states that given that the the absolute value of both the coarse and fine operator stability functions are less than or equal one, then there exists some constant so that the parareal solution is bounded for a fixed number of iterations k, as detailed in the proof omitted here, this bound can be written as $U_N^k \leq 2^{k_0} k_0 N^{k_0} \left|\bar{R}\right|^{N-k_0} U_0^0$ for all $k \leq k_0$.

Theorem 1 Assume we want to solve the autonomous differential equation

$$\frac{\partial y(t)}{\partial t} = \lambda y\left(t\right), \ y\left(0\right) = y_0, \ \ \lambda \in \mathbb{C}$$

using the parareal algorithm. Then stability is guaranteed for a fixed number of iterations as long as $\left|\bar{r}\left(z\right)\right| \leq 1$ and $\left|\bar{R}\left(z\right)\right| \leq 1$.

The stability guaranteed by theorem 1, which is easily obtained, unfortunately has been shown to not be sufficient and something stronger is needed. The behaviour that serves as the motivation to introduce another stability requirement

can be seen in the figure (4.7), section 4.5. In these figures the error as a function of iteration of the parareal solution of the two-dimensional heat diffusion equation discretized by the ADI method for both the coarse and fine solver is presented. The error of the parareal solution is seen, for some discretizations, to decay rapidly during the first few initial iterations, but unfortunately the error is seen to then increase again for larger values of k before ultimately converging towards the error of the fine propagator solution. Theorem 1 is valid as the solution is bounded for all k and eventually converges, but it is certainly not practical as the number of iterations for convergence makes the associated computational cost degrade any hope of parallel speed-up.

To meet this issue, [44] introduce a strong stability definition saying that when the parareal algorithm is stable for *all* possible number of time slots N and *all* number of iterations $1 \leq k \leq N$, we say that we have strong stability. When (3.6) is bounded by 1, we are guaranteed this to be the case. For the case of strong stability, we need to handle λ being complex or real separately.

Theorem 2 Assume we want to solve the autonomous differential equation

$$\frac{\partial y}{\partial t} = \lambda y,\ y(0) = y_0,\ 0 > \lambda \in \mathbb{R}$$

and that $-1 \leq r, R \leq 1$ where $r = r(\lambda \delta t)$ is the stability function for the fine propagator $\mathcal{F}$ using time-step δt and $R = R(\lambda \delta T)$ is the stability function for the coarse propagator $\mathcal{G}$ using time-step δT. We then have strong stability as long as

$$\frac{\bar{r}-1}{2} \leq \bar{R} \leq \frac{\bar{r}+1}{2}$$

where $\bar{r} = r(\lambda \delta t)^{\frac{\Delta T}{\delta t}}$ and $\bar{R} = R(\lambda \delta T)^{\frac{\Delta T}{\delta T}}$.

the proof follows straight from (3.6) when assuming that both r and R are real. It is not obvious which numerical solvers satisfy this stability condition. [44] provide a corollary to ease the interpretation. Assume that $z \to -\infty$ and that the fine propagator is infinitely close to the exact solution, meaning that $\bar{r} = 0$, strong stability for the parareal algorithm is then guaranteed if $R_\infty = \lim_{z \to -\infty} |R(z)| < \frac{1}{2}$, which means that all L-stable schemes fulfils theorem 2. Theorem 2 is derived assuming λ to be real. How do we guarantee strong stability for a constant coefficient ODE with imaginary eigenvalues? The first step is to write the now complex stability functions of the propagators as

$$\bar{r} = e^{x_{\bar{r}}} e^{\mathbf{i}\theta}$$
$$\bar{R} = e^{x_{\bar{R}}} e^{\mathbf{i}(\theta+\varepsilon)}$$

where θ is the argument, and ε the phase difference between $\bar{r}$ and $\bar{R}$. Both $x_{\bar{r}}$ and $x_{\bar{R}}$ are non-positive as both G and F must be stable meaning that $\bar{r}$ and $\bar{R}$ lie on or inside the complex unit circle i.e., $0 \leq \mathrm{e}^{x_{\bar{r}}}, \mathrm{e}^{x_{\bar{R}}} \leq 1$. In the analysis [44] they initially consider the case of pure imaginary eigenvalues. In this review however, we go straight to general case. The stability restriction (3.6) can then be rewritten

$$\begin{aligned}
&\left|\mathrm{e}^{x_{\bar{r}}}\mathrm{e}^{\mathbf{i}\theta} - \mathrm{e}^{x_{\bar{R}}}\mathrm{e}^{\mathbf{i}(\theta+\varepsilon)}\right| + \left|\mathrm{e}^{x_{\bar{R}}}\mathrm{e}^{\mathbf{i}(\theta+\varepsilon)}\right| \\
&\qquad = \left|\mathrm{e}^{x_{\bar{r}}} - \mathrm{e}^{x_{\bar{R}}}\mathrm{e}^{\mathbf{i}\varepsilon}\right| + \mathrm{e}^{x_{\bar{R}}} \\
&\qquad = \sqrt{\left(\mathrm{e}^{x_{\bar{r}}} - \mathrm{e}^{x_{\bar{R}}}\cos(\varepsilon)\right)^2 - \left(\mathbf{i}\mathrm{e}^{x_{\bar{R}}}\sin(\varepsilon)\right)^2} + \mathrm{e}^{x_{\bar{R}}} \\
&\qquad = \sqrt{\mathrm{e}^{2x_{\bar{r}}} + \mathrm{e}^{2x_{\bar{R}}}\cos^2(\varepsilon) - 2\mathrm{e}^{x_{\bar{r}}}\mathrm{e}^{x_{\bar{R}}}\cos(\varepsilon) + \mathrm{e}^{2x_{\bar{R}}}\sin^2(\varepsilon)} + \mathrm{e}^{x_{\bar{R}}} \\
&\qquad = \sqrt{\mathrm{e}^{2x_{\bar{r}}} + \mathrm{e}^{2x_{\bar{R}}} - 2\mathrm{e}^{x_{\bar{r}}}\mathrm{e}^{x_{\bar{R}}}\cos(\varepsilon)} + \mathrm{e}^{x_{\bar{R}}} \leq 1 \\
\Rightarrow\ &\mathrm{e}^{x_{\bar{R}}} \leq \frac{1}{2}\frac{1-\mathrm{e}^{2x_{\bar{r}}}}{1-\mathrm{e}^{x_{\bar{r}}}\cos(\varepsilon)}
\end{aligned}$$

again requiring the function to be bounded by 1. From the above derivation, theorem 3 is stated.

Theorem 3 Assume we want to solve the autonomous differential equation $y' = \lambda y$, where $\lambda \in \mathbb{C}$ and $Re(\lambda) \leq 0$, using the parareal algorithm. Assume that also both $\mathcal{G}$ and $\mathcal{F}$ are stable for the chosen scheme and time-step. The parareal algorithm is then guaranteed to be stable if

$$\mathrm{e}^{x_{\bar{R}}} \leq \frac{1}{2}\frac{1-\mathrm{e}^{2x_{\bar{r}}}}{1-\mathrm{e}^{x_{\bar{r}}}\cos(\varepsilon)} \tag{3.7}$$

where $\bar{r} = \mathrm{e}^{x_{\bar{r}}}\mathrm{e}^{\mathbf{i}\theta}$ and $\bar{R} = \mathrm{e}^{x_{\bar{R}}}\mathrm{e}^{\mathbf{i}(\theta+\varepsilon)}$ are the values of the stability function for $\mathcal{G}_{\Delta T}$ and $\mathcal{F}_{\Delta T}$.

It is important to stress that the theorems 2 and 3 are sufficient but not necessary for strong stability of the algorithm and particularly for smaller N and smaller K the restriction will be less severe. In the analysis of the case of pure imaginary eigenvalues using a symplectic solver for the fine propagator the criteria for strong stability reduces to $\sqrt{1+\mathrm{e}^{2x_{\bar{R}}} - 2\mathrm{e}^{x_{\bar{R}}}\cos(\varepsilon)} + \mathrm{e}^{x_{\bar{R}}} \leq 1 \Rightarrow \cos(\varepsilon) \leq 1$. Which is only true if $\varepsilon = 2a\pi$, $a \in \mathbb{Z}$. So if the coarse propagator $\mathcal{G}_{\Delta T}$ is out of phase with the fine propagator $\mathcal{F}_{\Delta T}$, then the predictor-corrector scheme can not be guaranteed to be strongly stable.

No numerical schemes has been found that guarantees the parareal algorithm to be stable for all possible eigenvalues and all possible number of subdomains

and number of iterations in time. In the case of pure imaginary eigenvalues it is particularly hard to guarantee strong stability which suggests that the numerical solution of some hyperbolic problems, and convection-diffusion problems with highly dominant convection may be unstable using the parareal algorithm.

The stability of the algorithm has been further studied by [6]. Here an abstract result is presented for a general partial differential equation, showing convergence of the algorithm provided that certain regularity conditions on the solution and initial condition is satisfied. The analysis is extended in the simplified case of linear partial differential equations with constant coefficients so to establish a more refined estimates for the stability of the algorithm, showing that the parareal algorithm is unconditionally stable for most discretizations of parabolic equations, but no so for hyperbolic equations.

These results are consistent with many observations available in the literature describing how instabilities can arise during long time integration of hyperbolic systems [14, 23, 25, 49]. Progress has since been made on the identification of these stability issues and in [18] the problem is identified as being deeply linked to the regularity of the solution as indicated in [6]. As it turns out, the problem is not really on the type or order of the equations, but on the regularity of the solution including the regularity of the initial condition. These results allows a better understanding of previous work, such as in [15] where stability problem are observed for a second-order hyperbolic problems but not for the parabolic and the first-order hyperbolic problem tested.

The instability that may arise when the plain parareal algorithm is applied to hyperbolic problems, is a consequence of the algorithm not conserving invariants, potentially leading to instabilities of the algorithm before convergence due to the lack of regularity that the solution develops over time.

A natural approach at handling this issues would be to use sympletic or symmetric integrators as both fine and coarse propagators. Unfortunately this is not viable as it can be shown that the parareal algorithm will not conserve these geometric properties of the underlying propagators. Various methods of making parareal conserve invariants have been proposed. In [24] and [15] a fairly involved correction procedure re-utilizing previously computed information is presented in order to improve the performance of the solution of a hyperbolic problem. A simpler strategy based on a symmetrization of the parareal in time algorithm has been proposed in [17]. The approach leading to a new multi-step scheme is not sufficient though, as resonance are artificially introduced that prevent the symmetric variant of behaving well.

In [18] a method based on a projecting of the solution during each iteration over an energy manifold defined on the fly is shown to stabilize the parareal

method applied to a linear wave equation and the non linear Burgers equation. However, results are ambiguous in the sense that even though the projection may make the algorithm converge for all N and k, the number of iterations needed for convergence in the examples are fairly high, and as pointed out in the concluding remarks they are high enough to pose a challenge in terms of attainable parallel efficiency.

Convergence

In the parareal method we introduce a fine operator $\mathcal{F}_{\Delta T}$ and a coarse operator $\mathcal{G}_{\Delta T}$ so to construct an iterative algorithm that converge towards the solution one would otherwise have obtained applying the fine propagator in a purely sequential manner. It is naturally of great interest to determine the rate of convergence. In the first publication on parareal [41], the accuracy of U_n^k was analysed for a fixed number of iterations as the interval length ΔT become small. The analysis was performed on a scalar linear model problem (3.8), which leads to the proposition 3.1.1.

$$\frac{\partial u}{\partial t} = \lambda u,\ u(0) = u_0,\ t \in [0, T]\ \ \lambda \in \mathbb{C} \tag{3.8}$$

Proposition 3.1.1 Let $\Delta T = T/N$, $T_n = n\triangle T$ for $n = 0, 1, \ldots, N$ so that $T_0 = 0$. Consider the initial value problem (3.8), with $\lambda \in \mathbb{R}$. Let $\mathcal{F}_{\Delta T} U_n^k$ be the exact solution at T_{n+1} of (3.8) with initial condition $u(t_n) = U_n^k$, and let $\mathcal{G}_{\Delta T} U_n^k$ be the corresponding backward Euler approximation with time-step ΔT. Then,

$$\max_{1 \le n \le N} \left| u(t_n) - U_n^k \right| \le C_k \Delta T^{k+1}$$

The implication of the proposition is essential as for a fixed iteration step k, the algorithm using the first order accurate backward euler method now behaves in ΔT like and $\mathcal{O}\left(\Delta T^{k+1}\right)$ method! The proposition was later extended to higher order and more general integration schemes in [6] [7], and in those papers it was shown that parareal is a method of order $\mathcal{O}\left(\Delta T^{m(k+1)}\right)$ when a method of order m is used as the coarse propagator $\mathcal{G}_{\Delta T}$ combined with an exact, or sufficiently accurate solver as the fine propagator $\mathcal{F}_{\Delta T}$.

The proposition 3.1.1 is only valid for fixed k as the constant C_k can be shown to grow with k. In [27] the same scalar linear model was further analysed, now instead assuming fixed ΔT and studying the algorithms behaviour as k becomes large. The analysis leads to new convergence results in the form of a super-linear

error bound on bounded time intervals, and linear convergence for unbounded time intervals. These results are then extended to model PDEs in the form of the heat equation and the pure advection equation.

3.2 Strong and Weak Scaling

It is of interest to know how the performance of the parareal algorithm scales with the number of compute units used to accelerate the computation of some evolution problem. There are two basic ways to measure the parallel performance of a given application, these are referred to as strong and weak scaling. When strong scaling is of interest, we measure how the solution time for a fixed problem scales with the number of compute units used. In the case of measuring weak scaling, the problem size assigned to each compute unit is held constant and additional units are used to solve a larger total problem. If the compute wall-time stays constant as more work and compute units are added to the computation, then we have obtained perfect weak scaling of the algorithm.

Typically it is harder to achieve good strong-scaling performance since the communication overhead for most algorithms increase in proportion to the number of compute units used. This is also the case for the classical spacial domain decomposition methods. When using domain decomposition to accelerate the solution of some partial differential equations, it is usually possible, in a purely algorithmic sense, to obtain ideal speed-up independent of problem size. But in a practical implementations, the communication time between compute units will degrade obtained parallel speed-up and effect the scaling properties.

As the ratio between compute to communication is often the limiting factor in parallel efficiency for this class of methods, the scaling properties are dependent on the problem size and the number of compute units applied. For very small problems or very many compute units, communication time degrade parallel speed-up so that linear strong or weak scaling should not be expected. In the limit of large problems, or very few compute units, one would on the other hand expect to observe linear behaviour when measuring strong and weak scaling for these methods. This is in the limits, actual measurements will usually be somewhere in between.

So how does these general considerations relate to parareal? In parareal, each compute unit will be solving an independent initial value problems over one of the intervals in the time decomposition. Thus, the number of time intervals is the maximum number of cores that can be used to accelerate the computation. If we where to disregard communication time and correction time and in addition

neglect the time used by the sequential coarse propagation, the parallel efficiency scales as $1/k$ as shown in chapter 2, since the entire problem will be solved k times with the fine propagator. Potentially, the number of intervals used in the scheme can effect the number of iterations needed for the algorithm to convergence, and thereby effect the scalability.

In addition, there exists various methods of distributing parallel work, and the number of intervals used will certainly effect the efficiency of the distribution. Finally, as with other methods of introducing concurrency in computations, the ratio between compute and communication will have an effect on the parallel efficiency as discussed above. As many things contribute to the parallel efficiency and all of them are expected to be dependent on the number of intervals, and thus compute units, the scalability of the algorithm is not easily analysed.

Here we refer to a few results from the literature regarding the purely algorithmic scalability of the scheme, that is, how the number of iterations needed for convergence scales with the number of intervals used, given that everything else is fixed. How the choice of distribution model for the parallel work relate to the parallel efficiency is covered in section 3.3

In [48] the scaling of the parareal algorithm applied to a two dimensional heat diffusion equation was investigated. In the paper by Maday and Turinici they obtained perfect scalability using a first order in time discretization and coupled with classical domain decomposition, with a ratio $r = \delta t/\delta T$ kept fixed and each time step length was inversely proportional to the number of processors. Essentially corresponding to weak scaling as the discretization becomes finer as compute units are added.

In [2] the same problem was analysed in the context of demonstrating the efficiency of a novel work distribution model. Here a second order accurate ADI method was used as the coarse and fine propagator. In [2], the ratio $r = \delta t/\delta T$ is also kept fixed, but the number of fine and coarse time-steps per interval ΔT decrease as the number of compute units increase as opposed to the investigation in [48] where the number of coarse and fine-steps per interval did not change. The results presented in [2] is thus of strong-scaling type since the problem is held constant while more compute units are added. In [2] they found that, for this particular problem, the number of iterations needed for convergence would increase as more compute units where added.

In [2] the scalability of the algorithm was also tested with a fixed coarse to fine propagator ratio $r = \delta t/\delta T$ and a fixed interval length ΔT, so that the number of compute units determine the time interval to be integrated. This is comparable to weak scaling as the problem size per compute units is fixed. [2] found perfect scalability in the tested time range.

Similar scalability test can be found in [9, 23, 28, 32, 54]. Typically the results presented are not on the scalability of the convergence rate of the algorithm specifically, but timings of actual implementation, including communication and distribution. In the section to follow, the impact of efficient distribution of parallel work is considered.

3.3 Distribution of parallel work

In the previous section we discussed how changing the number of intervals used in the algorithm may change the number of iterations needed for convergence and thereby affect the parallel efficiency of the scheme. As will be shown, another potential mayor influence on the parallel efficiency is affiliated with how the parallelism exposed by the parareal algorithm is exploited in an actual implementation.

The parareal algorithm originally received a lot of interest because analysis on the test problem shows that one can expect fast convergence of the algorithm towards the sequential fine propagator solution with a scheme that allows to be solved concurrently. Achieving good performance in practice has proved to be challenging though, due to the fundamental trade-off between the reduction of time required for an inherently sequential part of the algorithm and an increase in the number of iterations needed for convergence [43]. This inherent limitation in the algorithm has motived the search for alternative formulations such as the one presented in [61] where they attempt to improve the algorithm using the relationship between old and new simulations. The fundamental trade-off still exists though, and the topic of this section is methods of minimising the loss trough an effective distribution of parallel work.

This pseudo code for a simple implementation of the parareal algorithm as presented 2.1 can be interpreted as in the schematics in figure 3.1. Initially, a purely sequential coarse propagation is made throughout the problem in order to generate the initial conditions for the N IVPs, we call this iteration zero. Following iteration zero, the N IVPs can be solved in parallel and this followed by a purely sequential predictor-corrector method then constitutes iteration 1 and so forth.

As apparent in figure 3.1, we have a purely sequential part of the algorithm, and a part that can be solved in parallel. In the continuum of infinitely fine discretizations we are able to apply infinitely many intervals ΔT (and thus compute units). The purely sequential propagation however will take a fixed amount of time T, not influenced by the number of intervals applied. This

leads us to Amdahl's law. It seems that we have a program consisting of a sequential and a parallel part, and this sets a natural limit to the speed-up that can be obtained. Let us define $T_{\mathcal{F}}$ as the computational time consumed by one compute unit by applying the fine operator $\mathcal{F}_{\Delta T}$, in addition we define $T_{\mathcal{G}}$ as the computational time consumed by a single compute unit when applying the coarse operator $\mathcal{G}_{\Delta T}$. Neglecting the time spend on correcting, and the communication time between compute units, we can write the wall-time spend by the parareal algorithm given convergence in k iterations with N compute units as $NT_G + k\,(NT_G + T_F)$. The wall-time used applying the fine operator in a purely sequential is NT_F which leads us to the speed-up estimate

$$\Psi_{simple} = \frac{NT_F}{NT_G + k\,(NT_G + T_F)} \tag{3.9a}$$

$$= \frac{T_F}{T_G\,(k+1) + \frac{k}{N} T_F} \tag{3.9b}$$

$$= \frac{1}{\frac{T_G}{T_F}\,(k+1) + \frac{k}{N}} \tag{3.9c}$$

In the limit of infinitely many compute units, disregarding how the algorithm scales with N, we are left with a speed-up estimate $\frac{T_F}{T_G}\frac{1}{(k+1)}$. Thus, using such a simple approach at distributing creates an upper bound on the attainable speed-up given by the ratio between T_F and T_G. In correspondence with the simple computational complexity analysis in section 2.4 we denote the ratio R as $R = \frac{T_F}{T_G}$ with $R^{-1} = r$ and note that these ratios are essential in that they pose an upper bound on obtainable speed-up.

The figure 3.1 is drawn to scale using a ratio $R = 6$, with $N = 6$ intervals. Notice how five of the compute units are sitting idle for more than half the walltime. If the sequential part and the parallel part of the algorithm where truly separated, there wouldn't be much we could do about this except try to maximize R, that is minimizing the time used by the coarse propagator. Fortunately, interdependencies that allow for more efficient distribution of parallel work exists.

First we need to realise that a given fine propagation $\mathcal{F}_{\Delta T} U_{i+1}^{k+1}$ can be started as soon as the U_{i+1}^{k+1} parareal solution has been calculated using the information $\mathcal{G}_{\Delta T} U_i^{k+1}$, $\mathcal{F}_{\Delta T} U_i^k$ and $\mathcal{G}_{\Delta T} U_i^k$. From this arise the possibility of scheduling the work in a way so to minimize the idle time and thereby improve parallel efficiency.

This opportunity for improving the performance of the algorithm has been recognized by quite a few authors and several ways of distributing the parallel work has been proposed in the literature such as a pipelined model [50], task-scheduling [2] and an event-driven distribution [9].

The details of the different approaches is left for the interested reader to investigate. Here we only briefly mention the task-scheduling approach as proposed by [2]. Aubanel initially presents a manager-worker model where the manager is in charge of the coarse propagation and the corrections while the fine propagation is left for worker compute units. As initial conditions at time decomposition boundaries are ready they are shipped to workers that perform the fine propagation and return the final state to the manager, the manager then handles all corrections. The scheme is efficient at distributing the work but unfortunately memory consumption on the manager scales linearly with the number of intervals used as the manager needs to save the states of each interval. The memory consumption on the manager compute unit may pose a challenge for large problems using many compute units.

To counter this issue, [2] propose a fully distributed task scheduling model as presented in figurer 3.2 and later implemented on a large scale in chapter 7. Neglecting communication and correction time, the speed-up of this method can be written as

$$\Psi_{dist} = \frac{N}{N\frac{T_G}{T_F} + k\left(1 + \frac{T_G}{T_F}\right)} \tag{3.10}$$

Comparing this expression with (3.9) we notice something interesting. In the limit of N going to infinity with everything else fixed, the speed-up of the distributed model is only limited by the relation between the speed of the coarse and fine propagator. This means that with an effective distribution model we gain the potential to counter the $1/k$ loss in efficiency if sufficiently many compute units are available, to reach a maximum speed-up only bounded by the coarse propagator speed relative to the fine propagator speed.

In the context of the previous section on the topic of scalability, how can we expect the inclusion of distribution of parallel work to effect scalability? To answer this question, consider the expressions for parallel speed-up (3.10) and (3.9), estimating speed-up for different distribution models. Say we are given some fixed ratio R and iteration to convergence k for some problem, and say that we assume that increasing the number of intervals N will neither decrease or increase iterations needed for convergence. Now, by removing N in the denominator of the expressions, we have expressions for parallel efficiency as a function of number of intervals N given some fixed problem and discretization. We then notice how parallel efficiency will decrease for increasing number of intervals N in both the case of strong and weak scaling. Thus, from this fairly simple analysis we can conclude that it is not possible in a practical implementation of the parareal algorithm to obtain perfect linear scaling for *all* N in both the weak and strong scaling case.

As documented in [2,9,50] the use of an effective distribution model is paramount

in obtaining good parallel speed-up for real problems. This is unfortunate, as in the ideal world we would like the coarse propagator to simply be so fast that we need not care about idle time of compute units. Unfortunately such a "super fast coarse propagator" strategy is not viable as experience show that it typically degrades the convergence rate to a level that makes the upper bound on parallel efficiency unacceptable low.

One of the otherwise attractive features of parareal is that the very basic implementation as presented in the pseudo code in section 2.1 is simple to implement and easily wrapped around previously existing problem discretization. With the addition of effective parallel distribution models with elaborate methods of scheduling work, the implementation complexity of parareal unfortunately increase rapidly. The general structure of wrapping around existing propagators still exists though.

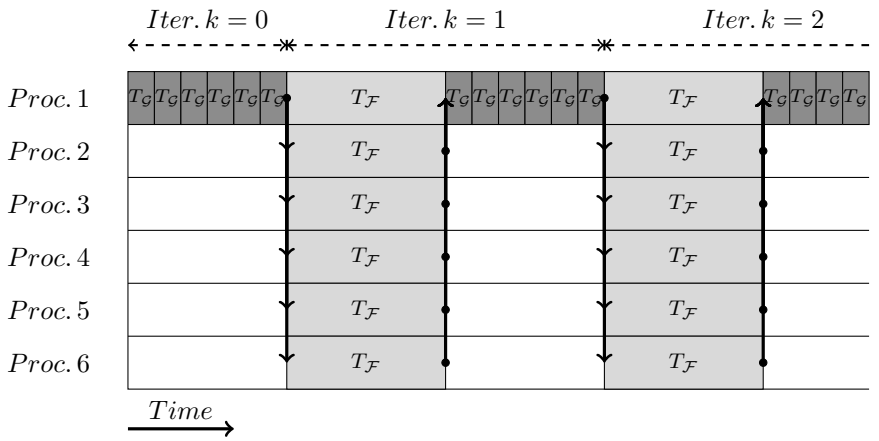


Figure 3.1: The simplest possible implementation of the parareal algorithm for the accelerated solution of an initial value problem. White space indicate compute unit idle time, dark grey indicate coarse propagation, light gray indicate fine propagation.

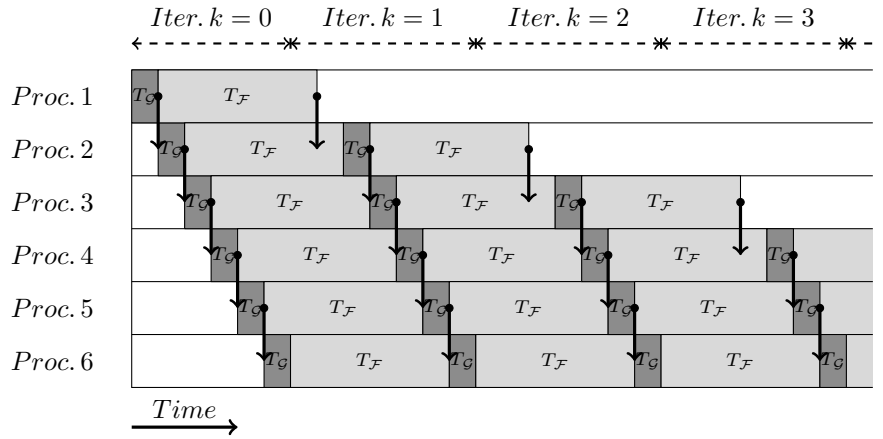


Figure 3.2: A task-based fully distributed distribution model as proposed by [2]. White space indicate compute unit idle time, dark grey indicate coarse propagation, light gray indicate fine propagation. This model does not rely on a manager to keep track of the computation as it involves, but rather need a rank zero machine which initialise the computation cascading down trough all available compute units.

3.4 Stopping criteria

The parareal method is an iterative method and so a stopping criteria is needed. It is a trivial task to show that the parareal algorithm converges to the accuracy of the fine operator within at most $k = N$ iterations, but clearly, iterating N times is not desirable as this would lead to no speed-up at all as shown in chapter 2. In the previous chapter we showed how the upper bound on parallel efficiency scales as $1/k$, so ideally we would hope for convergence in $k = 1$ iterations as this would lead to potential ideal speed-up. With the rapid decline of parallel efficiency as a function of iterations needed for convergence k, it is imperative that the iterative method stop as soon as convergence is reached.

But when are we satisfied with the parareal solution to the extend that it would be appropriate to say that the algorithm has converged? The method converges towards the solution one would otherwise have obtained applying the fine propagator $\mathcal{F}_{\Delta T}$ in a purely sequential fashion. The numerical fine propagator only solution is also subject to an error with respect to the true solution of whatever problem the algorithm is applied to, so in addition to the difference between the parareal algorithm and the fine solution, we also have an error on the fine solution itself. One could define the point at which the solution is converged in various ways, but an obvious choice would be that when the difference between the parareal solution and the fine solution is less than the difference between the fine solution and the true solution, then the error on the parareal solution is of the same order of magnitude as the error of the fine only solution, and then we say that the algorithm has converged. See figure 2.3 for a schematic drawing of the convergence procedure of parareal applied to an ODE IVP.

The challenge in determining when the algorithm has converged to acceptable accuracy lies in the unfortunate fact that we do not know the error of the parareal solution nor the error on the fine propagation only solution. For the sole purpose of testing the properties of the algorithm, one could initiate the test by first calculating the fine solution and a super fine solution, the later to use as an analytical solution, to the problem and use these for reference and comparison. However this has little practical relevance as why would anyone be interested in accelerating the computation of something we already know the solution of.

Various methods of a practical stopping criteria has been applied and proposed in the literature. A simple, but blind criterion would be to stop the iterations whenever a norm (any vector norm will do) of the difference between two consecutive parareal iterations falls below some prescribed threshold value. This approach is an often used method of stopping the algorithm, as many papers published on the parareal algorithm are with emphasis on identifying specific

properties of the algorithm rather than finding an optimal stopping criteria, examples include [55] [20] [30] [54] among many others. But as mentioned, the criteria is blind and one would need to do some tuning work in order to make sure that enough iterations are performed so that convergence is achieved, but at the same time that not to many iterations so the parallel efficiency is degraded beyond what is necessary. To be absolutely sure that convergence has been achieved, one could use the machine precision as a threshold value. However, the machine precision is certainly a lot smaller than the difference between the fine only and the true solution, and as such, one can be fairly sure that superfluous iterations, not adding any accuracy will be made, thus decreasing the parallel efficiency of the algorithm.

Another similar criterion based on the value of the maximum jump between the new prediction and the fine propagation of old parareal solution proposed in [64], this approach however is also blind in the sense that it would require tuning efforts before really being applicable to a given problem.

Despite the importance of a good stopping criteria for the parareal algorithm, the literature on the topic is surprisingly scarce. In [39] a more generally applicable stopping criteria is proposed. In the paper a method to determine convergence based on information from the fine propagation is presented and the method is tested using both the classical parareal as well as parareal in conjunction with time-adaptive integrators on a non-linear problem. The criteria used to determine convergence is stated in proposition 3.4.1

Proposition 3.4.1 Let $\mathcal{J}_n^k = \mathcal{F}_{\Delta T}\left(U_{n-1}^k\right) - U_n^k$ denote the jump, i.e., the difference between the solution provided by the fine integrator at T_n of the $k-th$ iteration and the parareal solution at same time and iteration. The parareal algorithm reaches the maximum accuracy that the fine grid solver can provide if the total error at iteration k

$$e_n^k = U\left(T_n\right) - U_n^k$$

has the same behaviour as the global error of the fine integrator for the subinterval $[T_{n-1}, T_n]$, i.e., e_n^k is $\mathcal{O}\left(\Delta T^p\right)$, where Δt is the time step of the fine integrator. This happens when

$$\left\|\mathcal{J}_n^k\right\| \leq \theta \cdot \left\|\mathcal{E}_{\mathcal{F},n}^k\right\| \text{ , } \theta \leq 1$$

where by $\mathcal{E}_{\mathcal{F},n}^k$ we denote the global error of the fine integrator on the current subinterval $[T_{n-1}, T_n]$.

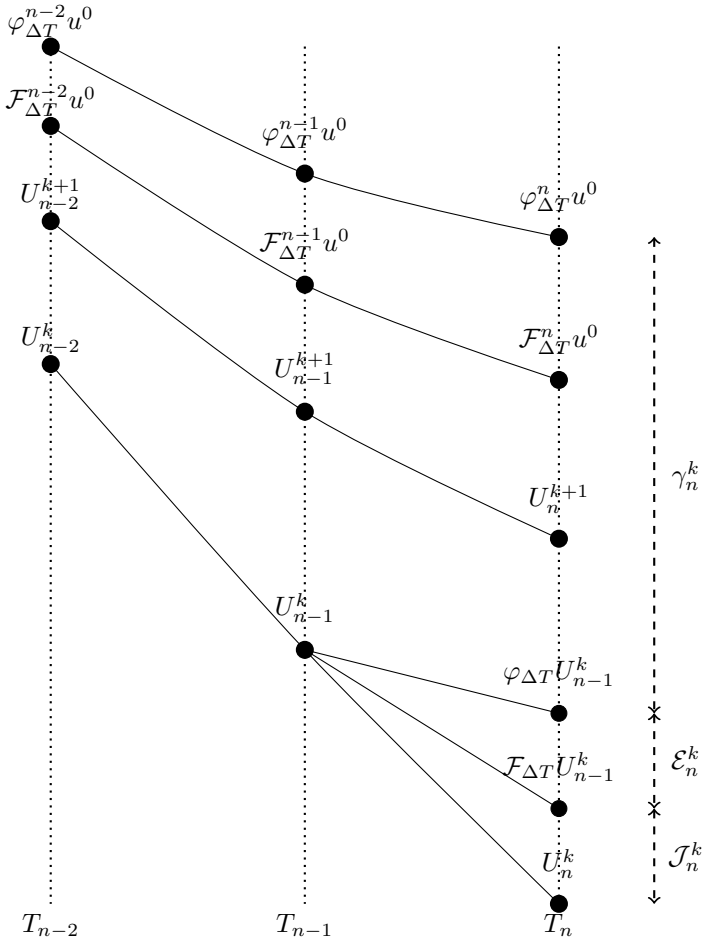


Figure 3.3: Schematics of the iterative solution by parareal. The exact propagator $\varphi_{\triangle T}$ is introduced as it is used in the proof of proposition 3.4.1. Applying the operator to the initial condition u^0 yields $\varphi_{\triangle T}^n u^0 = u(T_n)$. Make sure to distinguish between the dual use of n as both index and exponent.

The proof of proposition 3.4.1, involves the definition as visualized in the schematics in figure 3.3, we have that

$$e_n^k = \gamma_n^k + \mathcal{E}_n^k + \mathcal{J}_n^k \tag{3.11a}$$
$$\gamma_n^k = u\left(T_n\right) - \varphi_{\triangle T}\left(U_{n-1}^k\right) \tag{3.11b}$$
$$\mathcal{E}_{\mathcal{F},n}^k = \varphi_{\triangle T}\left(U_{n-1}^k\right) - \mathcal{F}_{\triangle T} U_{n-1}^k \tag{3.11c}$$
$$\mathcal{J}_n^k = \mathcal{F}_{\triangle T} U_{n-1}^k - U_n^k \tag{3.11d}$$

The first step of the proof involves a Taylor expansion around U_{n-1}^k of the function $\varphi_{\triangle T}\left(U\right)$, the result is used to generate an upper bound on the error as defined in (3.11a), which is then further expanded using basic operator norm definitions. The final step in the proof is recognizing that when convergence is reached ($k = N$), we have that $\mathcal{J}_n^k \approx 0$, which when combined with the error bound leads to the statement in proposition 3.4.1.

To apply the stopping method it is necessary to use a numerical integrator with an embedded error estimator so to estimate $\left\|\mathcal{E}_{\mathcal{F},n}^k\right\|$. The requirement makes Runge-Kutta integrators an obvious choice as there is plenty of high-order embedded methods to choose from here. The method is tested on the viscous burgers equation, using an SDIRK integrator, and was shown to be effective. It is important to note that the condition $\left\|\mathcal{J}_n^k\right\| \leq \left\|\mathcal{E}_{\mathcal{F},n}^k\right\|$ is sufficient, but not necessary, for converge and as such, the estimate is conservative in that convergence can happen before the criteria is met. The factor $0 < \theta \leq 1$ is included as a parameter to manually tune the criteria and make it less conservative if needed.

One unfortunate detail of the convergence criteria in proposition 3.4.1 is that the fine propagation on the parareal solution of the current iteration is needed to evaluate the criteria. We define iterations as in the pseudo code presented in section 2.1, so that k is the number of times that the fine propagator is applied in parallel over N intervals. This is an issue as the next iteration is needed to evaluate if the current iteration has converged, so even under ideal circumstances, proposition 3.4.1 only catch convergence one iteration after convergence has occurred.

For practical purposes, using the mentioned criteria to establish convergence in k iterations, means that $k + 1$ iterations are to be performed. Since the upper bound on efficiency scales as $1/k$, this would likely effect the result of the optimization problem of the accuracy choice of the coarse integrator $\mathcal{G}$.

In the case that one is interested in the solution over the entire time domain at the fine discretization level, the added iteration is not an issue as an additional parallel fine integration is needed to propagate information from the final

parareal solution over the span of the interior time domain.

The limited literature on the topic of an effective convergence criteria for the parareal algorithm is a challenge in terms of practical usability. It would be ideal to somehow establish a relation that guarantees convergence based on the jump between the coarse and fine propagation of the previous iteration with respect to the parareal solution of the current iteration, again one could assume the usage of an embedded error estimator on the integrators. Such a result would be a nice contribution to the literature on the parareal algorithm.

3.5 Various ways to reduce the coarse propagation

The key element in the parareal algorithm is the coarse propagator $\mathcal{G}$, it is this operator along with the predictor-corrector iterative method that constitutes the parareal scheme to accelerate, by the addition of multiple compute units, the otherwise sequential propagation of $\mathcal{F}$.

A natural straight forward way of constructing the coarse operator is to use the same numerical method on the same equation as the fine propagator $\mathcal{F}$, but simply take longer time steps so to be able to integrate an interval ΔT faster. This approach was taking in the introductory example in section 2.3 where a simple nonlinear ordinary differential equation was solved.

Are we allowed to be more creative in the construction of the coarse operator? Yes, yes we are! The only requirement by the fundamental analysis in 3.1 is that the operator must at least be stable if we are to hope for the parareal solution to be strongly stable. This opens up for a vast span of potential combinations and approaches at constructing the coarse propagator. A popular method in the literature is to reduce the physics of the problem, i.e. removing physical effects from the discretization that in any case is under-sampled by longer time-steps, thus reducing the computational load of the coarse propagator without diminishing its accuracy and thereby the rate of convergence.

The first such example of using simplified physics on the coarse propagator was with application to molecular dynamics simulation in [4] and a similar method was also used on [42] on another ode system simulating chemical reactions involving many scales in time, showing the method to be fairly robust. The method of reduced physics has also been applied to time dependent partial differential equations [24].

Another approach at constructing the coarse operator $\mathcal{G}$ for the parareal solution of partial differential equations is to use a coarse spacial discretization for the coarse operator. This has been done in [25] among others. Naturally one need to be aware that an operator to move from one space to another is needed. There are several good reasons to take such an approach. Often when integrating partial differential equations, we have a CFL(Courant-Friedrichs-Lewy) like restriction for stability which limit the time-step length of the coarse propagator. This limitation is typical when explicit schemes are used. If possible we would like to keep the coarse propagator explicit as implicit methods tends to be more computational expensive and we are in the search of minimizing the computational load of the coarse propagator. In addition to circumventing any CFL conditions, reducing both the spacial and temporal resolution for the coarse operator may potentially be a better trade-off in terms of speed/accuracy than reducing the temporal resolution alone. The take home message of this section is that a lot of creativity can be applied in constructing the coarse propagator as the parareal scheme itself does not pose limitations on the coarse operator other than it to be stable.

3.6 Combination with Domain Decomposition

A now classical approach for the solution of partial differential equations on a large scale is the application of domain decomposition methods. Domain decomposition is a class of methods to solve boundary value problems by splitting it into smaller boundary value problems on subdomains and iterating to coordinate the solution between adjacent subdomains. By applying some domain decomposition method to the boundary value problem in space, it is possible to accelerate the solution of evolution problems by increasing the speed of each integration step, the time-steps still being performed in a purely sequential manner.

The domain decomposition methods has been widely adopted as they have many benefits. Typically the methods lead to perfect speed-up given sufficiently fast communication and they also work well for the purpose of distributing large problems so to split memory consumption among different nodes.

The often encountered issue that arises when applying domain decomposition on a large scale is a degradation of performance when the number of subdomains to be computed concurrently increase. Eventually the spacial subdomains become very small and the compute units end up spending the majority of time to solve on communicating rather than computing. It is well known that communication speed pose an upper limit to the speed-up this class of algorithms can provide for

a fixed problem size. With modern hardware architectures expected to include an ever increasing number of cores, this pose an issue for the sole use of domain decomposition.

Domain decomposition works well as long as the subdomains are sufficiently large so that the compute time of individual domains is large enough that communication time is negligible. When this is the case, linear weak and strong scaling is possible which is a very attractive feature of domain decomposition. But what are we to do when subdomains reach a size where adding more compute units degrade parallel efficiency to a level so that adding more compute units does not accelerate the application? Here enters parareal, as mentioned elsewhere in this chapter the parareal algortihm can be wrapped around almost any stable type of coarse and fine numerical propagator, including discretizations which are already accelerated trough parallelism in other ways! Thus, instead of adding more cores to spacial domain decomposition, with increasingly diminishing returns, we could use additional compute units to introduce parallelism in the temporal direction by the parareal method. This combination was first proposed by Maday and Turinici in 2005 [48] providing a proof-of-concept implementation of a space-time parallel iterative method for solving the heat diffusion equation using both overlapping and non-overlapping domain decomposition methods.

Despite this possible combination being a central feature that is often mention as the motivation for parareal, surprisingly few practical studies exists in the literature. In a recent contribution [16], the parareal method is coupled with spacial domain decomposition into a space-time parallel scheme, applied to three-dimensional incompressible Navier-Stokes equation. The parallelization allows to employ additional cores to further speed up simulations after spatial parallelization has saturated. Numerical experiments performed on Cray XE6 simulate a driven cavity flow with and without obstacles using distributed memory parallelisation in both space and time. The application was tested with 2048 cores in total, showing that the space-time parallel methods can provide speed-up beyond the saturation of the spacial domain decomposition.

The most recent contribution on the topic of space-time parallel solvers is to appear on the SC12 conference, introducing a massively space-time parallel N-body solver [60]. In the contribution, temporal parallelism is applied on top of a PEPC (Pretty Efficient Parallel Coulomb) existing hybrid MPI/Pthreads spatial decomposition. For the time decomposition the PFASST method [50] is applied. PFASST is based on a combination of the iterations of the parareal algorithm with the sweeps of a spectral defered correction scheme. Results are presented from runs on up to 262,144 cores on the IBM Blue Gene/P installation JUGENE, demonstrating that the space-time parallel code provides speed-up beyond the saturation of the purely space-parallel approach.

3.7 Problems on which Parareal has been applied

It has been roughly a decade since the parareal algorithm in the currently used form of a predictor-corrector method was proposed in [4]. The method has been tested on a wide range of problems, but have still yet to become widely adopted. The latter probably due to other more commonly used methods still being sufficient for the parallelism in today's hardware for most uses. In this section, a non exhaustive list of problems on which parareal has been tested are presented as available in the literature.

Linear and non-linear parabolic problems. Parabolic problems are highly interesting as they cover a large class of problems in science and engineering. A comprehensive set of tests of parareal applied to both linear and nonlinear parabolic equations can be found in [62]. Early work. The parareal algorithm looks particularly promising for this kind of problems given that the stability requirement, proposition 3.4.1 is fulfilled convergence is fast.

Non-differential equation: The American Put. The pricing of an American option has been solved using the parareal algorithm in [7]. Equations of this type are of interest as they are frequently used in the financial world. The equation that was solved reads

$$\min\left(\partial_t u - \partial_{xx}^2 u, u - g\left(x\right)\right) = 0, \quad u\left(t=0\right) = g\left(x\right) = \max\left(\mathrm{e}^x - 1, 0\right) \tag{3.12}$$

In [7] they managed to obtain a speed-up of 6.25 using 50 time decompositions)(compute units). The results are by simulated parallelism though, calculated assuming negligible communication and correction time.

Molecular dynamics. One of the first examples of parareal applied to a real world problem was the solution of a molecular dynamics problem in [4] where the coarse propagator as implemented and tested using both a coarse time-step and a simplified physics model. The results looked promising. Again all solutions are simulated parallelism. The work on parareal for molecular dynamics simulations has later been extended in [3] where a new sympletic parareal variant is proposed and tested on a three-dimensional atomic lattices. The tests show attractive speed-up for lattices of more than 20,000 atoms given a proper choice of coarse propagator

Optimal control for Partial differential equations. The parareal algorithm was reinterpreted as a preconditioning procedure in [47] in order to extend the parallel methodology of the parareal algorithm to the problem of optimal control. The algorithm was applied to a test equation in optimal

control. The results are surprisingly good, with the algorithm itself showing superlinear speed-up. This is certainly not expected of parareal, but no explanation for the surprising result is provided in [47], though the author note that for more complex problems the super-linear speed-up is unlikely.

Stochastic ODEs and filtering problems. In [5] the parareal algorithm is tested on a stochastic ODE with filtering problems. The results are positive. But as noted by the author; when only statistical averages are needed, it is likely to be more efficient to simply generate more realisations rather than accelerate the generation of each realisation. The generation of independent realisations of the SDE is embarrassingly parallel and thus not subject to the same speed-up degration when adding more compute units as say domain decomposition for the solution of PDEs. For applications where only a single realization of the random process is of interest, such as in a filtering problems, parareal shows promising results.

Reservoir simulations. The properties of the parareal algorithm applied to reservoir simulations was investigated in [28] where parareal was applied to an in-house reservoir simulator Athena, simulating fluid flow in porous media. The implementation is one of the first demonstrations of obtaining speed-up for a practical simulation using the parareal algorithm.

Fluid, structure and fluid-structure computations. In [23] a comprehensive study of the parareal algorithm with application to a coupled fluid-structure model is presented. In this study all speed-up results are based on real parallel implementations. The scheme applied to the the problems is a slightly alternative algorithm that can be shown to be equivalent to the predictor-corrector form [23] for autonomous differential equations. The schemes are believed to be the same for all types of problems.

In the case of unsteady flow, a speed-up of 8.2 using 20 processors is achieved, corresponding to an efficiency of 41%. As noted this is not competitive compared to the speed-up possible with parallel methods in space. But potentially in competitive as an addition when speed-up by domain decomposition diminishes when split into many sub-domains. Whereas the algorithm behaved nicely in the case of unsteady flow, the structural dynamics problems experienced difficulties, see section 3.1 for a discussion on stability. Not surprisingly, the result is the same as for the fluid-structure model. If instabilities occur in the structure part of the computation, it pollutes the solution of the fluid problem part.

The work on parareal for fluid-structure problems has later been extended in [24] where a time-parallel framework for linear structural dynamics problems (PITA) was presented. In PITA a projection in each step of the parareal solution is introduced in order to avoid artificial resonance

and numerical instabilities when parareal is applied to the afore mentioned structural dynamics problems. The viability of the framework was demonstrated on complex strucutral dynamics problems from the aircraft industry. In [15] the work was extended to non-linear structural dynamics problems.

Navier-Stokes equation. The parareal algorithm with application to the Navier-stokes equation was investigated in [25]. For small Reynolds numbers the algorithm works well, but for large numbers it fails as the stability conditions discussed in 3.1 are violated. In [64] the parareal algorithm was applied to the unsteady incompressible Navier-Stokes equation showing that although the proposed methodologies are designed for a massive number of processors, significant speed-up can be obtained using a very small number of processors.

Plasma modelling. The parareal algorithm has been applied to the simulation of a fully-developed plasma turbulence model successfully in [55]. The results is far from trivial, and even unexpected since the exponential divergence of Lagrangian trajectories as well as the extreme sensitivity to initial conditions characteristic of turbelence set these type of simulations apart from the much simpler syystems to which the parareal algorithm has otherwise been applied to showed fast convergence. Speedup scales quite different to what is typical for spatial parallelization.

Acoustic-advection equation. The applicability of the parareal integration scheme for the solution of a linear, two-dimensional hyperbolic acoustic-advection system has been investigated by [54]. The paper adepts the modified parareal from [24] and employs it for the solution of a hyperbolic flow problem. It is demonstrated that the modified parareal is stable and can produce reasonable accurate solutions while allowing for a noticeable speed-up.

In [49] the application of parareal to a general acoustic wave propagation is investigated using a spatial discretization based on a spectral element approximation which allows flexible and accurate wave simulations in complex geological media. Initially a 1D acoustic wave equation in an homogeneous medium is tested to investigate convergence and stability. Tests confirm the stability issues as outlined by [23] and [6]. [49] notes that the stability issues are mitigated when using a time-discontinuous Galerkin discretization of the coarse propagator.

Quantum control. In [47] the theoretical modifications required to apply the parareal algorithm to quantum control are presented including preliminary results illustrating the feasibility of the approach and potential for large speed-up. Actual implementation speed-ups for quantum control problems are presented in [45].

3.8 Ongoing challenges and further work

In this chapter the vast amount of research that has gone into the parareal algorithm during the past decade has been coarsely sketched with the purpose of giving the reader an overview of the field. Despite the vast amount of published literature on the topic of the parareal-in-time algorithm, there are still many unanswered questions. Here we summarize a few important messages and discuss potential needs and missing contributions in the literature.

The stability of the parareal algorithm for linear autonomous differential equations has been thoroughly investigated, and expressions have been derived that guarantee stability of the parareal algorithm for all number of intervals N and all number of iterations $k \leq N$ when fulfilled. The stability for non-autonomous differential equations and for a general non-linear differentiation equation is still to be investigated. It is believed that doing so will require a much more involved analysis than what was presented in section 3.1. The stability analysis of the autonomous differential equations using different spatial discretizations for the operators $\mathcal{G}$ and $\mathcal{F}$ is also unexplored.

Given that we are certain that the algorithm is strongly stable on some problem, our next point of interest is in obtaining the largest possible speed-up given some number of compute units. Each iteration in the parareal scheme adds more computational work and by this degrade the parallel efficiency that can be obtained, it would be nice to see more research on the topic of effective stopping criteria's, with the only major result being that presented in [39].

The most important aspect in obtaining a high parallel efficiency is to choose a well balanced coarse propagator that provides sufficient accuracy to rapidly converge on all time sub-domains, but at the same time fast enough to minimize the time spent in this purely sequential propagation and the efficiency loss by compute units being idle waiting for compute work. Given the importance of the choice of coarse propagator, surprisingly few investigations on the optimal choice of coarse propagator exists. There are many papers available proposing different creative ways of constructing the coarse propagator, but not many studies exists on how to find the right trade-off between accuracy and speed of the coarse operator. Investigations on this topic is made in chapter 4 and 5.

The parareal algorithm has yet to find wide adaptation as a method of extracting parallelism in the solution of differential equations. This is likely do to the fact that other methods at accelerating the solution of differential equations trough parallelism has been shown to be very effective. For the parareal method, obtaining 50% parallel efficiency is great, typically in the range of 30-40% is observed when things work well on large scale problems. An ap-

proach where parallel efficiency of 100% is possible is the widely used spacial domain decomposition class of methods. An often mentioned motivation for the parareal algorithm is how adding an ever increasing amount of cores to a fixed problem (Strong scaling) typically diminishes speed-up returns when the spacial sub-domains become smaller as the assigned compute units spend more time communicating than computing. The potential opening for parareal is that when adding more compute units to a fixed problem, eventually, it may be more effective to begin decomposing the time direction than making the spacial sub-domains ever smaller. The good thing with parareal is that it allows to be wrapped around existing parallelism, including domain decomposition as documented in [48]. In 7 we apply parareal to a nonlinear free-surface water wave model investigation speedup and scalability for both the parareal method and spacial domain decomposition.

Despite this seemingly valid motivation, actual demonstrations of this combination being done is not often seen, with only two very recent publications on the subject [16, 60]. The lack of adoption may be in part due to other parallel methods still being sufficient at large and because adding another layer of parallelism increase implementation complexity, making developers reluctant to take such an approach.

The issue of communication being a show-stopper for parallelism is likely to increase in the future. For many years the amount of compute work that the CPUs could create have been growing much faster than the speed of communication, both on a node level(RAM speed and latency) and on a grid level(Network speed and latency). These developments may lead to the time-decomposition parallelism eventually becoming more widely used, in this context the parareal algorithm represents an exciting new way of using the ever increasing number of cores available in today's and tomorrows supercomputer platforms.

Chapter 4

Experimental investigation of convergence and stability

In chapter 2 figure 2.3 the solution procedure of the parareal algorithm applied to a simple ordinary differential equation was visualised. We noticed how convergence looks to happen in some 3-4 iterations, if need be one could measure this using a criteria as discussed in section 3.4. We would like to somehow qualitatively establish how the convergence rate depends on algorithm parameters such as the number of intervals N and the speed and accuracy of the coarse propagator $\mathcal{G}_{\Delta T}$ of the application of parareal to various differential equations. As mentioned in the summary of chapter 3, such discussion and observations are quite limited in published literature on parareal.

In the investigations to follow we measure the error of the parareal solution as a function of iterations count for various algorithm parameter combinations on four different types of initial value problems. As an initial problem we choose the Bernoulli equation using the forward euler method for both the coarse and the fine propagator. The problem is simple and with known analytical solution and as such there is nothing that would indicate irregular behaviour as long as the stability constraints on both the coarse and fine time-step are upheld. Next, we investigate the application of parareal to a linear system of ODEs with pure imaginary eigenvalues. For a system of pure imaginary eigenvalues, stability can not be guaranteed for all discretizations even when both the coarse and the

fine propagators are stable. See section 3.1 for a discussion on topic of parareal applied to hyperbolic problems in the literature. Finally we test parareal applied to a Stiff ODE system using an adaptive-in-time propagator followed by an investigation on parareal to a parabolic partial differential equations. The problems investigated are itemized below. All tests are implemented in Matlab. The numerical solvers used as propagators have been implemented using various finite difference discretions and schemes such as backwards euler, Runge-Kutta with adaptive timestep, Crank-Nicolson and ADI.

- Bernoulli equation, first order non-linear ODE using forward euler
- Linear system of ODEs with pure imaginary eigenvalues using backwards euler
- The Van der Pol oscillator, second order stiff ODE system using time adaptive ESDIRK2
- Diffusion equation with forcing term, parabolic second order PDE using Crank-Nicolson and ADI

Before embarking on the quest to qualitatively establish how the convergence rate depends on algorithm parameters we apply parareal to the test equation so to present the very basic convergence properties of the algorithm as this understanding is essential in the analysis of the results to follow. This chapter also serves as a precursor to the work presented in chapter 5 where we discuss and investigate how the parameter choice influence on convergence rate relate to the efficiency and speed-up of practical implementations with a given model of distribution of parallel work.

4.1 Test Equation

We apply parareal to the test equation 4.1 using the forward euler as both coarse and fine propagator. The test equation is coined so as it is often used in numerical stability analysis and thus it is a straight forward choice of IVP problem to present basic convergence properties of algorithm.

$$\frac{\partial y\left(t\right)}{\partial t} = \lambda y\left(t\right), \quad y\left(0\right) = 1 \tag{4.1}$$

using $\lambda = -1$ and solving on the interval $t \in [0, 1]$. We define parareal iterations as in the pseudo code in section 2.1. The convergence properties are independent of how the parallel work is being distributed. In this report and throughout, we define the number of parareal iterations as the number of times that the N initial value problems are solved in parallel using the fine propagator. Iteration zero is the prediction based on the initial coarse propagation with no fine propagations or correction.

In figure 4.1 the convergence rate of parareal on 4.1 is presented. The algorithm is applied using $N = 10$ intervals and a fine operator time-step $dt = 10^{-4}$ and three different values of dT given by $dT = R \cdot dt$ with $R = [50, 250, 1000]$. In figure 4.1a the error is measured with respect to the solution one would obtain using the fine propagator sequentially throughout the integration, and in figure 4.1b the error is measured with respect to the analytical solution of 4.1. As expected, the algorithm converge towards the fine solution, only limited by machine precision. Observe that the norm of the difference between the parareal solution and the fine solution convergence towards machine precision within in 5-7 iterations, while the parareal solution obtain the same order of accuracy as the fine solution with respect to the analytical solution in only 1-3 iterations. From this basic example it is clear that using the machine precision as stopping criteria on the norm of difference between two consecutive iterations would lead to many superfluous iterations as hinted in section 3.4.

Now let us instead fix the time-step length of fine propagator while varying the step-length of coarse step. Again we use $N = 10$ predictor/corrector intervals. The coarse propagator time-step is fixed at $dT = 2.5 \cdot 10^{-2}$ and the time-step of the coarse propagator is set to $dt = dT/R$ with $R = [50, 250, 1000]$. Results are presented in figure 4.1. Notice how the convergence rate is now constant while the final accuracy differs as expected. Thus, the fine propagator determines the accuracy of the final solution while the coarse propagator determines the rate of convergence. Note that changing the accuracy of the fine propagator can still change the number of iterations needed for convergence.

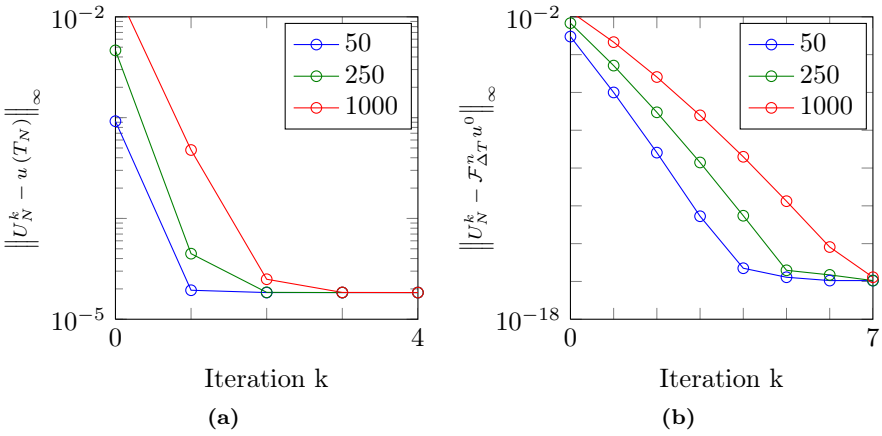


Figure 4.1: Error of the parareal solution after k iterations for fixed fine time-step length $\delta t = 10^{-4}$. a) Measured with respect to the solution of the pure fine operator $\mathcal{F}_{\Delta T}^N u_0$ and b) measured with respect to the analytical solution

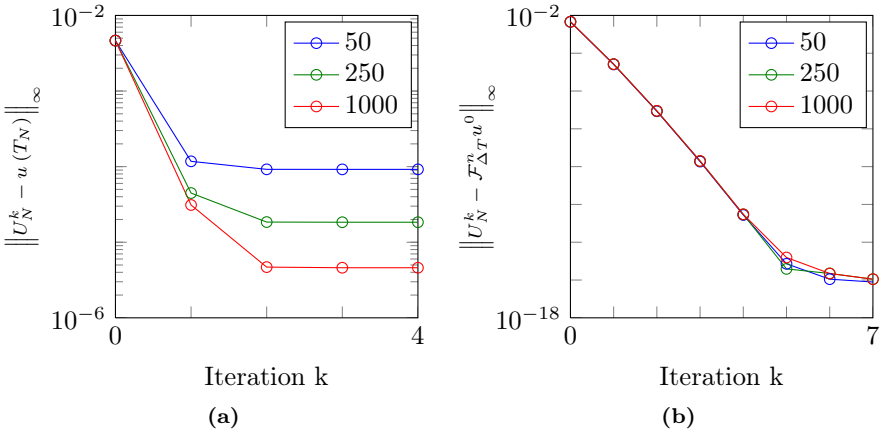


Figure 4.2: Error of the parareal solution after k iterations for fixed coarse time-step length $\delta T = 2.5 \cdot 10^{-2}$. a) Measured with respect to the solution of the pure fine operator $\mathcal{F}_{\Delta T}^N u_0$ and b) measured with respect to the analytical solution

4.2 Bernoulli

Now we apply the parareal algorithm to the Bernoulli differential equation 4.2. Bernoulli equations is a class of differential equations arising naturally upon examining the motion of a body subject to a resistance $F = c_1 v + c_2 v^\alpha$ where v is the velocity of the body. The equations are special because they are non-linear equations with known analytical solutions.

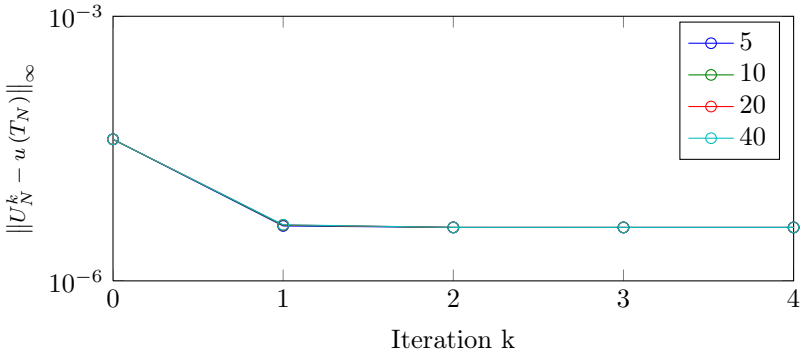
$$\frac{\partial y(t)}{\partial t} = 2\frac{y(t)}{t} - t^2 y(t)^2, \quad y\left(\frac{1}{2}\right) = \frac{40}{33} \tag{4.2}$$

with analytical solution $y(t) = \frac{5t^2}{t^5+1}$. We descritize the equation using a simple forward euler approach for both the coarse and the fine propagator. The equation is solved on the interval $t \in [0.5, 5.5]$ and the error is measured at $T = 5.5$ using the infinity norm. For the fine propagator a time-step of $\delta t = 5 \cdot 10^{-4}$ is used. In the figure 4.3 the error of the parareal solution as a function of iteration is presented using a time-step $\delta T = R \cdot \delta t$, $R = \{10, 50, 250\}$ for the coarse propagator in figures , and respectively.

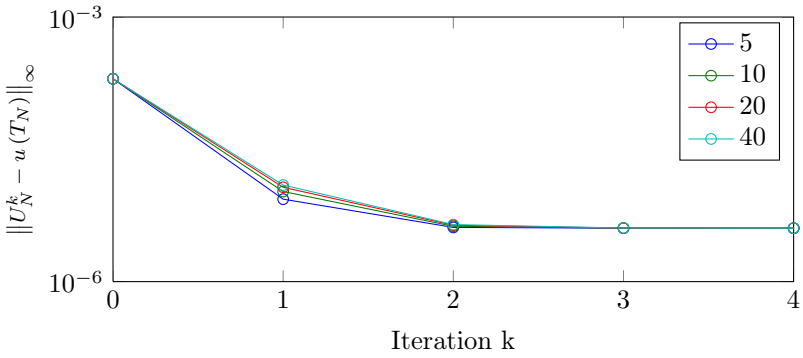
As observed when applying parareal to the test equation, the convergence rate depends on the ratio R since $R = \delta t/\delta T$ and δT is inversely proportional to the accuracy of the coarse propagator. Large R leads to slow convergence, but large R also indicates an inexpensive coarse solver which as we will see in chapter 5 greatly influence the attainable efficiency of the method in practical implementation when taking into account the distribution of parallel work.

Another interesting observation that can be made from figure 4.3 is how the rate of convergence for small R is largely unaffected by the number of intervals. In figure 4.3a convergence is reached in 1 iteration for all number of intervals tested, but for large R, figure 4.3c, the convergence rate is seen to decrease slightly with N. The convergence plots are in essence showing strong scaling for coarse propagators of different accuracy as we perform the test on a fixed problem. From the results we notice that when the number of iterations needed for convergence is small, the number of time decomposition intervals N does not change the convergence rate. But when the number of iterations needed for convergence is large, increasing the number of time intervals leads to a slower convergence rate.

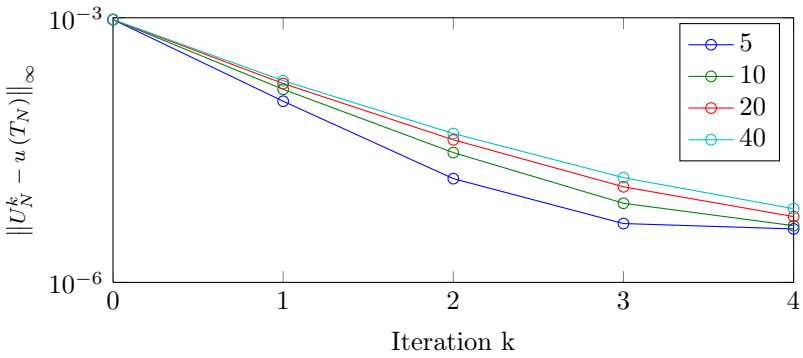
In context of the scaling discussions in section 3.2, we can in a purely algorithmic sense expect to achieve linear strong scaling with accurate propagators while less accurate(but fast) propagators will not have this property. This may work to explain some different results found in the literature on the topic such as in [48] and [2] where parareal has been applied to the same diffusion problem.



(a)



(b)



(c)

Figure 4.3: Convergence rate plots of the parareal solution of eq. 4.2. A forward euler method has been used for both the coarse and fine discretizations with a fine operator time-step of $\delta t = 5 \cdot 10^{-4}$.

4.3 Linear ODE system

In this section we examine the convergence properties of parareal on a linear system of ordinary differential equations with pure imaginary eigenvalues. As a consequence of the stability issues as described in 3.1 may arise for problems with pure imaginary eigenvalues. Matrices that satisfy the relation $\mathbf{A}^T = -\mathbf{A}$ are called skew symmetric and have pure imaginary eigenvalues. We choose such a skew symmetric form for the coefficient matrix with randomly choosing coefficients for a system on the form

$$\begin{bmatrix} \frac{\partial}{\partial t} y_1(t) \\ \frac{\partial}{\partial t} y_2(t) \\ \frac{\partial}{\partial t} y_3(t) \\ \frac{\partial}{\partial t} y_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 2 & 1 & -4 \\ -2 & 0 & 7 & -5 \\ -1 & -7 & 0 & 3 \\ 4 & 5 & -3 & 0 \end{bmatrix} \begin{bmatrix} y_1(t) \\ y_2(t) \\ y_3(t) \\ y_4(t) \end{bmatrix} \tag{4.3}$$

Computing the eigenvalues one arrives at $\bar{\lambda} = (10.0570\mathrm{i}, -10.0570\mathrm{i}, 1.6904\mathrm{i}, 1.6904\mathrm{i})$. To solve the system we apply a backwards euler method for both the coarse and fine propagator. The system is solved on the interval $t \in [0, 10]$, using the initial condition $\bar{y}(t) = [1, 1, 1, 1]$ and a fine propagator time-step $\delta t = 10^{-4}$. The convergence rate for various coarse to fine propagator ratios $R = 50, 500, 1000$ is presented in figure 4.4 for the number of time decompositions $N = 10, 50, 100$.

Many interesting things are to be noted in these figures. Let us try and relate the results to the stability criteria for a general linear system of ordinary differential equations with complex eigenvalues. According to theorem 3 in section 3.1, we can guarantee that the parareal algorithm converges for all iterations k if the relationship 3.7 is fulfilled for all eigenvalues of the coefficient matrix.

In order to compare the validity of the theorem, we build a little Matlab script to calculate the left and right hand side for all the eigenvalues in the matrix for each value of ΔT using the stability function of the backwards euler and some neat Matlab commands. The values are listed in table 4.1.

In the case presented in figure 4.4a with $R = 50$, the stability criteria 3.7 is fulfilled for all the eigenvalues, and we notice from the figure that the parareal solution converge for all values of N and k. Albeit, the convergence rate is nothing to brag about, but convergence in 7 iterations with 100 compute units still allows for a potential speed-up of up to $100/7 \approx 14.3$, disregarding communication and distribution of parallel work.

When a ratio of $R = 1000$ is used, the criteria 3.7 is strongly violated for all

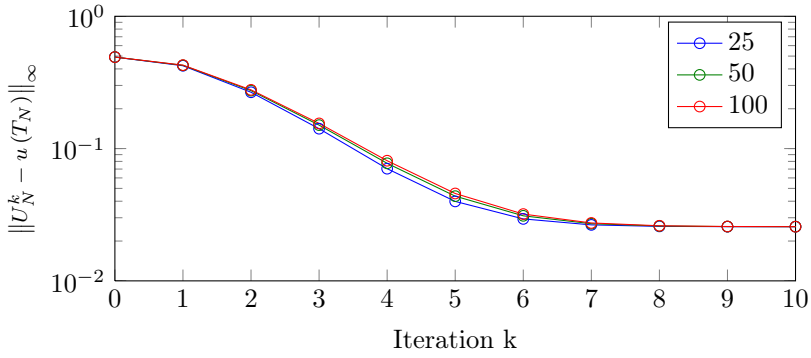
eigenvalues and we notice how the convergence rate presented in figure 4.4c depicts this. The error increases rapidly before only eventually converging in $k = N$ iterations.

A "middleroad" was also tested, using a coarse to fine propagator ratio of $R = 500$, the relation 3.7 is still violated for all eigenvalues, but less so than for $R = 1000$. Again we notice that the algorithm does not converge for all iterations k. Only after some 12-20 iterations does the parareal solution begin to converge upon the fine propagator solution, completely not viable in a parallel implementation.

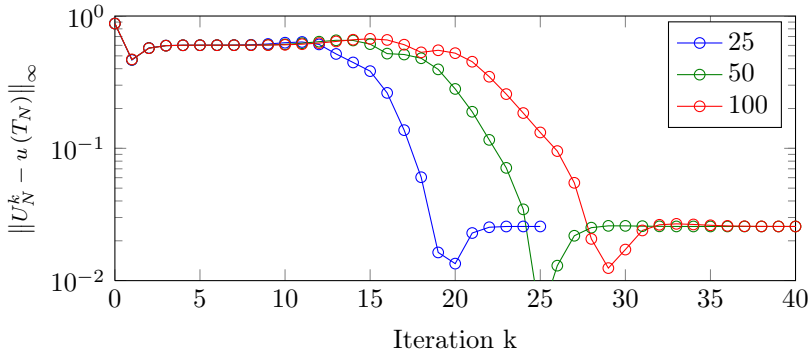
Upon these investigations we can conclude that theorem 3 presented in section 3.1 and proposed in [63] seems to do a good job in terms of predicting whether or not some coarse fine operator combination will converge for all k.

	$\Delta T = 0.005\, R = 50$		$\Delta T = 0.05\, R = 500$		$\Delta T = 0.1\, R = 1000$	
	LHS	RHS	LHS	RHS	LHS	RHS
λ_1	0.9905	0.7768	0.1050	0.0190	0.0304	0.0032
λ_2	0.9905	0.7768	0.1050	0.0190	0.0304	0.0032
λ_3	0.9999	0.9929	0.9313	0.9467	0.8686	0.5328
λ_4	0.9999	0.9929	0.9313	0.9467	0.8686	0.5328

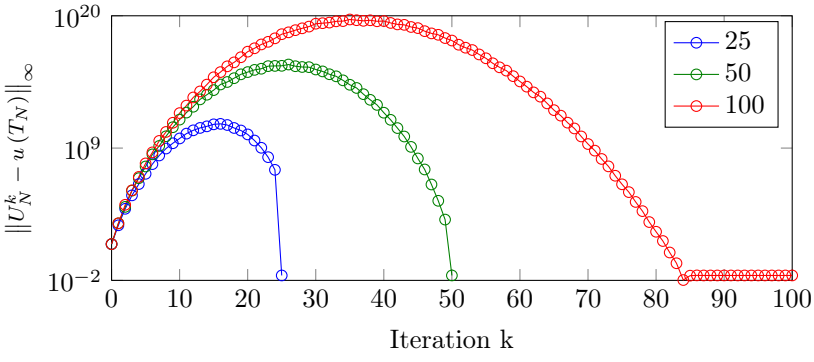
Table 4.1: The left and righ hand side of the relation 3.7 on 4.3 calculated on the discretizations as described in 4.3 using the stability function of the backwards euler method.



(a)



(b)



(c)

Figure 4.4: Converge rate of parareal applied to the system 4.3 for various discretizations.

4.4 Van der Pol Oscillator

We now consider the Van Der Pol Oscillator, a non-conservative oscillator with non-linear damping that reads

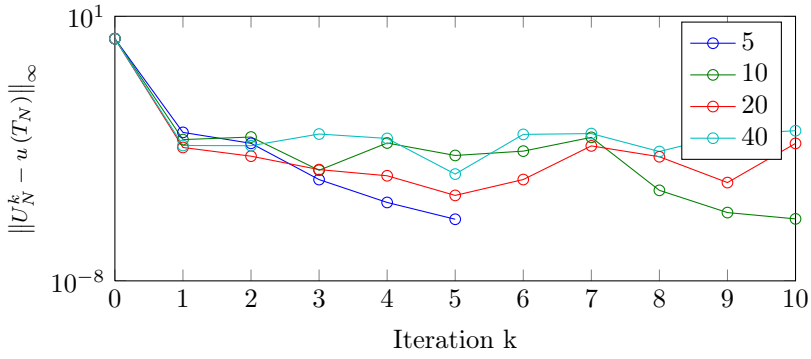
$$\frac{\partial^2}{\partial t^2} y(t) - \mu \left(1 - y(t)^2\right) \frac{\partial}{\partial t} y(t) + y(t) = 0 \tag{4.4}$$

To solve the second order ODE it is restated as a system of two coupled first order ordinary differential equations. With $y_1(t) = y(t)$, we write 4.4 on the form

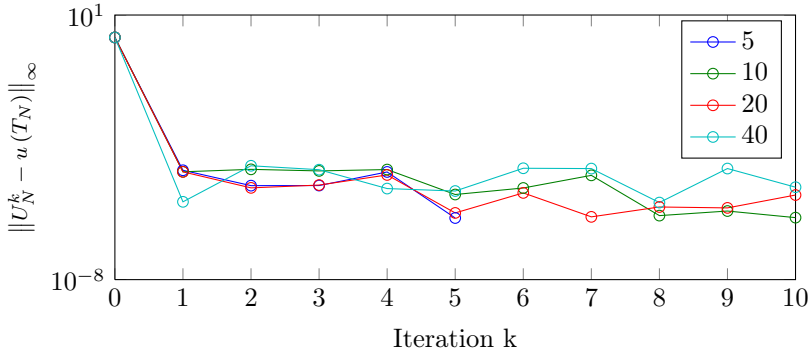
$$\begin{aligned} \frac{\partial}{\partial t} y_1(t) &= y_2(t) \\ \frac{\partial}{\partial t} y_2(t) &= \mu \left(1 - y_1(t)^2\right) y_2(t) - y_1(t) \end{aligned}$$

It can be shown that with $\mu = 100$ the problem becomes very stiff and in order to use an explicit method one would need to choose a time-step that is small relative to the time-scale of the rapid transient, which might be much to small for practical purposes. Instead it is usually preferable to apply an implicit method and for the same reasons it is useful to implement a dynamical way of controlling the step size for each iteration in the solution. The reason being that for a large period of time, the solution is only slowly varying, and long-time-steps are sufficient, while during other periods of time the solution varies rapidly and much smaller time-steps are needed to resolve the problem sufficiently. As a numerical propagator we choose the ESDIRK23 method combined with an asymptotic step-size controller. ESDIRK23 is a diagonally implicit Runga-Kutta integrator of second order accuracy with an embedded error estimator of third order accuracy. The embedded estimate given at each time-step is used to control the time-step length and derived error. That is, instead of specifying a time-step length for our operator, we specify some acceptable relative tolerance, and it is the job of the step-size controller to make sure that the error stays within the acceptable limit.

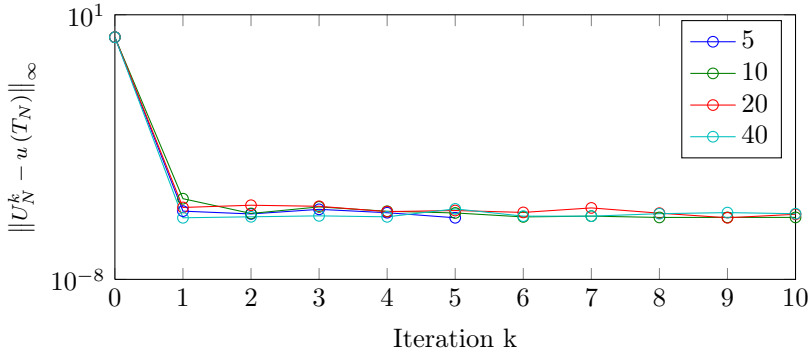
The convergence results of parareal applied to 4.4 using the ESDIRK23 method with an embedded error controller as both coarse and fine operator are presented in figure 4.5. A fine operator tolerance of $\delta t_{tol} = 10^{-9}$ is used with the coarse operator tolerances $\delta T_{tol} = [10^{-3}, 10^{-4}, 10^{-5}]$. Measuring the actual coarse to fine operator speed relationship we obtain roughly $R \approx \{51, 30, 16\}$. It is clear from 4.4 that for a coarse operator tolerance of $\delta T_{tol} = 10^{-5}$ the parareal solution converges after only one iteration. This certainly allows for a descent speed-up despite a low coarse to fine relationship of $R \approx 16$ as the effect of the slow operator can be partially hidden by using an efficient way of distributing parallel work as described in 3.3.



(a)



(b)



(c)

Figure 4.5: Visualization of the convergence rate of parareal applied the Van der Pol oscillator 4.4 using the ESDIRK23 method with asymptotic step-size control as both coarse and fine propagator. (a) $R \approx 52$ (b) $R \approx 29$ (c) $R \approx 17$.

4.5 Two-dimensional diffusion

In this section the convergence properties of the parareal method applied to the heat diffusion equation are investigated. The heat-diffusion equation is a classical example of a parabolic partial differential equation. The equation to be solved is the same as the one investigated in [48] in the context of spacial domain decomposition coupling and in [2] so to test various task-allocation algorithms aimed at improving the efficiency of the parareal algorithm. Using the same equation enables us to compare convergence results. The two dimensional heat diffusion equation with a forcing term reads

$$\frac{\partial u\left(x,y,t\right)}{\partial t} = \nabla^2 u\left(x,y,t\right) + 50\sin\left(2\pi\left(x+t\right)\right)\cos\left(2\pi\left(y+t\right)\right) \tag{4.5}$$

with $u\left(x,y,0\right) = 0$ and homogeneous Dirichlet boundary conditions. The equation is solved on the interval $t = [0,1]$ and in the spacial domain $x,y = [0,1]$. For a proper comparison the same numerical solver, the alternating direction implicit [19], as applied in [2] is used for both the coarse and fine propagator. The alternating direction implicit method is a modification of the locally-one-dimensional method (LOD).

The straight forward classical approach of solving a multidimensional diffusion equation is by applying a 5-point Laplacian in space and discretize in time using the trapezoidal method so to obtain the two-dimensional version of the Crank-Nicolson method. The method is second order accurate and unconditionally stable for any time-step. However, the implicit method requires the solution of system of equations of which the matrix is very large and very sparse and with a structures that makes it unsuitable for direct methods such as Gaussian elimination. An alternative approach is the LOD method as mentioned. The essential idea is to replace the fully coupled single time-step with a sequence of steps each of which is coupled in one only space direction resulting in a set of tridiagonal systems which can be solved with less computational effort.

The later is achieved by first applying Crank-Nicolson in the x-direction only solving over one time-step and then apply Crank-Nicolsion in the y-direction to the intermediate result. Physically this corresponds to modelling the diffusion in the x and y-directions over time as decoupled processes. Apart from some issues at boundaries that needs to be handled this works out nicely.

The alternating direction implicit method is a modification of the LOD method in which the two steps each involve discretization in only one spatial direction at the advanced time level, giving decoupled tridiagonal systems again, but coupled with discretization in the opposite direction at the old time level. The method

of this form is

$$U_{ij}^{*} = U_{ij}^{n} + \frac{k}{2}\left(D_y^2 U_{ij}^{n} + D_x^2 U_{ij}^{*}\right) + \frac{k}{2} f\left(x_{ij}, y_{ij}, \frac{t_{n+1} - t_n}{2}\right)$$
$$U_{ij}^{n+1} = U_{ij}^{*} + \frac{k}{2}\left(D_x^2 U_{ij}^{*} + D_y^2 U_{ij}^{n+1}\right) + \frac{k}{2} f\left(x_{ij}, y_{ij}, \frac{t_{n+1} - t_n}{2}\right)$$

with

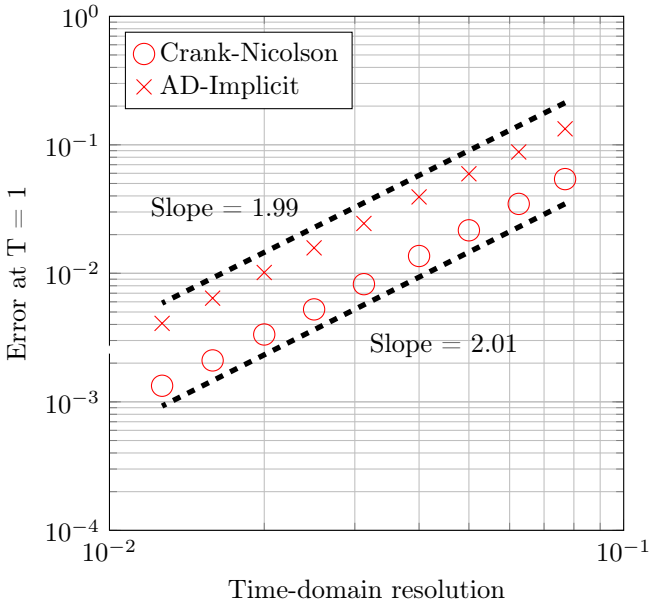
$$D_x^2 U_{i,j} = \frac{1}{h^2}\left(U_{i-1,j} - 2U_{i,j} + U_{i+1,j}\right)$$
$$D_y^2 U_{i,j} = \frac{1}{h^2}\left(U_{i,j-1} - 2U_{i,j} + U_{i,j+1}\right)$$

so to arrive at the matrices

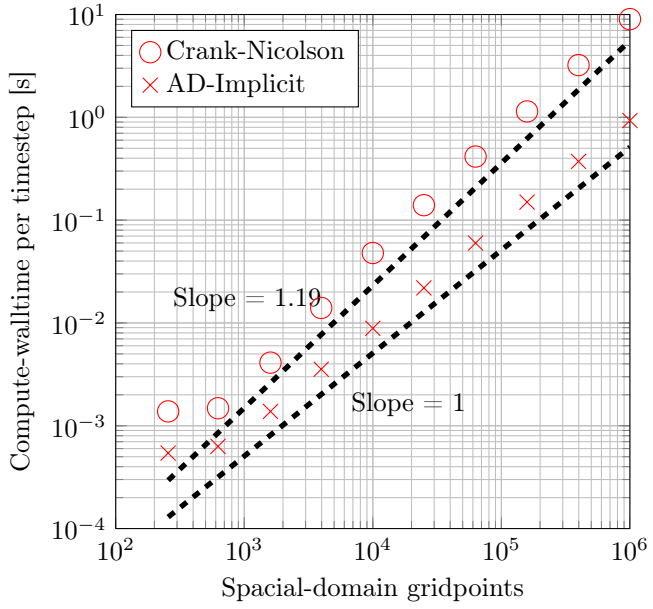
$$\left(I - \frac{k}{2} D_x^2\right) U^{*} = \left(I + \frac{k}{2} D_y^2\right) U^{n} + \frac{k}{2} f\left(X, Y, \frac{t_{n+1} - t_n}{2}\right)$$
$$\left(I - \frac{k}{2} D_y^2\right) U^{n+1} = \left(I + \frac{k}{2} D_x^2\right) U^{*} + \frac{k}{2} f\left(X, Y, \frac{t_{n+1} - t_n}{2}\right)$$

With $f(x, y)$ referring to the forcing term. In matrix form this again gives decoupled tridiagonal systems to solve in each step. Numerically there is a great advantage in decoupling the dimensions when possible by the physical process that is being modelled. By decoupled the space dimensions one obtain a sequence of tridiagonal systems to solve instead of a single large sparse matrix with a complicated structure. In the figures 4.6a and 4.6b the convergence and time to compute per iteration for the classical Crank-Nicholson approach and for the ADI approach is given.

The parareal algorithm is implemented using the ADI discretization method for both the coarse and fine propagator. A fine time-step length of $\delta t = 10^{-4}$ is used with coarse to fine propagator ratios $R = 25, 50, 100$. The convergence rate measurements are presented in figure 4.7. Notice how instabilities arise for some combinations of R and N, a bit surprising as we are dealing with a parabolic equation. Though it is important to note that there is no guarantee that the methods converge for all N and k simply because it is of parabolic type. The behaviour is quite remarkable, as we either observe divergence or very fast convergence, seemingly not much in between. The results does confirm the deductions from the analysis of parareal on linear constant coefficient problems 3.1, that the stability issues typically arise for large N and k.

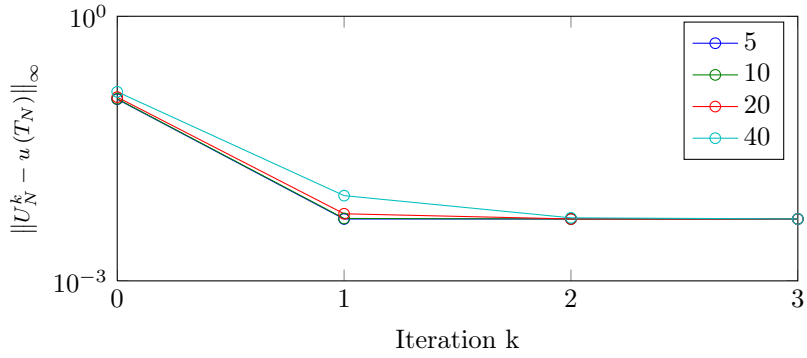


(a)

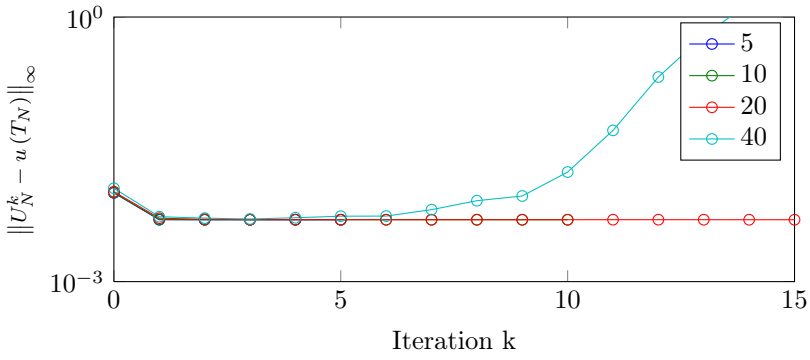


(b)

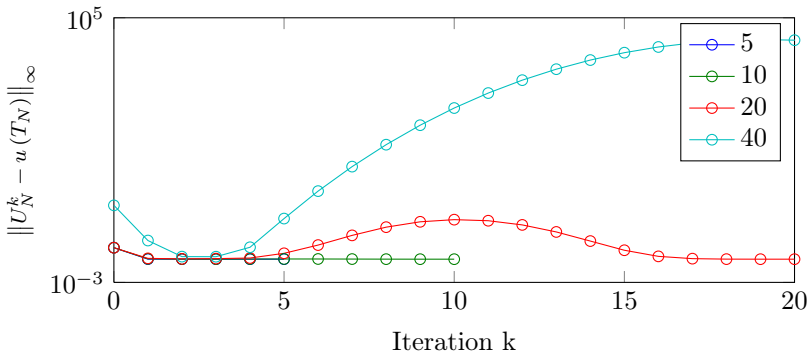
Figure 4.6: Convergence and error measurements by the integration of 4.5 on the domain $x \in [0,1]$, $y \in [0,1]$ and $t \in [0,1]$ using the ADI and the Crank-Nicholson method.



(a)



(b)



(c)

Figure 4.7: Visualization of the convergence rate of parareal applied to the two dimensional diffusion equation with forcing term 4.5 using the ADI method.

4.6 Summary

In this chapter we investigated the convergence properties of the parareal algorithm applied to a range of different problems with various coarse and fine numerical propagators. The parareal solution was shown to always converge given a sufficiently accurate coarse propagator.

By the tests performed a number of observations are worth noticing. First, it appears that the parareal algorithm in a purely algorithmic sense exhibits close to linear strong scaling when a sufficiently accurate coarse propagator is applied so that convergence happens in few iterations. Typically one or maybe two. In the case of choosing a faster but less accurate solver resulting in more iterations needed for convergence, the number of iterations needed for convergence was typically observed to increase as more time sub-domains was added to the problem.

It was also shown, not surprisingly, that increasing the accuracy of the coarse propagator leads to faster convergence. As discussed in chapter 3, section 3.3, one would expect the algorithm to converge faster when using a more accurate coarse solver, since more time is then spend propagating information forward. On the other hand, spending more time computing a fairly accurate coarse solver diminish the parallel efficiency since the coarse solver is used in a strictly sequential fashion.

In section 2.4 the parallel speed-up was estimated to be N/k (efficiency $1/k$), assuming negligible communication and correction time as well as an infinitely fast coarse propagator. From the observations made in this chapter we can already conclude that in particular, the later assumption may not be viable to archive for a coarse propagates that are to be sufficiently accurate so that the algorithm converges in sufficiently few iterations. Instead we may just note that $1/k$ constitutes and upper bound to the parallel efficiency that can be obtained.

The reasons for investigating the influence of time-subdomains and coarse propagator accuracy to that of the convergence rate is that all these factors are interconnected in determining the parallel efficiency that can be obtained and this will be the topic of the chapter to follow.

Chapter 5

Efficiency and speed-up

When using the parareal method to expose parallelism in the numerical integration of an IVP there is an inherent trade-of between the computational time spend on the coarse solver and the number of iterations needed for convergence as documented in chapter 4.

A natural question that arises in this context is how the choice of coarse propagator, given a number of time-subdomains, influence the speed-up and parallel efficiency of the algorithm. When implementing the algorithm, a developer will need to make a choice in terms of the accuracy of the coarse propagator, but it is not clear from the literature available what to consider when making such a choice. This is the topic of the chapter, and as will be documented, the question is not easily answered.

5.1 The issue of optimal efficiency

In chapter 4 the parareal algorithm was observed to obtain linear strong scaling for small k, and in section 2.4, an upper bound on parallel efficiency $1/k$ was derived. For this reason, an obvious approach at choosing the coarse propagator would be to find a sufficiently accurate operator $\mathcal{G}_{\Delta T}$ that makes the algorithm converge in a single iteration. By doing so we have maximised the upper bound on parallel efficiency and in a purely algorithmic sense, we can also expect almost linear strong scaling.

Indeed such a strategy of rephrasing the parareal iterative algorithm into a purely hierarchical one has already been proposed in [5]. But is this strategy always an optimal strategy? How much of a degradation in parallel efficiency will the speed, or lack thereof, of the coarse propagator induce?

We have already qualitatively established how the purely algorithmic behaviour of the parameters involved influence each other, but to address the posed question we also need to take into account the influence of the distribution of parallel work.

A few different models for the distribution exists as discussed in 3.3. The task-scheduling models proposed by [2] are heavily loops dependent which allows for the prediction of parallel speed-up given R, K and N are known while assuming negligible communication and correction time. In [2] a manager-worker based model and a fully distributed model where presented, which both was shown to minimize the idle time of compute units efficiently.

In the figures 5.1a and 5.1b, the parallel efficiency as a function of coarse to fine propagator speed R and number of time decomposition N is plotted for a fixed iteration to convergence count K as predicted when using a fully distributed task scheduling model. In the figurers 5.2a and 5.2b the parallel efficiency has been estimated using the manager/worker task scheduling model, again given a fixed number of iterations to convergence K.

The figures 5.2a and 5.2 depicts the impact of the distribution model on the parallel efficiency obtained by a practical implementation. We notice particularly how

- Parallel efficiency decrease with increasing K
- Parallel efficiency decrease with increasing N
- Parallel efficiency increase with increasing R

Also, from the investigations in chapter 4, we know that from a purely algorithmic point of view, K increase with increasing N if K is big, and that K decrease with increasing R. In the evaluation of parallel efficiency, a multitude of opposing factors exists, and it is not clear which factors are dominant and when, as it is likely both problem and discretization dependent. It might be possible though to make some general qualitative remarks on the behaviour. Such potential knowledge may be valuable when choosing a coarse propagator for some system in the quest to obtain the highest possible speed-up.

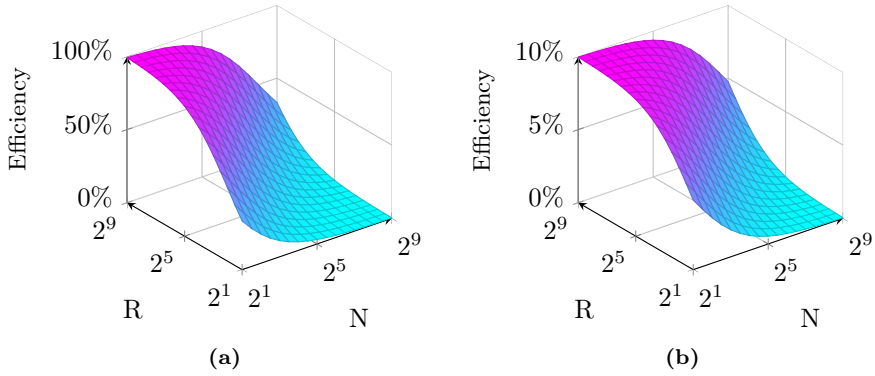


Figure 5.1: Predicted parallel efficiency as a function of R and N, using the fully distributed task scheduling model with fixed *(a)* $K = 1$ and *(b)* $K = 10$

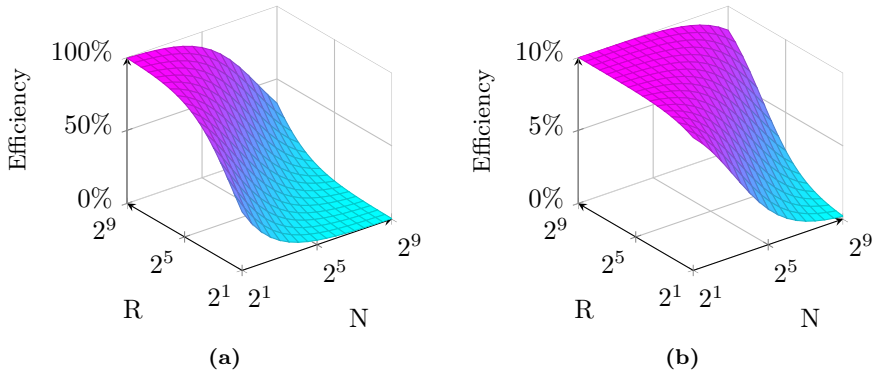


Figure 5.2: Predicted parallel efficiency as a function of R and N, using the manager-worker task scheduling model with fixed *(a)* $K = 1$ and *(b)* $K = 10$

We have established that the number of iterations needed for convergence K decrease with decreasing R. Also, from the figures 5.1 and 5.2 it is clear that the distribution of parallel work become more efficient with increasing R. With these opposing factors, among others, in the estimation of parallel efficiency, we expect that there for a given problem with a given coarse and fine solver exists a combination of parameters yielding the the highest possible parallel efficiency for that given problem. In this chapter we investigate this matter experimentally by implementing the parareal method on the range of problems presented in 4 using various solvers. The purpose, apart from proof-of-concept on various equations, is to quantify the range in parallel efficiency between an optimal choice of parameters and a poor choice. That is, how much of a penalty in parallel efficiency can be expected by making a less than optimal choice. The later enabling us to determine how much effort and concern should go into tuning the coarse propagator accuracy/speed trade-off and in addition potentially make some general observations that can serve as heuristics in tuning the trade-off between accuracy and speed.

5.2 Experimental investigations

To be able to make such generalizations as discussed in the previous section, we measure the number of iterations needed for the parareal solution to converge for each problem and discretization over a space of coarse to fine ratio's R given a fixed fine operator, and number of time sub-domains N. For each point in the space we have all the information needed to predict the parallel efficiency and speed-up given some distribution model only assuming negligible communication and correction time. In the previous chapter we visualized the convergence process of the parareal algorithm. In order to determine if convergence has happened, we need some stopping criteria as discussed in 3.4. The topic of the chapter is on optimal coarse operator choice and not optimal stopping criteria, therefore we pretend to have a perfect stopping criteria available. When the error at a given time-subdomain interval boundary is less than 1.1 times the error of the fine propagator solution at that same interval boundary, we are satisfied with the accuracy of the parareal solution and accept the parareal solution as converged. Throughout the rest of this report, chapter 6 and 7, we use the same criteria in determining convergence. The results of this brute force investigation are presented in the figures 5.3, 5.4, 5.5 and 5.6 for the parallel efficiency using a fully distributed and a manager-worker distribution model. In 5.7, 5.8, 5.9 and 5.10 the respective speed-up estimates are given. In all figures the highest parallel efficiency and speed-up obtaining for a given number of time-decompositions (compute units) are marked with a blue dots.

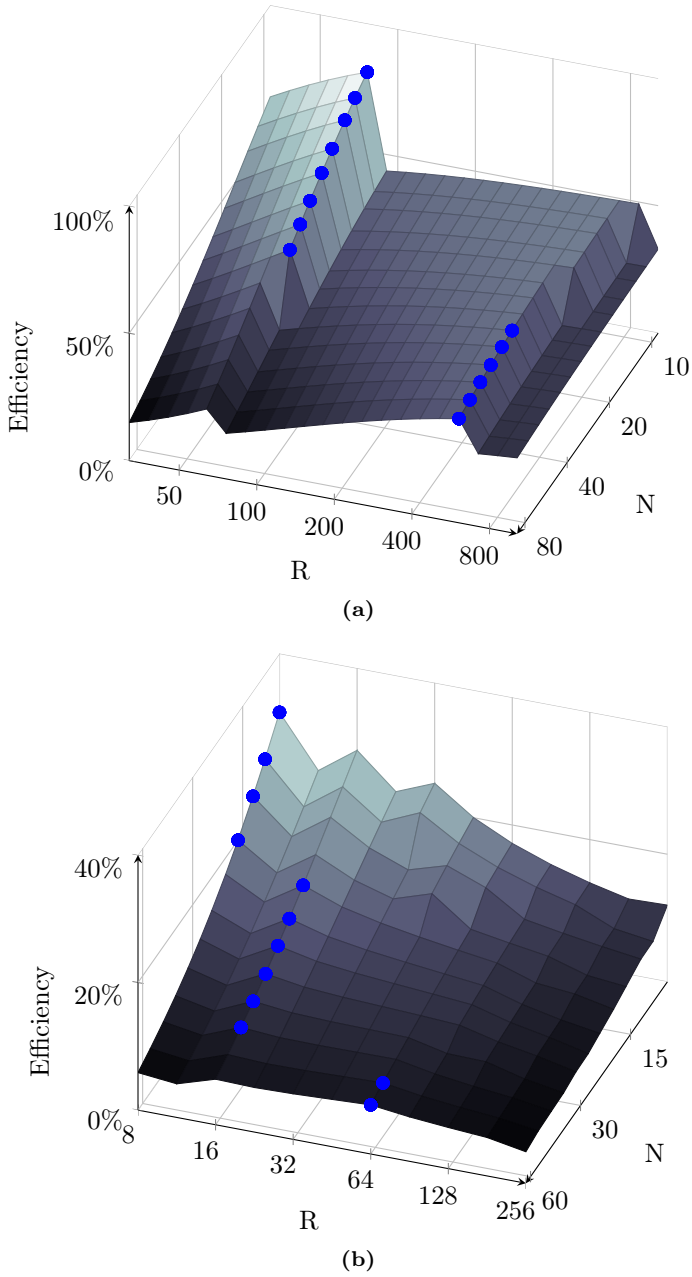
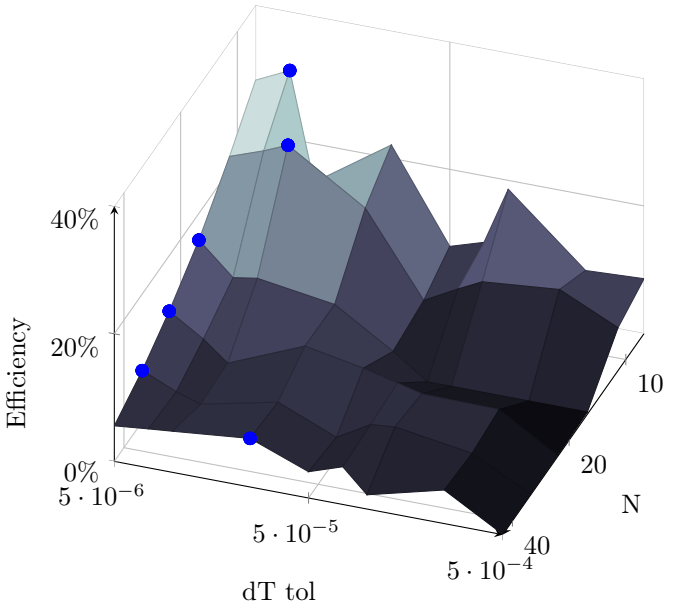
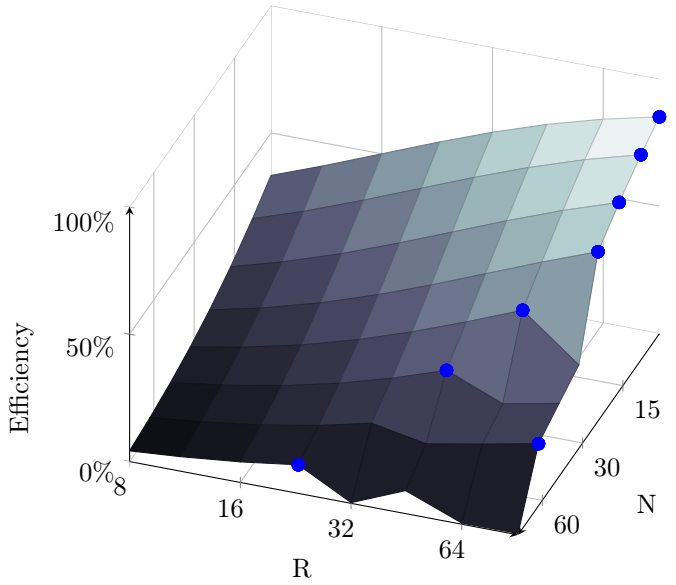


Figure 5.3: Predicted parallel efficiency using a fully distributed task scheduling model for *(a)* Bernoulli equation with forward euler *(b)* Linear ODE system with purely imaginary eigenvalues using backwards euler.



(a)



(b)

Figure 5.4: Predicted parallel efficiency using a fully distributed task scheduling model for *(a)* Van Der Pol oscillator with ESDIRK23 and asymptotic step-size control *(b)* Two dimensional diffusion equation with forcing term using a alternating direction implicit scheme.

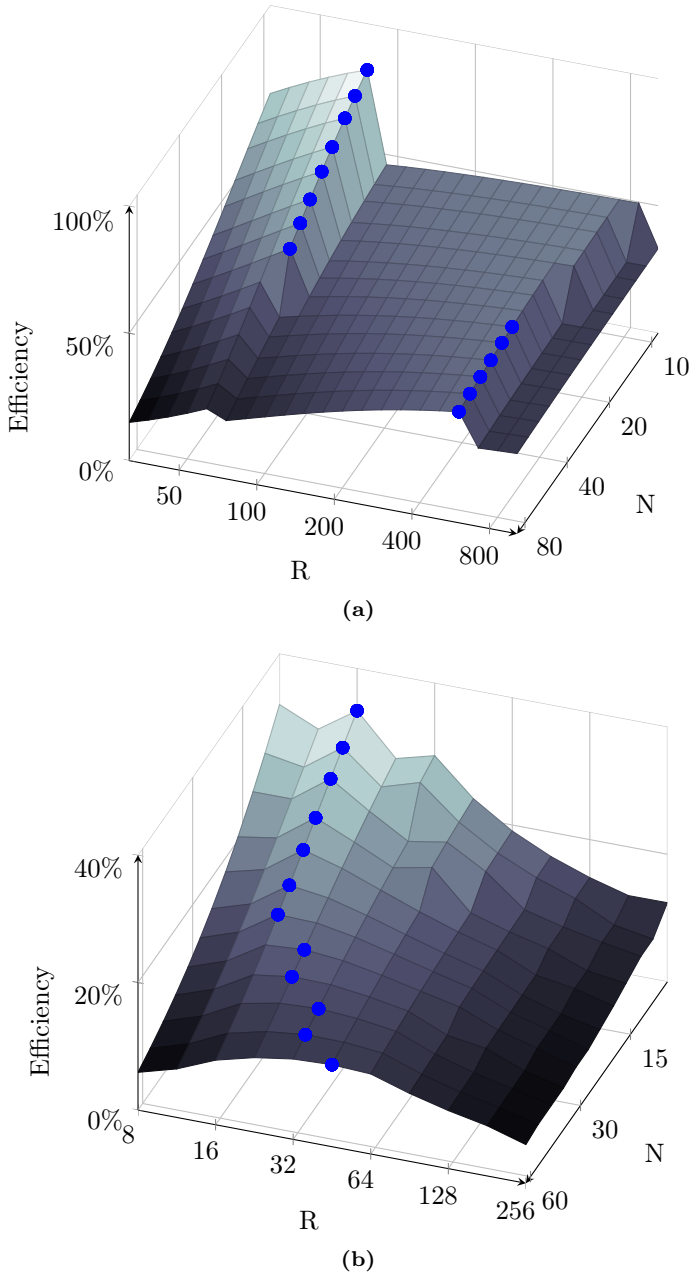
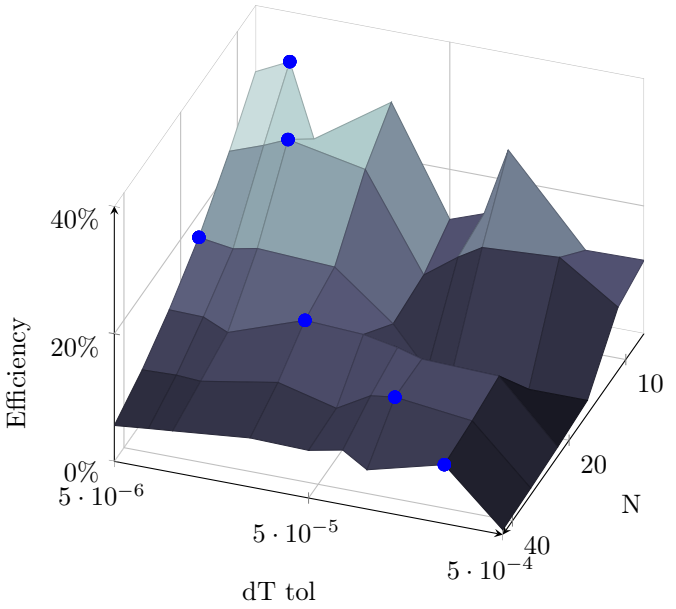
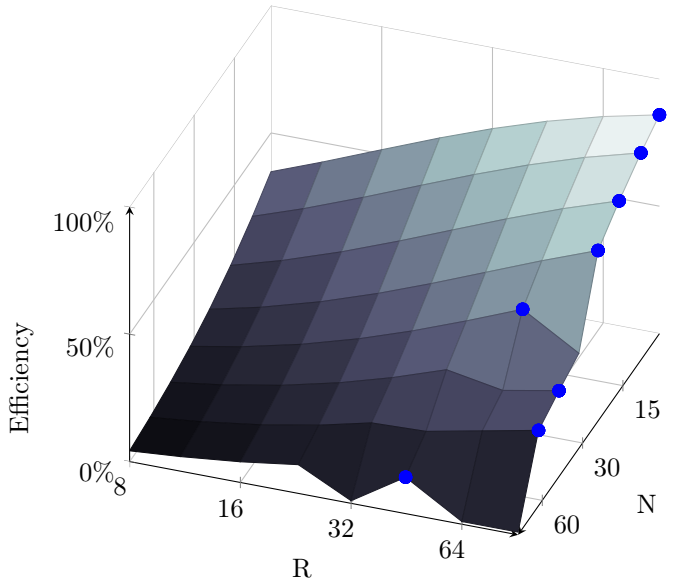


Figure 5.5: Predicted parallel efficiency using a manager-worker based task scheduling model for *(a)* Bernoulli equation with forward euler *(b)* Linear ODE system with purely imaginary eigenvalues using backwards euler.



(a)



(b)

Figure 5.6: Predicted parallel efficiency using a manager-worker based task scheduling model for *(a)* Van Der Pol oscillator with ESDIRK23 and asymptotic step-size control *(b)* Two dimensional diffusion equation with forcing term using a alternating direction implicit scheme.

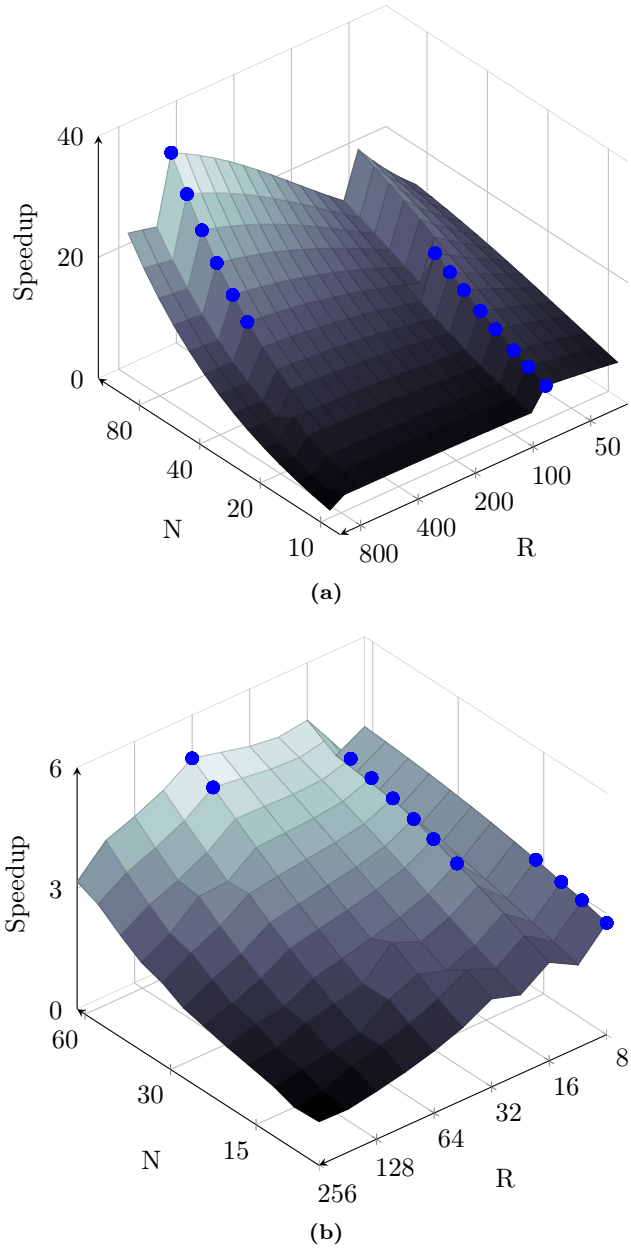
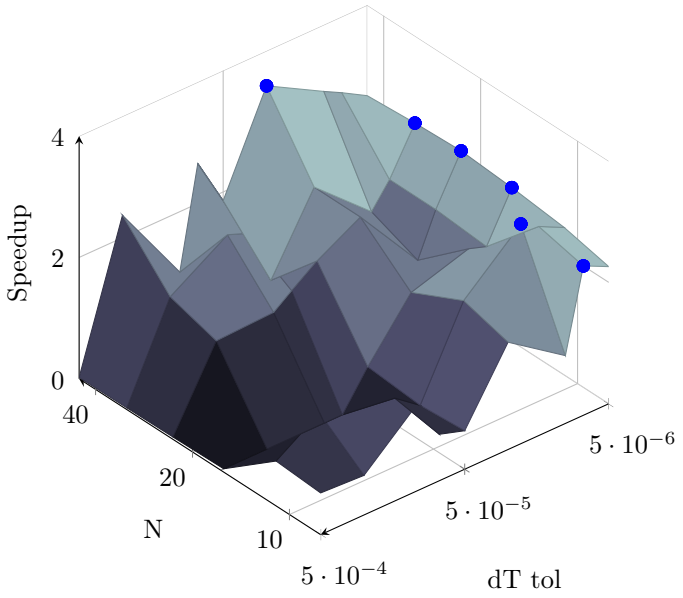
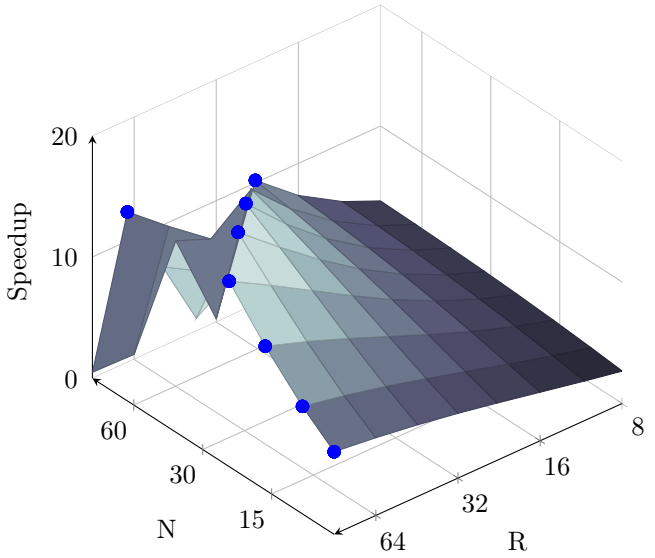


Figure 5.7: Predicted parallel speed-up using a fully distributed task scheduling model for *(a)* Bernoulli equation with forward euler *(b)* Linear ODE system with purely imaginary eigenvalues using backwards euler.



(a)



(b)

Figure 5.8: Predicted parallel speed-up using a fully distributed task scheduling model for *(a)* Van Der Pol oscillator with ESDIRK23 and asymptotic step-size control *(b)* Two dimensional diffusion equation with forcing term using a alternating direction implicit scheme.

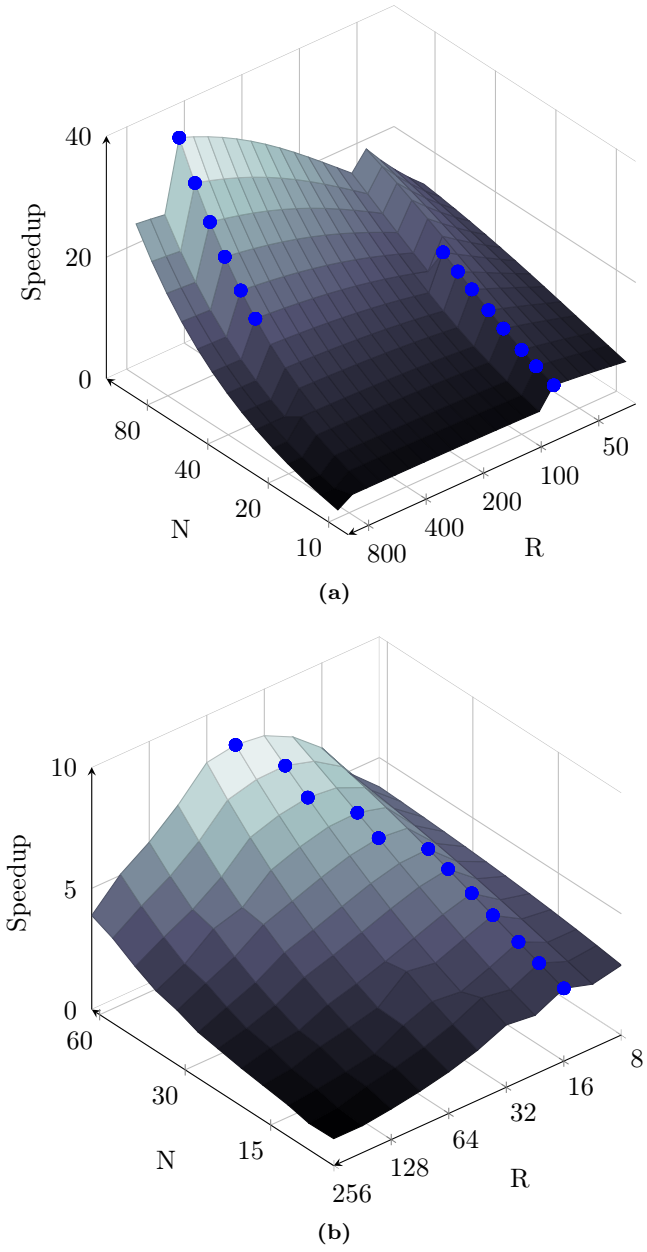


Figure 5.9: Predicted parallel speed-up using a manager-worker based task scheduling model for *(a)* Bernoulli equation with forward euler *(b)* Linear ODE system with purely imaginary eigenvalues using backwards euler.

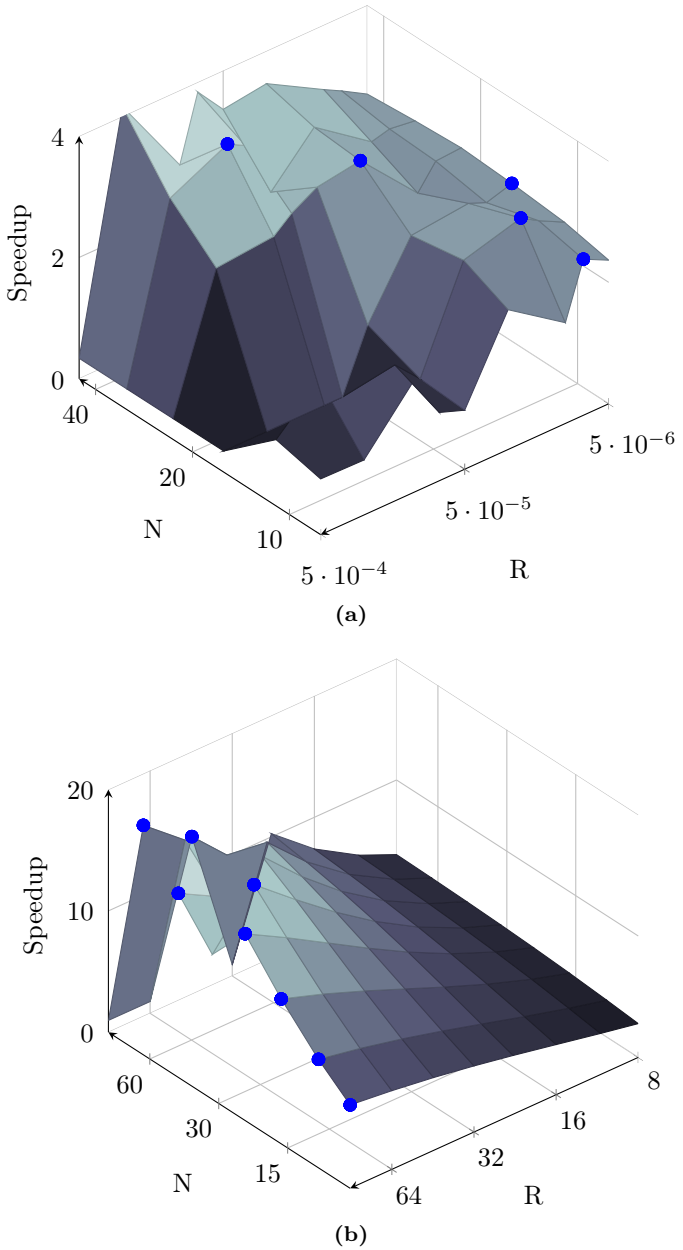


Figure 5.10: Predicted parallel speed-up using a manager-worker based task scheduling model for *(a)* Van Der Pol oscillator with ESDIRK23 and asymptotic step-size control *(b)* Two dimensional diffusion equation with forcing term using a alternating direction implicit scheme.

5.3 Summary

From the extensive survey presented in 5.2 many observations regarding the properties of the parareal algorithm can be made. In all figures the largest obtained parallel efficiency or speed-up for a given number of time-decompositions are marked with a blue dot as it is the behaviour of this optimum which is of interest. In this section we summarize and interpret on the results.

The results of the analysis on the Bernoulli equation are presented in figure 5.3a and 5.7a. For small N, choosing a small R so that the algorithm converges in a single iteration leads to the largest speed-up. We can interpret this as the dominant loss in parallel efficiency being the added computational work increasing with K. In the case of large N, the influence of the distribution of parallel work appear. For a large number of time-subdomains, it is seen to be more important to have a fast coarse propagator so that the idle time of compute units waiting for each other is minimized. As such, the optimum parallel efficiency is obtained for a larger R. Finally, we notice that given negligible communication and correction time, it should be possible to obtain a speed-up of almost 40 times, using 80 cores which is fairly impressive.

In the case of parareal applied to a system of linear ordinary differential equations with purely imaginary eigenvalues, we can make many of the same observations as with the application of parareal to the Bernoulli equation. One mayor difference though is how the efficiency decrease rapidly with increasing N due to a rapid increase in iterations needed by the algorithm to convergence for fixed R. In figure 5.7b the fully distributed task scheduling model has been used to predict speed-up, notice how efficiency degrade at a paste equivalent to the addition of new cores so that it is not really possible to accelerate the solution further by adding more time-decompositions. Using a manager-worker distribution model figure 5.9b improves the speed-up slightly, obtaining a speed-up of 8-9 using 60 compute units.

In figures 5.4a and 5.8a the results of the application of parareal to the Van der Pol oscillator with an ESDIRK23 scheme used as coarse and fine propagator with asymptotic stepsize controllers are presented. The predicted efficiency and speed-up does not follow the same smooth behaviour as previous investigations. This is not entirely unexpected as in the figure 4.5 the convergence was not particularly smooth either. Also, it seems to not be possible to do large scale acceleration. This is believed to be due to the discretization, as N becomes large the time sub-domains become smaller creating an upper limit to the time-step length and thereby limiting the effectiveness of the stepsize controller, so that the total number of used time-steps increased which adds to the computational burden of the parallel implementation and thereby degrade parallel efficiency.

Speed-up of around 4 seems viable.

The results of the application of parareal to the two dimensional diffusion equation 4.5 are presented in figure 5.4b and 5.8b. Surprisingly, we notice that as N increase, there is a tendency for small R to lead to optimal parallel efficiency! From the convergence investigations in section 4.5 we noted that for this discretization on the problem it appears that the algorithm either converges very fast or tend to diverge. Particularly for large k and N combined, the divergent behaviour is noticed. This suggest that a hierarchical structure may be appropriate for this particular equation and solver and we will simply have to accept the slow coarse propagator limitations on parallel efficiency. Despite these limitations, a fairly high speed-up is obtained of up to 15 using 30+ compute units.

The overall trend observed is that when a small number of compute units are applied for the acceleration of some fixed problem it is advisable to use an accurate coarse operator that converge very fast as the K times increase in computational workload will be the dominant factor in degradation of parallel efficiency. In the case of applying a large number of compute units to accelerate some fixed problem, it seems that the dominant loss in parallel efficiency will be in the distribution of parallel work. With a large number of compute units, much time is spend waiting for information to become available and as such, a fast coarse propagator is needed to minimize compute unit idle time and we must accept the increase in computational workload arising with the increasing number of iterations needed for convergence. These considerations seem to hold unless the combination of large K and N lead to a divergent behaviour of the solution, as was the case of the diffusion equation 4.5 with the ADI discretization. If this is the case, it might be better to implement parareal as a hierarchical structure as previously discussed.

Chapter 6

Application to nonlinear free surface flows

In chapter 4 and 5 the properties of the parareal algorithm on a range of simple differential equations was investigated. Due to the inherent challenging nature of extracting parallelism in the time domain and the derived added computational cost and communication by parareal, the algorithm is from a practical viewpoint only really interesting for larger compute intensive problems.

One such area of large-scale computing involves modelling of water waves in conjunction with ocean and coastal engineering. Such wave-models are of interest by engineers for predictive purposes in estimating flow kinematic and loads in designing offshore structures.

Recent development in the application of finite different methods to full potential flow theory has allowed for the efficient solution of nonlinear free surface waves that models the dispersive properties of water propagation at intermediate and deep waters well [21]. In [22] implementation on GPU hardware was demonstrated, showing speedup of an order of magnitude compared to that of otherwise efficient Fortran90 CPU single core code.

The purpose of this chapter and the following is to extend this work further by applying parareal as a wrap around so to enable the acceleration of the solution using multiple GPUs. In this chapter we first present the differen-

tial equation to be solved, as well as a brief overview on the finite difference based solver to be used. This presentation is followed by an investigation on the behaviour of the parareal algorithm to the full potential flow model where scalability, stability as well as convergence properties over the equation parameter space are measured in a Matlab implementation with simulated parallelism. For this purpose, Matlab code to initialize problem, generate analytical solution and integration in time of the wave model has been supplied by advisor Allan P. Engsig-Karup. In chapter 7 we build on the knowledge gained here and implement a full Cpp/CUDA/MPI model of parareal using GPUs as workers, measuring efficiency and speedup, comparing the results to that of of a classical domain decomposition approach.

6.1 Introducing the PDE system

The PDE system introduced below models free-surface potential flows above an uneven ocean bed. The model is derived assuming incompressible, inviscid and irrotational flow. We define the equations in a Cartesian coordinate system $(x, y, z) = (\mathbf{x}, z)$ with the xy-plane located at the still water level, see figure 6.1 In addition we define the still water depth $-h(\mathbf{x})$, as well as the wave perturbation (the free surface) $\mu(\mathbf{x}, t)$ and the potential $\phi(\mathbf{x}, z, t)$ of which the gradient $(u, v, \omega) = (\nabla, \partial_z)\phi$ is the fluid velocity defined on the entire domain of the fluid. $\nabla = (\partial_x, \partial_y)$ is the horizontal gradient operator. With these definitions the model takes on the form of two coupled partial differential equations

$$\partial_t \eta = -\nabla\eta \cdot \nabla\tilde{\phi} + \tilde{\omega}(1 + \nabla\eta \cdot \nabla\eta) \tag{6.1a}$$

$$\partial_t \tilde{\phi} = -g\eta - \frac{1}{2}\left(\nabla\tilde{\phi} \cdot \nabla\tilde{\phi} - \tilde{\omega}^2(1 + \nabla\eta \cdot \nabla\eta)\right) \tag{6.1b}$$

Where g is the gravitation constant $9.82 m/s^2$,$\tilde{\phi} = \phi(\mathbf{x}, \eta, t)$ and $\tilde{\omega} = \partial_z\phi|_{z=\eta}$. In order to integrate the equation so to find $\eta(\mathbf{x}, t)$ and $\phi(\mathbf{x}, \eta, t)$, we need to find $\tilde{\omega}$. But how so? In doing so we need to solve the Laplace problem so to find the potential in the entire domain. With kinematic boundary conditions at the vertical sides and at the bottom, the Laplace problem can be written as

$$\begin{aligned} \phi &= \tilde{\phi}, \;\; z = \eta, \\ \triangle\phi + \partial_{zz}\phi &= 0, \;\; -h \leq z < \eta \\ \mathbf{n} \cdot (\nabla, \partial_z)\phi &= 0, \;\; (\mathbf{x}, z) \in \partial\Omega \end{aligned}$$

with $\mathbf{n} = (n_x, n_y, n_z)$ being the outward pointing normal vector to the domain boundary surfaces, $\partial\Omega$. As defined above, the boundary conditions in the problem are moving and thus time-dependent, by employing a transform of $\eta(\mathbf{x}, t)$

in the vertical coordinate as proposed in [40] the problem can be transformed into one with a fixed domain, see figure 6.2 The vertical transform is defined as

$$\sigma = \frac{z + h\left(\mathbf{x}\right)}{\eta\left(\mathbf{x}, t\right) + h\left(\mathbf{x}\right)}, \ 0 \leq \sigma \leq 1 \tag{6.2}$$

With the above transform, the Laplace problem takes on the form

$$\Phi = \tilde{\phi} \tag{6.3a}$$

$$\nabla^2\Phi + \nabla^2\sigma\left(\partial_\sigma\Phi\right) + 2\nabla\sigma\cdot\nabla\sigma\left(\partial_\sigma\Phi\right) + \left(\nabla\sigma\cdot\nabla\sigma + \left(\partial_z\sigma\right)^2\right)\partial_{\sigma\sigma}\Phi = 0 \tag{6.3b}$$

$$\mathbf{n}\cdot\left(\nabla, \partial_z\sigma\partial_\sigma\right)\phi = 0 \tag{6.3c}$$

With 6.3a for $\sigma = 1$, 6.3b for $0 \leq \sigma < 1$ and 6.3c for $(\mathbf{x}, \sigma) \in \partial\Omega$. Furthermore, when Φ is known, the vertical velocity can be determined from

$$\omega = \partial_z\phi = \partial_z\sigma\partial_\sigma\Phi \tag{6.4}$$

All numerical tests throughout this chapter and the following are based on the solution of two dimensional waves. We expect the results to be qualitatively similar to what would be obtained solving three dimensional waves. The discretization approach is based on a flexible-order finite difference method which have been demonstrated to be robust, efficient, and accurate, enabling numerical solution of a broad range of important applications. A complete analysis and experimental validation can be found in [21] and [10]. For the discretization a free surface grid consisting of N_x points are defined along the horizontal axis where the surface variables $\tilde{\phi}$ and η are to be computed. In addition to the horizontal points N_x, N_z points are defined in the vertical direction below each horizontal point at the free surface, so to solve the transformed Laplace problem 6.3. The N_z points can be arbitrarily spaced, however for all investigations in this report we use a uniform grid. Choosing nearby points in the x-direction allows finite difference schemes for the one-dimensional first and second derivatives in (x, σ) to be developed for each of the x and σ positions on the grid. The resulting discrete Laplace problem is a linear system of equations $\mathcal{A}\mathbf{\Phi} = \mathbf{b}$ where $\mathcal{A}$ of rank $n = N_x \cdot N_z$. $\mathcal{A}$ is in general a large sparse non-symmetric matrix and $\mathbf{\Phi}$ a vector of values for the unknown scalar velocity potential to be found while $\mathbf{b}$ is a vector accounting for in-homogenous boundary conditions. For the time-integration of the free-surface conditions in 6.1, an explicit four-stage fourth order Runge-Kutta (ERK4) is used as the fine propegator. The ERK4 and several other integrators are tested in section 6.3 for their applicability as a coarse propagator for the parareal algorithm. An important property of the nonlinear pde system 6.1 introduced is that under certain ideal circumstances with fixed constant water depth it is possible to construct an analytical solution to the equations. In constructing an analytical solution to the equations a number of parameters are needed. Aside from physical constants such as the gravitation

constant, the code requires a parameter $k \cdot h > 0$, where $k = 2 \cdot \pi / L$ is the wave number, L being the wave length, multiplied with the still-water level depth h. The quantity typically characterize the dispersion properties of the problem. In addition to $k \cdot h$, a dimensionless parameter H/L is needed. H/L is expressed as the relationship between wave hight measured from wavetop to wavebottom and wavelength L. H/L is passed as a dimensionless parameter between 0 and 100% of the limit of wave breaking $(H/L)_{max}$. As such H/L can also be seen as a measure of the amount of energy present in the wave. Small H/L means that surface elevation is negligible, corresponding to linear waves, so the parameter is also a measure of how non-linear the problem is.

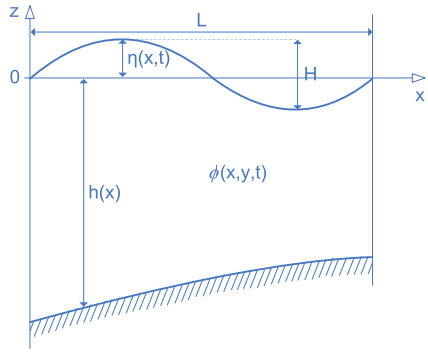


Figure 6.1: Definitions for the derivation of 6.1. The xy-plane is located at still water level and the water depth is defined as $-h(\mathbf{x})$ with the free surface perturbation given by $\eta(\mathbf{x}, t)$ and the potential $\phi(\mathbf{x}, z, t)$. Figure by Stefan L. Glimberg

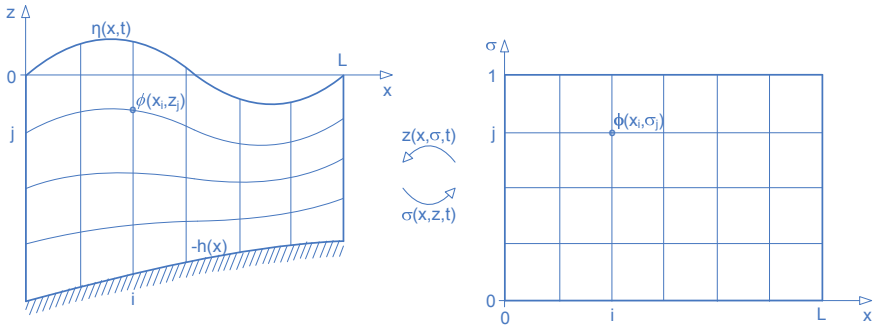


Figure 6.2: A visual presentation of the transform of $\eta(\mathbf{x}, t)$ in the vertical coordinate by 6.2. Figure by Stefan L. Glimberg

6.2 Weak and Strong Scaling

One of the often mentioned advantages of the parareal algorithm over other methods of extracting parallelism in the time domain, such as for example parallel Runge-Kutta integrators, is that there is no natural upper bound to the number of independent compute units that can be applied to the algorithm aside from the fineness of integration. However, since the efficiency and therefore speed-up scales inversely with the number of iterations needed to convergence, it is of great interest to know how the iteration count scales both weakly and strongly with the number of predictor-corrector intervals that can be independently scheduled to different compute units. In chapter 4 the convergence of a range of differential differential problems was visualized and it was deducted that for $K = 1$ and potentially $K = 2$ it is possible to observe linear strong and weak scaling of the algorithm. Here a few similar tests are performed on the nonlinear wave model.

Figures 6.3a and 6.3b show weak scaling results of a matlab implementation with simulated parallelism. The results are for intermediate depth and deep water respectively. The time per predictor-corrector interval in the parareal algorithm is kept constant at 1/16 wave periods, and the number of iterations to convergence are measured integrating over 2,4,8,...,128 intervals. Thus, the longest integration consists of eight wave periods using 128 predictor-corrector intervals which means that up to 128 compute units could be added in the acceleration of the parallel part of the parareal algorithm. A spacial discretization of $N_x = N_z = 25$ is used with a finestep calculated from a courant number 0.25. In the integration, RK4 is used as both the coarse and the fine propagator, with $R = 8 : 1$ being the ratio between the time of the fine propagator to the time of the coarse propagator to integrate over given a fixed interval.

Empirical strong-scaling results are presented in figures 6.4a and 6.4b for intermediate and deep waters respectively. The total integration time is now fixed at 8 wave periods, and instead the time per predictor/corrector interval various with the number of predictor/corrector intervals assigned. The iterations to convergence is again measured for 2,4,8,...128 intervals.

We notice that for the given choice of coarse propagator accuracy we obtain what looks to be linear strong and weak scaling in a purely algorithmic sense for the deep water wave. For the intermediate depth water wave, the number of iterations needed for convergence of the algorithm increase with the number of intervals regardless of whether the time-interval that is integrated is scaled with the number of intervals or kept fixed. Despite the lack of scaling in the case of intermediate depth water, the number of iterations needed for convergence seem to only grow slowly with the number of intervals. In the worst case observed,

the number of iterations to convergence go from 2 to 6 when moving from 2 to 128 intervals with a fixed integration time.

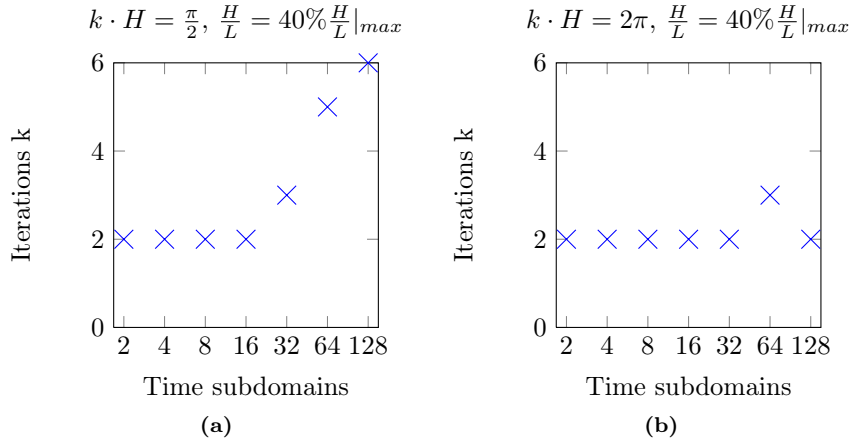


Figure 6.3: Weak scaling. Fixed timestep length ΔT, integration time T increase with the number of time subdomains N

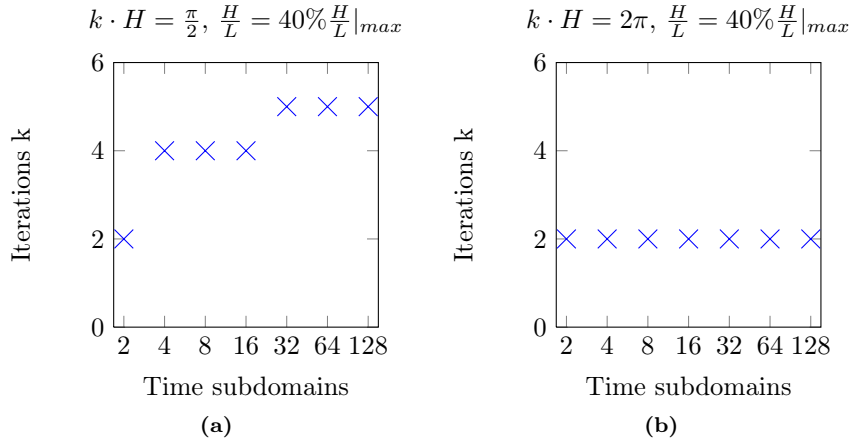


Figure 6.4: Strong scaling. Fixed total integration time T, timestep length ΔT decrease with number of time subdomains N

6.3 Stability of long time integration

The PDE system 6.1 presented is of hyperbolic nature. This is a course of concern regarding the use of the Parareal method to solve the equation. Numerous studies have shown that instabilities may arise in the numerical solution of hyperbolic equations accelerated by parareal [14] [23] [25] [49].

From a theoretical viewpoint the issue has also received a fair amounts of attention. In [63] stability conditions for parareal applied to the solution of a system of linear coupled ordinary differential equations was derived and it was showed that for systems with pure imaginary eigenvalues or complex eigenvalues, where the imaginary part much larger than real part, it is difficult to guarantee stability of the algorithm for all number of intervals and all iterations, suggesting that hyperbolic and convection diffusion problems with highly dominant convection will experience instabilities. In addition, in [6] it was shown that for linear partial differential equations with constant coefficient, parareal is unconditionally stable for most discritizations of parabolic equations but not so for hyperbolic equations.

Due to the potential convergence and stability issues of the parareal method applied to hyperbolic problems, we test and measure convergence of the parareal algorithm applied to the problem 6.1 under long time integration for various parameter combinations so to establish whether to expect the parareal algorithm to be stable and whether to expect good convergence properties for the fully nonlinear wave problem.

As in the previous section we implement the algorithm using an explicit four stage fourth order runge-kutta method as both the fine and the coarse propagator and simply using a longer timestep for the coarse propagator. In each of the figures 6.5, 6.6 and 6.7 six different combinations of wave parameters are used, $\frac{H}{L} = \{10\%, 40\%, 70\%\}\, \frac{H}{L}|_{max}$, with $\frac{H}{L}|_{max}$ being the ratio at which the wave brakes, and $kh = \{\pi/4, 2\pi\}$, which corresponds to shallow/intermediate and deep water respectively. In each figure a different coarse to fine timestep ratio is used. The ratios $r = \{4, 8, 12\}$, may seem fairly low, but they are chosen as a consequence of the investigations in chapter 4 that pointed to the highest speed-up being obtained with fast convergence in few iterations. The apparent reason for the later being that the upper bound of the numerical efficiency scales as $1/k$, so it seem to often be better to use a fairly accurate solver so to converge in as few iterations as possible, ideally during the first corrector-predictor update.

In all figures, the algorithm is used to integrate a single wave resolved with $N_x \times N_z = 65 \times 65$ points over 100 waves periods, each predictor-corrector

interval being the length of 1 wave period. Thus, in essence the measurements can also be interpreted as a larger test of the weak scaling properties of the parareal algorithm, as the number of cores that can be assigned to compute increase with the number of wave-periods to be integrated.

In figures 6.5 to 6.7, the dashed black bold line indicates the error measured using a purely sequential fine propagator. The blue solid line is the error of the initial coarse prediction, iteration zero. The lines that follows, i.e., green, red, cyan, magenta, yellow and grey are depicting iteration 1,2,...,6 respectively. Doing only six iterations seems to be sufficient to uncover the qualitative behaviour under various parameter combinations.

The use of 65 points of resolution in the z direction is more than needed for practical purposes, particularly for shallow and intermediate depth water it is possible to use fare fewer points with a limited impact on accuracy. However, in order to make sure that errors are completely dominated by the difference in timestep length only, we rule out other factors such as the N_z resolution dependency that should otherwise be tuned and minimized as much as possible.

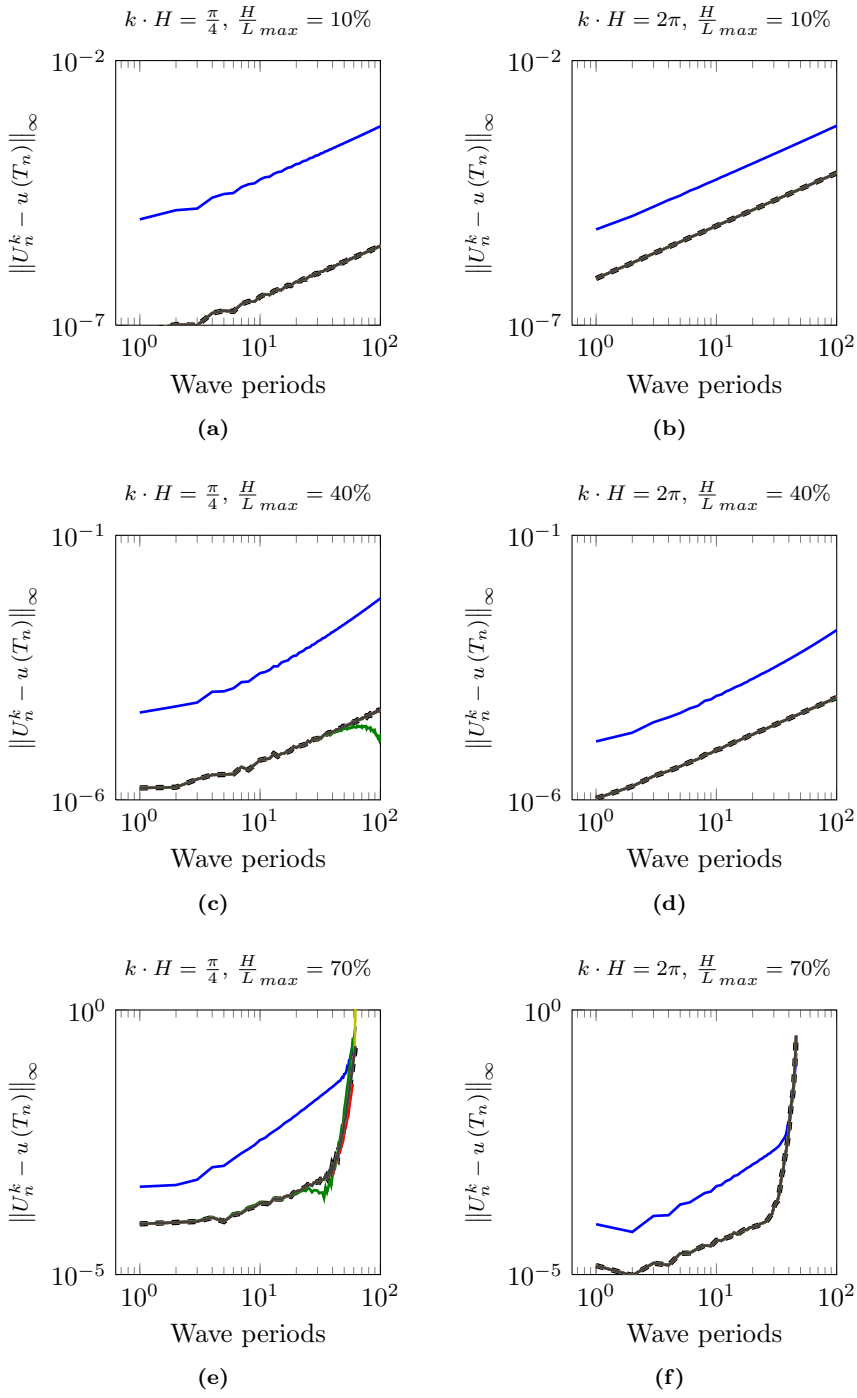


Figure 6.5: Long time integration using RK4 as $\mathcal{G}_{\Delta T}$ with $R = 4$

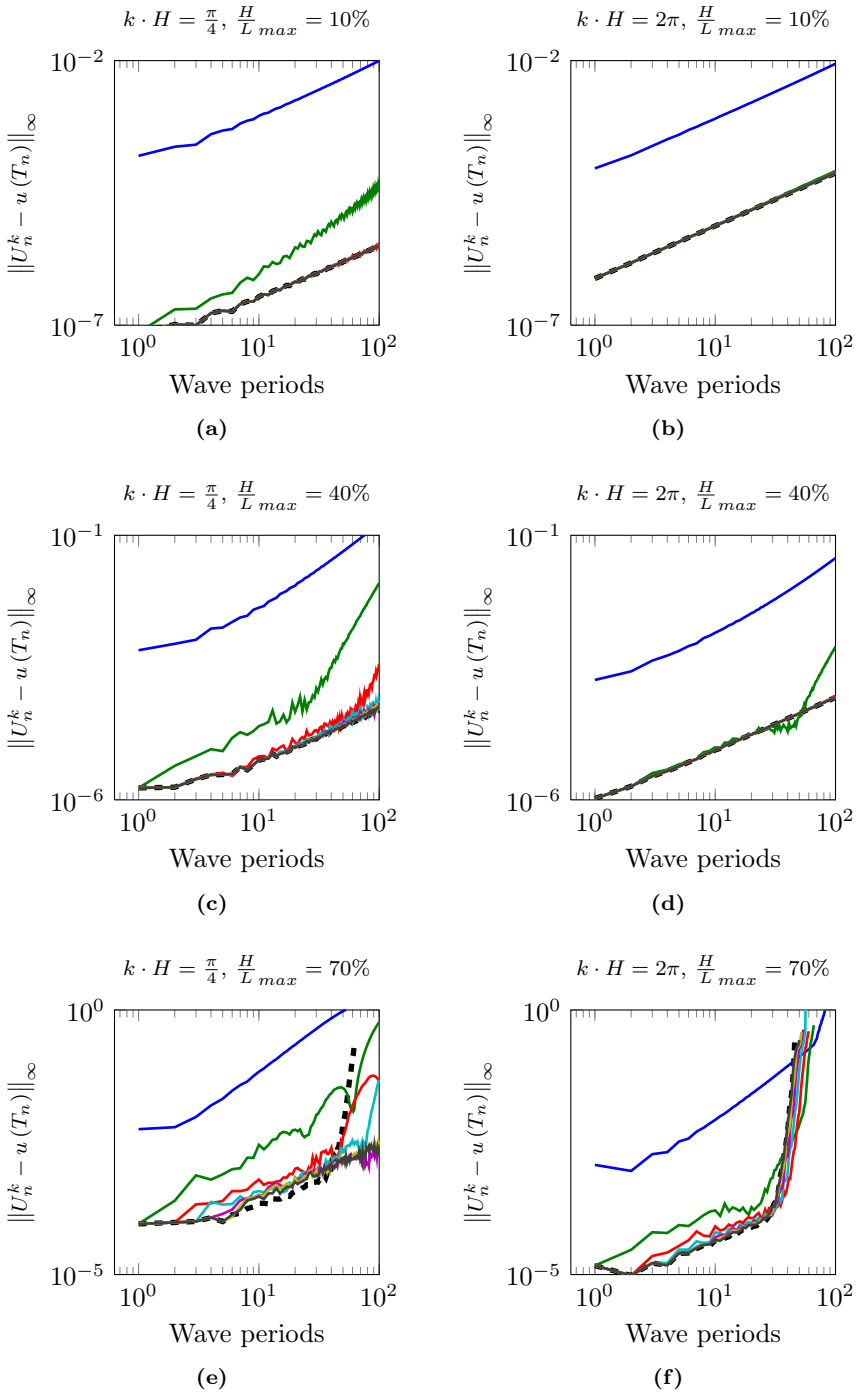


Figure 6.6: Long time integration using RK4 as $\mathcal{G}_{\Delta T}$ with $R = 8$

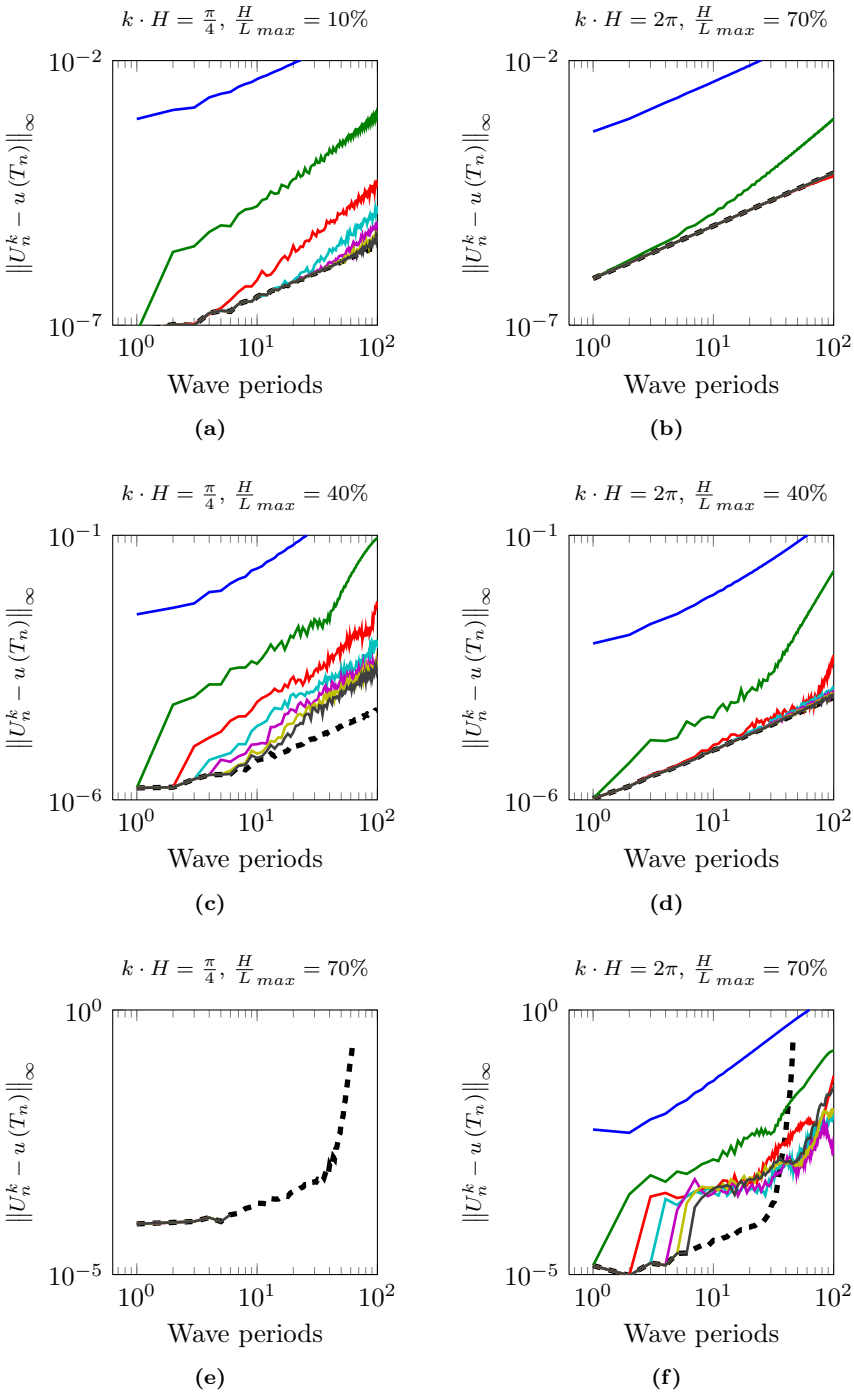


Figure 6.7: Long time integration using RK4 as $\mathcal{G}_{\Delta T}$ with $R = 12$

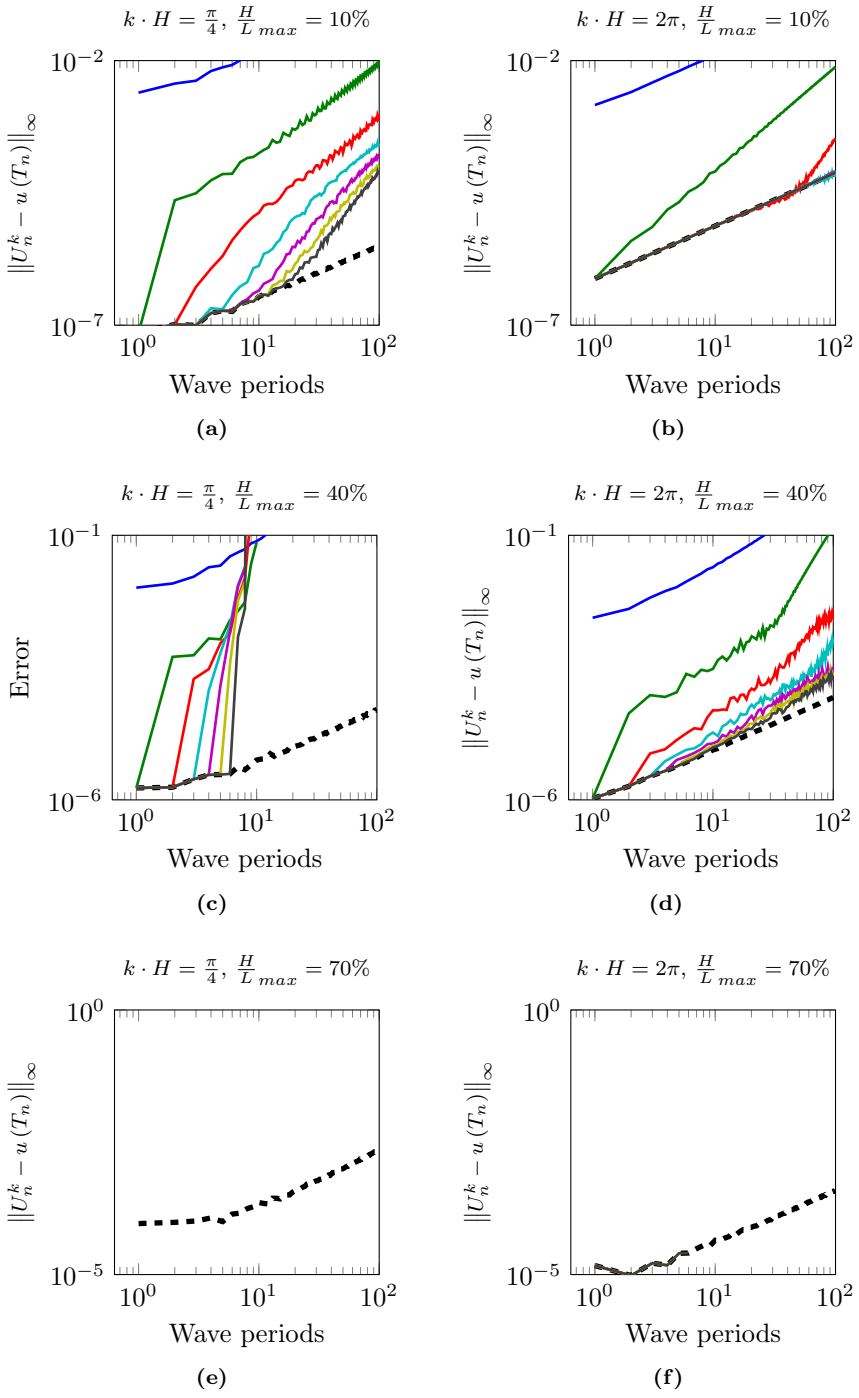


Figure 6.8: Long time integration using RK4 as $\mathcal{G}_{\Delta T}$ with $R = 16$ and Savitzky-Golay filter applied.

With a fine, and thus slow, coarse propagator we observe instant convergence by the first iteration to 100 wave periods for all parameter combinations in figure 6.5. Besides the fast convergence properties, it is also worth noticing how the purely sequential fine propagator becomes unstable after around 30-40 wave periods and the error increases rapidly for large $\frac{H}{L}$ waves in figure 6.5e and 6.5f. Further investigations into this by visualising the wave profile leads to the conclusion that high frequency modes rise and these eventually break down information in the wave profile.

Despite good convergence results, the fine to coarse propagator time consumption is $R = 4$, which is fairly slow and likely to lead to low efficiency as compute units will be waiting for one another throughout large parts of the algorithm, and as such may not be of much interest.

In figure 6.6 results are presented using a faster coarse solver with ratio $R = 8$. This has a number of implications to the convergence speed of the algorithm. Only the deep water wave with small wave amplitudes converge instantly over the entire 100 wave period integration. It also becomes clear that the algorithm converge faster on deep water than on shallow/intermediate depth water and in addition increasing energy and wave amplitude seems to degrade the speed of convergence

A particularly interesting thing to note here is what happens in figure 6.6e. As previously noted the purely sequential solver develop instabilities due to the introduction of high frequency modes when integrating high energy waves. What is striking here is how the parareal algorithm actually works to stabilize the solution, and eventually make it converge in less than 6 iterations without the introduction of an otherwise necessary filtering in each integration step! Fairly impressive. The same is not observed when integrating deep water waves with the parareal algorithm shown in figure 6.6f, here the parareal algorithm solution also develop high frequency modes that destabilize the solution.

The results for fine to coarse ratio $R = 12$ presented in figure 6.7 generally yield the same overall observations as what can be made from figure 6.6 for $R = 8$. As expected slower convergence is happening, but that is expected, given a less accurate coarse propagator. The most important thing to note here is that in the figures 6.7c, 6.7e and 6.7f we notice beginning instabilities arising where progressive iterations actually deteriorate the solution as found in paper, similar to what was demonstrated in [18] on applying parareal to a linear wave problem and the viscous burgers problems. However, we take note that for sufficiently accurate solver the algorithm will converge however slowly. A coarse to fine time ratio of $R = 12$ is still a lot less than what we would really like to wish for, but decreasing the accuracy of the coarse operator further seems to make the algorithm diverge. For the sake of experiments, we try with ratio $R = 16$

while also applying a Savitzky-Golay [57] smoothing filter after each iteration to try and filter out high frequencies that are introduced in the solution, error measurements are presented in figure 6.8. In the figures 6.8e and 6.8e the very first iteration is not stable for the choice of coarse operator and so the parareal algorithm never initializes. The filter is seen to help in the sense that the fine solution is now stable for all parameter choices throughout the integration of 100 waves, but the filter does not seem to improve the convergence properties of the parareal scheme.

Comparing the convergence results to that of the simple equations investigated in chapter 4, it seems that the problem 6.1 requires a very accurate coarse solver in order to converge. With such a "slow" fast solver, it is imperative that the parallel code is scheduled efficiently to avoid idle cores as much as possible. It is worth investigating ways of increasing the speed of the coarse solver as the speed of the coarse solver may prove to be the biggest challenge in obtaining a decent acceleration of the integration. A possible route to a faster coarse integrator could be to choose a different integrator more suitable when only low accuracy is needed.

In the four stage fourth order explicit Runge-Kutta method used as both the coarse and fine solver in all previous test, the Laplace problem, which is expensive to solve, is evaluated once at every stage. Replacing the ERK4 integrator with a three stage third order Runge-Kutta means only 3/4 as much work per timestep as the ERK4 solver and may prove to be more effective numerically for low accuracy solutions. The impact being that a ratio of say $R = 12$ between timesteps would actually correspond to a ratio of $R = 16$. The figures 6.9 and 6.10 contain convergence results of long time integration using an ERK3 integrator for the coarse propagation and an ERK4 integrator for the fine propagation. Unfortunately it seems that the gain in fine to coarse ratio using fewer stages in each integration step are offset by a need for finer timesteps of the coarse solver to obtain the same convergence rate, roughly cancelling out any speed benefits of the coarse propagator.

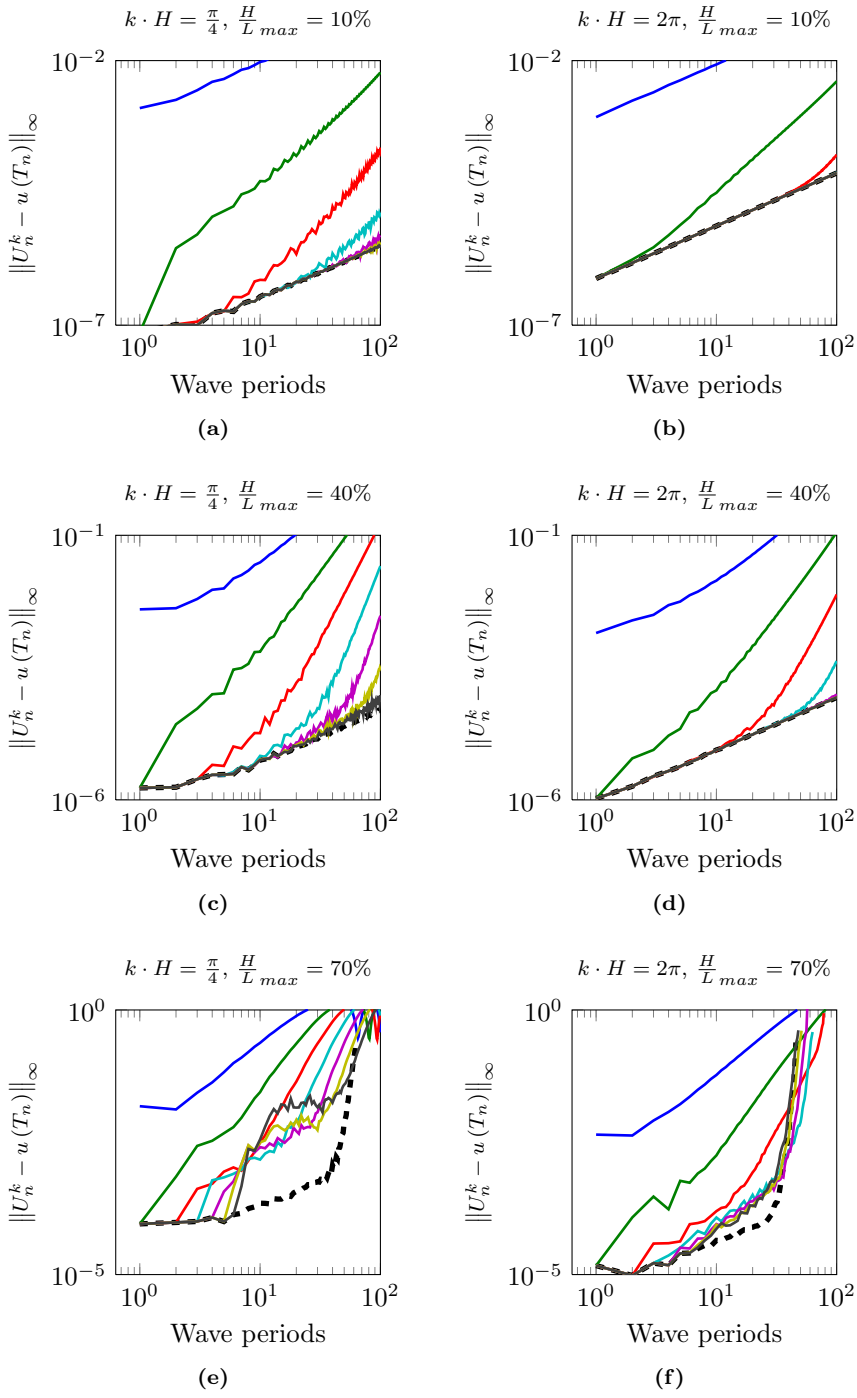


Figure 6.9: Long time integration using RK3 as $\mathcal{G}_{\Delta T}$ with $R = 6(8)$

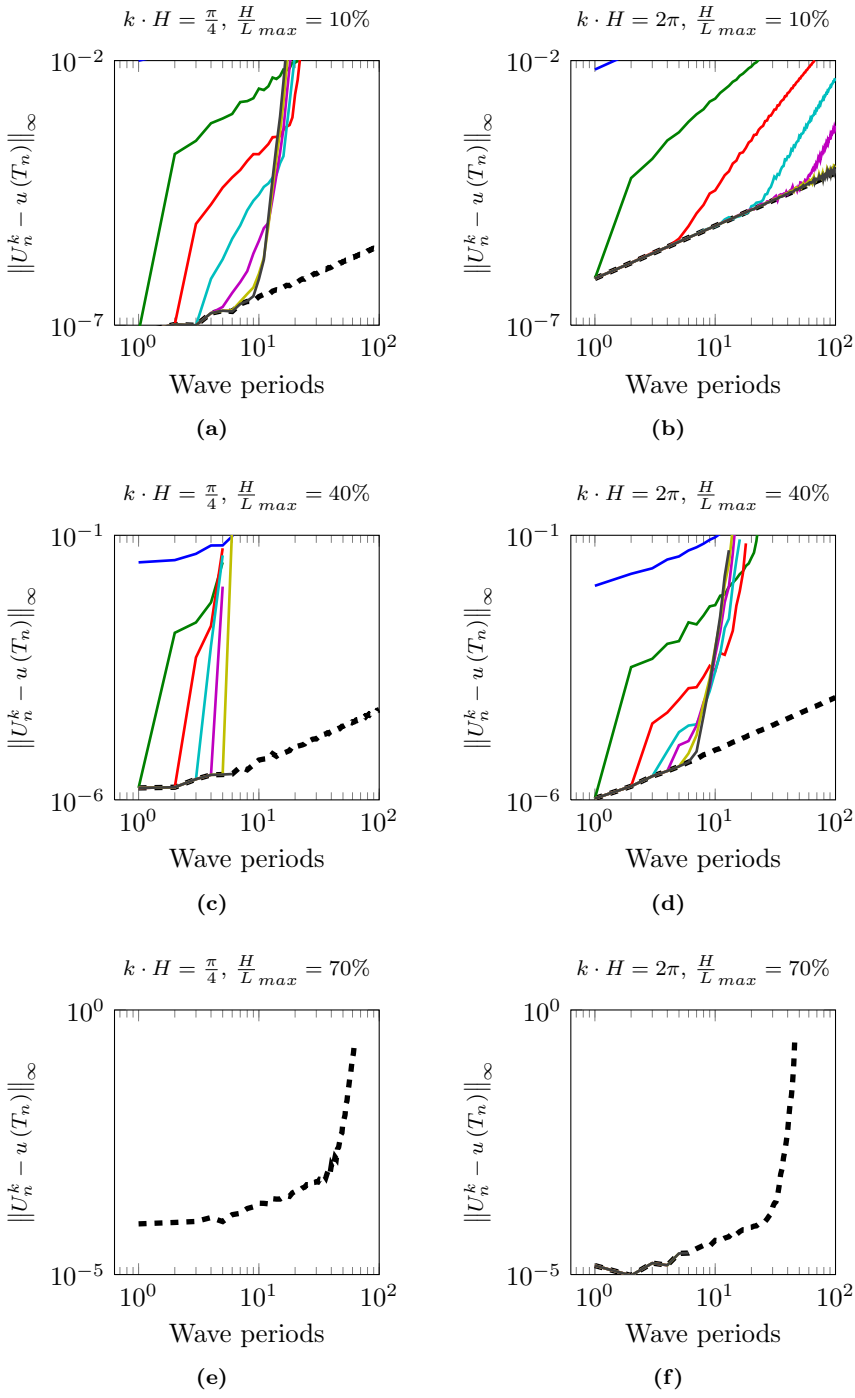


Figure 6.10: Long time integration using RK3 as $\mathcal{G}_{\Delta T}$ with $R = 12(16)$

In the hunt for a faster coarse integrator, a few experiment where performed using a coarse spacial grid as discussed in section 3.5, the idea is here to have to coarse solver simply solve on a coarse spacial grid and then use some operator to interpolate from one space to another. Unfortunately the usage of a coarse grid did not work out well. The algorithm showed divergent behaviour for all discretizations, it seems that the information lost in the interpolation in general made the algorithm diverge before converging in $k = N$ iterations.

As a final attempt at accelerating the coarse propagator we experiment with a combined explicit implicit propagator Runge Kutta propagator. A commonly used second order accurate 3-stage diagonally implicit Runga Kutta method is implemented with a slight modification, in each stage the Laplace problem to get $\tilde{\omega}$ is evaluated *explicitly* followed by an *implicit* integration of 6.1. Long time integration results for this approach are presented in 6.11 for $R = 6(8)$ and in 6.12 for $R = 12(16)$. If we compare the resulting long time integration presented in figure 6.12b and 6.12a for $R = 12(16)$ with the results using RK4 as coarse propagator for $R = 16$ presented in the figures 6.8c and 6.8d we see vast improvements as the parareal solution now converges in each step in the case of shallow/intermediate depth water. For $R = 6(8)$ the application of a modified DIRK method as a coarse propagator leads to a slower convergence rate than the equivalent implementation using the RK4 method as a coarse propagator.

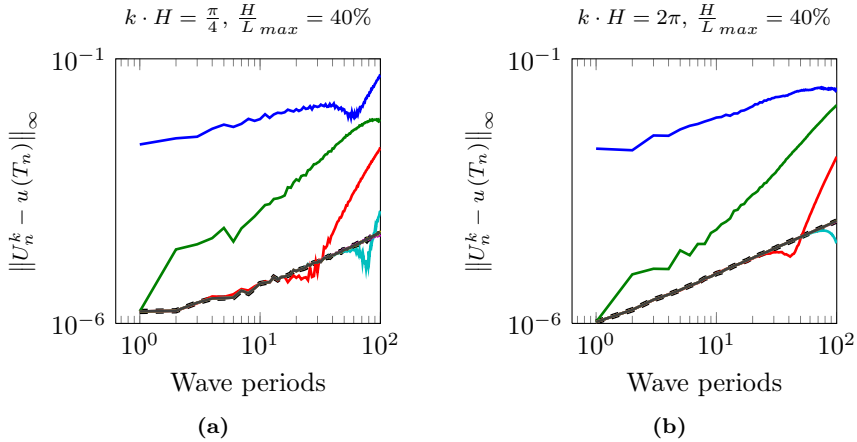


Figure 6.11: Long time integration using modified DIRK2 as $\mathcal{G}_{\Delta T}$ with $R = 6(8)$

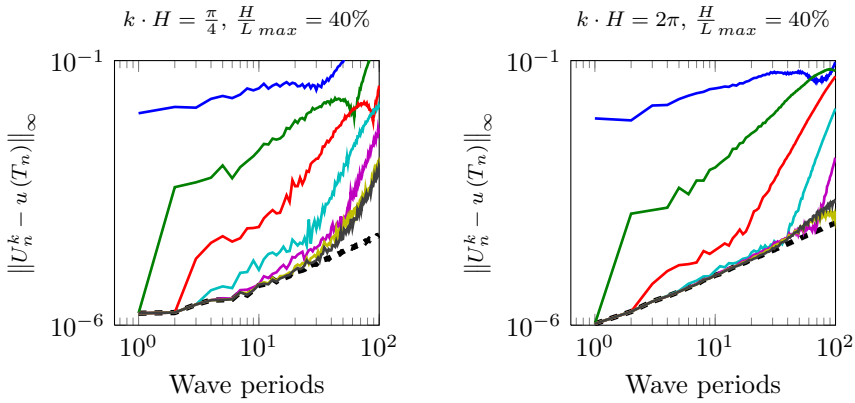


Figure 6.12: Long time integration using modified DIRK2 as $\mathcal{G}_{\Delta T}$ with $R = 12(16)$

6.4 Convergence speed on parameter space

In building an initial condition for the numerical solution of the system 6.1 of two coupled partial differential, a number of wave parameters arise. From the long-time integration tests in previous sections it became clear that the rate of convergence is heavily dependent on these physical parameters such as water depth and energy in wave. In this section we measure the number of iteration required for convergence over a parameter space spanned by $h = \{0.1, 0.2, ..., 1\}$, in a $1m$ long tub with $1m$ long waves so that $kh = \{0.2\pi, 0.4\pi, ..., 2\pi\}$ and $\frac{H}{L} = \{10\%, 20\%, .., 80\%\}\frac{H}{L}_{max}$ as fast convergence is paramount in obtaining speedup. The number of iterations to convergence is independent of how the parallel work is being scheduled, and all tests are thus carried out in a Matlab implementation with simulated parallelism. The convergence criteria is kept strict at 1.1 times the error of the reference solution using a purely sequential fine integrator only. See figure 6.13, strictly sequential fine error as a function of parameter space, used as convergence criteria. This is of course for practical applications not a viable way of determining convergence, and a number of practical methods have been proposed such as [39] which could be implemented using a Runge-Kutta method with embedded error estimation method. However, for simplicity we focus only on whether or not convergence has happened, and not on the various approaches of determining so.

For each integration in the parameter space, a single wave resolved by $\{N_x, N_z\} = \{49, 49\}$ is integrated over two wave periods. Again we use a large N_z to make sure that the error is completely dominated by the time resolution under all circumstances. The finestep is calculated from a Courant number of 0.25. The same integrator is used for both the coarse and fine propagator so that the coarse timestep is simply R times bigger than the fine step. The integration over the parameter space is done for six different combinations of $R = \{4, 8, 12\}$ and $N = \{4, 16\}$ with N being the number of time subdomains. The results of these measurements are presented in figure 6.14. As was indicated in section 6.3 figure 6.5 for long time integrations, a fine to coarse time ratio of $R = 4$ yields instant converge over the entire parameter space as depicted in figure 6.14a and 6.14b. Again we notice how increasing the timestep length of the coarse solver decrease the speed of convergence. Also, it is quite obvious that the parareal algorithm shows slower convergence for shallow water and large amplitude waves and as such, parareal may not be suitable for the acceleration of the solution of such systems. It is not obvious what choice of fine to coarse timestep solver ratio leads to the largest speedup. The question is a matter of trade-off between fast convergence using an expensive coarse solver, or slower convergence using a faster but less accurate solver. A slow coarse solver will lead to compute units spending a lot of time waiting for each other leading to low efficiency, on the other hand with fast solvers but many iteration to convergence, more compute work needs to be done since the entire problem is to be solved k times. To evaluate, or

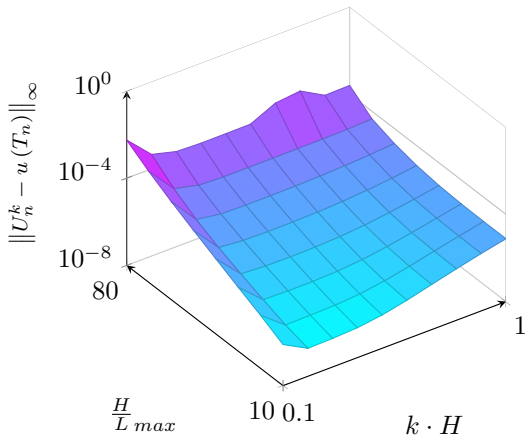


Figure 6.13: The error at the final timestep of the integration of a single wave over two wave periods using an ERK4 integrator in a strictly sequential fashion.

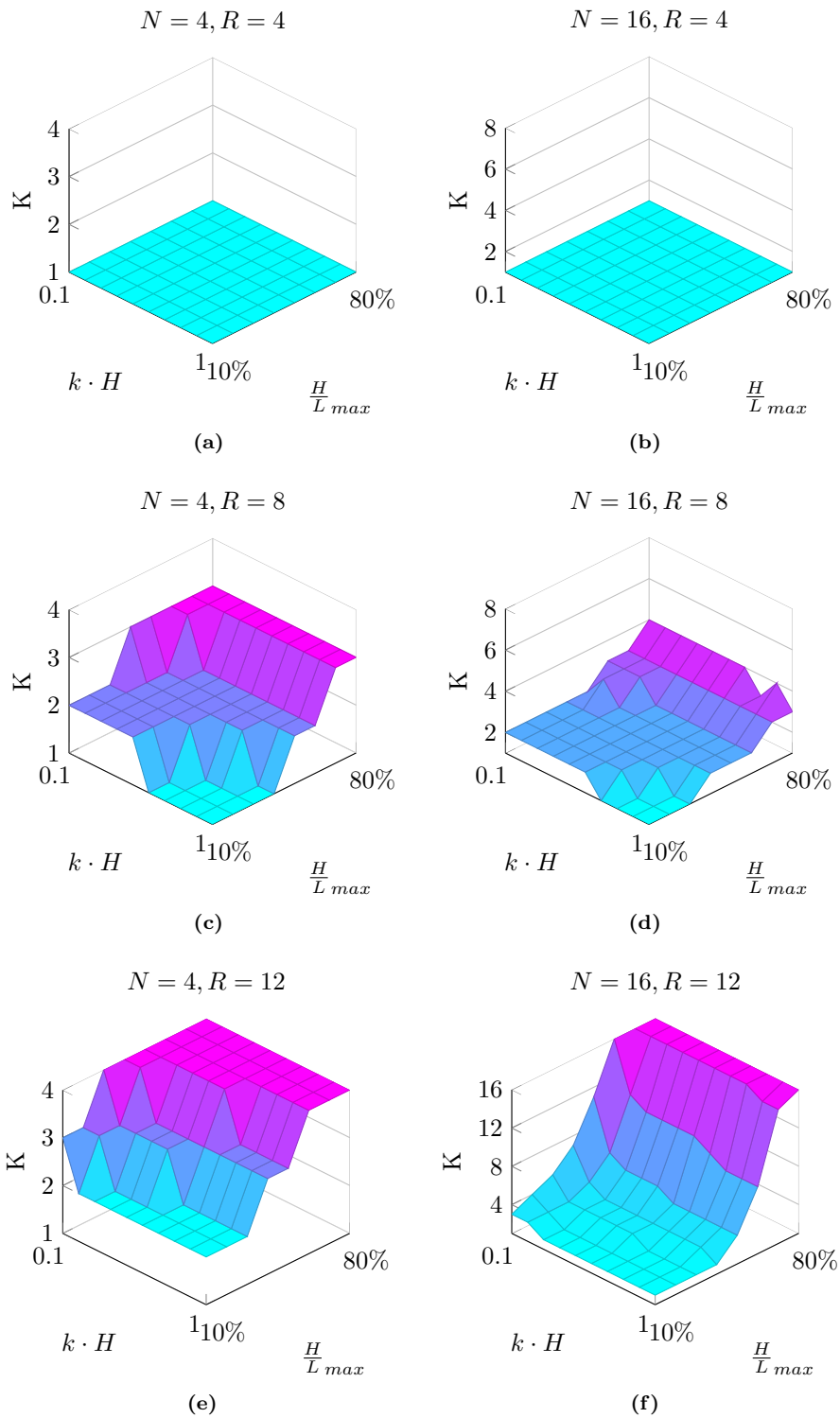


Figure 6.14: Iterations K to convergence.

estimate, which ratio R is better for a given parameter combination, we need to establish how to distribute the compute work in the algorithm. The approach taken by E. Eubanel [2] of splitting work into interdependent tasks is clear in style of implementation and results show great improvements over a simple strictly sequential-parallel style implementation. More importantly it conforms well with the work that will be presented in chapter 7 where a Cpp/CUDA/MPI based implementation using GPUs as workers is developed and tested. With multiple levels of parallelism on a heterogeneous hardware setup, the algorithmic complexity increases rapidly and data locality becomes non-triviel and as such for implementation purposes the task-dependent approach is estimated to be a good choice as it is less implementation heavy than say an event based method of but delivers similarly high efficiency. The actual implementation of the algorithm is discussed further in chapter 7. Having decided on an approach to distribute work in the parareal algorithm, it is possible to estimate the speed-up. Aubanel presents two task-scheduling based approaches of distributing the work, a manager/worker and a fully distributed algorithm. Aubanel also derives expressions for the speed-up as a function of fine to coarse time ratio $R = r^{-1}$ and iterations to convergence k under the assumption of instant communication and neglecting the correction time, as mentioned in section 3.3, these tests showed that the expressions given below are fairly accurate.

$$\psi_{MW(k=1)} = \frac{N}{(2N-1)\,r+1} \tag{6.5a}$$

$$\psi_{MW(k>1)} = \frac{N}{2\,(N-1)\,r+k} \tag{6.5b}$$

and

$$\psi_{FD} = \frac{N}{Nr+k\,(1+r)} \tag{6.6}$$

Applying these expressions to the measurements 6.14, one can calculate the expected speed-up assuming negligible correction and communication time. In figure 6.15 the speedup was estimated based on the simplest possible distribution model as presented in figurer 3.1, while in 6.16 it was estimated using the fully distributed model. From the measurements it is clear that due to a fairly slow coarse propagator being needed for convergence of the parareal algorithm on the fully nonlinear wave problem 6.1 a highly efficient method of distributing work is necessary to obtain reasonable speed-up. For this reason the more involved fully distributed model proposed by [2] will be used in chapter 7 for the implementation using GPUs as worker units. The implementation of GPUs as worker units in the parareal algorithm has to the knowledge of the author been proposed, but not attempted, before and as such, the results constitutes a novel contribution to the available literature.

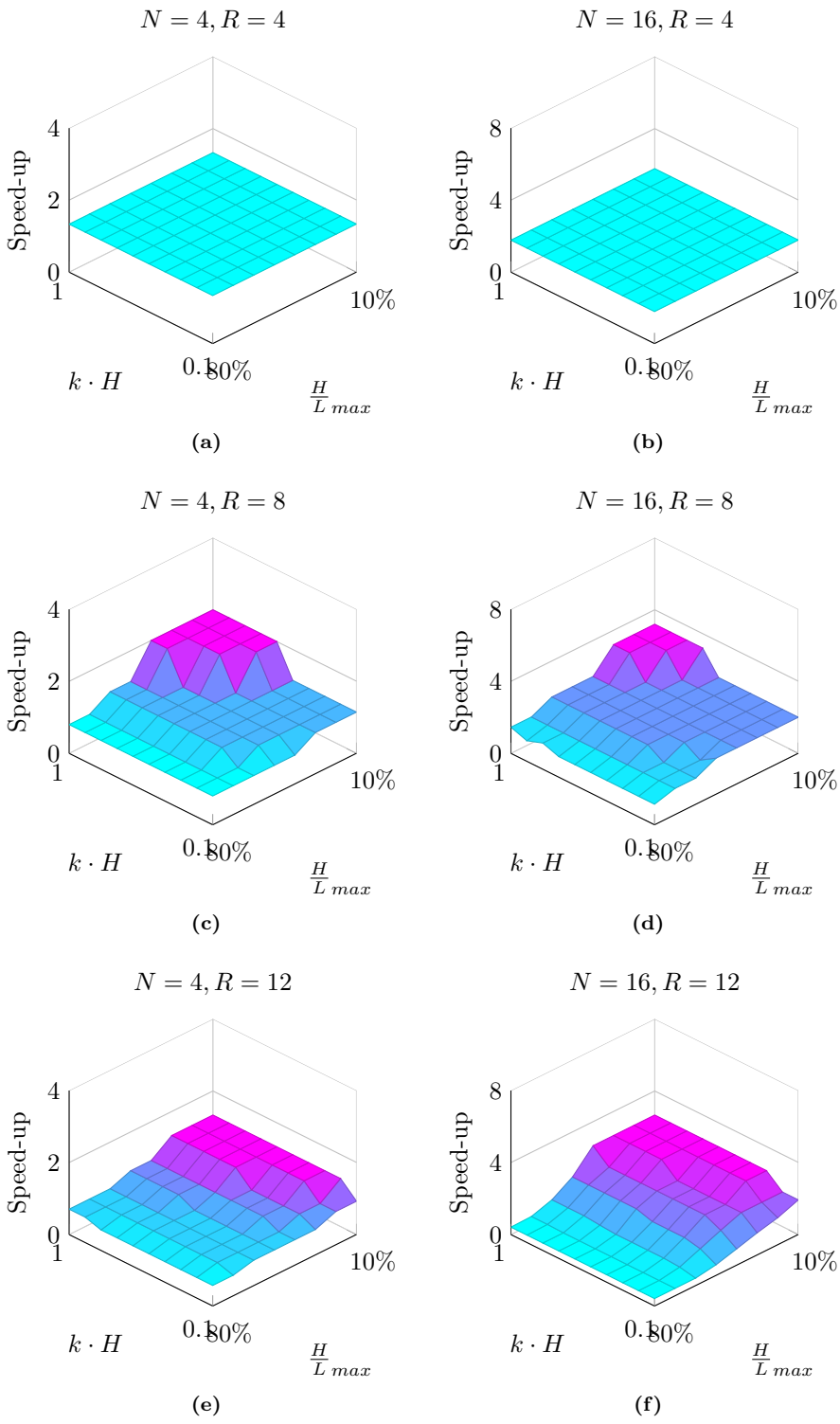


Figure 6.15: Estimated speedup based on measurements 6.14 given simplest possible distribution model

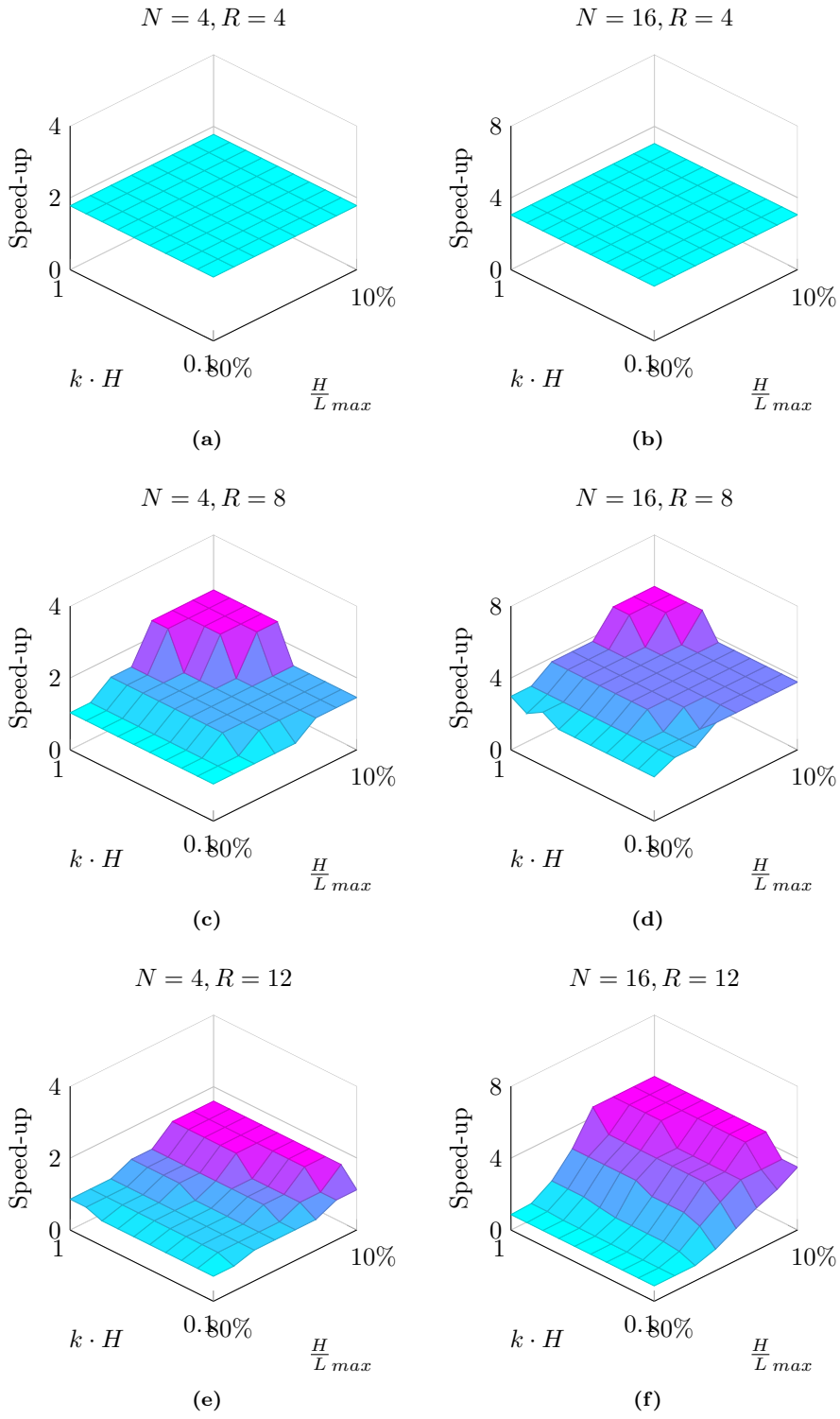


Figure 6.16: Estimated speedup based on measurements 6.14 given fully distributed task scheduling model

6.5 Summary

In the beginning of this chapter a fully non-linar wave model was presented together with a finite difference based solver. The properties of the parareal algorithm applied to this problem was investigated in terms of stability, convergence and estimated speed-up. Clearly the parameters H/L and kh influence the effectiveness of the added parallelism by the parareal method. The efficiency can be seen to range from zero to more than fifty percent for deep water waves at moderate wave amplitude. The algorithm show neither weak nor strong scaling properties on the problem 6.1 however the convergence speed degrades fairly slow with the amount of predictor-corrector intervals added so this should not be an issue in terms of obtaining speedup with 10s or even 100s of compute units.

Despite the hyperbolic nature of the non-linear wave model, it was shown that with a sufficiently accurate coarse solver the parareal algorithm will converge on the problem. Though, when using a fine to coarse time ratio of $R = 12 : 1$ and $R = 16 : 1$ instabilities such as those presented in [18] and [15] among others would appear. Due to the nature of the problem, a fairly accurate, and thus slow, coarse propagator is needed. This is an issue in terms of the overall speed-up that can be attained and with the use of such a "slow" fast coarse propagator it is strictly necessary to apply an effective method of distributing the parallel compute work in order to obtain speed-up. Fortunately it is here possible to build upon previous work such as that presented in [2].

The issue of slow convergence indicate that the algorithm is less suitable for this particular problem, which is not entirely unexpected given its hyperbolic nature. But as will be shown later, the finite difference based solver for the wave-problem has another property that has a positive implication on the application of parareal. In the parareal algorithm, the solution state needs to be communicated between consecutive time intervals and for this particular problem the state takes up very little space compared the time to integrate a small interval. Perfectly suited for the parareal algorithm and as will be demonstrated in the chapter to follow it is possible to obtain speedup with parareal on the fully nonlinear wave model.

Chapter 7

A Large-Scale GPU Based Implementation

Using the massively parallel GPUs to accelerate the solution of the nonlinear free-surface water wave model 6.1 has been shown to speed-up the time to solution by an order of magnitude [22]. This is impressive, let us now see if we can obtain further acceleration of the solution by the application of the parareal scheme to the two dimensional problem.

In the previous chapter we investigated the algorithmic properties of parareal applied to 6.1, using the numerical scheme presented in chapter 6. With a thorough knowledge of the convergence rate to be expected and it's relation to wave parameters, we move on to implement the parareal algorithm using GPUs as worker compute units. The parallel work is scheduled using the fully distributed wave model as proposed by [2] and presented in section 3.3.

In this chapter the Cpp-MPI-CUDA implementation is presented along with measured speed-up results. Initial tests are conducted on a node level measuring speed-up using 2 and 4 GPUs. The results are compared to what is obtainable using a spacial domain decomposition approach also on node level. In the quest to accelerate the solution further, the method is implemented on a grid level and tested on the Brown University Oscar GPU cluster using up to 16 GPUs. The chapter ends with a discussions on the strength and weaknesses of parareal with

respect to spacial domain decomposition, included discussions on the prospect of combining the algorithms in different hardware layers so to exploit different properties of each algorithm by the introduction of a third layer of parallelism.

7.1 The fully distributed task scheduling model

The fully distributed task scheduling model was first introduced in the literature review section 3.2 and a schematical description of the distribution of work can be found in figure 3.2.

The implementation of parareal on top of the parallelism in the GPU based solver is build using components available in the GPUlab library by Stefan L. Glimberg, Phd student at DTU Informatics. The implementation to be tested is thus build using C++ and MPI for the distribution of parallel work and CUDA for the GPU based solver. Upon starting the program, an MPI process is initialised for each GPU available on a given node. Each MPI thread has it's own unique id, MPI_rank, which is used in identification when communication between ranks are needed. The id's are enumerated as $0, 1, .., N-1$. MPI_rank zero is the initiating worker unit that performs the first coarse propagation from the initial state. Upon finishing, id zero sends the resulting state to id one, and from here on the computation cascades down trough all iterations and intervals. The pseudo code for the implementation is stated in algorithm 2.

As we are dealing with a heterogeneous CPU-GPU system, on top of which we impose an additional layer of parallelism, data locality becomes non-trivial. The implementation is made so that all communication happens trough MPI send and receive calls in which CUDA transfer calls between device and host are wrapped. When GPUs on a grid need to communicate a solution state, the commutation route is GPU->CPU->CPU->GPU. This is a lot of data transfer, and it is a fair concern that it may degrade performance. The communication time could potentially be reduced by employing CUDA GPUdirect as available in CUDA 5.0 in the communication between GPUs, but as will be shown, the amount of data that needs to be communicated is sufficiently small compared to the integration time that even for smaller problems the communication time is effectively hidden by the computational work.

Algorithm 2 Fully distributed parareal implementation

$convergeNext \leftarrow$ FALSE
if $id = 0$ **then**
  $\tilde{U}_0^0 \leftarrow y_0$
  $\tilde{U}_1^0 \leftarrow \mathcal{G}_{\Delta T}\tilde{U}_0^0$
  send $\tilde{U}_1^0$ to GPU id 1
  $\hat{U}_1^0 \leftarrow \mathcal{F}_{\Delta T}\tilde{U}_0^0$
  $U_1^1 \leftarrow \hat{U}_1^0$
  send $converge$ and U_1^1 to GPU id 1
  exit \\GPU $id = 0$ finished.
else
  receive $\tilde{U}_{id}^0$ from GPU $id - 1$
  $\tilde{U}_{id+1}^0 \leftarrow \mathcal{G}_{\Delta T}\tilde{U}_{id}^0$
  if $id! = N - 1$ **then**
    send $\tilde{U}_{id+1}^0$ to GPU $id + 1$
  end if
end if
$U_{id}^0 \leftarrow \tilde{U}_{id}^0$
for $k = 1$ to K_{max} **do**
  $\hat{U}_{id+1}^{k-1} \leftarrow \mathcal{F}_{\Delta T}U_{id}^{k-1}$
  if $convergeNext$ **then**
    $converge \leftarrow$ TRUE
    $U_{id+1}^k \leftarrow \hat{U}_{id+1}^{k-1}$
    send $converge$ and U_{id+1}^k to GPU $id + 1$
    exit \\GPU id finished.
  end if
  receive $converge$ and U_{id}^k from GPU $id - 1$
  $\tilde{U}_{id+1}^k \leftarrow \mathcal{G}_{\Delta T}U_{id}^k$
  $U_{id+1}^k \leftarrow \tilde{U}_{id+1}^k + \hat{U}_{id+1}^{k-1} + \tilde{U}_{id+1}^{k-1}$
  if $converge$ & $|U_{id+1}^k - \mathcal{F}_{\Delta T}^{id+1}y_0| > \epsilon$ **then**
    $converge \leftarrow$ FALSE
    $convergeNext \leftarrow$ TRUE \\Converges in next iteration
  end if
  if $id! = N - 1$ **then**
    send $converge$ and U_{id+1}^k to GPU $id + 1$
  end if
  if $converge$ **then**
    exit \\GPU id finished.
  end if
end for

7.2 Single node implementation

In this section we present results of wall-time measurements of the implementation of the parareal scheme using GPUs as workers as explained in the previous section. As communication and correction time may degrade performance, particularly for smaller problems, the wall-time of the computation is measured as a function of problem size. To test the implementation, a deep water wave $k \cdot H = 2\pi$ with small amplitude $\frac{H}{L} = 10\%\frac{H}{L}|_{max}$ is generated. By the investigations made in chapter 6, we expect the parareal solution to converge in a single iteration on this problem with a timestep $R = 8$ longer than the fine timestep using RK4 as both coarse and fine propagator. Each wave is 1 meter in length and resolved using $N_x = N_z = 33$ points, and the problem consists of solving the wave model over one wave period in time. The problem size is then increased by simply adding more of the same waves, increasing the spacial dimension, while still only integrating one wave period forward in time. The time-steps of the fine propagator is calculated using a courant number of 0.25 for a total of 128 fine propagator time-steps in the integration of a wave-period.

The GPU based propagator used to integrate each time sub-domain uses an iterative method to solve the Laplace problem, and as such we need to specify a convergence tolerance. This tolerance needs to be different for the coarse and fine propagator implementations as the Laplace problem in the coarse propagator does not need to be solved to same accuracy. The tolerances needed are estimated, and actual coarse to fine time ratio spend on integrating an interval ΔT might be slightly different than $R = 8$.

The compute walltime measured as a function of problem size is presented in figure 7.2 for the parareal scheme and in figure 7.1 using spacial domain decomposition as implemented in the GPUlab library by Stefan L. Glimberg. The application was tested on a node consisting of an Intel Xeon E5620 @ 2.4Ghz with 12GB ram and 2x NVIDIA Geforce GTX 590 graphics cards, each GTX590 holds two Fermi generation GPUs with 1.5GB of dedicated memory.

In figure 7.3 the speedup, as calculated from the data presented in figures 7.2 and 7.1, is presented. It is observed that the parallel speed-up obtained corresponds nicely with what can be predicted assuming negligible communication and correction time. Spacial Domain decomposition as expected works very well for large problem sizes, going towards perfect linear scalability. The classic domain decomposition method is also observed to be much more sensitive to communication issues than the parareal algorithm. For smaller problem sizes the parareal algorithm is observed to be much faster as the parallel efficiency of the spacial domain decomposition method decrease rapidly as a consequence of smaller spacial domains leading to an increase in the communication to compu-

tation ratio.

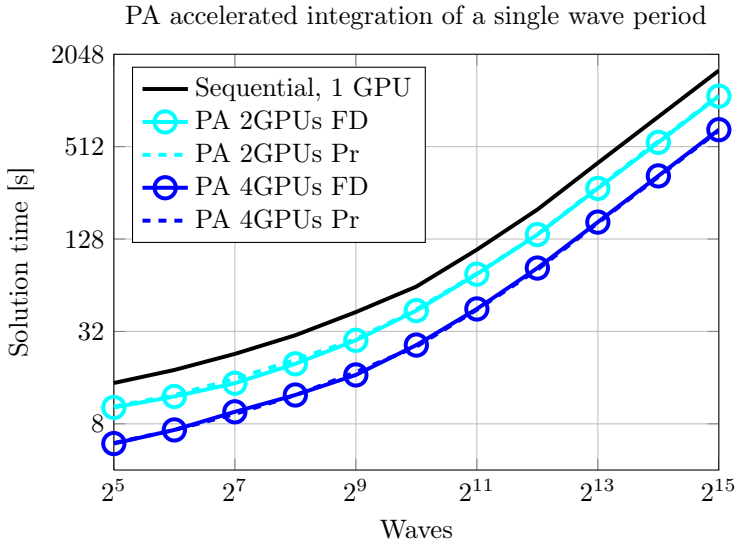


Figure 7.1: The solution wallclock time measured as a function of problem size given parallel acceleration using the parareal scheme.

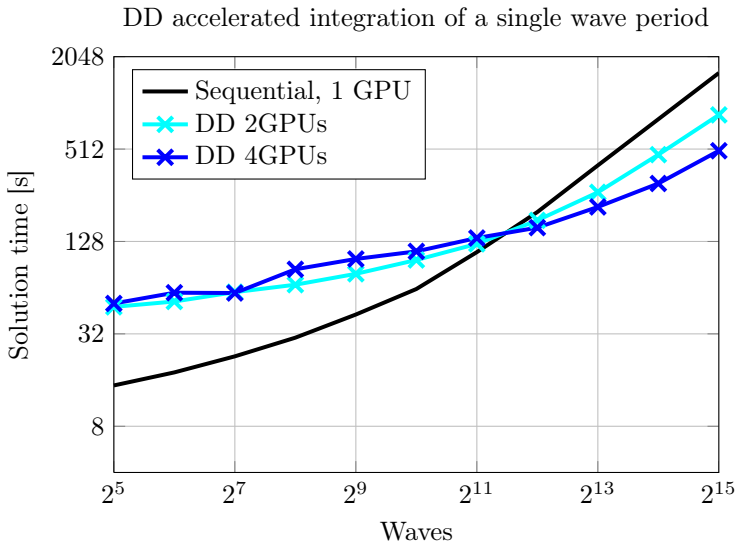


Figure 7.2: The solution wallclock time measured as a function of problem size given parallel acceleration using spacial domain decomposition.

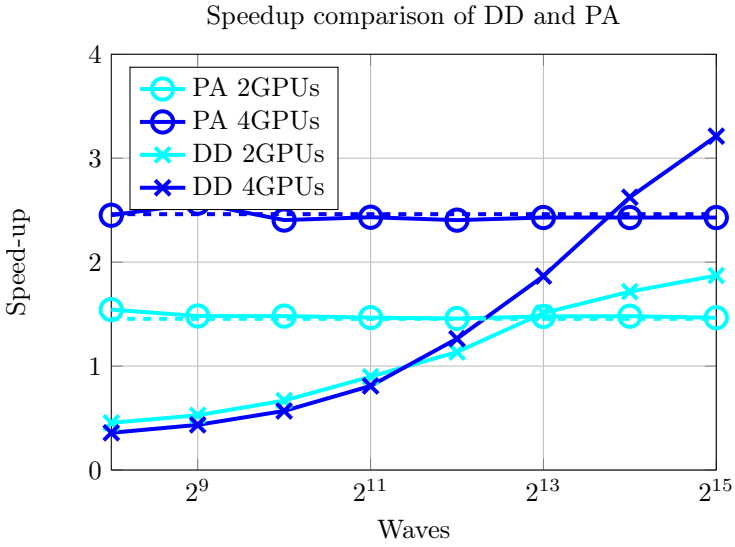


Figure 7.3: Speedup as computed by the results presented in figures 7.1 and 7.2. Notice how the parareal scheme is much less sensitive to the size of the problem solved, also notice how spacial domain decomposition allows for linear speed-up under ideal conditions.

7.3 Grid level implementation

In the previous section we obtained a speed-up of more than 2 using 4 GPUs on a single node. This is very nice, but we would like to see if it is possible to do even better. For the purpose of investigating so, the application was tested on the Oscar GPU cluster at Brown University. The GPU cluster consists of 44 nodes, each with two NVIDIA Tesla M2050 GPUs for a total of 88 GPUs available on the cluster. The walltime and speed-up measurements on the same problem as tested in previous section, integrating over one wave period in time, can be found in table 7.1. Notice how for an increasing amount of cores, we experience diminishing returns as the added acceleration upon adding more GPUs decrease rapidly. This effect can be attributed to the fairly slow coarse propagator, being only roughly 8 times faster than fine propagator. As additional GPUs are added, the GPUs spend an increasing amount of time waiting for one another. This effect is also apparent in the speed-up estimate 6.6. Letting N go to infinity with fixed k and r, the speed-up is seen to be limited only by the speed of the coarse propagator. When adding more GPUs this limit is approached asymptotically.

Speedup	Walltime	GPUs
-	26.96s	1
2.4	11.06s	4
3.6	7.45s	8
4.2	6.45s	16

(a)

Speedup	Walltime	GPUs
-	106.7s	1
2.4	43.72s	4
3.6	29.59s	8
4.2	25.31s	16

(b)

Table 7.1: Walltime measurements on the integration of a single wave period for a problem size (a) 128 waves and (b) 1024 waves.

7.4 Summary

In this chapter a proof of concept acceleration of the fully non-linear free surface water wave model 6.1 using the parareal scheme with GPUs as compute units was presented. It was shown to be possible to obtain more than 50% parallel efficiency when using a small amount of GPUs. In addition, we noticed how large scale speedup is limited by the speed of the coarse propagator relative to the fine propagator.

The results was compared to speedup measurements of the acceleration using domain decomposition to distribute the computation onto more GPUs. In the comparison between the two approaches it is of particular interest how the parareal scheme seem to be much less influenced by communication than the spacial domain decomposition approach. The parareal algorithm is not able to obtain as high speedup as domain decomposition under ideal conditions though, and this even though wave parameters for which optimal performance of the parareal algorithm could be expected was chosen.

Concluding remarks

The parareal algorithm has been presented along with everything that a developer will need to know in order to build an efficient parareal implementation to accelerate the solution of some evolution problem. A thorough literature review has been presented along with a large amount of information and references to theoretical and experimental results with the parareal algorithm as available in the literature.

In chapter 4 and 5 the convergence properties of the algorithm with application to a number of different initial value problems was presented and from the analysis, a heuristic for optimal coarse propagator accuracy choice was proposed. The investigations indicate that the best strategy when seeking to accelerate the solution of some IVP with a small number of compute units is to choose a sufficiently accurate propagator, that converge fast, as the added compute work, increasing with a factor k, is dominant in the degradation of parallel efficiency. In the case that acceleration is sought using many compute units, it is advisable to choose a computational fast but less accurate propagator, as the dominant loss in parallel efficiency will be on the idle time of compute units waiting to receive the initial conditions needed. These heuristics seemed consistent given that no stability issues for the choice of k and N are present. In case of the latter, it may be better to employ a simple hierarchical algorithm making sure to choose a sufficiently accurate coarse propagator so to converge in a single iteration.

In terms of applicability of parareal, there are a few issues that requires to be addressed. It is hard to predict beforehand how much of a computational speed-up can be obtained by applying the parareal algorithm to some problem.

In addition, the effectiveness of the algorithm also depends a great deal on finding a coarse propagator satisfying the accuracy needed for convergence, at the cheapest possible computational expense. Having to find an appropriate operator type and experiment to find the best accuracy for optimal parallel efficiency makes the procedure of obtaining parallel speed-up more involved than otherwise expected. A positive note on the parareal scheme though, is how it is easily wrapped around any other propagator used, and it should be possible to build parareal library allowing for fast parallelization of any given operator combination without having to deal with the implementation of complicated distribution models as these are universal for any combination of propagators.

An often mentioned motivation for parareal is how it is possible to combine the method with the successful spacial domain decomposition methods. When to many compute units are added to a spacial domain, the subdomain eventually become very small and communication time reduce parallel efficiency to a point where no additional speed-up is obtained as observed in chapter 7. The proposition is then to use added compute unit beyond the points of saturation to parallelize in time rather than to parallelize further in space.

But this is not necessarily the only argument for the application of parareal. Parareal and spacial domain decomposition was shown to have very different properties in terms of communication and latency sensitivity. Modern large scale compute grids tend to be multi-layered with each layer having different properties in terms of memory access, speed and latency. It seems reasonable that this multi-layered structure should be reflected in the algorithms applied to such machines. The parareal method in this context looks favourable for grid implementation as it is much less sensitive to latency and communication speed, while spacial domain decomposition may be an excellent choice on a nodal level as these methods can achieve linear speed-up given the availability of sufficiently fast low latency communication. With regards to the nonlinear free-surface water wave model presented in chapter 6, there is a potential for additional speed-up by employing two separate layers of parallelism on top of the parallelism already found in the GPU based propagator. Such investigations are left for future work.

As a closing remark, we highlight how the parareal method may prove to be a small but significant contribution in the challenges of extracting parallelism in the solution of differential equations so to exploit the compute capabilities of modern many core hardware architectures at extreme scale. A multitude of papers have been published on parareal during the last decade, with recent publications investigating practical results of large scale implementations, combining parareal and spacial domain decomposition, this may be a hint that the algorithm is maturing to a level at which it will be more widely adopted.

Bibliography

[1] K. Asanovi, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick, *The landscape of parallel computing research: A view from berkeley*, Technical Report EECS-2006-183, University of California Berkeley, 2006.

[2] E. Aubanel, *Scheduling of tasks in the parareal algorithm*, Parallel Computing, 37 (2010), pp. 172–182.

[3] C. Audouze, M. Massot, and S. Volz, *Sympletic multi-time step parareal algorithms applied to molecular dynamics*. Not formally published, 2009.

[4] L. Baffico, S. Bernard, Y. Maday, G. Turinici, and G. Zérah, *Parallel in time molecular dynamics simulations*, Physical Review E., 66 (2002).

[5] G. Bal, *Parallelization in time of (stochastic) ordinary differential equations*. Not formally published, 2002.

[6] ——, *On the convergence and the stability of the parareal algorithm to solve partial differential equations*, in Domain Decomposition Methods in Science and Engineering, R. Kornhuber, R. H. W. Hoppe, D. E. Keyes, J. Periaux, O. Pironneau, and J. Xu, eds., vol. 40, SIAM, Springer-Verlag Berlin, 2004, pp. 425–432.

[7] G. Bal and Y. Maday, *A "parareal" time discretization for non-linear pde's with application to the pricing of an american put*, in Recent development in domain Decomposition Methods, L. Pavarino and A. Toelli,

eds., vol. 23 of Lecture Notes in Computational Science and Engineering, Springer-Verlag Berlin, 2002, pp. 189–202.

[8] A. Bellen and M. Zennaro, *Parallel algorithms for initial value problems for nonlinear vector difference and differential equations*, Journal of computing and applied mathematics, 25 (1989), pp. 341–350.

[9] L. A. Berry, W. Elwasif, J. M. Reynolds-Barredo, D. Samaddar, R. Sanchez, and D. E. Newman, *Event-based parareal: A data-flow based implementation of parareal*, Journal of Computational Physics, 231 (2012), pp. 5945–5954.

[10] H. B. Bingham and H. Zhang, *On the accuracy of finite-difference solutions for nonlinear water waves.*, Journal of Engineering Mathematics, 58 (2007), pp. 211–228.

[11] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, *State-of-the-art in heterogeneous computing*, Scientific Programming, 18 (2010), pp. 1–33.

[12] K. Burrage, *Parallel and sequential methods for ordinary differential equations*, The Clarendon Press Oxford University Press, Burlington, MA, USA, 1995.

[13] P. Chartier and B. Philippe, *A parallel shooting technique for solving dissipative odes*, Computing, 51 (1993), pp. 209–236.

[14] F. Chouly and M. A. Fernandez, *An enhanced parareal algorithm for partitioned parabolic- hyperbolic coupling*, AIP Conference Proceedings, 1168 (2009), pp. 1517–1520.

[15] J. Cortial and C. Farhat, *A time-parallel implicit method for accelerating the solution of non-linear structural dynamics problems*, Journal for Numerical Methods in Engineering, 77 (2009), pp. 451–470.

[16] R. Croce, D. Ruprecht, and R. Krause, *Parallel-in-space-and-time simulation of the three-dimensional, unsteady navier-stokes equations for incompressible flow*, preprint, ICS, 03 2012.

[17] X. Dai, C. L. Bris, F. Legoll, and Y. Maday, *Parareal algorithms for hamiltonian systems.* Submitted to Mathematical Modelling and Numerical Analysis Nov, 2010.

[18] X. Dai and Y. Maday, *Stable pararael in time method for first and second order hyperbolic system.* Submitted Jan, 2012.

[19] J. Douglas and H. H. Rachford, *On the numerical solution of heat conduction problems in two and three space variables*, Transactions of the american mathematical society, (1956), pp. 421–439.

[20] W. R. Elwasif, S. S. Foley, D. E. Bernholdt, and L. A. Berry, *A dependency-driven formulation of parareal: Parallel-in-time solution of pdes as many-task application*, (2011), pp. 15–24.

[21] A. P. Engsig-Karup, H. B. Bingham, and O. Lindberg, *An efficient flexible-order model for 3d nonlinear water waves*, Journal of Computational Physics, 228 (2009), pp. 2100–2118.

[22] A. P. Engsig-Karup, M. G. Madsen, and S. L. Glimberg, *A massively parallel gpu-acceleratedmodel for analysis of fully nonlinear free surface waves*, International Journal for Numerical Methods in Fluids, 10.1002/fld.2675 (2011).

[23] C. Farhat and M. Chandersris, *Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid-structure applications*, International Journal for Numerical Methods in Engineering, 58 (2003).

[24] C. Farhat, J. Cortial, C. Dstillung, and H. Bavestrello, *Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamics responses*, Int. Journal for Numerical Methods in Engineering, 67 (2006), pp. 697–724.

[25] P. Fischer, F. Hecht, and Y. Maday, *A parareal in time semi-implicit approximation of the navier stokes equations*, in 15th International Conference on Domain Decomposition Methods in Science and Engineering, R. Kornhub, R. Hoppe, J. Periaux, O. Pironneau, O. Widlund, and J. Xu, eds., vol. 40, SIAM, Springer-Verlag Berlin, 2004, pp. 433–440.

[26] M. Gander and S. Vanderwalle, *On the superlinear and linear convergence of the parareal algorithm*, in 16th International Conference on Domain Decomposition Methods in Science and Engineering, vol. 55, Springer Berlin, 2007, pp. 291–298.

[27] M. Gander and S. Vandewalle, *Analysis of the parareal time-parallel time-integration method*, SIAM Journal of scientific computing, 29 (2007), pp. 556–578.

[28] I. Garrido, G. Fladmark, and M. Epedal, *Parallelization in time for reservoir simulation*. Not formally published, 2003.

[29] I. Garrido, B. Lee, G. E. Fladmark, and M. E. Espedal, *Convergent iterative schemes for time parallelization*, Mathematics of computation, 75 (2006), pp. 1403–1428.

[30] J. Geiser and S. Guttel, *Coupling methods for heat transfer and heat flow: Operator splitting and the parareal algorithm*, Journal of mathematical analysis and applications, 388 (2012), pp. 873–887.

[31] W. Hackbusch, *Parabolic multigrid methods*, Computing methods in applied sciences and engineering, VI (1984), pp. 189–197.

[32] C. Harden, *Realtime computing with the parareal algorithm*, master thesis, Florida State University, College of Arts and Sciences, School of Computational Science, 2008.

[33] L.-P. He and M. He, *Parareal in time simulation of morphological transformation in cubic alloys with spatially dependent composition*, Communications in Computational Physics, 11 (2012), pp. 1697–1717.

[34] K. R. Jackson, *A survey of parallel numerical methods for initial value problems for ordinary differential equations*, IEE Transactions on Magnetics, 25 (1991), pp. 3792–3797.

[35] K. R. Jackson and S. N. rsett, *The potential for parallelism in runge-kutta methods. part 1: Rk formulas in standard form*, SIAM Journal of Numerical Analysis, 32 (1995), pp. 49–82.

[36] B. Khalaf and D. Hutchinson, *Parallel algorithms for initial value problems: parallel shooting*, Parallel Computing, 18 (1992), pp. 661–673.

[37] M. Kiehl, *Parallel multiple shooting for the solution of initial value problems*, Parallel Computing, 20 (1994), pp. 275–295.

[38] E. Lelarasmee, A. E. Ruehli, and A. L. Sangiovanni-Vincentelli, *The waveform relaxation method for time-domain analysis of large scale integrated circuits*, IEEE Trans. on CAD of IC and Syst., 1 (1982), pp. 131–145.

[39] B. Lepsa and A. Sandu, *An efficient error control mechanism for the adaptive 'parareal' time discretization algorithm*, in Proceedings of the 2010 Spring Simulation Multiconference, R. McGraw and E. Imsand, eds., no. 87, Society for Computer Simulation International, 2010.

[40] B. Li and C. A. Fleming, *A three dimensional multigrid model for fully nonlinear water waves*, Coastal Engineering, 30 (1997), pp. 235–258.

[41] J.-L. Lions, Y. Maday, and G. Turinici, *Résolution d'edp par un schéma en temps pararéel*, C.R. Acad Sci. Paris Sér. I math, 332 (2001), pp. 661–668.

[42] Y. Maday, *Parareal in time algorithm for kinetic systems based on model reduction*, in High-Dimensional Partial Differential Equations in Science and Engineering, A. Bandrauk, M. Delfour, and C. LeBris, eds., vol. 41 of CRM Proceedings & Lecture Notes, American Mathematical Society, 2007, pp. 183–194.

[43] Y. Maday, *The parareal in time algorithm*, Technical Report R08030, Universite Pierré et Marie Curie, 2008.

[44] Y. Maday, E. R. nquist, and G. A. Staff, *The parareal-in-time algorithm: Basics, stability and more*, (2006).

[45] Y. Maday, J. Salomon, and G. Turinici, *Monotonic parareal control for quantum systems*, SIAM JOURNAL ON NUMERICAL ANALYSIS, 45 (2007), pp. 2468–2482.

[46] Y. Maday and G. Turinici, *A parareal in time procedure for the control of partial differential equations*, C. R. Math. Acad. Sci. Paris, 335 (2002), pp. 387–392.

[47] ——, *Parallel in time algorithms for quantum control: Parareal time discretization scheme*, International Journal of Quantum Chemistry, 93 (2003), pp. 223–228.

[48] ——, *The parareal in time iterative solver: A further direction to parallel implementation*, in 15th International Conference on Domain Decomposition Methods in Science and Engineering, R. Kornhub, R. Hoppe, J. Periaux, O. Pironneau, O. Widlund, and J. Xu, eds., vol. 40 of Lecture Notes in Computational Science and Engineering, Springer-Verlag Berlin, 2004, pp. 441–448.

[49] D. Mercerat, L. Guillot, and J. P. Vilotte, *Application of the parareal algorithm for acoustic wave propagation*, AIP Conference Proceedings, 1168 (2009), pp. 1521–1524.

[50] M. Minion, *A hybrid parareal spectral deferred corrections method*, Communications in Applied Mathematics and Computational Science, 5 (2010), pp. 265–301.

[51] W. L. Miranker and W. Liniger, *Parallel methods for the numerical integrating of ordinary differential equations*, Mathematics of Computation, 21 (1967), pp. 303–320.

[52] A. S. Nielsen, A. P. Engsig-Karup, and B. Dammann, *Parallel programming using opencl on modern architectures*, Technical Report IMM-2012-05, Technical University of Denmark, 2012.

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

[54] D. Ruprecht and R. krause, *Explicit parallel-in-time integration of linear acoustic-advection system*, Computers & Fluid, 59 (2012), pp. 72–83.

[55] D. Samaddar, D. E. Newman, and R. Sanchez, *Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm*, Journal of Computational Physics, 229 (2010), pp. 6558–6573.

[56] H. Samuel, *Time domain parallelization for computational geodynamics*, Geochemistry Geophysics Geosystems, 13 (2012), pp. 1–16.

[57] A. Savitzky and M. J. E. Golay, *smoothing and differentiation of data by simplified least squares procedures*, Analytical Chemistry, 36 (1964), pp. 1627–1639.

[58] J. Shalf, S. Dosanjh, and J. Morrison, *Exascale computing technology challenges*, in Lecture Notes in Computer Science, J. Palma, M. Dayde, O. Marques, and J. Lopez, eds., vol. 6449, 9th International Conference on High Performance Computing for Computational Science, Springer, 2011, pp. 1–25.

[59] J. Simoens and S. Vandewalle, *Waveform relaxation with fast direct method as preconditioner*, SIAM Journal on Scientific Computing, 21 (2000), pp. 1755–1773.

[60] R. Speck, D. Ruprecht, R. Krause, M. Emmett, M. Minion, M. Winkel, and P. Gibbon, *A massively space-time parallel n-body solver*, tech. report, 2012.

[61] A. Srinivasan and N. Chandra, *Latency tolerance through parallelization of time in scientific applications*, Parallel Computing, 31 (2005), pp. 777–796.

[62] G. Staff, *Convergence and stability of the parareal algorithm: A numerical and theoretical investigation*, master thesis, Norwegian University of Science and Technology, Department of Mathematical Science, 2003.

[63] G. Staff and E. M. R. nquist, *Stability of the parareal algorithm*, in Lect. Notes in Comput. Sci. Eng., R. Kornhub, R. Hoppe, J. Periaux, O. Pironneau, O. Widlund, and J. Xu, eds., vol. 40 of Lecture Notes in Computational Science and Engineering, Springer-Verlag Berlin, 2005, pp. 449–456.

[64] J. Trindade and J. Pereira, *Parallel-in-time simulation of two-dimensional, unsteady, incompressible laminar flows*, Numerical Heat Transfer Part B-Fundamentals, 50 (2006), pp. 25–40.

[65] A. E. S. V. Sarkar, W. Harrod, *Software challenges at extreme scale*, Communications in Applied Mathematics and Computational Science, (2010), pp. 60–65.