GMD Research Series

GMD –
Forschungszentrum
Informationstechnik
GmbH

Reinhold Hess

Dynamically Adaptive Multigrid on Parallel Computers for a Semi-Implicit Discretization of the Shallow Water Equations

N° 9/1999

GMD – Forschungszentrum Informationstechnik GmbH
Schloß Birlinghoven
D-53754 Sankt Augustin, Germany
Telefon +49 -2241 -14 -0
Telefax +49 -2241 -14 -2618
http://www.gmd.de

Anschrift des Verfassers/Address of the author:
Reinhold Hess
Deutscher Wetterdienst
Forschung und Entwicklung
Postfach 100465
D-63004 Offenbach a. M.

Die vorliegende Veröffentlichung entstand im/
The present publication was prepared within:
Institut für Algorithmen und Wissenschaftliches Rechnen (SCAI)
Institute for Algorithms and Scientific Computing
http://www.gmd.de/SCAI/

Die Deutsche Bibliothek - CIP-Einheitsaufnahme:
Hess, Reinhold:
Dynamically Adaptive Multigrid on Parallel Computers for a Semi-Implicit Discretization of the Shallow Water Equations / Reinhold Hess.
GMD – Forschungszentrum Informationstechnik GmbH. - Sankt Augustin:
GMD – Forschungszentrum Informationstechnik, 1999
(GMD Research Series ; 1999, No. 9)
Zugl.: Köln, Univ., Diss., 1999
ISBN 3-88457-358-6

ISSN 1435-2699
ISBN 3-88457-358-6

Abstract

Numerical weather forecasting and climate predictions require enormous computing power since high resolution and accuracy are necessary to achieve reliable results. The overall idea of this dissertation on scientific computing is to develop and implement a numerically highly efficient basic meteorological model for parallel environments. Numerical acceleration is achieved by adaptive multigrid in two ways:

Firstly, the resolution of the finite difference discretization is locally adapted to the actual requirements of the weather situation. High resolution is provided only where it is necessary (e. g. strong low pressure areas). Since the weather situation changes in a time dependent simulation the refinement areas have to be adapted. This dynamic adaptation is controlled by a refinement criterion based on the estimation of the local spatial discretization error and performed fully automatically.

Secondly, multigrid is used to increase the sizes of stable time steps. The applied semi–implicit time scheme results in a Helmholtz equation, which is solved by MLAT. The cycling is adapted to the stability requirements of the actual model problem. In general, one very cheap multigrid cycle suffices.

The numerical algorithm is portably implemented on parallel environments by the grid partitioning approach with explicit message passing. A dynamic load balance algorithm considers neighborhood relationships in order to reduce data transfer.

Keywords: Multigrid, Adaptivity, Helmholtz equation, Parallel Solver, Shallow Water Equations

Zusammenfassung

Numerische Wettervorhersagen und Klimasimulationen erfordern enorme Rechenleistungen; hohe Auflösung und Genauigkeit sind notwendig, um zuverlässige Ergebnisse zu erzielen. Im Rahmen dieser Dissertation des wissenschaftlichen Rechnens wurde ein numerisch hoch–effizientes meteorologisches Modell für Parallelrechner entwickelt. Numerische Effizienz wird dabei mit einem auf zweierlei Weise adaptiven Mehrgitterverfahren erreicht:

Zum einen wird die räumliche Auflösung der finiten–Differenzen Diskretisierung an die tatsächlichen Erfordernisse der vorliegenden Wetterverhältnisse angepaßt. Hohe Auflösung wird nur in Gebieten angewandt, wo sie wirklich notwendig ist (z. B. in starken Tiefdruckgebieten). Da sich die Wetterverhältnisse während der Simulation ändern, müssen die Verfeinerungsgebiete angepaßt werden. Diese dynamische Anpassung wird vollautomatisch von einem Verfeinerungskriterium gesteuert, das auf einer Schätzung des lokalen Diskretisierungsfehlers beruht.

Zum anderen wird Mehrgitter verwendet, um die Größe stabiler Zeitschritte zu erhöhen. Die semi–implizite Zeitdiskretisierung resultiert in einer skalaren helmholtz–ähnlichen Gleichung, die in jedem Zeitschritt mit adaptivem Mehrgitter (MLAT) gelöst wird. Der Mehrgitterzyklus ist an den Stabilitätsanforderungen des vorliegenden Modellfalls angepaßt.

Das numerische Verfahren wurde mittels Gebietszerlegung und explizitem Datenaustausch (MPI) portabel auf Parallelrechnern mit verteiltem Speicher implementiert. Ein dynamischer Lastausgleichsalgorithmus berücksichtigt Nachbarschaftsbeziehungen, um den erforderlichen Datenaustausch zu reduzieren.

Schlagwörter: Mehrgitter, Adaptivität, Helmholtzgleichung, Paralleler Löser, Flachwassergleichungen

Contents

List of Figures

List of Tables

Zusammenfassung in deutscher Sprache

Die vorliegende Arbeit ist in englischer Sprache geschrieben, da diese zur Zeit in wissenschaftlichen Texten bevorzugt wird und damit eine größere Sichtbarkeit ermöglicht. Im folgenden ist eine Zusammenfassung in deutsch gegeben.

Wissenschaftliches Rechnen beschäftigt sich mit den Disziplinen Mathematik und Informatik mit dem Ziel, wissenschaftliche Probleme mit Hilfe von elektronischen Rechenanlagen zu lösen.

Im vorliegenden Fall wurde ein hoch–effizientes Verfahren zur Lösung eines grundlegenden Modells der Meteorologie — die Flachwassergleichungen — für aktuelle Parallelrechner mit verteiltem Speicher entwickelt und implementiert. Numerische Beschleunigung wird dabei mit einem auf zweierlei Weise adaptiven Mehrgitterverfahren erreicht:

Zum einen wird die räumliche Auflösung der finiten–Differenzen Diskretisierung an die tatsächlichen Erfordernisse der vorliegenden Wetterverhältnisse angepaßt. Hohe Auflösung wird nur in Gebieten angewandt, wo es wirklich notwendig ist (in starken Tiefdruckgebieten, an Wetterfronten oder über unregelmäßigem Terrain), wogegen es genügt, ruhige Regionen mit ausreichend niedrigerer Auflösung zu berechnen. Da sich die Wetterverhältnisse während der zeitabhängigen Simulation ändern (Wetterfronten und Tiefdruckgebiete ziehen vorbei), müssen die Verfeinerungsgebiete angepaßt werden. Diese dynamische Anpassung wird vollautomatisch von einem Verfeinerungskriterium gesteuert, das auf einer Schätzung des lokalen Diskretisierungsfehlers basiert. Dabei sind allgemein geformte Verfeinerungsgebiete möglich.

Zum anderen wird Mehrgitter verwendet, um die Größe stabiler Zeitschritte zu erhöhen. Dazu wurde eine semi–implizite Zeitdiskretisierung entwickelt, für die in jedem Zeitschritt nur eine skalare helmholtz–ähnliche Gleichung gelöst werden muß. Verwendet man Ergebnisse eines expliziten Zeitschemas als Anfangswerte für die Iteration dieser Gleichung, so ist die angestrebte Genauigkeit bereits erreicht; Mehrgitteriterationen werden dann nur noch durchgeführt um die Stabilität des Verfahrens zu gewährleisten.

Dabei kann der Mehrgitterzyklus an die Stabilitätsanforderung des vorliegenden Modellfalls angepaßt werden, woduch im allgemeinen ein sehr einfacher Zyklus mit wenigen groben Gittern genügt. Dies ist insbesondere auf Parallelrechnern von Vorteil. In diesem Zusammenhang wird Mehrgitter eher als *adaptiver Stabilisator* denn als Löser benutzt wie bisher üblich.

Das numerische Verfahren wurde mittels Gebietszerlegung und explizitem Datenaustausch portabel auf Parallelrechnern mit verteiltem Speicher implementiert. Abbildung 1 stellt die erreichten Beschleunigungen der Simulation aufgrund algorithmischer Verbesserungen und Parallelisierung dar. Die Laufzeiten des meteorologischen Flachwassermodells mit voll–adaptiven lokalen Verfeinerungen wurden auf einem Parallelrechner IBM SP2 gemessen. Für ein später detailliert beschriebenes Modellproblem ist die Simulation mit semi–implizitem Zeitschrittverfahren nahezu doppelt so schnell wie eine rein explizite Referenzrechnung mit kleineren Zeitschrittweiten. Lokale Verfeinerungen schließlich führen in diesem Fall zu einer drastischen Beschleunigung der Simulation um den Gesamtfaktor 16 (bei nahezu gleicher Genauigkeit).

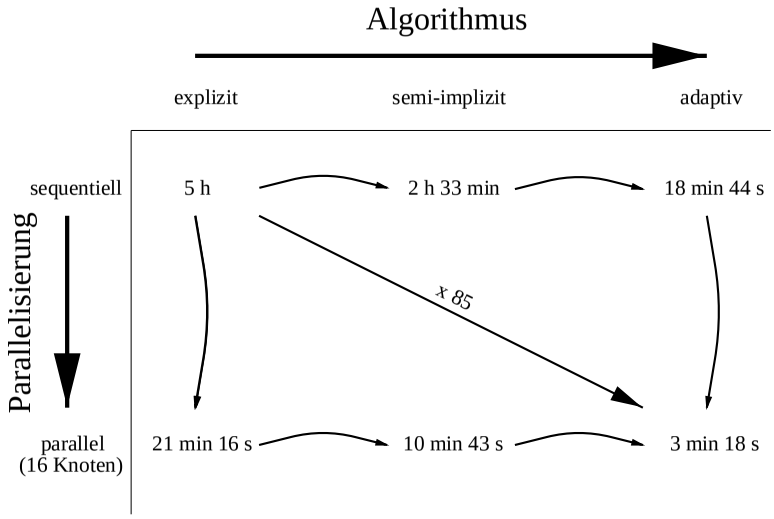

Abbildung 1: *Beschleunigung durch Algorithmus und Parallelität*

Die Parallelisierung des dynamisch adaptiven und semi–impliziten Verfahrens beschleunigt die Simulation um den Faktor 2.9 auf vier Rechenknoten und um 5.7 auf sechzehn Knoten, was insgesamt eine 85–fach schnellere Berechnung im Vergleich zu einer rein expliziten sequentiellen Simulation ergibt. Der weitere Geschwindigkeitsgewinn von vier auf sechzehn Rechenknoten zeigt die Skalierbarkeit des adaptiven Verfahrens, jedoch wird der Rechenaufwand zur Simulation der Flachwassergleichungen mit den beschriebenen algorithmischen Verbesserungen sehr klein im Verhältnis zu den zusätzlich eingebrachten Kosten der Parallelisierung. Aufgrund der geringen

Anzahl von Rechenoperationen pro Gitterpunkt stellen die Flachwassergleichung bezüglich der Parallelisierung einen schlimmsten Fall im Vergleich mit komplexeren insbesondere drei–dimensionalen meteorologischen Simulationen dar.

Um dieses dynamisch adaptive Verfahren zu entwickeln und zu implementieren, und die dabei auftretenden Schwierigkeiten zu lösen, sind eine Reihe von Kenntnissen aus den Wissenschaften *Mathematik*, *Informatik* und *Meteorologie* nötig. Diese werden im weiteren genannt. Die meisten dieser Schwierigkeiten sind bereits für block–strukturierte, adaptive und parallele Algorithmen bekannt; ebenso sind bereits eine Reihe von Lösungsansätzen veröffentlicht. Nichts desto trotz mußten neue Ideen gefunden und die bereits existierenden Ansätze angepaßt werden, um sie zu einem *laufenden* vollständig selbst–anpassenden parallelen Programm zusammenzuführen.

Anwendung Als meteorologische Anwendung werden die Flachwassergleichungen modelliert, welche sowohl Strömung (Wind) als auch Wellenausbreitung beschreiben, und damit wesentlich komplexer als Standardmodellprobleme sind. Außerdem wurde das Verfahren auch hinsichtlich allgemeinerer meteorologischer Modelle entwickelt, weshalb ein block–strukturierter Diskretisierungsansatz bevorzugt wurde.

Zeitschema Explizite Zeitdiskretisierungen führen bei hohen räumlichen Auflösungen, wie sie bei adaptiven Verfahren ermöglicht werden, zu extrem kleinen Zeitschritten. Um längere Zeitschritte zu verwenden, wurde ein semi–implizites Zeitschema entwickelt. Dieses benötigt nur zwei Zeitebenen und ermöglicht eine vereinfachte Verwaltung der Verfeinerungsstrukturen.

Löser Das semi–implizite Zeitverfahren kann über eine skalare helmholtz–ähnliche Gleichung berechnet werden. Diese muß in jedem Zeitschritt auf der aktuellen Verfeinerungsstruktur mittels Parallelrechner gelöst werden. Dazu bieten sich parallele Mehrgitterverfahren an.

Kriterium Die Modellgebiete, in denen höhere Auflösung nötig erscheint, werden während der Simulation automatisch durch ein Verfeinerungskriterium erkannt und angepaßt. Im vorliegenden Fall wird ein rein mathematisches Kriterium angewandt, das auf einer Schätzung des lokalen Diskretisierungsfehlers beruht.

Blockung In Hinblick auf die Anwendung Meteorologie werden die Verfeinerungsgebiete durch rechteckige, strukturierte und nicht–überlappende Blöcke überdeckt, die an den gröberen Gittern orientiert sind. Diese Blockung der Verfeinerungsgebiete erfolgt in einer Weise, so daß die Größe und Anzahl der entstehenden Blöcke bereits für die Parallelisierung geeignet sind. Darüber hinaus sollten einmal entstandene Blöcke so lange wie möglich wiederverwendet werden können.

Einbettung Die Verfeinerungsblöcke werden an ihren Rändern mit Daten des darunterliegenden gröberen Gitters versorgt. Da bei zeitlich veränderlichen Verfeinerungen die Lösungen auf den groben Gittern durch die genaueren Ergebnisse der Verfeinerungen verbessert werden, kann es zu unerwünschten Oszillationen an den Verfeinerungsrändern kommen. Dies wird durch ein spezielles Interpolationsverfahren vermieden.

Lastausgleich Parallele Effizienz hängt wesentlich von einer ausgeglichenen Verteilung des Rechenaufwands ab. Dies muß bereits bei der Blockung der Verfeinerungsgebiete berücksichtigt werden. Darüber hinaus wird im Lastausgleichs–Algorithmus als Nebenbedingung auch der entstehende Austausch von Volumendaten reduziert.

Implementierung Zuletzt ist die Implementierung eines solchen komplexen parallelen Programms sehr aufwendig. Spezielle Datenstrukturen wurden entwickelt, um die nötigen Informationen über die einzelnen Verfeinerungsblöcke zu verwalten, deren Anzahl und Größe sich dynamisch ändern. Die Blöcke werden den einzelnen Knoten des Parallelrechners dynamisch zugeteilt, welche entsprechend den veränderlichen Nachbarschaftsbeziehungen Daten austauschen. Dabei wurde konsequent asynchroner Datenaustausch eingesetzt, um Überlappung von Rechnung und Kommunikation zu ermöglichen und um bestmögliche parallele Effizienzen zu erreichen. Das Verfahren ist in etwa 10 000 Programmzeilen der Computersprache ‚C' implementiert.

Besondere Sorgfalt wurde auf die Kombination von Adaptivität und Parallelismus vor dem zeitabhängigen Hintergrund gelegt. Der Austausch von Volumendaten, der bei räumlichen Verfeinerungen allgemein stattfindet, zerstört den Rand–Volumen–Effekt der Gitterzerlegungsmethoden und kann die parallele Effizienz adaptiver Mehrgitterverfahren beschränken. Aus diesem Grunde wird die Anzahl der Initialisierungen und Umverteilungen der Verfeinerungsblöcke reduziert. Es wird versucht, die Verfeinerungsblöcke so lange als möglich beizubehalten, und die Verfeinerungsstruktur im Verlauf der Simulation eher anzupassen, als in jedem Schritt neu zu erzeugen. Darüber hinaus wird Volumenaustausch verringert, indem die Verfeinerungsblöcke und die zugehörigen tragenden gröberen Blöcke vorzugsweise dem selben Rechenknoten zugewiesen werden.

Die Bedingung wurde aufgehoben, unabhängig von der Anzahl der eingesetzten Rechenknoten das exakt selbe Ergebnis zu erzeugen. Die Blockung des Verfeinerungsgebiets wird für die vorhandene parallele Umgebung optimiert. In dieser Abhängigkeit werden mehr oder weniger Punkte, die ursprünglich nicht durch das Verfeinerungskriterium gekennzeichnet wurden, in die Verfeinerungsblöcke miteinbezogen. Dadurch kann das Rechenergebnis geringfügig variieren.

Aufgrund des hohen Potentials lokal adaptiver Verfahren, Rechenzeit zu sparen, finden zur Zeit eine Reihe von Forschungsprojekten in diesem Bereich statt (siehe Abschnitt 2.5). Dennoch wurde bisher noch kein tatsächlicher Durchbruch adaptiver Verfeinerungen in meteorologischen Simulationen erreicht. Die vorliegende Arbeit fügt Adaptivität und Parallelismus für eine Simulation mit semi–implizitem Zeitschema zusammen und wird als ein wichtiger Schritt in Richtung der Anwendung von parallelen adaptiven Verfeinerungen in der numerischen Wettervorhersage betrachtet.

Chapter 1

Introduction

In summary, then, scientific computing
draws on mathematics and computer science
to develop the best ways to use computer systems
to solve problems from science and engineering.

Citation from: G. Golub and J. M. Ortega,
Scientific Computing. An Introduction with Parallel Computing,
Academic Press, 1993, p. 3 .

Numerical weather forecasting and climate predictions require enormous computing power to solve partial differential equations prescribing meteorological and physical processes where high resolution and accuracy are necessary to achieve reliable results for medium and long–range predictions. Six of the twenty most powerful computer systems in the world are dedicated to weather research[1]. All of them are parallel architectures with the number of nodes ranging between 116 and 1024 .

However, to exploit the high computing power provided by parallel computers, existing codes are adapted to parallel computers and new parallel algorithms are developed. Since it is an expensive waste to run numerically inefficient algorithms on parallel machines, the requirement for modern algorithms is numerical *and* parallel efficiency.

The overall idea of this dissertation on scientific computing is to develop and implement a numerically highly efficient basic meteorological model for parallel environments. Numerical acceleration is achieved by adaptive multigrid in two ways:

Firstly, the resolution of the finite difference discretization is locally adapted to the actual requirements of the weather situation. High resolution is provided only where it is necessary (e. g. strong low pressure areas, weather fronts, steep orography). Calm regions are calculated with sufficient

[1] June 1998, Dongarra, Meuer, and Strohmaier [26]

lower mesh size. Since the weather situation changes in a time dependent simulation (weather fronts and cyclones move and evolve), the refinement areas have to be adapted during the simulation. This dynamic adaptation is controlled by a refinement criterion based on the estimation of the local spatial discretization error and performed fully automatically. Very general refinement areas are supported. In this way, the number of grid points can be essentially reduced.

Secondly, multigrid is applied to increase the sizes of stable time steps. For that a semi–implicit time scheme is developed, which results in a scalar Helmholtz–(like) equation. Using predictions from an explicit time scheme as initial values for this Helmholtz–(like) equation the required accuracy is already achieved. Multigrid iteration then is performed only for stability reasons and the cycling is adapted to the stability requirements of the actual model problem. In general, one very cheap multigrid cycle with a small number of coarse grids suffices. This is very desirable in a distributed parallel environment. In this context multigrid is used as an *adaptive stabilizer* rather than as a solver as hitherto.

The numerical algorithm is portably implemented on parallel environments by the grid partitioning approach with explicit message passing. Figure 1.1 shows the acceleration due to the algorithmic improvements and parallelization. The run time measurements of the meteorological model, which solves the Shallow Water Equations with fully dynamically adaptive local refinements, were performed on an IBM SP2. For a model problem,

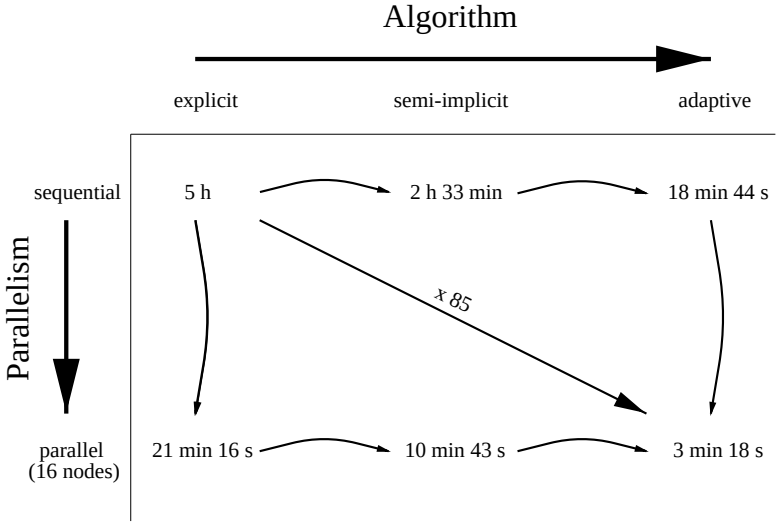

Figure 1.1: *Algorithmic and parallel acceleration*

which is described later in detail, the simulation with a semi–implicit time scheme is about two times faster than the reference computation with a purely explicit time scheme and smaller time steps. The local refinements

finally lead to a drastic improvement by a factor 16 in total in this case.

Parallelization of the dynamically adaptive and semi–implicit method accelerates the simulation by a factor 2.9 on 4 nodes and 5.7 on 16 , which results in a computation 85 times faster than the explicit sequential model. The further increase in speed from 4 to 16 nodes shows that the adaptive method is scalable in general. With the mentioned algorithmic improvements, however, the computational load for the Shallow Water Equations becomes very small in comparison to the parallelization overhead. Because of the small number of floating point operations per grid point the Shallow Water Equations are a worst case problem with respect to parallelization compared to more complex or even three–dimensional meteorological simulations.

In order to develop and implement this dynamically adaptive and parallel model different matters of the disciplines *mathematics*, *computer science*, and *meteorology* are combined in order to solve the difficulties listed below. Most of them are already known to appear in block structured adaptive and parallel methods, and some solution approaches have been reported (some references are provided). Nevertheless, a number of new ideas had to be developed and the existing approaches were partly further improved and partly tailored for the presented method in order to bring them together to a *running* fully self–adaptive parallel implementation.

Application Concerning a meteorological application, the time dependent Shallow Water Equations (2D) are implemented; they form the dynamic core of numerical weather simulations and are often used as a basic meteorological model (e. g. Barros [8], McBrian [52] and Williamson et al. [84]). This hyperbolic system includes advection and wave propagation and is more complex than standard exercise problems and thus more difficult to calculate with adaptive refinements and to parallelize. The high propagation speed of the waves gives reason for a semi–implicit time scheme. However, the design of the adaptive model fit even to more advanced meteorological systems of equations. The discretization is based on structured grids, which are commonly used in meteorological simulations, because they allow more computational optimizations compared to unstructured grids.

Time Scheme High spatial resolutions, which become available with local refinements, cause extremely small time steps with explicit schemes. A semi–implicit time scheme is applied therefore to calculate the fast waves implicitly and to reduce the time step limitations. Whereas a semi–implicit three–time–level scheme is already known (Haltiner and Williams [34]), a semi–implicit (non–semi–Lagrange) time scheme with only two time levels had to be developed. Especially with dynamically adaptive refinements two–time–level schemes are advantageous, since

this facilitates the administration of the refinement structures. The stability of the semi–implicit scheme is theoretically analyzed.

Solver For the computation of the semi–implicit time discretization a scalar Helmholtz–(like) equation has to be solved efficiently on a grid with local refinements with a parallel computer. This is performed by a parallel multigrid algorithm; multigrid can handle local refinements very naturally (e. g. Bai and Brand [6] and Joppich and Mijalković [43]), and by applying multigrid as smoother rather than solver, the coarse grid problem (Hempel and Schüller [36]) of parallel multigrid is solved.

Criterion The regions where higher spatial resolution is beneficial have to be detected automatically by an appropriate refinement criterion during the simulation. A purely mathematical criterion is applied that is based on an estimation of the local truncation error of the model equations (Stoer and Bulirsch [74], Bai and Brand [6], and Berger and Oliger [14]). Since the mesh size adapts only in space, the local truncation error with respect to space of the discretization of the Shallow Water Equations is estimated[2]. Not only the solution of the model equations is prone to oscillate at the refinement boundaries, the boundaries themselves tend to oscillate between two grid points of the coarse grid in fully dynamically adaptive simulations. Special care is taken to keep the boundaries of the refinement areas stable.

Blocking With regard to the application the refinement areas are designed as unification of rectangular and structured aligned blocks. The blocks are non–overlapping and oriented to the global grid in order to be used efficiently within the multigrid algorithm for the semi–implicit time scheme. However, the points specified by the criterion define irregularly shaped refinement areas in general. These areas have to be arranged in blocks (including occasionally few additional grid points) to create the current refinement structure (compare Lemke [47]).

With dynamically adaptive refinements it is important to use the blocks as long as possible. Initialization of new blocks not only costs local memory copies, but also introduces additional (volume) communication in a parallel environment. Therefore, continuity in time of the refinement structure is very important, and the refinement structure should be adapted rather than created fully anew in each adaptation step.

Nesting When integrating the model equations on a locally refined grid, the blocks of the refinement structure have to be embedded into the global

[2] The Helmholtz–(like) equation, which results from the implicit evaluation of some of the terms of the Shallow Water Equations, is solved for stability purposes only. Its truncation error is not a suitable refinement criterion.

grid. In a two–way interaction the boundary values of the refinements are supported by the global grid, whereas the global grid is improved by the interior fine grid values (Berger and Oliger [14] and Zhang et al. [85]).

For time dependent problems care has to be taken to avoid oscillations and instabilities of the solution at the refinement boundaries, especially for hyperbolic equations, which have no inherent diffusive and smoothing property as do parabolic equations (Grell, Dudhia, and Stauffer [31]).

Load Balance Parallel efficiency substantially depends on load balance. Whenever the refinement structure changes during simulation, a re–mapping of the blocks to the computational nodes is performed (e. g. Ritzdorf and Stüben [63] and Elbern [27]). Not only grid points have to be distributed as equally as possible, also the amount of transferred data and neighborhood relationships should be considered (Böhm and Speckenmeyer [16]). For this reason, load balancing is already taken into account when blocking the points that are detected to be refined in an adaptation step.

Implementation Last but not least, the implementation of such a complex parallel model is rather demanding. The refinement blocks, their number and sizes, and the computing nodes to which they are mapped change during simulation. The processors exchange messages, which vary in origin, destination, length, and content. Data structures have to be developed to administrate this information when designing the adaptive model for parallel computers.

A number of software libraries exist to support grid oriented parallelizations with adaptive mesh refinement (e. g. CLIC: Hempel and Ritzdorf [35], RSL: Michalakes [57], LPARX/KELP: Baden [5], and P++/AMR++: Lemke, Witsch, and Quinlan [46]). However, parallelization is performed directly with the explicit message passing interface MPI without use of a special communication library. The main reason is efficiency: the numerical algorithm is very efficient and with the low number of floating point operations of the Shallow Water Equations the data transfer must be highly optimized as well to achieve reasonable speed–ups on up–to–date parallel computers. The adaptive model is therefore designed for asynchronous communication to reduce synchronization and enables concurrent execution of communication and computation.

Special care is taken to combine adaptivity and parallelism in this time–dependent frame. The volume data communication, which appears in gen-

eral in adaptive methods, destroys the boundary–volume effect of grid partitioning parallelization approaches and saturates the parallel efficiency of adaptive cycles when solving increasing problem sizes (compare Ritzdorf and Stüben [63]). Therefore, the number of new initializations and re–mappings of refinement blocks are reduced, and the refinement structure is kept continuous in time as far as possible. Moreover, the exchange of volume data between coarse grid and refinement is reduced by introducing neighborhood conditions in the load balancing algorithm.

The requirement to reproduce the same results when changing the number of engaged computational nodes was released to optimize the number and sizes of the refinement blocks. The points that are additionally included in the refinement block may differ and the results can vary slightly.

The model is implemented in about 10 000 lines of the computer language 'C' to administrate the complex problem of changing refinement structures in a distributed parallel environment.

Due to the high potential of adaptive methods in saving computing time, much research is currently being done in this area even in combination with parallel computers (Section 2.5 "Related Work"). However, no real breakthrough of dynamically adaptive local refinements for meteorological applications on block structured grids has been achieved yet. The work presented in this paper, which combines adaptive multigrid and parallelism for a semi–implicit simulation, has to be regarded as an important step towards the application of adaptive multigrid in meteorological simulations.

The outline of this paper is as follows: Chapter 2 presents an overview of the meteorological application and the methodical aspects of adaptive multigrid and parallel computing used in this work. The Shallow Water Equations are derived from basic conservation laws and theoretically analyzed. Their ranges of dependence provide some insight into the behavior of their analytical solutions, which is fruitfully used when stabilizing the semi–implicit time scheme. As there is comprehensive literature about adaptive multigrid and parallel computing, only the essential ideas of these subjects are mentioned as far as they are relevant to the completeness of this paper (references for details are given in the sections).

The conceptual design of the adaptive method is given in Chapter 3 . A semi–implicit two–time–level discretization is developed and its stability is analyzed. The resulting Helmholtz–(like) equation of this time scheme is iterated by a multigrid algorithm that is adapted to provide stability. The introduction of local refinements along with the estimation of the local spatial truncation error as refinement criterion, the partitioning by hypergrids, and the nesting of the refinement blocks by blending are presented. At the end of this chapter, parallelization aspects such as load balancing, asynchronous communication, and parallel algorithms are discussed.

In Chapter 4 some details of the implementation of the adaptive method are documented. Data structures used to manage information for the changing refinement blocks and work units as basic computational units for coarse grain parallelization are presented.

The selected model problem *artificial cyclone* is given in detail in Chapter 5. The adaptive method is validated by comparing the results with a non–adaptive highly resolved simulation and an available analytical solution. Explicit and semi–implicit time stepping are compared and the benefits of adaptivity and parallelism are measured in run times.

Finally, Chapter 6 summarizes this paper and gives an outlook with respect to the usage of the adaptive method for more operational and three–dimensional meteorological models.

Before continuing, I wish to thank Professor Dr. Ulrich Trottenberg, who promoted my research in numerical weather forecasting at the Institute for Algorithms and Scientific Computing (SCAI) at GMD — German National Research Center for Information Technology. He introduced me to this important and fascinating field and gave me the opportunity to write this dissertation in optimal surroundings. I also thank Professor Dr. Ewald Speckenmeyer for letting me let me participate in the Graduate School "Scientific Computing" at the University of Cologne.

Special thanks to Dr. Wolfgang Joppich for always taking time for many questions and intense discussions. I would like to thank him also for reading my manuscript. I learned much from him during my time at the Institute.

I also want to thank many colleagues at the Institute. It is a pleasure to work so closely with many friendly and helpful specialists in different areas of mathematics and computer science. I wish I could mention them all by name, but the list would bee too long.

Chapter 2

Overview

2.1 About Meteorological Applications

Today's weather predictions have a quality of about 0.95 for one–day forecasts (24–hour predictions). This is considerably better than the 0.80 attained in the late sixties (Meyers [56][1]). This essential increase[2] is a result of higher spatial resolutions as well as of improved models and more accurate initial data.

Still higher resolutions are desirable; they not only provide more accurate predictions, but also resolve more small–scale features over complex terrain or of an evolving weather system (Baillie, Michalakes, and Skålin [7]). However, when increasing the resolution in all three dimensions, the computational demands increase dramatically by $\mathcal{O}(n^4)$, with n denoting the number of grid points of a grid dimension. Stability constraints enforce smaller time steps when decreasing the spatial mesh size.

One way to increase the resolution is to use more powerful machines. In 1992 a breakthrough in parallel computing for numerical meteorology was achieved by the parallelization of the operational Integrated Forecasting System (IFS) of the European Centre for Medium Range Weather Forecasts (ECMWF) in Reading (Gärtel, Joppich, and Schüller [30]). This system is still running with an improved degree of parallelism to provide a global 10–day weather forecast every day.

In the meantime, a number of other weather models have been parallelized for distributed memory computers (e. g. MC2: Thomas et al. [81], HIRLAM: Skålin and Bjørge [69] and MM5: Michalakes [58]) and most of the currently developed new generation of comprehensive weather prediction models is already designed for parallel machines (e. g. Schättler and Krenzien [68]).

[1] The quality is measured by correlation coefficients for the 500 hPa–level.

[2] The increase is essential, which becomes obvious when considering the misses to the perfect correlation of 1.0 (reduced from 0.2 to 0.05).

As high end parallel computers are already being used in numerical weather prediction and because it is extremely difficult and expensive to exploit the performance of even faster systems (e. g. Cassirer et al. [22]), research on fast and parallel algorithms is still important.

A very promising idea in this context is therefore to adapt the resolution locally to the actual requirements. This is already being done to some extent: *Global models* predict the weather of the whole world with a typical spatial resolution between 30 km and 100 km, whereas *regional models* provide higher resolution and a more detailed forecast for designated regions (e. g. for Germany or for Europe). Since regional weather forecasts for the next few days are globally influenced, global simulations are necessary in order to provide boundary data for regional medium–range weather predictions.

Although global and regional models simulate weather and also climate with very similar methods, their different computational domains have algorithmic consequences. Because global simulations are performed on a sphere, *spectral models* (Orzag [59] and Machenhauer [50]) are often applied, which are well suited for this domain in combination with large scales and periodic boundary conditions. A drawback of spherical discretizations along longitudes and latitudes is the singularities at the poles and the unintentionally high resolution in their surroundings.

For regional simulations *grid point models* with easier to handle finite–difference discretizations based on Cartesian grids are usually used. Problems with spherical anisotropies can be effectively solved by using a rotated coordinate system, so that the computational domain is sufficiently far from the poles. Moreover, grid point approaches have an inherent locality, which is advantageous for simulating local phenomena and also with regard to parallel computers (grid partitioning). Because of this advantage global grid point models based on triangular grids are also currently being developed (Majewski [51]).

Moreover, it is also desirable to adapt the mesh size locally *within* global and regional models and to provide high resolution only where it is beneficial in order to save computational costs elsewhere. This can be done with *static* refinement areas that are known and defined prior to the simulation (e. g. for regions of special interest or with complex orography) or even *dynamically*, by adapting the resolution to the requirements of the evolving weather situation (e. g. for weather fronts or cyclones). Since the weather is being simulated itself and is not known beforehand, the regions that require high resolution have to be detected by a suitable mathematical or meteorological criterion and adapted automatically during simulation.

A *one–way interaction*, which is the boundary support for the fine grid with coarse grid data, is sometimes sufficient for static refinements. With dynamically adapted refinements, however, *two–way interactions* are necessary, where the more accurate results of the fine grids also have a feedback

on the coarse grid. This prevents the solutions on the fine and coarse grids from diverging from each other, which would hinder the release of the refinement regions, once they have been introduced. Moreover, with a two–way interaction a locally more accurate simulation can improve global accuracy and the overall quality of the forecast.

Especially with dynamic local refinements higher resolutions and more accurate predictions become available at lower computational costs.

The structure of commonly used regional weather prediction models implies constraints for parallelization with respect to data decomposition and load balance. Comprehensive meteorological models consist of two main parts: the *dynamics* and the *physics*.

In the dynamics, the partial differential equations describing fluid motion are integrated. Advection and gravity pressure balance are integrated from time step to time step with various alternatives for the time scheme. With *(fully) explicit* Eulerian[3] schemes the sizes of the time steps are limited by the fastest moving waves, which are the gravity waves (in hydrostatic models). To save computing time *split–explicit* schemes are often applied where the slower advection is integrated with larger steps and the gravity waves by small time steps. Another alternative is to compute the gravity waves implicitly, which resolves the time step limitation; only large time steps are performed. Such a time scheme is called *semi–implicit* and is applied for the adaptive method presented in this paper.

However, also with semi–implicit time schemes the time step sizes are still restricted for stability rather than accuracy, and semi–implicit *semi–Lagrange* schemes, which do not limit the time steps for stability, become popular (Staniforth and Côté [73] and Bates [10]). Both semi–implicit Eulerian and semi–implicit semi–Lagrangian schemes require the solution of a Helmholtz–(like) equation for every time step. Much of the design of this paper is therefore usable for semi–Lagrangian schemes as well.

The physics is the second main part of weather forecast models. It comprises the physical processes like radiation, precipitation, evaporation, etc. Usually the physics are computed after the dynamics and complete a full time step of the model. Since the physical processes are strongly coupled in the vertical direction (e. g. radiation and rainfall are almost purely vertical processes), grid decompositions for parallelization in this direction are avoided. By partitioning the grid only horizontally, existing sequential physics packages can be re–used and, moreover, a high degree of parallelism can be achieved for this part of the simulation.

Typically, the physics require about one–third of the computational costs of the dynamics, but much higher costs arise depending on the complexity of the physical models. Furthermore, the costs vary considerably depending

[3] non–Lagrangian

on the different physical processes that are simulated for the local weather simulation (e. g. convection (ascending and descending air) arises only locally). This is a serious difficulty for parallel computers, since imbalances of computational work are introduced. Dynamic mapping algorithms have been developed for this reason (e. g. Elbern [27]).

In contrast to the physics, the dynamics have essential horizontal dependencies. With horizontal grid partitionings, data transfers must take place on parallel systems with distributed memory. In this paper we will concentrate on the dynamics part. On the one hand, it is the basis of full weather prediction models; on the other hand, it is algorithmically more demanding in combination with parallel systems (as usually only horizontal partitionings are applied).

Basic model equations of the dynamics in three space dimensions are the *Primitive Equations* (e. g. Kwizak and Robert [44]). However, since the motion of the atmosphere is predominantly horizontal, simplified two–dimensional models often serve as reasonable model problems. Model equations describing two–dimensional atmospheric flow are the *Shallow Water Equations*, which are presented in detail in Section 2.2 .

2.2 The Shallow Water Equations

The Shallow Water Equations (SWE) are considered model problem for meteorological applications. They form the dynamic basis of comprehensive weather prediction models and are often used as a first approach when developing and implementing new numerical weather prediction models (e. g. Cassirer, Hess, Jablonowski, and Joppich [21]) and also for benchmarks on parallel computers (e. g. McBrian [52]) . The SWE already describe meteorological phenomena (advection, Rossby–waves and gravity waves) that have to be considered when designing new algorithms for numerical weather simulations. Essential characteristics like accuracy and stability can be investigated (e. g. in combination with adaptive refinements).

Nevertheless, the SWE require essentially less effort and computer power, when implementing and performing validation and test calculations in comparison to three–dimensional weather models. They are also simple enough to allow theoretical investigation like the representation of domains of dependence and influence (Section 2.2.2), stability analyzes (Section 3.1.3), or even the comparison with analytical solutions in some cases (Section 5.1) .

The name of the SWE comes from hydrodynamics, since they describe the flow and the waves of water in case the depth of water is small compared to the radius of curvature of its surface (Stoker [75], e. g. they are used to model breaking waves at shallow beaches).

Considering the atmosphere as an incompressible fluid and assuming the hydrostatic law (2.3) (which is a good approximation for large–scale

motions, Haltiner and Williams [34]) the SWE are a first approximation to atmospheric flows. In this sense the atmosphere is regarded as *shallow* with respect to its extension around the globe.

In this paper the SWE are applied in time–dependent and nonlinear form with two space dimensions and in spherical coordinates. Formulated as in Equation (2.7) they describe the following three phenomena:

- *Gravity Waves*
  These are in general fast waves that propagate mediated by the gravity force due to variations in hydrostatic pressure in the atmosphere. They are the equivalent of water waves. Although they play a role in the so–called *geostrophic adjustment* process (Randall [61]), their influence in weather forecasts is small. Therefore they are generally considered disturbing oscillations (meteorological *noise*), when they propagate in numerical weather models.

- *Advection*
  This describes the motion of the atmosphere (wind). Advection is generally much slower than the speed of the gravity waves.

- *Rossby Waves*
  These are large–scale waves caused by the varying Coriolis force depending on latitude. Very important in global weather models, they are of minor interest in local and small–scale motions.

In Table 2.1 the main synoptic pressure levels and their average heights (Meyers [56]) are listed. Also included are the velocities of the corresponding gravity waves according to Formula (2.14). The high velocity of the gravity waves and their minor influence on the weather have to be considered when discretizing the SWE (Sections 2.5 and 3.1.1).

The 500 hPa–level has an outstanding position, since about half of the mass of air of the atmosphere is below and half above. Nevertheless, we will concentrate later in Chapter 5 on the more extreme levels with 850 hPa and 200 hPa.

pressure [hPa]	height [km]	wave speed [m/s]
100	16.0	396
200	12.0	343
300	9.0	297
500	5.5	232
700	3.0	172
850	1.5	121

Table 2.1: *Main pressure levels and speed of gravity waves*

In Section 2.2.1 the SWE are derived from conservation laws and the hydrostatic approximation. A more mathematical insight into the behavior of the solutions of the SWE is presented in Section 2.2.2 .

2.2.1 Derivation

For the systematic completeness of this paper the SWE are derived from the laws of conservation of mass and momentum and with the assumption of the hydrostatic law (compare Courant and Friedrichs [24]) .

Let x and y be the horizontal coordinates and z the vertical coordinate of a Cartesian coordinate system. The water extends from the bottom $z=0$ to the free surface $z=\tilde{h}$, so $\tilde{h}$ is a function of x and y and the time t . With the velocity components u , v , and w of the coordinates x , y , and z the *continuity equation* reads

$$u_x + v_y + w_z = 0 \tag{2.1}$$

and the *conservation of momentum* states

$$\begin{aligned} \rho\frac{du}{dt} &= -p_x \\ \rho\frac{dv}{dt} &= -p_y \\ \rho\frac{dw}{dt} &= -p_z - g\,\rho \quad , \end{aligned} \tag{2.2}$$

with pressure p and acceleration of gravity g . Constant density ρ is assumed.

The *hydrostatic law*

$$p = g\,\rho\,(\tilde{h} - z) \tag{2.3}$$

states that p depends only on the height of the column of water above. This can be shown to be a good approximation for shallow water (Stoker [75]) .

With Equation (2.3) the pressure gradients p_x , p_y , and p_z become independent on z and with Equation (2.2) also the horizontal velocities u and v , in case they were independent on z at some initial state. The vertical acceleration $\frac{dw}{dt}$ becomes zero. The first two equations of System (2.2) then formulate the first two equations of System (2.5) for the *geopotential* $h = g\,\tilde{h}$.

Equation (2.1) is integrated over the height of the column of water

$$0 = \int_0^{\tilde{h}} (u_x + v_y + w_z)\; dz = \tilde{h}\,(u_x + v_y) + w\Big|_{z=0}^{z=\tilde{h}} \quad , \tag{2.4}$$

and with the kinematic conditions

$$w\Big|_{z=0} = 0 \qquad \text{and} \qquad w\Big|_{z=\tilde{h}} = \frac{d\tilde{h}}{dt}$$

the third equation of System (2.5) follows

$$\begin{aligned} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial h}{\partial x} &= 0 \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial h}{\partial y} &= 0 \\ \frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y} + h\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) &= 0 \quad . \end{aligned} \tag{2.5}$$

This system is called the time dependent SWE in convective formulation for the geopotential h.

Sometimes, the SWE are used in linearized form assuming constant values for the coefficients of the derivatives. In one dimension the linearized system reads

$$\begin{aligned} \frac{\partial u}{\partial t} + u_0\frac{\partial u}{\partial x} + \frac{\partial h}{\partial x} &= 0 \\ \frac{\partial h}{\partial t} + u_0\frac{\partial h}{\partial x} + h_0\frac{\partial u}{\partial x} &= 0 \quad , \end{aligned} \tag{2.6}$$

with constant u_0 and h_0. Theoretical investigations like linear stability analyzes can be performed with this simplified system.

In the meteorological context the Coriolis force and the acceleration due to orography are also included in the SWE. The Coriolis force results from the rotation of the earth and is of essential importance for large–scale atmospheric flow. The Coriolis parameter $f = 2\,\Omega\,\sin\varphi$ is defined by the angular velocity $\Omega = \frac{2\pi}{86400s}$ of the earth and the latitude φ. With h^s denoting the orography (or the height of the bottom of the water) and with the Coriolis force the SWE in Cartesian coordinates are written

$$\begin{array}{llll} \overbrace{\qquad\qquad\qquad}^{advection} & \overbrace{\quad}^{Cor.\ f.} & \overbrace{\qquad\qquad\qquad}^{grav.\ terms} & \\ \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} & - f\,v & + \frac{\partial (h+h^s)}{\partial x} & = 0 \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} & + f\,u & + \frac{\partial (h+h^s)}{\partial y} & = 0 \\ \frac{\partial h}{\partial t} + u\frac{\partial h}{\partial x} + v\frac{\partial h}{\partial y} & & + h\left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right) & = 0 \quad , \end{array} \tag{2.7}$$

in which the terms prescribing advection, Coriolis force, and gravity waves are indicated.

2.2.1.1 Spherical Coordinates

For regional area models the SWE are solved on a rectangular section of the globe that is bounded by two longitudes and two latitudes. System (2.7) is transformed into spherical coordinates $\lambda \in [0, 2\pi]$, $\varphi \in [-\pi, \pi]$, and the radius

of the earth a. Details of the transformation are provided in Section A.1. On the rotating sphere the Coriolis force is described in good approximation by the Coriolis terms $\left(f + \frac{u \tan\varphi}{a}\right) v$ and $\left(f + \frac{u \tan\varphi}{a}\right) u$ (see Haltiner and Williams [34]). The full system of equations as applied and implemented in this work then reads

$$\begin{aligned}
\frac{\partial u}{\partial t} + \frac{u}{a\cos\varphi}\frac{\partial u}{\partial \lambda} + \frac{v}{a}\frac{\partial u}{\partial \varphi} - \left(f + \frac{u\tan\varphi}{a}\right) v + \frac{1}{a\cos\varphi}\frac{\partial (h + h_s)}{\partial \lambda} &= 0 \\
\frac{\partial v}{\partial t} + \frac{u}{a\cos\varphi}\frac{\partial v}{\partial \lambda} + \frac{v}{a}\frac{\partial v}{\partial \varphi} + \left(f + \frac{u\tan\varphi}{a}\right) u + \frac{1}{a}\frac{\partial (h + h_s)}{\partial \varphi} &= 0 \\
\frac{\partial h}{\partial t} + \frac{u}{a\cos\varphi}\frac{\partial h}{\partial \lambda} + \frac{v}{a}\frac{\partial h}{\partial \varphi} + \frac{h}{a\cos\varphi}\left(\frac{\partial u}{\partial \lambda} + \frac{\partial (v\cos\varphi)}{\partial \varphi}\right) &= 0 \quad .
\end{aligned} \tag{2.8}$$

2.2.1.2 Relation to the Euler Equations

It should be mentioned here that the SWE are equivalent to the Euler Equations for polytrophic gases with adiabatic exponent $\gamma = 2$, see Equation (2.9). To see that, the pressure $\tilde{p} = \frac{1}{2} g \tilde{h}^2$ and the density $\tilde{\varrho} = \tilde{h}$ are introduced. The SWE (2.5) can then be formulated

$$\begin{aligned}
u_t + u u_x + v u_y &= -\frac{\tilde{p}_x}{\tilde{\varrho}} \\
v_t + u v_x + v v_y &= -\frac{\tilde{p}_y}{\tilde{\varrho}} \\
\tilde{\varrho}_t + (\tilde{\varrho} u)_x + (\tilde{\varrho} v)_y &= 0 \quad ,
\end{aligned} \tag{2.9}$$

where $\tilde{p} = \frac{1}{2} g \tilde{\varrho}^2$ holds.

The speed of sound of the Euler Equations (2.9), $\tilde{a} = \sqrt{\frac{\mathrm{d}\tilde{p}}{\mathrm{d}\tilde{\varrho}}}$, is equal to the speed of the gravity waves of the SWE as given in Equation (2.14).

2.2.2 Characteristic Surfaces

With regard to the discussions in Sections 3.1 and 3.2 the theory of characteristics is applied to the SWE. For a more general discussion of this theory, which is valuable especially for hyperbolic initial value problems, we refer to Sauer [65] or Hirsch [39].

2.2.2.1 Ranges of Dependence and Influence

The *characteristics* give useful information about the behavior of partial differential equations, especially for hyperbolic systems such as the SWE. Deteriorations and discontinuities (shocks) propagate along them, and they define domains of dependence and influence.

In three dimensions (two space and one time dimension) the characteristics are characteristic surfaces. For their derivation a smooth plane $t = t(x, y)$ with continuous derivatives $r = t_x$ and $s = t_y$ is assumed. Inner derivatives of a function $f(t, x, y)$ with respect to the plane t are derivatives in direction to the tangents of the plane; in x– and y–direction: $\frac{\delta f}{\delta x} = f_x + r\, f_t$ and $\frac{\delta f}{\delta y} = f_y + s\, f_t$, respectively. The partial derivative f_t is called an outer derivative.

Transformation of System (2.5) to inner derivatives results in a linear system for u_t, v_t, and h_t

$$\begin{aligned}
(u\,r + v\,s - 1)\,u_t + r\,h_t &= u\,\frac{\delta u}{\delta x} + v\,\frac{\delta u}{\delta y} + \frac{\delta h}{\delta x} \\
(u\,r + v\,s - 1)\,v_t + s\,h_t &= u\,\frac{\delta v}{\delta x} + v\,\frac{\delta v}{\delta y} + \frac{\delta h}{\delta y} \\
h\,r\,u_t + h\,s\,v_t + (u\,r + v\,s - 1)\,h_t &= u\,\frac{\delta h}{\delta x} + v\,\frac{\delta h}{\delta y} + h\left(\frac{\delta u}{\delta x} + \frac{\delta v}{\delta y}\right) \quad .
\end{aligned} \tag{2.10}$$

In case there are given values for u, v, and h on the plane, the inner derivatives and the right hand side of Equation (2.10) are also determined. If

$$\mathcal{R} := \begin{vmatrix} u\,r + v\,s - 1 & 0 & r \\ 0 & u\,r + v\,s - 1 & s \\ h\,r & h\,s & u\,r + v\,s - 1 \end{vmatrix} = 0 \tag{2.11}$$

is *not* true, the outer derivatives u_t, v_t, and h_t are given, which provide constraints for the solution of the partial differential equation in the neighborhood of t.

However, in case Equation (2.11) holds, the outer derivatives can exist only, if also

$$\mathcal{V} := \begin{vmatrix} u\,r + v\,s - 1 & 0 & u\,\frac{\delta u}{\delta x} + v\,\frac{\delta u}{\delta y} + \frac{\delta h}{\delta x} \\ 0 & u\,r + v\,s - 1 & u\,\frac{\delta v}{\delta x} + v\,\frac{\delta v}{\delta y} + \frac{\delta h}{\delta y} \\ h\,r & h\,s & u\,\frac{\delta h}{\delta x} + v\,\frac{\delta h}{\delta y} + h(\frac{\delta u}{\delta x} + \frac{\delta v}{\delta y}) \end{vmatrix} = 0 \quad , \tag{2.12}$$

which poses a condition for the values u, v, and h on the plane t, and represents the fact that solutions to the partial differential equation evolve along planes with $\mathcal{R} = 0$. [4]

Equation (2.11) is a partial differential equation of the first order; its solutions are the characteristic surfaces. For the SWE there are three different real solutions, which is the mathematical background as to why the SWE in Formulation (2.5) are classified as hyperbolic. The solutions are

$$u\,r + v\,s - 1 = \begin{cases} 0 \\ \pm\sqrt{h\,(r^2 + s^2)} \end{cases} \quad . \tag{2.13}$$

[4] The conditions $\mathcal{R} = 0$ and $\mathcal{V} = 0$ can be used to solve a hyperbolic equation on a coordinate system based on the characteristics. In two space dimensions this was carried out (Hess [37]) for the steady potential flow problem.

A linearization by keeping u, v, and h constant provides a linear approximation for the characteristic surfaces. The first solution of Equation (2.13) degenerates into a line in the direction of the advection with $\frac{dx}{dt} = u$ and $\frac{dy}{dt} = v$; the other two solutions define a cone with $\frac{dx}{dt} = u \pm \frac{\sqrt{h}\, r}{\sqrt{r^2+s^2}}$ and $\frac{dy}{dt} = u \pm \frac{\sqrt{h}\, s}{\sqrt{r^2+s^2}}$. Hence,

$$\left(\frac{dx}{dt} - u\right)^2 + \left(\frac{dy}{dt} - v\right)^2 = h \quad . \tag{2.14}$$

This result means, that disturbances propagate with the speed $\sqrt{h} = \sqrt{g\tilde{h}}$ relative to the advection[5].

In Figure 2.1 the characteristic cone of a point P is displayed, and the corresponding linear approximations of its domain of dependence and zone of influence are shown for the preceding and succeeding time step. The domain

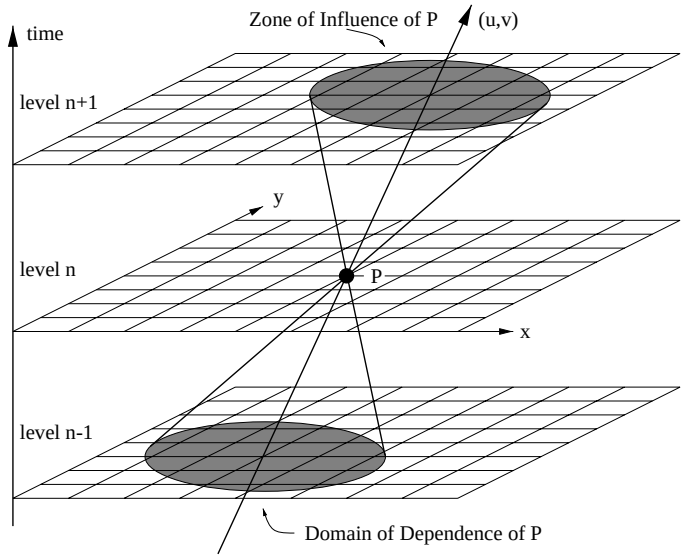

Figure 2.1: *Domain of dependence and zone of influence*

of dependence of a point P is the region of the initial values (or values of the precedent time level[6]) that contribute to the solution in P. It is limited by the characteristics that define the maximal speed of propagation within the hyperbolic system. Points outside this region can not influence the solution in P. Conversely, the zone of influence of P is the region that is influenced

[5] Since $\sqrt{h}$ is higher in the crest of the wave than in the trough, waves tend to break in shallow water as experienced at the beach.

[6] In case the partial differential equations are solved as a sequence of boundary value problems as in our case.

by the value in P. It is also bounded by the characteristic surfaces; points outside this region do not depend on the value in P.

With regard to the discretized SWE the domain of dependence of P consists of a number of grid points of the previous time level as displayed in Figure 2.1. This result is important for the numerical solution of the hyperbolic system (see Section 3.2). All information that P depends on physically should be supported by the numerical scheme.

2.2.2.2 The Initial Value Problem

When solving partial differential equations it is essential to formulate a *well posed problem*[7]. From the mathematical point of view a *global* shallow water model can be classified as a hyperbolic initial value problem with periodical boundary conditions, which is well posed.

In operational environments the result of global simulations is often used as boundary data for *local* models. However, when prescribing all variables (h, u, and v in case of the SWE) as Dirichlet values for the local model an over–determined problem results, and no solution exists in general (e. g. Sauer [65]). Consequently, local models are usually *nested* into global models. The values of the global model are introduced in a smooth way in a boundary region that has a width of several grid points. Such a nesting procedure is applied in this work for the local refinements and presented in Section 3.3.3 in detail.

When embedding a local model into a global model in this way the local model can be considered as part of the global model, especially if a two–way interaction is applied, so that the combined problem stays well posed.

2.3 Adaptive Multigrid

For a detailed description of multigrid techniques we refer to Brandt [17], Joppich and Mijalković [43], and Trottenberg, Oosterlee, and Schüller [82]. Here only the rough idea is mentioned, details are given only as far as they are necessary for this paper.

A number of iterative solvers (e. g. Jacobi, Gauss–Seidel, SOR) eliminate high frequency error components very effectively, while low frequency components are reduced at a much lower rate. Within multigrid algorithms these solvers are used as *smoothers* to eliminate the high frequency error components, while low frequency components are transferred to a *coarse grid*. The coarse grid has considerably fewer grid points than the original fine grid, and these low frequency components can be eliminated more efficiently there.

[7] A mathematical problem is called "well posed" if a unique solution exists, and if it depends on the given initial and boundary conditions in a continuous way.

Moreover, because these low–error components of the fine grid have higher frequencies with respect to the coarse grid, these error components can usually be reduced in the same way as on the original fine grid. High frequencies (with respect to the coarse grid) are smoothed again, and the remaining low frequencies are transferred to an even coarser grid. This procedure results in a recursive algorithm with several levels of coarse grids.

The idea of *Full Multigrid (FMG)* is worth mentioning, although it is not used in this paper: Coarse grid approximations of the solution are used in order to find reasonable initial values for the finer grids. These initial values are improved by multigrid iterations that use the coarse grids again. This method has an interesting property for an elliptic boundary value problem as Problem (2.15): After only one FMG–iteration the algebraic error is in the order of the discretization error (under some assumptions, see Brandt [17]). Since this accuracy is supposed to be sufficient in most cases, an optimal order $\mathcal{O}(n)$ algorithm (n denote the number of grid points) results for the solution of the problem.

In Section 2.3.1 a two–level scheme is presented that forms the basis of multi–level iterations. The discretization on two grids with different mesh sizes makes it possible to estimate the *local truncation error* in space. This error can be used to formulate a *refinement criterion* to define regions of the computational domain where higher spatial resolution is appropriate, see Section 2.3.2 . The combination of multigrid and local refinement is very natural; two–level and multilevel schemes need only be slightly adapted (Section 2.3.3).

2.3.1 Full Approximation Scheme (FAS)

In order to formulate a two–level *Full Approximation Scheme (FAS)* an elliptic Dirichlet boundary value problem is defined on a rectangular two–dimensional domain $\mathbf{G} = [0,1] \times [0,1]$ by

$$\begin{aligned} L\,u &= f \quad \text{in} \quad \overset{\circ}{\mathbf{G}} \\ u &= g \quad \text{in} \quad \partial\mathbf{G} \quad , \end{aligned} \tag{2.15}$$

where L is a (not necessarily linear) second order operator, f is a smooth right hand side and g are the boundary values. To solve this problem numerically it is discretized on a grid

$$\mathbf{G}_h = \left\{ (x_i, y_j) \middle| x_i = i\,\Delta x,\ y_j = j\,\Delta y;\ 0 \le i \le N_x,\ 0 \le j \le N_y;\ i,j \in \mathbb{N} \right\} \quad .$$

The subscript $h = (\Delta x, \Delta y) = (1/N_x, 1/N_y)$ defines the mesh sizes in both directions and N_x and N_y the number of intervals in x– and y–direction, respectively.

(1)	*Pre–Smoothing*	$u_h := \mathcal{S}^{\nu_1}(u_h, L_h, f_h)$	on $\overset{\circ}{\mathbf{G}}_h$
(2)	*Computation of Residuals*	$r_h := f_h - L_h\, u_h$	on $\overset{\circ}{\mathbf{G}}_h$
(3)	*Restriction*	$r_H := I_h^H\, r_h$	on $\overset{\circ}{\mathbf{G}}_H$
(4)	*Update Right Hand Side*	$f_H := L_H\, \hat{I}_h^H\, u_h + r_H$	on $\overset{\circ}{\mathbf{G}}_H$
(5)	*Solve Coarse Grid Problem (Apply Multigrid Cycle)*	$u_H := L_H^{-1}\, f_H$	on $\overset{\circ}{\mathbf{G}}_H$
(6)	*Calculation of Increments*	$e_H := u_H - \hat{I}_h^H\, u_h$	on $\overset{\circ}{\mathbf{G}}_H$
(7)	*Prolongation*	$e_h := I_H^h\, e_H$	on $\overset{\circ}{\mathbf{G}}_h$
(8)	*Addition of Correction*	$u_h := u_h + e_h$	on $\overset{\circ}{\mathbf{G}}_h$
(9)	*Post–Smoothing*	$u_h := \mathcal{S}^{\nu_2}(u_h, L_h, f_h)$	on $\overset{\circ}{\mathbf{G}}_h$

Table 2.2: *Full approximation scheme (FAS), two–grid scheme*

The discretized problem reads:

$$\begin{aligned} L_h\, u_h &= f_h \quad \text{for} \quad u_h \in \overset{\circ}{\mathbf{G}}_h \\ u_h &= g_h \quad \text{for} \quad u_h \in \partial\mathbf{G}_h \quad , \end{aligned} \tag{2.16}$$

where L_h, f_h, and g_h are discretizations of L, f, and g, respectively. $\overset{\circ}{\mathbf{G}}_h$ denotes the inner points and $\partial\mathbf{G}_h$ the boundary points of $\mathbf{G}_h$.

A basic two–grid FAS scheme is formulated for Problem (2.16) in Table 2.2. The sign ':=' defines assignments in the algorithmic sense rather than mathematical equalities or definitions. With standard coarsening ($H = 2h$) the coarse grid $\mathbf{G}_H$ has half as many intervals as $\mathbf{G}_h$. Steps (1)–(9) are discussed in detail:

(1) (Pre–Smoothing) The high frequency error components are reduced by applying a smoothing operator $\mathcal{S}$ [8]. The superscript ν_1 defines the number of smoothing steps to be performed.

(2) (Computation of Residuals) The residuals of the fine grid values are calculated. They represent for the most part the remaining low frequency error components that have not been effectively damped by

[8] For Laplace, Poisson, or Helmholtz equations often red–black Gauss–Seidel iterations are used.

the smoothings.

(3) (Restriction) The residuals are transferred to the coarse grid $\mathbf{G}_H$ applying a grid transfer operator I_h^H . Usually, an averaging of neighboring fine grid values is performed to transfer the low frequency components correctly. Examples for I_h^H are *full weighting (FW)* and *half weighting (HW)*[9]. In stencil notation:

$$\overset{(FW)}{I_h^H} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_h^H \quad \text{resp.} \quad \overset{(HW)}{I_h^H} = \frac{1}{8}\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix}_h^H .$$

(4) (Update Right Hand Side) The right hand side of the coarse grid equation is calculated. The coarse grid operator L_H is an approximation of L_h , and usually the stencil of L_h is used[10]; however, it is applied on $\hat{I}_h^H u_h$, a restriction of the fine grid function u_h . The standard choice of $\hat{I}_h^H$ is *simple injection (INJ)*.

This right hand side f_H is calculated in this way rather than using a restriction $\hat{I}_h^H f_h$ of the original problem. This represents the fact that the coarse grid is used to calculate the original problem (2.16) with the accuracy of the fine grid. This divergence can be used to define a refinement criterion as discussed in Section 2.3.2 .

(5) (Solve Coarse Grid Problem) The coarse grid equation has to be solved, which is essentially cheaper than solving the original fine grid problem, since the coarse grid has about four times fewer grid points (with standard coarsening, $H = 2h$) and a direct solver might therefore be applicable. However, it is not necessary to solve the coarse grid equation exactly to achieve good convergence rates within the two–level scheme for the fine grid equation. The coarse grid equation can also be treated in the same way as the original fine grid problem using an even coarser grid. This idea results in a recursive multigrid FAS with several levels of coarse grids.

(6) (Calculation of Increments) The increments of the coarse grid values that result from the solution (iteration) of the coarse grid equation are calculated. These values represent the coarse grid correction.

(7) (Prolongation) The coarse grid correction is transferred to the fine grid. Only increments are interpolated to the fine grid (and not the coarse grid values itself) to prevent large interpolation errors. The

[9] In combination with red–black Gauss–Seidel smoothers HW can be optimized to *half injection (HI)*.

[10] An alternative is a Galerkin operator defined by the *Galerkin condition* $L_H = I_h^H L_h I_H^h$ with I_h^H and I_H^h being adjoint to each other.

transfer operator I_H^h defines the interpolation from the coarse to the fine grid. For bi–linear interpolation the stencil reads

$$I_H^h = \frac{1}{8} \left[\begin{array}{ccc} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{array} \right]_H^h \quad .$$

(8) (Addition of Correction) The transferred corrections are added to the fine grid values.

(9) (Post–Smoothing) A number of ν_2 smoothing steps $\mathcal{S}$ are carried out to reduce high error components that may have been introduced by the interpolation of the coarse grid correction.

Obviously u_H tends to u_h in case of convergence, although there is no direct assignment as $u_H = \hat{I}_h^H u_h$ within the FAS.

This two–level FAS as well as multi–level schemes can be parallelized by *grid partitioning* (see Section 2.4.3).

2.3.2 Local Discretization Error as Refinement Criterion

The local truncation error in space can be estimated by comparing the discretizations of the boundary value problem on the two grids $\mathbf{G}_h$ and $\mathbf{G}_H$.

The right hand side of the coarse grid equation of Step (5) of the FAS can be written

$$f_H = L_H \hat{I}_h^H u_h + r_H = L_H \hat{I}_h^H u_h + I_h^H (f_h - L_h u_h) = \tau_H^h + I_h^H f_h \quad , \quad (2.17)$$

introducing τ_H^h, the relative local discretization error of $\mathbf{G}_H$ and $\mathbf{G}_h$,

$$\tau_H^h = L_H \hat{I}_h^H u_h - I_h^H L_h u_h \quad . \quad (2.18)$$

This value represents the improvement of the accuracy of the discretization on grid $\mathbf{G}_h$ relative to $\mathbf{G}_H$. If an asymptotic expansion for the local discretization error τ_H of the discretization on grid $\mathbf{G}_H$ of the continuous problem of Equation (2.15) is assumed, τ_H can be extrapolated using τ_H^h.

A refinement criterion with a purely mathematical background[11] is yield that limits the local discretization error τ_H^h due to

$$|\tau_H| < c_{crit} \quad . \quad (2.19)$$

The points which do not fulfill the Criterion (2.19) are marked to be refined. An example is given in Figure 2.2 (left), where a critical region is indicated

[11] The physical or meteorological background of an application can also provide refinement criteria, e. g. the gradients of the solution.

by a thick bent line. The points that should be refined are indicated by dots.

However, in most cases it is not obvious how to define c_{crit} in order to achieve a maximal increase in accuracy by minimal sizes of the refinement region. A reasonable choice for c_{crit} is presented in Section 3.3.1 along with the application of the refinement criterion (2.19) to the two–dimensional time–dependent SWE (2.7).

2.3.3 Multilevel Adaptive Technique (MLAT)

The refined grids are defined in this paper (due to Bai and Brandt [6]) with a refinement ratio of $1:2$,, and the fine grid lines are aligned to the global grid (in the refinement area coarse grid lines coincide with grid lines of the refinement patches)[12].

In Figure 2.2 (left) the points selected by a refinement criterion lead to the refinement area, that is hatched in the picture. The resulting refined grid is displayed in Figure 2.2 (right). In general, the refinement may have a very irregular shape.

The idea of the *Multilevel Adaptive Technique (MLAT)* is to realize a refined grid as a composed grid $\mathbf{G}_{\mathcal{H}}$ consisting of a global grid $\mathbf{G}_H$ and a grid $\mathbf{G}_h$ with smaller mesh size that is called local refinement. The fine grid is placed on the global grid to increase the resolution locally and the accuracy locally as well as globally[13].

The global grid $\mathbf{G}_H$ has two functions simultaneously:

- At grid points outside the local refinement, the coarse grid is used to calculate the solution as if no refinement existed.
- At grid points inside the local refinement the solution is calculated on the fine grid. The points of the coarse grid are used to increase the rate of convergence of the multigrid iteration and to solve the corresponding coarse grid equation within the FAS.

The problem to be solved on the composed grid $\mathbf{G}_{\mathcal{H}} = \mathbf{G}_H \cup \mathbf{G}_h$ is (here, H denotes the mesh size of the global grid in contrast to h in System (2.16))

[12] Other approaches are arbitrary rotated refinement areas (e. g. Berger and Oliger [14]); different refinement ratios are possible as well.

[13] Another philosophy of local refinement is not to put additional refinement patches on a global grid, but to reduce the global fine grid $\mathbf{G}_h$ to those parts where high resolution is really necessary. The coarse grid that was formerly used in the multigrid context only to reduce low error components now becomes visible where no refinement exists and carries the solution there (e. g. McCormick [54]).

$$
\begin{aligned}
L_h\, u_h &= f_h \quad && \text{for} \quad && u_h \in \overset{\circ}{\mathbf{G}}_h \\
u_h &= I_H^h\, u_H && \text{for} && u_h \in \partial\mathbf{G}_h \\
u_H &= \hat{I}_h^H\, u_h && \text{for} && u_H \in \overset{\circ}{\mathbf{G}}_H \cap \overset{\circ}{\mathbf{G}}_h \\
L_H\, u_H &= f_H && \text{for} && u_H \in \overset{\circ}{\mathbf{G}}_H \setminus (\overset{\circ}{\mathbf{G}}_H \cap \overset{\circ}{\mathbf{G}}_h) \\
u_H &= g_H && \text{for} && u_H \in \partial\mathbf{G}_H \quad .
\end{aligned}
\tag{2.20}
$$

System (2.20) is closed and can be solved with the MLAT algorithm that is presented in Table 2.3 . In contrast to the FAS, the update of the right hand side in Step (4) is now restricted to points where the required fine grid values are available. For the other points $(\overset{\circ}{\mathbf{G}}_H \setminus (\overset{\circ}{\mathbf{G}}_H \cap \overset{\circ}{\mathbf{G}}_h))$ the usual discretization of f is applied in order to solve the coarse grid problem in step (5) . Another divergence from the FAS is the interpolation of the inner boundaries $\partial\mathbf{G}_h$ of the refinement in Step (9). The interpolation operator $I\!I_H^h$ usually is of a higher order (bi–cubic), since function values and not only increments are interpolated, and because discretization order and rate of convergence could be affected otherwise.

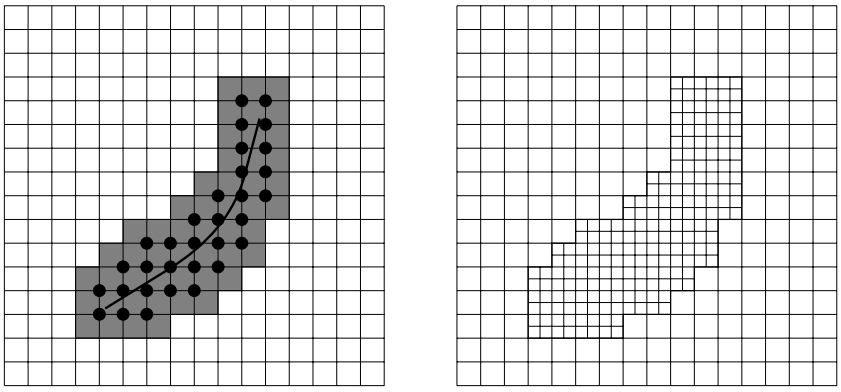

Figure 2.2: *Refinement area (left) and corresponding refined grid (right)*

However, this bi–cubic boundary interpolation was skipped when using MLAT in the context of the time dependent SWE. The details of how MLAT is actually applied in this paper are given in Section 3.3.4 .

2.4 Parallel Programming

During the last decade parallel computers have become a cutting–edge technology of high–performance and scientific computing (e. g. Stüben [77]). Whereas single processors have almost reached their maximal theoretical computational performance, that is limited by physical reasons (speed of

(1)	*Pre–Smoothing*	$u_h := \mathcal{S}^{\nu_1}(u_h, L_h, f_h)$	on $\overset{\circ}{\mathbf{G}}_h$
(2)	*Computation of Residuals*	$r_h := f_h - L_h\, u_h$	on $\overset{\circ}{\mathbf{G}}_h$
(3)	*Restriction*	$r_H := I_h^H\, r_h$	on $\overset{\circ}{\mathbf{G}}_H \cap \overset{\circ}{\mathbf{G}}_h$
(4)	*Update Right Hand Side*	$f_H := L_H\, \hat{I}_h^H\, u_h + r_H$	on $\overset{\circ}{\mathbf{G}}_H \cap \overset{\circ}{\mathbf{G}}_h$
(5)	*Solve Coarse Grid Problem (Apply Multigrid Cycle)*	$u_H := L_H^{-1}\, f_H$	on $\overset{\circ}{\mathbf{G}}_H$
(6)	*Calculation of Increments*	$e_H := u_H - \hat{I}_h^H\, u_h$	on $\overset{\circ}{\mathbf{G}}_H \cap \overset{\circ}{\mathbf{G}}_h$
(7)	*Prolongation*	$e_h := I_H^h\, e_H$	on $\overset{\circ}{\mathbf{G}}_h$
(8)	*Addition of Correction*	$u_h := u_h + e_h$	on $\overset{\circ}{\mathbf{G}}_h$
(9)	*Boundary Interpolation*	$u_h := I\!I_H^h\, u_H$	on $\partial\mathbf{G}_h$
(10)	*Post–Smoothing*	$u_h := \mathcal{S}^{\nu_2}(u_h, L_h, f_h)$	on $\overset{\circ}{\mathbf{G}}_h$

Table 2.3: *Multilevel adaptive technique (MLAT)*

light, elementary charge), increasing performance is still achieved by developing *parallel computers*. Up to thousands of single processors are combined in parallel systems. These parallel architectures can be classified into *Single Instruction Multiple Data (SIMD)* machines and *Multiple Instruction Multiple Data (MIMD)* machines, depending on whether single or multiple instruction streams are used.

SIMD systems perform the same instruction at the same time, but applied to different data. They can be programmed using array constructs that are supported by special compilers. Vector computers can be regarded as a particular SIMD class. Each processor of a MIMD system, on the other hand, performs its own instructions on different data. The data organization of MIMD machines can be *shared*, where each processor has accesses to a common global memory, or *distributed*, where each processor is provided only with its own local memory. In the latter case the processors are logically connected by data transfers in the form of messages. Distributed memory architectures allow parallel systems with a much higher number of processors, whereas shared memory systems are limited in size because of hardware and efficiency reasons.

This paper concentrates on MIMD architectures with distributed memory and the application of the *explicit message passing* programming model. This is a more general approach than *shared memory programming*, and can

be applied to a much greater variety of numerical algorithms. It provides maximal computational performance and portability; however, it is the most expensive and difficult parallelization approach.

In Section 2.4.1 basic characteristics of parallel algorithms are discussed. For explicit message passing the *Message Passing Interface (MPI)* (see Section 2.4.2) is the current standard that guarantees portability for many available parallel platforms. Parallelization of multigrid algorithms is effectively done by *grid partitioning*. An example is provided in Section 2.4.3 .

2.4.1 Characteristics of Parallel Algorithms

For modern numerical algorithms parallelism has become a very important characteristic in addition to numerical efficiency. Basic measures for parallel algorithms are parallel speed–up S_p and parallel efficiency E_p, defined as

$$S_p := \frac{T_1}{T_p} \quad \text{and} \quad E_p := \frac{S_p}{p} \quad , \tag{2.21}$$

where p is the number of processors, and T_1 and T_p are the solution times of the parallel algorithm calculated with one and with p processors, respectively. Using T_1 as reference time in Formula (2.21) neglects the fact, that sometimes a faster sequential algorithm exists. High parallel efficiencies can not be seen as a good result, however, if they are achieved by parallelizing a numerically slow algorithm. Therefore for a fair comparison the time T^{best} of the fastest *available* sequential algorithm should be used instead of T_1 .

However, when calculating a large problem it is sometimes not possible to solve it with only one processor and to measure T_1 (or T^{best}) at all (because of memory or time restrictions). The following approximations for S_p and E_p can be used in these cases:

$$\tilde{S}_p := \frac{\sum_{i=1}^{p} T^i_{calc}}{\max_{i=1,\ldots,p} (T^i_{calc} + T^i_{comm})} \quad \text{and} \quad \tilde{E}_p := \frac{\tilde{S}_p}{p} \quad , \tag{2.22}$$

where T^i_{calc} and T^i_{comm} denote the execution times for calculation and communication of processor number i, respectively, $i = 1, \ldots, p$.

$\tilde{S}_p$ and $\tilde{E}_p$ are only rough approximations, because in general parallelization introduces additional algorithmic and software overhead, and because usually not all calculation of an algorithm can be executed in parallel. The sequential fraction $\alpha \in [0, 1]$ of an algorithm limits the maximal speed–up according to *Amdahl's Law*

$$S_p \leq \frac{1}{\alpha + \frac{1-\alpha}{p}} \leq \frac{1}{\alpha} \quad . \tag{2.23}$$

This seems to be a severe restriction of parallelism; however, Amdahl's Law does not take into account that with a larger number of processors larger problems can also be solved.

In general it is not possible to distribute the computational work equally among the computational nodes. The load imbalance of a parallel algorithm limits parallel efficiency, since it leads to sequential parts or at least to parts with a reduced degree of parallelism. It should be balanced therefore as well as possible among the available processors. The load balance factor

$$\lambda_p = \frac{T_{calc}^{average}}{T_{calc}^{max}} \quad , \tag{2.24}$$

with $T_{calc}^{average} = \frac{1}{p} \sum_{i=1}^{p} T_{calc}^{i}$ and $T_{calc}^{max} = \max_{i=1,\ldots,p} T_{calc}^{i}$ can be used to estimate the parallel efficiency $\tilde{E}_p$. If T_{comm}^{i} is negligible in Equation (2.22);

$$\tilde{E}_p = \lambda_p \tag{2.25}$$

results. Since the maximal execution time T_{calc}^{max} determines the execution time of the whole parallel application, the maximal computational load is to be minimized.[14]

Parallel algorithms need to have special qualities in order to use the computational performance of parallel computers as well as possible and to achieve high parallel speed–ups and efficiencies.

- The *communication load* of a parallel algorithm is given by the number and sizes of the messages to be transmitted during the algorithm. It obviously should be minimized. An important factor of a parallel system is the relation between the performance of communication and the performance of computation. The time for buffering messages before and after transmits can not always be neglected and contributes to the parallel overhead.

- The *granularity* of algorithms defines the portions of computational work between the communication steps. Message exchanges not only cost time but also *synchronize* the processes. This leads to extra waiting times in case there are different load imbalances in the computational steps of an algorithm. Coarse granularity and large portions of work facilitate parallelization with good parallel efficiencies.

- The *scalability* of an algorithms describes the effect of an increasing number of processors on the parallel performance. There are two points of view:

[14] Sometimes the ratio between minimal and maximal execution time is used as the load balance factor. However, the resulting value is less important.

- The increased number of processors can be applied to a fixed problem size. Scalability then expresses how much faster the problem can be solved, as defined by parallel efficiency and speed–up.
- The other possibility is to increase the problem size according to the number of processors so that the portion of work on each processor stays constant. Of course, with increased computer power larger problems can be solved. This is an essential quality of parallel algorithms.

In this paper we concentrate on the first point of view and always compare the parallel adaptive algorithm with a reference problem of fixed size in order to show the acceleration of the simulation due to parallelism and adaptivity.

2.4.2 Explicit Message Passing — MPI

In the explicit message passing programming model the data interchanges are explicitly inserted by calls of communication subroutines. The *Message Passing Interface (MPI)* (Message Passing Interface Forum [28]) is the current standard interface[15].

Explicit message passing (by use of MPI) has two main advantages:

- It is the most portable parallel programming approach. It is possible to run MPI programs on most currently distributed memory parallel machines with an MPI interface. Even clusters of workstations can be used (a corresponding implementation of MPI already exists, however the lower communication performance reduces the parallel efficiency). Moreover, experience shows that explicit message passing runs very efficiently on shared memory parallel platforms.

- It is the fastest and the most general parallelization approach. In comparison to the data parallel programming model a larger class of numerical algorithms can be parallelized. Because of the direct control of the communication of the parallel system, in general coarser granularity and lower communication load can be achieved.

The disadvantage of explicit message passing is that it is a very difficult and time–consuming way of parallelization.

The most important functionality of MPI (besides initialization) is the *sending* and *receiving* of data. Different modes for sending and receiving exist. The standard is the blocking mode that is initiated by calling the subroutines `MPI_Send` and `MPI_Recv` of the parallel interface MPI. In blocking

[15] Other portable interfaces are *Parallel Virtual Machine (PVM)* (Sunderam et al. [78]) and *Parallel Macros (PARMACS)* (Calkin et al. [20]).

mode the data is copied from or into its memory buffer before the subroutine returns. In this way the buffer can be reused immediately after the call for the next communication that takes place. When receiving a message the processor is blocked until the message the processor is waiting for arrives. Not only waiting time is lost, also special care has to be taken to avoid *deadlocks*.

The subroutines of the non–blocking mode `MPI_Isend` and `MPI_Irecv` only initiate communications without completing them. Special subroutines such as `MPI_Wait` or `MPI_Test` can be used in order to *test* whether messages have arrived, or to wait at a later stage of the algorithm. Between initiation and completion of the messages other computation or communication tasks can be performed (of course the message buffer must not be accessed in this case). Not only is the interleave of computation and communication supported[16], more general communication pattern can be used (two examples of algorithms with non—blocking communication are given in Section 3.4.3). Of course, the complexity of the parallel algorithm is increased.

Besides sending and receiving much more functionality exists in MPI, e. g. there are global operations such as *broadcasting* or *reducing* data.

2.4.3 Grid Partitioning

For multigrid algorithms the *grid partitioning* parallelization approach is well suited. The global grid is subdivided and distributed among the processors; an example is given in Figure 2.3 . A grid with 9×9 points is

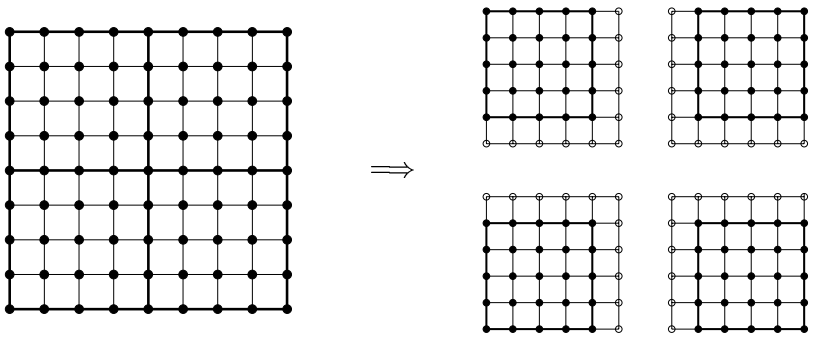

Figure 2.3: *Grid partitioning, overlap width 1*

partitioned into 2×2 subgrids. Points of the boundaries between the subgrids (inner boundaries) are distributed to all adjacent subgrids. This is not necessarily the case, but it has advantages in combination with local refinements and multigrid (see Section 3.3).

The subgrids are extended by one grid line at the inner boundaries and form *overlap areas* (indicated by hollow points in the figure). In this way

[16] On some machines (e. g. IBM SP2) it is possible to transfer data in the background concurrently to computations using designated communication processors.

grid operations (e. g. smoothing steps) can be applied on each processor in parallel. After one or more grid operations (depending on the overlap width) the overlap regions are updated by data exchanges. The coarse grids of multigrid algorithms can be partitioned in the same way, so that corresponding parts of the fine and coarse grids reside on the same processor. Inter–level exchanges (i. e. restriction and prolongation) are then performed without additional communication.

In Hess and Joppich [38] a multigrid algorithm that is parallelized by grid partitioning is applied to a Helmholtz equation. The comparison to an algorithm with Fast Fourier Transformation and Gauss Elimination that is parallelized by a transposition strategy shows good numerical and parallel performance of (parallel) multigrid for reasonable large problem sizes.

The computational work of grid operations increases with the total number of grid points, whereas only lower–dimensional boundary data has to be exchanged. Because of the boundary–volume effect, coarse granularity at low communication load is achieved for large partition sizes. However, if the sizes of the partitions are small, the portion of work of the grid operations may become small in comparison to the communication overhead of the boundary exchanges.

Small partitions are a problem especially for parallel multigrid algorithms if the small coarse grids are partitioned for the same high number of processors as the fine grid. A remedy for this is the so–called *agglomeration*; at certain coarse levels all data is gathered to one or a smaller number of processors. The sizes of the partitions of the coarse grids are increased, and the coarse grid correction can be calculated with a reduced communication load for the boundary exchanges. Afterwards the coarse grid results are redistributed again. This procedure affects load balancing, some of the processors are not engaged in computations of the agglomerated coarse levels; however, the imbalance is small if the agglomerated grids are small.

2.5 Related Work

In this section some other work on adaptive methods for numerical simulations is mentioned. Because of the attractiveness of adaptive refinements in general and especially in meteorology, comprehensive research is currently being done in this area. However, only selected references are given in the following with no claim to completeness.

Static refinements have already become operational in meteorological weather forecasts in different scenarios. The Deutscher Wetterdienst (DWD) runs a global weather prediction model and uses its results to support boundary values for two regional models with higher resolutions (e. g. Schättler and Krenzien [67]) in today's operational mode. However, as in most cases, there is only a one–way interaction from the global grid to the regional model; the

more accurate results from the regional models are not used to improve the global forecast.

Comprehensive models with two–way interaction also exist (e. g. Grell, Dudhia, and Stauffer [31] and Sathye et al. [64]), where the feedback from the refinement prevents the solutions on the related grids from diverging from each other. Since the simulations on the related grids must run synchronously in this case, two–way interactive simulations are usually performed within one model, which is a demanding concern for parallel software engineering.

Other approaches of static refinement in meteorology exist as well (e. g. Staniforth and Côté [73] present a model where the resolution of a global grid is increased for the desired region by a continuously decreasing mesh size).

The situation is completely different for dynamically adaptive simulations. In most cases the user has to specify the refinement areas and their dynamic movement in advance of the simulation. However, since the information where the refinement areas are needed is rarely available before the simulation, interest in truly adaptive mesh refinement is growing (Baillie, Michalakes, and Skålin [7]).

As early as 1984 Berger and Oliger [14] published a fundamental paper on Adaptive Mesh Refinement (AMR) for hyperbolic partial differential equations. Numerical experiments in one and two dimensions with moved and arbitrarily rotated rectangular grids were presented. The applied model equations were simple (basically the scalar advection equation), and the refinements were restricted to only a small number of large blocks in selected model scenarios.

Nevertheless, based on this paper the first (as claimed by the authors) truly dynamically adaptive simulation of large–scale atmospheric flows was published in 1987 by Skamarock, Oliger, and Street [70] . One of the two reported model problems is a barotrophic cyclone where a rectangular refinement block is tracking the cyclone controlled by a Richardson–type estimation of the truncation error. However, the refinement is restricted to only one block and the applied nesting of the refinement block was not performing fully satisfactorily. Oscillations and discontinuities at the refinement boundaries restricted the model to fully explicit time schemes. Based on these early studies research on adaptive refinements continues until the present. Recent results are (1997):

Berger and LeVeque [15] published results of wave propagation; generally shaped local refinements are introduced into a two–dimensional, boundary–fitted, and logically rectangular grid. Although only results for the advection problem have been reported, an extension to the Euler Equations and for three dimensions is intended.

Fulton [29] presented an adaptive model for hurricane track prediction

that is based on a simple scalar vorticity conservation model. The adaptive refinement method is very restricted: Nested rectangular blocks of a given number and size are adjusted to the center of the vortex. Adaptive multigrid based on Bai and Brandt [6] (MLAT) is applied to solve the Poisson equation for the streamfunction in this paper.

With regard to refinements on distributed parallel computers Michalakes [58], who parallelized a regional weather model with local refinements based on the Penn State/NCAR MM5 with the RSL communication library (Michalakes [57]), has to be mentioned. For this three–dimensional forecast system reasonable parallel performance is reported for a simulation including two nested (however static) refinement blocks.

Sometimes the Shallow Water Equations are simulated with triangular meshes and finite element discretizations. In contrast to atmosphere models with structured refinement blocks a number of parallel and adaptive methods with finite element discretizations exist for marine simulations (e. g. Bastian [9] and Behrens [12]). The unstructured grids facilitate the discretization of the shores.

Finally, there are also very different approaches in adaptive meteorologic simulations, e. g. Göttelmann [32] developed an adaptive global shallow water model based on wavelets on the sphere.

In comparison to the approaches with block structured refinements that are mentioned above, the dynamically adaptive simulation that is presented in this paper is applied to the SWE[17] with general and fully automatically controlled refinement areas. Moreover, a semi–implicit time scheme is implemented and the model is parallelized for distributed memory parallel systems. With this combination the model goes far beyond the limits of the other known approaches.

[17] The advection equation is much simpler than the SWE, since no gravity waves occur.

Chapter 3

Conceptual Design

This chapter presents details of the concepts that are used and combined in the dynamically adaptive model.

In Section 3.1 the applied time and space discretization is introduced. A semi–implicit time scheme is developed to increase the sizes of stable time steps that can be efficiently integrated by solving a scalar Helmholtz–(like) equation. The space discretization is based on rectangular grids bounded by longitudes and latitudes as are common for regional weather models. Stability is theoretically investigated by a Von Neumann method applied to a simplified one–dimensional version of the SWE.

It is not necessary to solve the Helmholtz–(like) equation exactly. Section 3.2 introduces the general idea of using multigrid as stabilizer and adapting the cycling to the stability requirements that arise from the implicit treatment of the fast gravity waves.

The central part of the chapter is Section 3.3 that describes the design of the local refinements. A computationally efficient estimation of the local truncation error of the model equations is applied as an a–priory refinement criterion, in which it is taken into account that refinement takes place only in space. The detected refinement area is partitioned into blocks by recursive coordinate bisection in combination with the use of hypergrids. In this way the number of initializations of new refinement blocks is reduced. MLAT iterations are applied to the Helmholtz–(like) equation on the refined grid.

Parallelization concepts are given in Section 3.4 . The administration of the block structure in the explicit message passing parallel environment is presented. The load balancing algorithm computes a mapping of the refinement blocks to the available computational nodes in which coarse–fine neighborhood relations are considered as side conditions to reduce the required amount of communication. In order to achieve best parallel performance asynchronous communication is applied to reduce synchronization and to enable the concurrent execution of computation and communication. Exemplary parallel parts of the dynamically adaptive algorithm as the applied

parallel MLAT algorithm conclude this chapter.

3.1 Discretization

3.1.1 Semi–Implicit Time Discretization

With local refinements, high resolutions become available, and the size of stable time steps is extremely limited when using explicit time schemes due to the Courant–Friedrichs–Levy (CFL) criterion. Therefore a semi–implicit Eulerian[1] two–time–level discretization was developed. The fast gravity waves are calculated implicitly, whereas advection and Coriolis force are evaluated in an explicit way. In this way the time step size is no longer restricted by the fast gravity waves, but only by the generally much slower advection. The stability criterion becomes less severe and larger time steps available. By calculating advection and Coriolis terms explicitly the implicit system of equations can be reduced to a scalar Helmholtz–(like) equation and solved efficiently by a multigrid algorithm.

In contrast to the semi–implicit Leap–Frog method with three time levels in Kwizak and Robert [44], a two–time–level scheme is developed here.[2] Especially with local refinements, two–time–level schemes are advantageous. The refinement structures of only two time levels have to be administered, which reduces overhead and memory requirements. Moreover, the number of additional interpolations is reduced.

Figure 3.1 symbolizes the semi–implicit two–time–level scheme. Integration is performed from time level n to $n+1$. The advection and Coriolis terms are integrated by a two–step Lax–Wendroff scheme (e. g. Hirsch [39]) with use of an auxiliary time level $n+\frac{1}{2}$. For the gravity terms the implicit Eulerian scheme is applied.

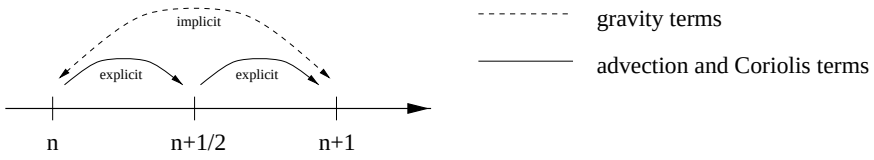

Figure 3.1: *Semi–implicit two–level time discretization*

The time discretization is presented in detail in the following. For simplicity the discretization is given for the SWE in Cartesian coordinates (2.7), the discretization of the equations in spherical coordinates is straightforward. The formulas are provided in the appendix, Equations (3.1) – (3.6) correspond to (A.4) – (A.9).

[1] meaning non–(semi)–Lagrangian

[2] Semi–implicit semi–Lagrangian schemes with two time levels have been already investigated in Bates [10] and in Temperton and Staniforth [79].

For the first half–step values of the auxiliary level $n+\frac{1}{2}$ are calculated.

$$\begin{aligned} u^{n+\frac{1}{2}} &= \bar{u}^n - \frac{\Delta t}{2}\left(u^n u_x^n + v^n u_y^n - f v^n + h_x^n + h_x^s\right) \\ v^{n+\frac{1}{2}} &= \bar{v}^n - \frac{\Delta t}{2}\left(u^n v_x^n + v^n v_y^n + f u^n + h_y^n + h_y^s\right) \\ h^{n+\frac{1}{2}} &= \bar{h}^n - \frac{\Delta t}{2}\left(u^n h_x^n + v^n h_y^n + h\left(u_x^n + v_y^n\right)\right) \quad , \end{aligned} \tag{3.1}$$

the bars on values denote averages of surrounding grid points as defined in Equation (3.8). These averages provide the stability of the Lax–Wendroff scheme.

The first half–step still includes the gravity terms. This provides better accuracy especially for the model problem of Section 5.1, where the pressure, which is caused by gravity, and the inertia moments are in equilibrium.

To reduce interpolation and roundoff errors the time scheme is formulated in the following for the increments from time level n to $n+1$

$$\begin{aligned} \delta u &= u^{n+1} - u^n \\ \delta v &= v^{n+1} - v^n \\ \delta h &= h^{n+1} - h^n \quad . \end{aligned} \tag{3.2}$$

For the second half–step of the two–step Lax–Wendroff scheme, at first the increments including only advection and Coriolis terms are calculated using the values of the auxiliary time level $n+\frac{1}{2}$

$$\begin{aligned} \delta u^{adv} &= -\Delta t\left(u^{n+\frac{1}{2}}\, u_x^{n+\frac{1}{2}} + v^{n+\frac{1}{2}}\, u_y^{n+\frac{1}{2}} - f\, v^{n+\frac{1}{2}}\right) \\ \delta v^{adv} &= -\Delta t\left(u^{n+\frac{1}{2}}\, v_x^{n+\frac{1}{2}} + v^{n+\frac{1}{2}}\, v_y^{n+\frac{1}{2}} + f\, u^{n+\frac{1}{2}}\right) \\ \delta h^{adv} &= -\Delta t\left(u^{n+\frac{1}{2}}\, h_x^{n+\frac{1}{2}} + v^{n+\frac{1}{2}}\, h_y^{n+\frac{1}{2}}\right) \quad . \end{aligned} \tag{3.3}$$

In this way the second half–step including the implicit gravity terms can be formulated

$$\begin{aligned} \delta u &= \delta u^{adv} - \Delta t\left(h_x^n + \mu\left(\delta h\right)_x + h_x^s\right) \\ \delta v &= \delta v^{adv} - \Delta t\left(h_y^n + \mu\left(\delta h\right)_y + h_y^s\right) \\ \delta h &= \delta h^{adv} - \Delta t\, h^n\left(u_x^n + \mu\left(\delta u\right)_x + v_y^n + \mu\left(\delta v\right)_y\right) \quad , \end{aligned} \tag{3.4}$$

where the homotopy parameter μ can be used to select different time schemes for the gravity terms. In detail:

$$\mu = \begin{cases} 0 & : \text{ explicit Eulerian scheme (unstable)} \\ 1/2 & : \text{ Cranck–Nicholson (2nd order)} \\ 1 & : \text{ implicit Eulerian scheme (1st order, highly damping)} \end{cases}$$

The explicit Eulerian scheme ($\mu = 0$) is known to be unstable. Numerical experiments show that also the Cranck–Nicholson scheme ($\mu = \frac{1}{2}$) is unstable in combination with the two–step Lax–Wendroff scheme for advection (at least when applied to the central space discretization of Section 3.1.2).

The implicit Eulerian scheme ($\mu = 1$) is conditionally stable in combination with the Lax–Wendroff scheme for Coriolis force and advection (the stability analysis of Section 3.1.3 confirms the results of the numerical experiments). It is of first order accuracy only and highly damping. Applied to the gravity terms the damping effect is a very good feature: The overall accuracy of the solution of the SWE in the meteorological context depends at most on advection and Coriolis force. The high frequency gravity waves are considered disturbing oscillations and with the use of the implicit Eulerian scheme they are effectively damped out (compare Hirsch [39] and Haltiner and Williams [34]). In the following we will restrict ourselves therefore to the case $\mu = 1$.

For an explicit reference time scheme the gravity terms are integrated with the two–step Lax–Wendroff scheme as the advection and Coriolis terms (providing second order accuracy over all).

$$\begin{aligned} \delta u^{exp} &= \delta u^{adv} - \Delta t \left(h_x^{n+\frac{1}{2}} + h_x^s \right) \\ \delta v^{exp} &= \delta v^{adv} - \Delta t \left(h_y^{n+\frac{1}{2}} + h_y^s \right) \\ \delta h^{exp} &= \delta h^{adv} - \Delta t\, h^n \left(u_x^{n+\frac{1}{2}} + v_y^{n+\frac{1}{2}} \right) \quad . \end{aligned} \tag{3.5}$$

System (3.4) is implicit (for $\mu = 1$), since derivatives of δu, δv, and δh appear in the right hand side. To solve this system efficiently, the first two equations are derivated with respect to x and y, respectively, and inserted into the third equation. In this way a Helmholtz–(like) equation for δh results

$$\begin{aligned} -\nabla^2 \delta h + \frac{\delta h}{\Delta t^2\, h^n\, \mu^2} &= \\ = \frac{\delta h^{adv}}{\Delta t^2\, h^n\, \mu^2} + \frac{\nabla^2 (h^n + h^s)}{\mu} - \frac{u_x^n + \mu\, (\delta u^{adv})_x + v_y^n + \mu\, (\delta v^{adv})_y}{\Delta t\, \mu^2} & \quad , \end{aligned} \tag{3.6}$$

where ∇^2 denotes the Laplacian. This scalar equation can be solved very efficiently with a multigrid algorithm even on a refined grid. Note that in this discretization the coefficient of δh, $1/\Delta t^2\, h^n\, \mu^2$, is variable in space.[3] Nevertheless, we will call this coefficient the Helmholtz constant and consequently the Helmholtz–(like) equation (3.6) a Helmholtz equation from now

[3] This time scheme corresponds to version no. 3 of the three–level schemes compared in Barros [8] (Coriolis force explicit, nonlinear mass–divergence implicit).

(1) *Calculate values of auxiliary time level* $n+\frac{1}{2}$ *with Eq. (3.1)*
(2) *Calculate advection and Coriolis force of the increments, Eq. (3.3)*
(3) *Calculate Helmholtz constant and right hand side of Eq. (3.6)*

---

(4) *Solve Helmholtz equation (3.6)*

---

(5) *Calculate increments* δu *and* δv *with Eq. (3.4)*
(6) *Complete time step with Eq. (3.2)*

Table 3.1: *Semi–implicit time discretization*

on for simplicity. When using multigrid it is not necessary to perform a linearization and to fix the Helmholtz constant.

Once the Helmholtz equation has been solved, the increments δu and δv can be evaluated. The computation of the total time scheme is displayed in Table 3.1 . The algorithm consists of 6 steps that are arranged in pre–solving, solving, and post–solving (separated by horizontal lines).

3.1.2 Space Discretization

The space discretization is based on a structured spherical grid. Equations (A.4) – (A.9) (respectively (3.1) – (3.6)) are discretized on a Cartesian grid $\mathbf{G}_h$ of the spherical coordinates λ and φ .

$$\mathbf{G}_h = \left\{ (\lambda_i, \varphi_j) \;\middle|\; \lambda_i = i\,\Delta\lambda,\ \varphi_j = j\,\Delta\varphi;\ 0 \le i \le N_\lambda,\ 0 \le j \le N_\varphi;\ i, j \in \mathbb{N} \right\} \quad ,$$

N_λ and N_φ denote the number of intervals in λ– and φ–direction, respectively. The subscript h of $\mathbf{G}_h$ can be read as multi–index $(\Delta\lambda, \Delta\varphi)$ defining the mesh sizes in both directions[4].

The values for u , v , and h of the time levels n and n+1 reside on the grid nodes (Arakawa–A grid, non–staggered grid). This simplifies restriction and prolongation of multigrid especially with local refinement.

At the boundary $\partial\mathbf{G}_h$ Dirichlet values are assumed (but consider the special nesting method in Section 3.3.3); at interior points $\overset{\circ}{\mathbf{G}}_h$ first and second derivatives are discretized symmetrically; e. g. the Laplacian ∇^2 in spherical coordinates (Equation (A.3)) is discretized using a five–point stencil.

[4]For local area models the spherical coordinates can be rotated in a way so that the poles of the rotated system are far away from the local area. In that case (almost) no anisotropies occur.

Applied to a point (i, j) it looks like this:

$$\nabla^2 = \frac{1}{\Delta\lambda^2\, a^2 \cos^2\varphi_j} \begin{bmatrix} & \beta_{j+\frac{1}{2}} & \\ 1 & -(2+\beta_{j+\frac{1}{2}}+\beta_{j-\frac{1}{2}}) & 1 \\ & \beta_{j-\frac{1}{2}} & \end{bmatrix} \quad , \tag{3.7}$$

$$\text{with} \quad \beta_{j\pm\frac{1}{2}} = \left(\frac{\Delta\lambda}{\Delta\varphi}\right)^2 \cos\varphi_j \, \cos\varphi_{j\pm\frac{1}{2}} \quad .$$

The values of the auxiliary time level $n+\frac{1}{2}$ reside exceptionally in between the grid–points as displayed in Figure 3.2 . In this way the time step restriction with regard to the explicit two–step Lax–Wendroff scheme is given by the CFL condition 1 , which is larger by a factor of $\sqrt{2}$ than it would be without this exception (compare Hirsch [39]). For the semi–implicit scheme an analysis for the sizes of stable time steps is given in Section 3.1.3 .

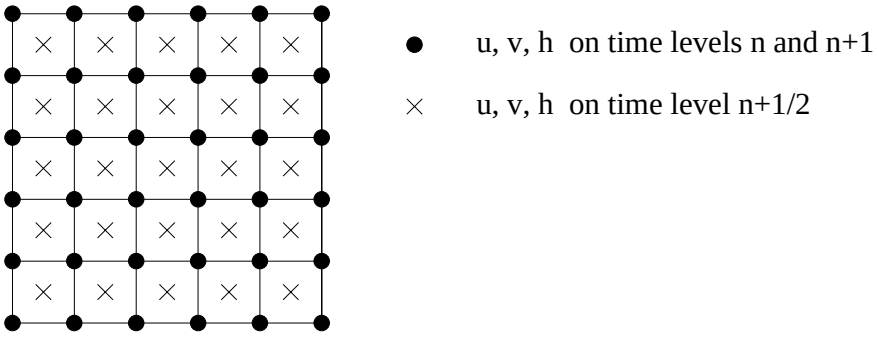

Figure 3.2: *Space discretization*

The averaged values $\bar{u}$, $\bar{v}$, and $\bar{h}$ that introduce additional dissipation to stabilize the Lax–Wendroff scheme are defined as (e. g. for $\bar{u}$)

$$\bar{u}^n_{i+\frac{1}{2},\,j+\frac{1}{2}} = \frac{u^n_{i+1,\,j+1} + u^n_{i+1,\,j-1} + u^n_{i-1,\,j+1} + u^n_{i-1,\,j-1}}{4} \quad . \tag{3.8}$$

The discretization is the same on all refinement levels. No special discretization at the boundaries of the refinement areas is necessary (see Section 3.3.3).

3.1.3 Stability Analysis

The combination of the two stable time schemes (Lax–Wendroff and implicit Eulerian or Cranck–Nicholson) does not guarantee stability. Moreover, the stability of the combined schemes also depends on the applied space discretization. A linear Von Neumann stability analysis is applied for a two–dimensional simplified version of the SWE, therefore, to derive the stability constraints.

For a Von Neumann stability analysis the semi–implicit time discretization is applied to the simplified linear system (2.6). The first half–step results in

$$\begin{aligned} u_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= \frac{1}{2}\left(u_{i+1}^{n}+u_{i}^{n}\right)-\frac{\sigma\, u_0}{2}\left(u_{i+1}^{n}-u_{i}^{n}\right)-\frac{\sigma}{2}\left(h_{i+1}^{n}-h_{i}^{n}\right) \\ h_{i+\frac{1}{2}}^{n+\frac{1}{2}} &= \frac{1}{2}\left(h_{i+1}^{n}+h_{i}^{n}\right)-\frac{\sigma\, u_0}{2}\left(h_{i+1}^{n}-h_{i}^{n}\right)-\frac{\sigma\, h_0}{2}\left(u_{i+1}^{n}-u_{i}^{n}\right) \quad , \end{aligned} \tag{3.9}$$

where $\sigma = \frac{\Delta t}{\Delta x}$. The second semi–implicit half–step:

$$\begin{aligned} u_i^{n+1} &= u_i^n - \sigma\, u_0\left(u_{i+\frac{1}{2}}^{n+\frac{1}{2}} - u_{i-\frac{1}{2}}^{n+\frac{1}{2}}\right) - \\ &\quad - \frac{\sigma\,\mu}{2}\left(h_{i+1}^{n+1} - h_{i-1}^{n+1}\right) - \frac{\sigma\,(1-\mu)}{2}\left(h_{i+1}^{n} - h_{i-1}^{n}\right) \\ h_i^{n+1} &= h_i^n - \sigma\, u_0\left(h_{i+\frac{1}{2}}^{n+\frac{1}{2}} - h_{i-\frac{1}{2}}^{n+\frac{1}{2}}\right) - \\ &\quad - \frac{\sigma\,\mu\, h_0}{2}\left(u_{i+1}^{n+1} - u_{i-1}^{n+1}\right) - \frac{\sigma\,(1-\mu)\, h_0}{2}\left(u_{i+1}^{n} - u_{i-1}^{n}\right) \quad . \end{aligned} \tag{3.10}$$

When inserting the fist half–step into the second one System (3.10) can be formulated

$$\mathbf{A}\begin{pmatrix} u_i^{n+1} \\ h_i^{n+1} \end{pmatrix} = \mathbf{B}\begin{pmatrix} u_i^{n} \\ h_i^{n} \end{pmatrix} \quad , \tag{3.11}$$

the coefficients of $\mathbf{A}$ and $\mathbf{B}$ are given in the appendix (Equation (B.1)). Introducing Fourier harmonics $e^{Ii\varphi}$ of phase angle $\varphi \in [-\pi, \pi]$

$$\begin{aligned} u_i^n &= U^n e^{Ii\varphi} \\ h_i^n &= H^n e^{Ii\varphi} \quad , \quad I^2 = -1 \quad , \end{aligned}$$

and defining the iteration matrix $\mathbf{G} = \mathbf{A^{-1}}\,\mathbf{B}$, the Equation (3.11) can be written

$$\begin{pmatrix} U^{n+1} \\ H^{n+1} \end{pmatrix} = \mathbf{G}\begin{pmatrix} U^{n} \\ H^{n} \end{pmatrix} \quad . \tag{3.12}$$

The matrix $\mathbf{G}$ depends on the phase angle φ and prescribes the growth of the Fourier harmonics in time. Scheme (3.11) is stable if the amplitudes U^n and H^n of the harmonics do not grow. This is guaranteed, if the spectral radius of $\mathbf{G}$ is smaller or equal to 1 for all $\varphi \in [-\pi, \pi]$.

The eigenvalues of $\mathbf{G}$ are numerically evaluated for different values of u_0, h_0, μ, and σ. The results of the stability analysis are:

1. For $\mu = 1$ (implicit Eulerian) the scheme is conditionally stable. Time steps with up to $\sigma = 0.7/u_0$ are within the stability region and show stable results, whereas instability occurs for little longer steps (independently on h_0). For this value of σ and for gravity wave speeds $\sqrt{3}$

and 10 times faster than advection ($h_0 = 10\,{u_0}^2$ and $h_0 = 100\,{u_0}^2$) the moduli of the eigenvalues of $\mathbf{G}$ are displayed in Figure 3.3 (left and right) in dependence on φ. The moduli are smaller than 1 throughout, which shows stability.

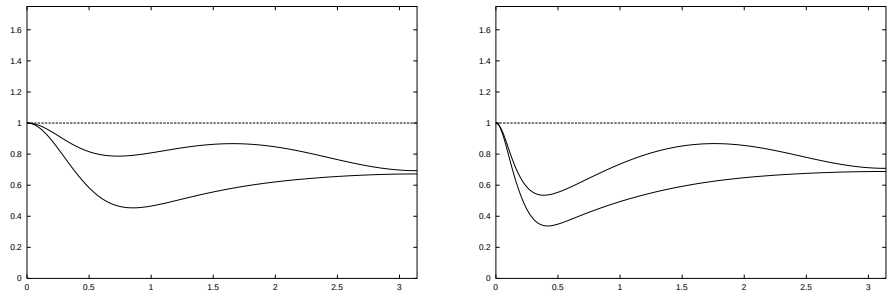

Figure 3.3: *Eigenvalues for implicit Eulerian scheme ($\mu = 1$)*

2. In case $\mu = \frac{1}{2}$ (Cranck–Nicholson) the moduli of the eigenvalues become larger than 1 even for much smaller time step sizes. For the same values of σ and h_0 as before the corresponding norms are displayed in Figure 3.4. The time scheme is unstable, it is therefore not possible to combine the Cranck–Nicholson scheme for the gravity terms with the Lax–Wendroff scheme for the advection (at least not for the applied space discretization)

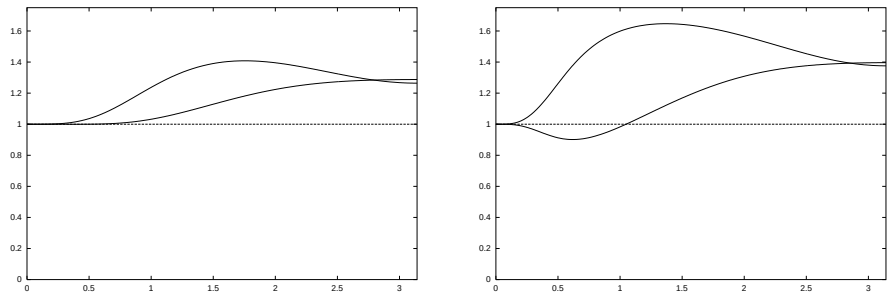

Figure 3.4: *Eigenvalues for Cranck–Nicholson scheme ($\mu = \frac{1}{2}$)*

These results confirm numerical experiments. For $\mu = \frac{1}{2}$ the integration becomes unstable after a few time steps, and for $\mu = 1$ stability is achieved for $\sigma < \sqrt{2}/u_0$ only.

Instability due to the nonlinearity of the SWE was not detected to be a problem when applying the implicit Eulerian scheme for the gravity waves. With its strong damping, additional horizontal diffusion is not necessary.

3.2 Multigrid as Stabilizer

Another way multigrid is used in an adaptive sense besides local refinement is developed in the following. The general idea is to apply multigrid as stabilizer rather than solver; the cycling is adapted to the stability requirements of the model problem that mainly depend on the speed of the gravity waves in the shallow water model.

In most cases the time steps of meteorological models are limited because of stability; larger steps could provide sufficient accuracy. This is the motivation of the implicit calculation of the gravity waves within the semi–implicit time scheme of Section 3.1.1 .

The limitation of stable time steps due to the CFL criterion has a mathematical background. Hirsch [39] writes:

> *The CFL stability condition ... expresses that the mesh ratio $\Delta t/\Delta x$ has to be chosen in such a way that the domain of dependence of the differential equation should be contained in the domain of dependence of the discretized equations.*

This means that in the example in Figure 2.1 on Page 34 the values of the grid points in the interior of the domain of dependence must contribute to the solution in P. Conversely, if only the values of neighbor points of P are used (within an explicit scheme), the domain of dependence must not exceed the area defined by these points. Implicit schemes usually depend on all points of the computational domain and thus unconditional stability can result. However, since in our case the time step sizes are still limited because of the explicit discretization of the advection, unconditional stability for the gravity waves is not necessary.

Moreover, the semi–implicit time scheme is transformed for computation into a scalar Helmholtz equation. If the results from an explicit time scheme are used as initial values[5], the accuracy of the explicit scheme is already provided. The Helmholtz equation has to be iterated only to provide stability.

This idea is different to the way parabolic equations are solved with multigrid, in which calculations on the coarse levels are used to save computation on the finest grid (Brandt [19]). This is possible since parabolic problems tend to become smooth in time and converge to a steady state. The complexity of local phenomena of the solutions of hyperbolic equations

[5] In other words, the explicit scheme is used as a predictor, whereas an iteration of the Helmholtz equation can be considered a correction. As initial values the explicit results for the second Lax–Wendroff step (Equation (3.4) for $\mu = 1$) can be used. However, to avoid the introduction of high frequency errors by the explicit evaluation of the gravity terms as initial values, it is preferable to use the increments excluding the gravity terms of Equation (3.3) instead (Brandt [18]). Numerical experiments showed little improvement of accuracy.

(as the SWE) however is constant in time in general (e. g. for the model problem in Section 5.1), and the fine grid can only be temporally and locally reduced (as with dynamic local refinements).

In Figure 3.5 the domains of dependence with respect to the advection and to the faster gravity waves are displayed. The time step size is adapted in such a way that the domain of dependence of the advection is included within a distance of one grid point. An explicit time scheme that evaluates the two neighbor points of P can therefore be applied in a stable way. For the faster gravity waves, however, information on distant grid points has to be included.

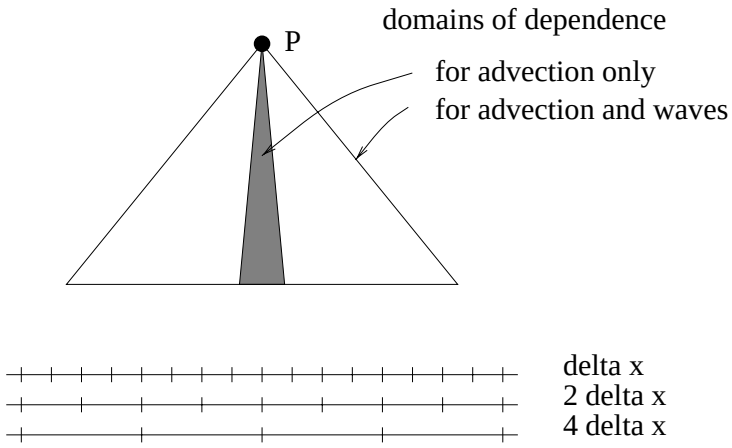

Figure 3.5: *CFL criterion on coarse grids*

This can be done by point relaxations (e. g. Gauss–Seidel or SOR). However, within one iteration of an usual 5–point stencil, information passes only one grid point. Therefore, seven iterations are necessary to transport any information from the outer points in Figure 3.5 to P. Moreover, it is not sure whether seven iterations suffice. The averaging of values within point relaxations reduces the influence of the outer points on P after this minimal number of relaxations.[6] Figure 3.5 presents the situation of the model problem of Section 5.1 in case of *fast* gravity waves, numerical experiments show that nine Gauss–Seidel iterations are necessary and sufficient.

The use of coarse grids in a multigrid algorithm accelerates this transport of information, on a grid with mesh size $2\Delta x$ it is twice as fast and even faster with coarser grids. Therefore, the coarse grids are still important to reduce the number of operations on the finest level, in which the number of coarse grids and the number of applied smoothing steps are optimized to

[6] Actually, this is the reason why single–grid relaxations show slower convergence with increased mesh size (h–dependent convergence rate). With a higher number of grid points the boundary values are transported to the center more slowly.

achieve minimal computational costs for the execution of one time step with the semi–implicit scheme.

Since the time step size is defined by the speed of advection, the multigrid cycling is adapted to the *ratio* of gravity waves and advection. The benefits of this method are presented in Section 5.2 , where a comparison with a common solution strategy takes place.

The idea to iterate only for stability is especially valuable with respect to parallelism, because a smaller number of coarse levels is necessary. This solves the *coarse grid problem* of parallel multigrid.

3.3 Adaptive Local Refinements

In the following the aspects of adaptive local refinement concerning this paper are mentioned. Here, a local refinement is defined as unification of rectangular and structured aligned blocks as is common for MLAT and already described in Section 2.3.3 .

In Figure 2.2 (left, Page 41) a computational domain is covered by a grid $\mathbf{G}_H$ with mesh size H . A critical region is indicated by a thick bent line.[7] To adapt the refinement areas dynamically during simulation a computational refinement criterion has to specify critical points where higher spatial resolution is appropriate. These points are indicated by dots in the figure. The shaded refinement area results, that may have very irregular shape in general. Good results are obtained by the refinement criterion presented in Section 3.3.1 that is based on an estimation of the local truncation error of the model equations.

After the critical points are determined, they are arranged in a number of rectangular blocks to create the block structured refinement. Usually the blocks contain a few grid points in addition to the critical points. This can be more efficient than introducing a higher number of blocks and additional overhead with it. Block structures with one and four blocks corresponding to the example of Figure 2.2 are indicated by thick lines in Figure 3.6 (left and right). This block–structured definition of refinements has two main advantages in comparison with refinements based on unstructured grids:

1. Rectangular structured blocks allow a high degree of computational optimization and performance. Moreover, since meteorological simulations are usually based on structured grids therefore, the incorporation of existing modules (e. g. for physical processes) into the adaptive method is facilitated. The same discretization (applied to different mesh sizes) can be used on fine and coarse grids.

2. The refinement blocks can be distributed to the available processors as grid partitions for parallelization. In this way the parallelization has

[7] In a meteorological application this could be a (part of a) weather front.

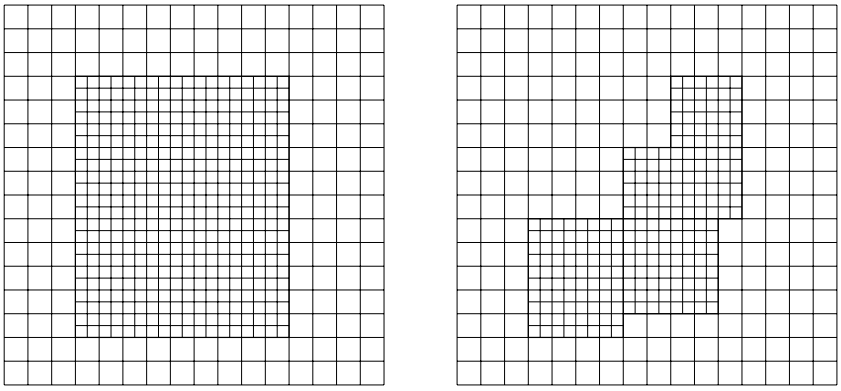

Figure 3.6: *Refinement block structure I*

much coarser granularity as it would have if each block was computed in parallel individually.

The rectangular refinement blocks and the global coarse grid have corresponding properties (actually, refinement blocks are implemented by the same data structure as the partitions of the global grid, see Section 4.1.1). Several levels of refinement can be introduced straightforward in a recursive way.

A suitable blocking algorithm that also takes continuity of the block structure and load balancing into account is presented in Section 3.3.2 .

To integrate the model equations the refinement structure is supported with boundary data from the global grid. However, special care is taken to avoid oscillations at the refinement boundaries in time dependent simulations (e. g. Skamarock, Oliger, and Street [70]). The embedding procedure called *nesting* is considered a major difficulty for simulations with block–structured dynamically adaptive refinements. In Section 3.3.3 theoretical analyzes are performed and two different approaches are discussed.

An additional interaction of coarse and fine grids results from the solution of the semi–implicit time scheme on the locally refined grid with multigrid. The application of MLAT for the given environment is presented in Section 3.3.4 .

Table 3.2 presents an overview of the time loop of the dynamically adaptive simulation. At the very beginning of the simulation a refinement structure is created, in later time steps the refinement is adapted. Then the explicit parts of the time discretization and the Helmholtz constant and the right hand side of the Helmholtz equation are calculated on all levels (Steps (1)–(3) of Table 3.1). If the semi–implicit time scheme is applied, a MLAT–cycle is performed on the locally refined grids (Step (4)). Thereafter the time step is completed (Steps (5)–(6)), and the two–way interactive nesting is performed.

Adaptive Time Loop

```
while time < end
  ( adaptation of refinement
    pre–solve all levels
    if implicit=true
      ( multigrid cycle
    post–solve all levels
    refinement nesting
```

Table 3.2: *Adaptive time loop*

3.3.1 Refinement Criterion

A refinement criterion based on the local discretization error τ_H for an elliptic boundary value problem is introduced in Section 2.3.2 .

Unfortunately, this criterion is not applicable to the Helmholtz equation of the implicit time scheme. Numerical experiments showed good results for static refinement (if the refinement areas were created only at the beginning of the simulation and not adapted further on), but when dynamic adaptation is realized, the refinement areas change very frequently (almost randomly) from time step to time step. This is understandable since the Helmholtz equation is solved for stability and because of the implicit treatment of the gravity waves only. Advection and accuracy is dominated by the explicit terms of the SWE. Therefore, it is appropriate to estimate the local discretization error to these explicit terms and to use this estimation for a refinement criterion.

Berger and Oliger [14] applied a refinement criterion based on the local truncation error on hyperbolic partial differential equations. An estimation for the local discretization error is determined by integrating the model equations with different resolutions and applying a Richardson extrapolation. One time step is performed with size Δt and mesh size Δx and two time steps with sizes $\frac{\Delta t}{2}$ and $\frac{\Delta x}{2}$. If time and space discretization are of the same order of accuracy, the extrapolation is straightforward. Limiting the maximal local discretization error in every time step is an appropriate criterion to achieve higher global accuracy (compare Stoer and Bulirsch [74] for ordinary differential equations). The refinement criterion of Berger and Oliger is tailored to the actual situation of this paper. Because of the semi–implicit time scheme, refinement in time does not take place (see Section 3.3.4), the time step size is constant for all grid levels and it depends on the smallest

mesh size. Therefore only the local discretization error in space is used for the refinement criterion. Moreover, it is not clear how to apply the double integration idea of Berger and Oliger to the semi–implicit time scheme in an efficient and stable way (within MLAT the coarse grids are used to calculate the coarse grid correction).

In this paper the local discretization error in space for a simplified explicit time scheme is determined in Section 3.3.1.1 for a refinement criterion. Since this value can be evaluated without loss of efficiency before the implicit integration of the SWE, an *a–priory* criterion can be formulated that predicts the appropriate refinement area for the following time step. If the refinement area is enlarged by one grid point on all sides, it is guaranteed that a critical weather phenomenon does not advect outside the refinement during the current time step (one grid point suffices, since the CFL criterion restricts the advection to less than one grid point per time step). The minimal size of a refinement area is therefore 3 times 3 points.

Since the decision whether to introduce or to keep a local refinement block at mesh size $\frac{h}{2}$ is based on coarser grids with mesh sizes h and $2h$, high oscillations on the finest grid (meteorological *noise*) can not affect the refinement criterion. It does not seem to be a problem that the criterion is less accurate because it is evaluated on the coarser grids.

In Section 3.3.1.1 the discretization error is estimated; in Section 3.3.1.2 the refinement criterion is applied to the system of the SWE, and threshold values are chosen in a way to control the global accuracy of the model equations.

3.3.1.1 Local Spatial Discretization Error

For a computationally efficient a–priory refinement criterion the local discretization error with respect to space of the explicit Eulerian scheme is used. It is shown that this value is a good estimation for the corresponding local spatial discretization error of the two–step Lax–Wendroff scheme.

For the semi–implicit time discretization the local discretization error is more expensive to approximate. Therefore the refinement criterion based on the local discretization error for the explicit Eulerian scheme is also used for the semi–implicit scheme. Numerical experiments show very good results.

For simplicity the following derivation is given for the one–dimensional linearized equations (2.6). A central space discretization of the explicit Eulerian scheme yields

$$\begin{aligned} \frac{h_i^{n+1} - h_i^n}{\Delta t} &= -u_0 \frac{h_{i+1}^n - h_{i-1}^n}{2\Delta x} - h_0 \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} \\ \frac{u_i^{n+1} - u_i^n}{\Delta t} &= -u_0 \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} - \frac{h_{i+1}^n - h_{i-1}^n}{2\Delta x} \quad . \end{aligned} \tag{3.13}$$

Assuming h_i^n and u_i^n to be the exact analytical solutions of Equation (2.6) at time step n and grid point i, and assuming sufficient smoothness, the development in Taylor series gives

$$\begin{aligned} h_i^{n+1} &= h_i^n + \Delta t (u_t)_i^n + \frac{\Delta t^2}{2}(u_{tt})_i^n + \mathcal{O}(\Delta t^3) \\ h_{i\pm1}^n &= h_i^n \pm \Delta x (u_x)_i^n + \frac{\Delta x^2}{2}(u_{xx})_i^n \pm \frac{\Delta x^3}{6}(u_{xxx})_i^n + \mathcal{O}(\Delta x^4) \quad . \end{aligned} \tag{3.14}$$

Substitution of these series in Equation (3.13) yields

$$\begin{aligned} h_t - u_0 h_x - h_0 u_x &= -\frac{\Delta t}{2} h_{tt} + u_0 \frac{\Delta x^2}{6} h_{xxx} + h_0 \frac{\Delta x^2}{6} u_{xxx} + \mathcal{O}(\Delta t^2, \Delta x^3) \\ u_t - u_0 u_x - h_x &= -\frac{\Delta t}{2} u_{tt} + u_0 \frac{\Delta x^2}{6} u_{xxx} + \frac{\Delta x^2}{6} h_{xxx} + \mathcal{O}(\Delta t^2, \Delta x^3) \quad , \end{aligned} \tag{3.15}$$

where the right hand sides are the local discretization errors of the scheme.

The explicit Eulerian scheme with central space discretization is of consistency order 1 in time and 2 in space. The terms $-\frac{\Delta t}{2} h_{tt}$ and $-\frac{\Delta t}{2} u_{tt}$ prescribe negative viscosities. Since they amplify disturbances, the explicit Eulerian scheme is unstable, as is well known. Nevertheless, this scheme can be used for the refinement criterion, since stability is not important here.

Since the resolution is adapted only in space and not in time, a suitable refinement criterion has to detect the local discretization error depending on the spatial resolution only. It can be seen that the local discretization errors in space depend on third derivatives of the solution; however, these derivatives are not evaluated directly. Using Richardson extrapolation, the local spatial discretization errors are efficiently estimated by computing the right hand sides of the explicit Eulerian scheme (3.13) on two corresponding grids with mesh sizes h and $2h$ (actually, it is sufficient to calculate the differences of the corresponding values, which is already the desired result up to a constant factor).

For the two–step Lax–Wendroff scheme, which is actually applied for advection, additional terms appear in the local discretization error:

$$\begin{aligned} h_t - u_0\, h_x - h_0\, u_x \;=\; & -\frac{\Delta t}{2} h_{tt} + u_0\,\frac{\Delta x^2}{6} h_{xxx} + h_0\,\frac{\Delta x^2}{6} u_{xxx} + \\ & + u_0^2\,\frac{\Delta t}{2} h_{xx} + u_0\, h_0\,\frac{\Delta t}{2} u_{xx} + \\ & + u_0^2\,\frac{\Delta t\,\Delta x^2}{24} h_{xxxx} + u_0\, h_0\,\frac{\Delta t\,\Delta x^2}{24} u_{xxxx} + \mathcal{O}(\Delta t^2, \Delta x^3) \\ u_t - u_0\, u_x - h_x \;=\; & -\frac{\Delta t}{2} u_{tt} + u_0\,\frac{\Delta x^2}{6} u_{xxx} + \frac{\Delta x^2}{6} h_{xxx} + \\ & + u_0^2\,\frac{\Delta t}{2} u_{xx} + u_0\,\frac{\Delta t}{2} h_{xx} + \\ & + u_0^2\,\frac{\Delta t\,\Delta x^2}{24} u_{xxxx} + u_0\,\frac{\Delta t\,\Delta x^2}{24} h_{xxxx} + \mathcal{O}(\Delta t^2, \Delta x^3) \end{aligned} \tag{3.16}$$

In contrast to before, here the terms $\frac{\Delta t}{2} h_{xx}$ and $\frac{\Delta t}{2} u_{xx}$ represent positive viscosities that stabilize the scheme. These terms, and the others that depend on the temporal discretization only, need not be used for the refinement criterion, as Δt is not adapted. The coefficients of the fourth derivatives in space depend on Δt and Δx^2. If the CFL criterion $\Delta t \leq \Delta x / u_0$ is taken into account, these coefficients can be seen as of higher order $\mathcal{O}(\Delta x^3)$ and can be ignored, too. The remaining terms are identical to those used for the explicit Eulerian scheme that can therefore be used to estimate the local discretization error in space.

For the semi–implicit time scheme with the implicit evaluation of the gravity terms the situation is more complicated. Nevertheless, the refinement criterion based on the explicit time discretization works very well for the semi–implicit scheme as well, as numerical experiments show.

This refinement criterion is efficiently evaluated, because a cheap time scheme is used to estimate the local discretization error of the actually applied discretization.

3.3.1.2 Threshold Values

The refinement criterion $|\tau_H| < c_{crit}$ (Equation (2.19)) is generalized to the systems of the SWE by applying it to the local discretization error of each component separately and by combining the resulting values. Consequently, a threshold value for each component of the system has to be defined.

In Figures 3.7, 3.8, and 3.9 the estimated local discretization errors τ_h^h, τ_h^u, and τ_h^v of h, u, and v, respectively, are displayed. They correspond to the model problem in Figure 5.1 (top), and are evaluated on a global grid with the finest resolution that is used for the adaptive simulation. Note that τ_h^h is small in the center of the cyclone. Without the consideration of τ_h^u

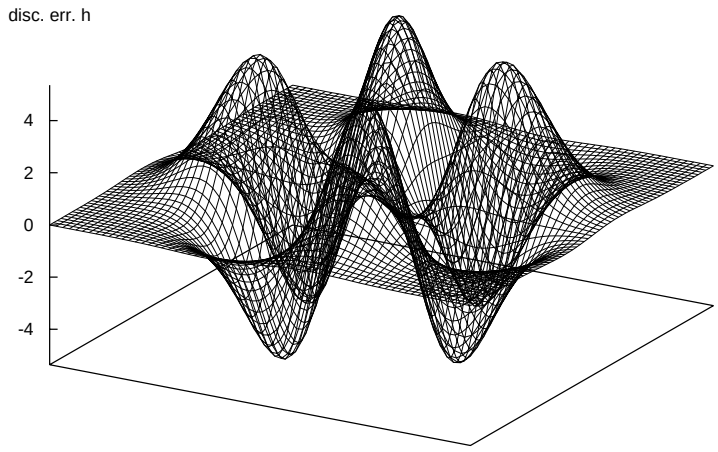

Figure 3.7: *Local discretization error of h*

and τ_h^v for the criterion the resulting refinement area would be a hollow ring around the vortex.

In our case the velocity components u and v of the SWE are of the same physical quality, and a common threshold value can be used. The applied refinement criterion reads

$$r_{crit} := \frac{|\tau_H^h|}{c_{crit}^h} + \frac{|\tau_H^u|}{c_{crit}^{u,v}} + \frac{|\tau_H^v|}{c_{crit}^{u,v}} < 1 \quad , \tag{3.17}$$

where τ_H^h , τ_H^u , and τ_H^v denote the local discretization errors of the prognostic variables on a coarse grid, and c_{crit}^h and $c_{crit}^{u,v}$ are the threshold values for h and for u and v, respectively. Recall that the decision whether to refine or not is based on computation on the already existing coarse grid with mesh size H.

A general idea to attain reasonable threshold values is presented in the following: The accuracy of non–adaptive models is determined by the resolution of the global uniform grid; in this paper the major objective of local refinement is to achieve the order of this accuracy at computationally lower costs. The results of an adaptive model must be comparable to the results of a model that uses the maximal resolution globally.

In order to achieve this, the local discretization error of the adaptive simulation is limited in every time step by the maximal local discretization error of the uniform model with the highest resolution (as displayed in

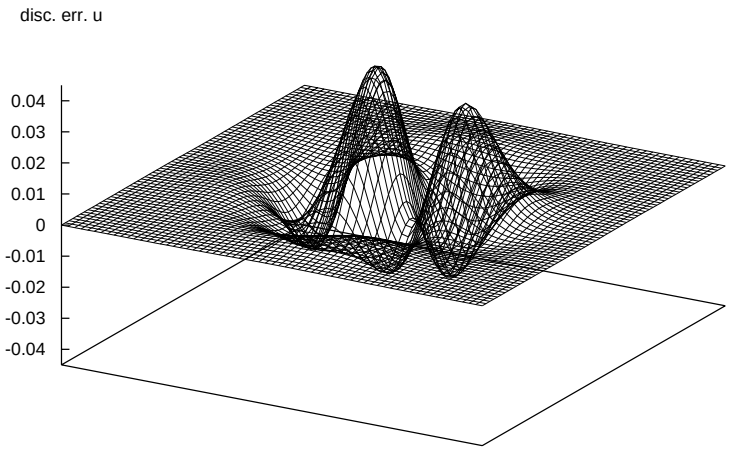

Figure 3.8: *Local discretization error of u*

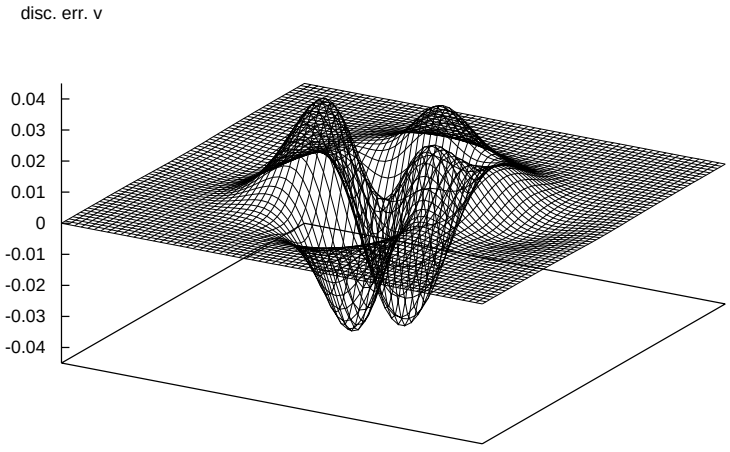

Figure 3.9: *Local discretization error of v*

Figures 3.7, 3.8, and 3.9). If the local accuracy necessary at a grid point is already achieved with lower resolution, a refinement of this point is not necessary. If the accuracy is worse, the point is selected until the highest resolution is applied.

To determine the maximal absolute values of the local discretization errors, it is sufficient to execute one time step of the model with the desired resolution. The resulting value for r_{crit} is displayed in Figure 3.10.

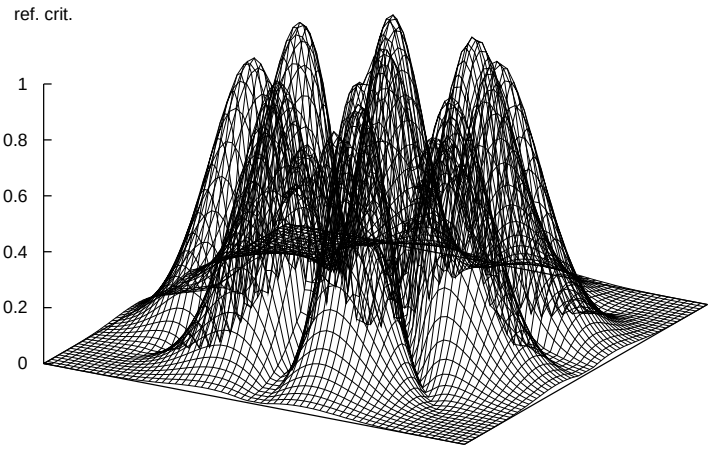

Figure 3.10: *Refinement value* r_{crit}

Since the refinement criterion is second order in space, it is approximately four times larger when evaluated on the next coarser grid. This means that values larger than 0.25 in Figure 3.10 define points where the highest resolution is necessary. Values smaller than 0.25 indicate points where the refinement value of the coarse grid is smaller than 1, and where no refinement according to Criterion (3.17) takes place. The isolines $\frac{1}{4}$, $\frac{1}{16}$, $\frac{1}{64}$, and $\frac{1}{256}$ of r_{crit} are displayed in Figure 3.11. They estimate the refinement boundaries of the model problem in Figure 5.1 (top).

Another problem occurs for simulations with dynamically adaptive refinements: The boundaries of the detected refinement areas tend to oscillate in time, the local refinement increases and decreases by one boundary point alternatively in each adaptation step. A similar phenomenon also appears when solving ordinary differential equations with adaptive step size; it can be controlled by limiting the size of the next step to 1.5 times the previous step. According to this idea, the decrease in the refinement area is controlled

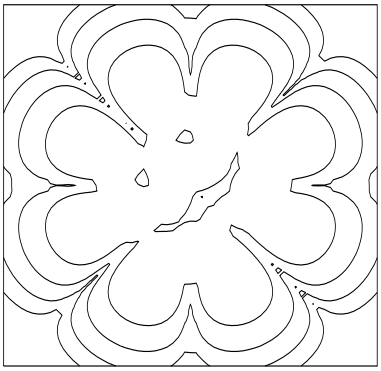

Figure 3.11: *Isolines of refinement value* r_{crit}

$1 < r_{crit}$	:	*refinement necessary*
$\alpha < r_{crit} \leq 1$	:	*do not alter refinement*
$r_{crit} \leq \alpha$	:	*refinement not necessary*

Table 3.3: *Refinement criterion*

by a reduced threshold α as given in Table 3.3. In case a grid point is already included in the refinement, it is marked to be refined already if r_{crit} exceeds α but not 1 ($\alpha = 0.1$ is found to be a good value). However, also points where a refinement is definitely not necessary ($r_{crit} \leq \alpha$), can be included in a refinement block because of the blocking algorithm (see Section 3.3.2).

3.3.2 Dynamic Blocking

Once the critical points have been determined by the refinement criterion, they are arranged in rectangular blocks. To prevent the appearance of a large number of very small blocks, a small number of grid points that have not been selected by the refinement criterion can be included additionally. Small blocks increase computational and communication overhead, since additional work is introduced at the block boundaries in general, and because data exchanges have to take part in the update of the overlap areas in parallel systems (see Section 2.4.3).

An appropriate partitioning algorithm, where the acceptable number of unnecessarily refined blocks and the resulting block sizes is tunable by a threshold value, is given in Section 3.3.2.1 .

However, two more things have to be considered when blocking the refinement area.

- The dynamic adaptation of the refinement area introduces initialization (copies), interpolations, and volume communication whenever the

blocks of the refinement structure change; therefore, local adaptations of the refinement areas must not change the whole refinement structure.

- For parallelization a good load balance and an equal distribution of the refinement grid points is important. In general a perfect balance can not be achieved with block structured refinements; however, a creating blocks with a limited maximal size facilitates the determination of a sufficiently good balance.

Because of these two constraints, the blocking algorithm is applied on pre–partitioned blocks that are defined by *hypergrids* (see Section 3.3.2.2).

3.3.2.1 Recursive Coordinate Bisection

For the blocking of the critical points an algorithm based on recursive coordinate bisection is applied that is given in Table 3.4. Steps (1)–(4) more

(1) *Determine the minimal covering block*
(2) *Calculate blocking quality*
(3) *If this quality is acceptable then finish*
(4) *Else subdivide the block and restart with (1) for both parts*

Table 3.4: *Recursive coordinate bisection*

precisely:

(1) The minimal block containing all selected points is determined by the minimal and maximal x– and y–coordinates of the critical points. As an example the resulting block for the scenario of Figure 2.2 (left) is displayed in Figure 3.6 (left).

(2) The ratio of the number of critical points to the total number of points included in the blocks is computed. This ratio is a measure of the quality of the block structure.

(3) If this ratio exceeds a tunable threshold value, the number of unnecessarily included points is considered as acceptably small, and the actual block is accepted for the current refinement structure[8].

(4) If there are still too many unnecessarily included points, the block is subdivided into two smaller blocks at the center of its longer side. The algorithm is applied in a recursive way on both new blocks.

[8]Because of the limited minimal size of the refinement blocks, it is assured that the recursion terminates for all threshold values less than or equal to 1.

In our example a refinement structure consisting of four blocks as displayed in Figure 3.6 (right) may result. The threshold value is determined in a way that achieves a good compromise between the number of unnecessary points and the number of blocks created.

3.3.2.2 Blocking by Hypergrids

To reduce the number of new initializations of new blocks and to limit the maximal block size, the refinement area is partitioned by a pre–defined *hypergrid* prior to the application of the recursive coordinate bisection algorithm to the resulting partitions.

If the coordinate bisection algorithm is applied straightforwardly, it often happens that a small adaptation of the refinement area results in a very different block structure. This may cause many initializations and high computational and communication overhead during a dynamically adaptive simulation.

Two examples of hypergrids that define four and sixteen default partitions are indicated by thick lines in Figure 3.12 (left and right). The application of Algorithm 3.4 results in the displayed block structures. In this

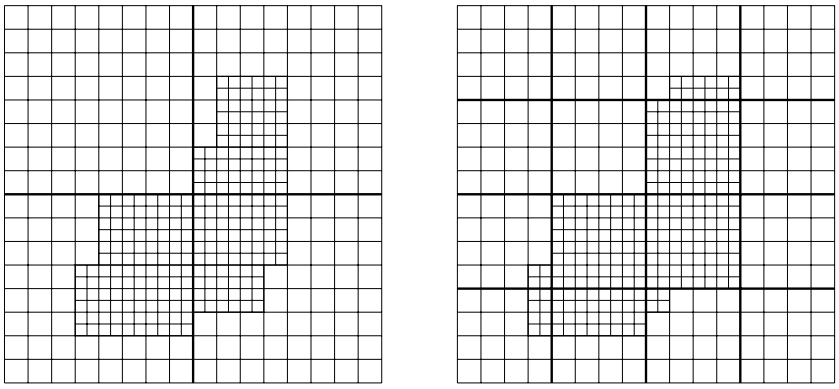

Figure 3.12: *Blocking by hypergrids*

way, local adaptations of the refinement area result in only local changes of the block structure. Most of the blocks can generally be reused in the next time step.

Moreover, the resulting blocks have a maximal size that limits load imbalances when the blocks are distributed to the processors for parallelization. The hypergrid has to be optimized for the parallel system and the application. Many and smaller blocks facilitate good load balancing; however, additional work for computation and communication is introduced. As for coarse grid agglomeration (introduced in Section 2.4.3) it might be faster to calculate a small block on only one processor. Such tuned sub–optimal load balancing can be considered *fine grid agglomeration*.

Another advantage of the hypergrid technique is that the computation of the blocking can be performed in parallel.

3.3.3 Nesting the Refinement Areas

The refinement blocks have to be supported by the embedding coarse grids with values at the boundary of the refinement structure. This embedding is performed every time step during simulation. Conversely, the coarse grids are updated by the more accurate data of the refinement. This two–way interaction is carried out in dynamically adaptive models to prevent the solutions of the corresponding fine and coarse grids form diverging from each other in time. Without the update of the coarse grids it would not be possible to reduce the refinement areas again.

The solutions of coarse and fine grids are calculated simultaneously. In this context the refinement procedure can be regarded as a *coupling problem*, where a partial differential equation[9] is solved separately on different grids, and after every time step the grids are coupled together. If there are several levels of refinement, several stages of nesting become necessary, too.

In the following two different nesting approaches are presented in detail. Theoretical studies in Section 3.3.3.1 show that a straightforward second order discretization of the SWE as commonly applied within MLAT (for refined boundary value problems, see System 2.20) is not stable. To prevent the introduction of oscillations, special discretizations at the refinement boundaries (e. g. based on a finite volume discretization with control volumes as indicated in Figure 3.13) are necessary. However, numerical experiments have not shown satisfactory results especially when adapting and moving the refinements dynamically.

The second approach is based on a blending of coarse and fine grid values that are multiply computed on both grids in an overlap area (see Section 3.3.3.2) for details. No special discretization is necessary at the boundaries of the refinement, and even different model equations can be solved on the grids and coupled together in smooth transition.

3.3.3.1 Discretizing the Boundary

The idea of this approach is to discretize the boundaries of the refinement in a special way, preferably using finite volume techniques with control volumes as indicated in Figure 3.13 with hashed lines[10]. This is necessary, since straightforward second order discretizations are not stable, which is shown by a stability analysis with the matrix method in the following.

[9] Not necessarily the same partial differential equation, since in general it is possible to use different models for different resolutions in the simulation.

[10] This idea leads to *composite grid stars* as in McCormick [54] .

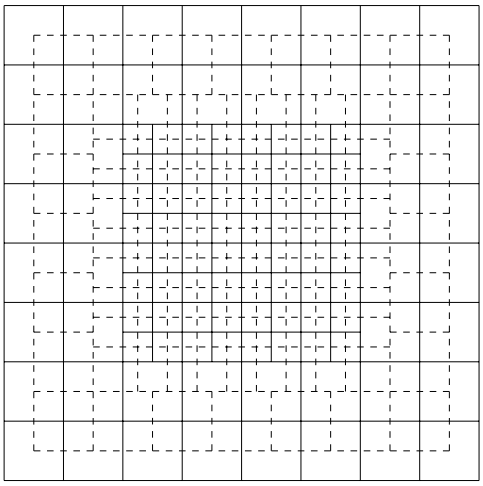

Figure 3.13: *Control volumes of finite volume discretization*

For simplicity the one–dimensional advection equation is used as a model problem for the stability analysis (advection is included in the SWE)

$$\frac{\partial u}{\partial t} + a\,\frac{\partial u}{\partial x} = 0 \quad , \tag{3.18}$$

with Dirichlet boundary conditions

$$\begin{aligned} u(0,x) &= f(x),\ 0 \le x \le 1 \\ u(t,0) &= g(t), \quad t > 0 \quad . \end{aligned}$$

After a discretization in time a system of ordinary differential equations results

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = \mathbf{S}\mathbf{u} + \mathbf{q} \quad , \tag{3.19}$$

where the vector $\mathbf{u}$ contains the unknowns $u_i = u(\frac{i}{n})$, $i = 1, \ldots, n-1$ on a grid with n intervals, and the vector $\mathbf{q}$ contains boundary values. The structure matrix $\mathbf{S}$ defines the discretization of Equation (3.18) in space.

The following statement is true (Hirsch [39]):

> *The analytical solution of system* (3.19) *remains bounded, if the real parts of the eigenvalues of* $\mathbf{S}$ *are negative or zero. If* 0 *is an eigenvalue, it must be simple.*

If the analytical solution of System (3.19) is not bounded, it is not possible to discretize it in time in a consistent and stable way. Therefore the eigenvalues of $\mathbf{S}$ can be used to investigate space discretizations with respect to stability. A space discretization satisfying Hirsch's condition is called stable.

At first a non–refined central discretization of Equation (3.19) is considered

$$\frac{\mathrm{d}u_i}{\mathrm{d}t} = -\frac{a}{2\Delta x}(u_{i+1} - u_{i-1}), \quad i = 1, \ldots, n-1 \quad , \tag{3.20}$$

with boundary conditions

$$\begin{array}{l} u_0(t) = g(t) \\ u_{n+1}(t) = h(t) \end{array} \quad , \quad t > 0 \quad .$$

In matrix notation Equation 3.20 can be written

$$\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t} = -\frac{a}{2\Delta x} \begin{pmatrix} 0 & 1 & & & \\ -1 & 0 & 1 & & \\ & -1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & -1 & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ \\ \vdots \\ \\ u_n \end{pmatrix} + \begin{pmatrix} \frac{a}{2\Delta x}\, g(t) \\ 0 \\ \vdots \\ 0 \\ -\frac{a}{2\Delta x}\, h(t) \end{pmatrix} \quad . \tag{3.21}$$

The eigenvalues of the structure matrix $\mathbf{S}_{(3.21)}$ of Equation (3.21) are purely imaginary and in pairs different (the eigenvalues are $2\,I\cos\left(\frac{j\pi}{n+1}\right)$, $j = 1, \ldots, n$, Lomax [49]), therefore Discretization (3.20) is stable.

Based on this result for uniform meshes discretizations with local refinements are examined in the following. In Figure 3.14 a one–dimensional refined grid is displayed. The unknowns u_i, $i = 1 \ldots, 11$ are numbered with i and x denotes their x–coordinates. The discretizations of the refinement

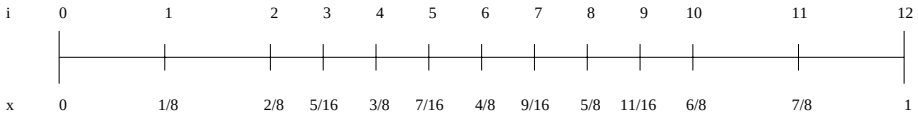

Figure 3.14: *One–dimensional refined grid*

boundary points u_2 and u_{10} are especially critical.

A straightforward central finite difference discretization as commonly applied for adaptive boundary value problems that can be solved by MLAT (compare System 2.20 and Table 2.3) is

$$
\begin{aligned}
\frac{\mathrm{d}u_1}{\mathrm{d}t} &= -\frac{a}{2\Delta x}\,(u_2 - u_1) \\
\frac{\mathrm{d}u_2}{\mathrm{d}t} &= -\frac{a}{2\Delta x}\,(u_4 - u_1) \\
\frac{\mathrm{d}u_i}{\mathrm{d}t} &= -\frac{a}{\Delta x}\,(u_{i+1} - u_{i-1}) \quad , \quad i = 3, \ldots, 9 \\
\frac{\mathrm{d}u_{10}}{\mathrm{d}t} &= -\frac{a}{2\Delta x}\,(u_{11} - u_8) \\
\frac{\mathrm{d}u_{11}}{\mathrm{d}t} &= -\frac{a}{2\Delta x}\,(u_{12} - u_{10}) \quad .
\end{aligned}
\tag{3.22}
$$

Note that the refinement boundaries u_2 and u_{10} are discretized in the same way as the coarse grid points u_1 and u_{11}, and the points u_3 and u_9 are not used. The structure matrix $\mathbf{S}_{(3.22)}$ is

$$
\mathbf{S}_{(3.22)} = -\frac{a}{2\Delta x}
\begin{pmatrix}
0 & 1 & & & & & & & & & \\
-1 & 0 & 0 & 1 & & & & & & & \\
 & -2 & 0 & 2 & & & & & & & \\
 & & -2 & 0 & 2 & & & & & & \\
 & & & -2 & 0 & 2 & & & & & \\
 & & & & -2 & 0 & 2 & & & & \\
 & & & & & -2 & 0 & 2 & & & \\
 & & & & & & -2 & 0 & 2 & & \\
 & & & & & & & -2 & 0 & 2 & \\
 & & & & & & & -1 & 0 & 0 & 1 \\
 & & & & & & & & & -1 & 0
\end{pmatrix} ,
$$

the corresponding eigenvalues are numerically calculated and displayed in Figure 3.15. Two eigenvalues with positive real parts exist; Discretiza-

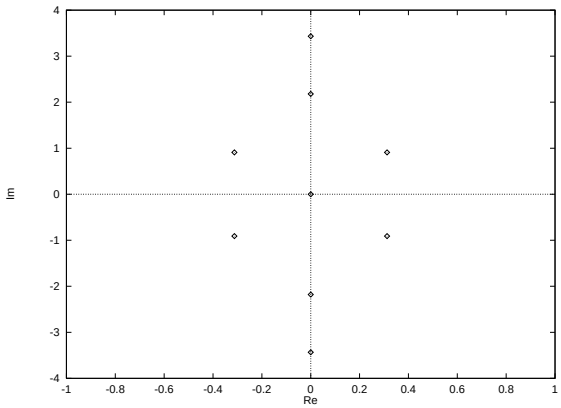

Figure 3.15: *Eigenvalues of second order finite difference discretization*

tion (3.22) is unstable. The same is true for larger examples with different sizes of grids and refinement areas.

This shows that a special discretization of the refinement boundaries is necessary. An example involving the values u_3 and u_9 for the boundary discretization is

$$\begin{aligned} \frac{\mathrm{d}u_2}{\mathrm{d}t} &= -\frac{a}{\Delta x}\left(u_3 - \frac{u_1+u_2}{2}\right) \\ \frac{\mathrm{d}u_{10}}{\mathrm{d}t} &= -\frac{a}{\Delta x}\left(\frac{u_{11}+u_{10}}{2} - u_9\right) \end{aligned} \quad , \tag{3.23}$$

with structure matrix $\mathbf{S}_{(3.23)}$,

$$\mathbf{S}_{(3.23)} = -\frac{a}{2\Delta x}\begin{pmatrix} 0 & 1 & & & & & & & & & \\ -1 & -1 & 2 & & & & & & & & \\ & -2 & 0 & 2 & & & & & & & \\ & & -2 & 0 & 2 & & & & & & \\ & & & -2 & 0 & 2 & & & & & \\ & & & & -2 & 0 & 2 & & & & \\ & & & & & -2 & 0 & 2 & & & \\ & & & & & & -2 & 0 & 2 & & \\ & & & & & & & -2 & 0 & 2 & \\ & & & & & & & & -2 & -1 & -1 \\ & & & & & & & & & -1 & 0 \end{pmatrix} .$$

The eigenvalues are presented in Figure 3.16 . Only one single eigenvalue is

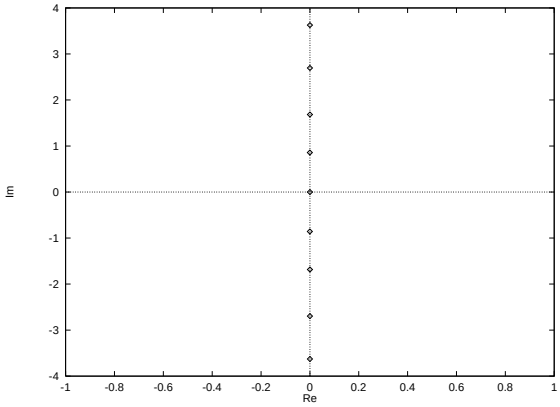

Figure 3.16: *Eigenvalues of composite grid star discretization*

0, all others are purely imaginary. With a suitable time scheme, stability can be achieved with this space discretization.

This result is not only true for the special grid of Figure 3.14 . In many other cases with larger global grids and larger refinement areas no instable

cases could be found. (No proof — the analytical calculation of the eigenvalues — was found). Nevertheless, Discretization (3.23) is a very good candidate for a stable discretization.

Based on these results, the SWE (2.7) were discretized on a refined grid in a straightforward generalization in two space dimensions. Although Discretization (3.23) is consistent of first order only, almost global second order accuracy was achieved in numerical experiments with static refinements.

With adaptive dynamic refinements, however, the situation becomes more complicated. When the grid is adapted, the refinement areas change. Points of the refinement boundary become points in the interior of the new refinement area or points of the coarse grid outside the refinement. In both cases grid point values resulting from a first order discretization were used to initialize points in a homogeneous second–order discretization. Spurious oscillations appeared in numerical experiments when the refinements were dynamically adapted, and the accuracy of the global solution was affected.

It is not clear whether this is a first order – second order problem only. Also with more elaborate refinement boundary discretizations (e. g. with conservative discretizations of higher order as in Hundsdorfer et al. [41]) the boundary values are used to initialize interior refinement points with a different homogeneous discretization.

But even if the oscillations can be reduced, a severe disadvantage occurs with special refinement boundary discretizations: The implementation costs increase substantially. Each side (left, right, upper and lower) of the block structured refinement has to be discretized separately, and the same is true for the four corners. Moreover, with higher order discretizations in space, not only the boundary itself has to be discretized in a special way, but possibly also the points adjacent to the boundaries of the refinement.

Also the multigrid transfer operations (i. e. restriction and prolongation) to compute the semi–implicit time scheme might require an elaborate adaptation at the refinement boundaries.

In a number of numerical experiments with different boundary discretizations the implementation costs were experienced as a severe drawback of this nesting approach. With regard to more complicated model equations than the SWE it was decided to apply the blending interpolation approach for nesting as described in the next section.

3.3.3.2 Blending Interpolation

The blending approach of refinement nesting is published by Zhang et al. [85] and has been implemented in the Canadian model MM5 (Grell, Dudhia, and Stauffer [31]) . This approach is adapted to the situation in this paper[11] and described in the following.

[11] In the model MM5 the mesh refinement ratio is 1 : 3 and staggered grids are used.

The general idea is to introduce an area where fine and coarse grids overlap and values for both grids are computed. The resulting coarse and fine grid values are *blended* according to

$$\varphi_b = \lambda \varphi_c + (1 - \lambda) \varphi_f \quad , \quad \lambda \in [0, 1] \quad , \tag{3.24}$$

where φ is any prognostic variable, and the subscripts b, c, and f denote blended, coarse grid, and fine grid values, respectively. The value λ defines the mixing ratio; λ decreases smoothly from 1 to 0 from the coarse grid boundary of the blending area to its fine grid boundary.

In Figure 3.17 a blending area with a width of four fine grid points is shaded. In this area the coarse grid values are interpolated (with bi–linear

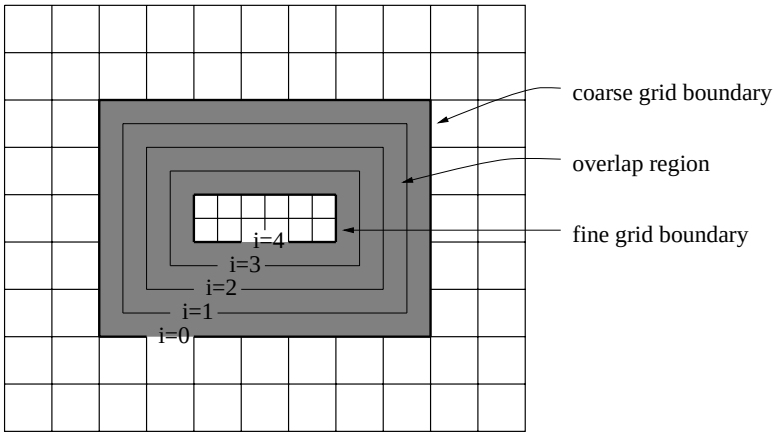

Figure 3.17: *Blending area with width 4*

or bi–cubic interpolations) to the mesh size of the fine grid and the blending according to Equation (3.24) is performed. For the two–way interaction the solution of the fine grid including the new blended fine grid values is restricted to the coarse grid.

The mixing ratio λ depends on the distance from the outer boundary of the nesting and is indicated by the value i in Figure 3.17. Formula (3.25) for λ was found to work very well.

$$\lambda = \cos^2 \left(\frac{i}{n} \frac{\pi}{2} \right) \quad , \tag{3.25}$$

where n is the thickness of the blending area (for the actual implementation $n = 4$ applied). The graph of Equation (3.25) is given in Figure 3.18 .

This blending approach provides a very smooth transition from the coarse to the fine grid. Moreover, the consecutive grids stabilize each other in the blending area.

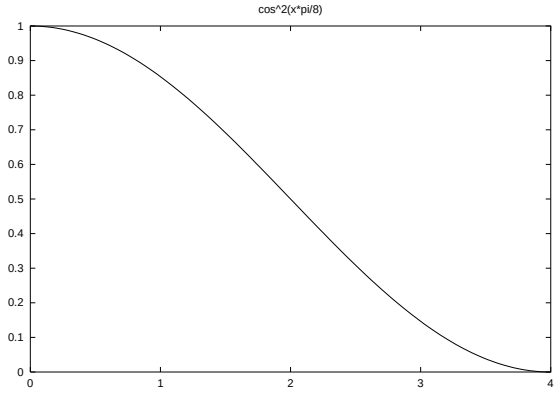

Figure 3.18: *Graph of the mixing ratio* λ

- High frequency erroneous oscillations on the fine grid can not be resolved on the coarse grid and are damped with the interpolated values at the fine grid boundaries.

- High frequency erroneous oscillations on the coarse grid do not appear on the fine grid and are similarly damped by the restrictions from the fine grid.

An important advantage of this approach is that no special discretization of the model equations is necessary. Also different discretizations can be applied without additional implementation costs incorporating the adaptive refinements. Even the computation of different models depending on the refinement level and on the corresponding mesh size becomes possible.

In general refinement structures with a number of blocks the determination of the distance i of a fine grid point from the outer boundary of the blending area can be very tedious. An example is shown in Figure 3.19 .

On the left side a refinement structure consisting of 3 blocks including the shaded blending area is displayed. The central block is indicated by dashed lines[12]. Thin lines show the distance in number of grid points from the outer boundary as in Figure 3.17 . The right side of Figure 3.19 shows a three–dimensional graph of the distance from the outer boundary for the central block.

The algorithm to determine the distance i is rather technical; only the rough idea is presented here: At first all points of a block are initialized with the distance i from the block boundaries without regard to adjacent refinement blocks. Then i is increased for each neighbor block and for all

[12] For boundary sections with adjacent refinement blocks no nesting takes place and the blending area is used to exchange the fine grid boundary values.

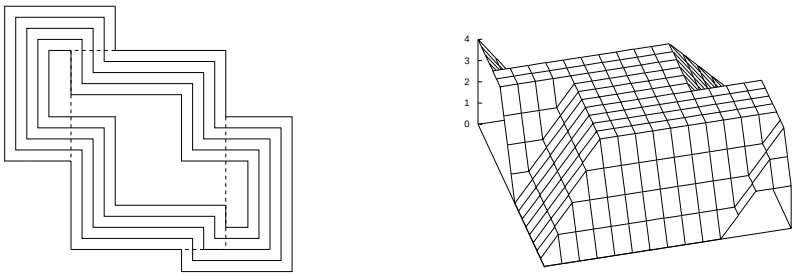

Figure 3.19: *Nesting for irregular domains*

points in the intersection of the two blocks according to the formula

$$dist(P, \bigcup_k \Omega_k) = \max_k \left(dist(P, \Omega_k) \right) \quad , \tag{3.26}$$

where P is a grid point and $\Omega_k \, , k \in \mathbb{N}$, is the sequence of blocks containing P .

It is not clear how to guarantee conservativity with blending interpolation. However, for local models conservativity is less important than for long–term global simulations (Zhang et al. [85]) . Moreover, for dynamically adaptive methods this seems to be a minor problem, since critical areas should be kept *inside* the refinement area, where they do not disturb the nesting at the refinement boundaries[13].

3.3.4 The Helmholtz equation with Local Refinements

For the semi–implicit time discretization the Helmholtz equation (3.6)[14] has to be solved on refined grids. The discretized Helmholtz equation is written

$$L_h \, u_h := (-\nabla_h^2 + c_h) \, u_h = f_h \quad , \tag{3.27}$$

where c_h is the Helmholtz constant, f_h the right hand side, and the discretized increment of the geopotential δh is denoted u_h in this section. The subscript h is the mesh size.

As mentioned before, the Helmholtz constant c_h is not actually a constant value, but it is pointwise variable given by $c_h = 1/(\Delta t^2 h_h^n \mu^2)$, due to

[13] Of course, the nesting method has to be robust enough to work also in critical situations. The implemented nesting method succeeded in tests where the advected cyclone of Section 5.1 crossed the boundary of a static refinement.

[14] Equation (A.9) in spherical coordinates, respectively

the spatial variation of the geopotential h_h^n. However, the variation in c_h does not cause special difficulties for the multigrid algorithm[15].

Since $c_h > 0$ it is known that a unique solution of Equation 3.27 exists. Moreover, the positive c_h improves the smoothing rates of Jacobi and Gauss–Seidel relaxations, and guarantees that smoothing rates at least as good as for the Laplace or Poisson equation can be achieved.

The right hand side f_h is computed with values of the prognostic values h_h, u_h, and v_h in each time step. This is a serious complication in comparison with the Laplace– or Poisson–Equation with a given right hand side. Oscillations of the prognostic variables will immediately lead to oscillations in the right hand side. The feedback to the prognosis variables of the next time step by solving the Helmholtz equation will very likely lead to instabilities in time.

Especially at the boundaries of the refinements special care has to be taken when computing the right hand side. Section 3.3.3.2 presents how the solutions of the prognostic variables on fine and corresponding coarse grids are blended in order to avoid instabilities. Nevertheless, for the computation of these variables with the semi–implicit discretization care has also been taken to prevent the introduction of oscillations and to achieve optimal results.

In general the MLAT algorithm of Table 2.3 is applied. Red–black Gauss–Seidel relaxations[16] are used for smoothing, and half injection is performed as restriction. V–cycles are applied, and bi–linear interpolation prolong the coarse grid corrections.

The number of smoothing steps ν_1 and ν_2 to be performed before and after coarse grid correction as well as the number of coarse levels is adapted to the speed of the gravity waves. The idea to adapt multigrid as stabilizer of Section 3.2 is consequently applied also in combination with adaptive refinements. If the explicit values are used as initial solution, they are copied to the coarse grid together with the right hand side of the coarse grid equation, see Section 3.4.4.

A refined grid is displayed in Figure 3.20. The fine grid $\mathbf{G}_h$ is extended by overlap regions of two grid points in width (shaded in the figure). When using the points of this overlap as interior points of the refinement, the refinement boundary can be discretized as ordinary interior points. In this way it is avoided, that interpolated coarse grid values at the refinement boundaries are evaluated for the computation of second order derivatives

[15] When solving the Helmholtz equation with a Fourier algorithm, usually a linearization of the geopotential is performed to get a constant c_h. The possibility of treating variable c_h is an advantage of multigrid.

[16] The spherical coordinates affect smoothing rates of point relaxations because of anisotropies in the proximity of the poles. However, for local weather models the coordinates can be rotated in a way that the poles are far away from the domain of the model.

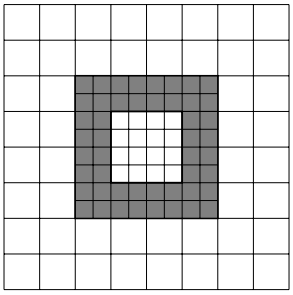

Figure 3.20: *Helmholtz equation on the refined grid*

that are necessary for the determination of the right hand side f_h. Finite differences of interpolated values are prone to deteriorate accuracy and to introduce oscillations. The overlap regions are also used for nesting and for boundary data transfer.

As an optimization the bi–cubic boundary interpolation was skipped. If the initial values at the outer boundary of the overlap region are equal to the coarse grid values, is it sufficient to interpolate the coarse grid increments to keep the values identical. Step (8) of MLAT, Table 2.3, is applied to $\mathbf{G}_h$ rather than on the inner points $\overset{\circ}{\mathbf{G}}_h$ only, and step (9) was skipped. The inner boundaries are interpolated bi–linearly only, however oscillations are kept out of the inner refinement region because of the overlap region, and because only one V–cycle is applied, which is sufficient in general.

3.4 Parallelization with Distributed Memory

Whereas the computational demands of the SWE are low, the communication requirements are comparable to computationally more demanding models. Moreover, with the application of the semi–implicit time scheme and with local refinements the simulation becomes numerically highly efficient. To achieve reasonable acceleration on modern parallel systems, the parallelization has to be of comparable quality. It is particularly desirable to port fast numerical algorithms to parallel computers, since it is not reasonable to *buy* high parallel speed–up with slow numerical performance.

In this paper major concern has been spent for parallelization to achieve good parallel efficiencies also for this worst case problem of meteorological simulations. With a number of design decisions adaptive multigrid and parallelism are combined in an efficient way. As already mentioned in the previous sections:

- Parallelization is performed by explicit message passing and the grid partitioning approach (Sections 2.4.2 and 2.4.3). With explicit message passing very general communications are supported and optimiza-

tions are possible. Portability for most modern parallel architectures is provided.

- The transfers of fully two–dimensional data (volume communication) within the dynamic adaptation of the refinements and the multigrid cycle are systematically reduced with the special hypergrid–blocking method (Section 3.3.2). Moreover, the maximal size of the refinement blocks is chosen depending on the available number of processors. For this optimization the requirement to reproduce exactly the same results when changing the number of applied processors was abandoned.

- Multigrid is applied as stabilizer and the number of coarse grids is drastically reduced. In this way the coarse grid problem of parallel multigrid is solved (Section 3.2).

More details about the parallelization of the adaptive method are given in the following. The administration of the distributed block structure along with neighborhood relationship information is discussed in Section 3.4.1 . In Section 3.4.2 the load balancing algorithm that is applied to distribute the refinement blocks to the processors is defined. Asynchronous communication provides the possibility of reducing synchronization among the processors and executing computation and communication concurrently. The concept of nonblocking asynchronous communication is presented in Section 3.4.3 . Finally, as the computational core of this paper, a parallel MLAT algorithm with asynchronous communication is provided in Section 3.4.4 .

3.4.1 Distributed Information

To reduce administration overhead and to support full flexibility in parallelization the processes are organized in a flat hierarchy. Each processor is provided with just the information necessary for computing that includes the grid point data of the designated partitions (blocks), dimension and location of the other partitions for global load balancing, and neighborhood relationships for nesting, multigrid, and boundary exchange. There is no host process that controls the computational tasks; the computing nodes are enabled to organize themselves[17]. In this way even some of the overhead to administrate the local refinements can be performed in parallel (e. g. blocking of critical points and the computation of the neighborhood relationships). Especially for the global decision base load balancing algorithm each processor is provided with the current block structure that is described in the following.

The block structure is a graph with the refinement blocks as nodes and with edges representing the neighborhood relationships. Each refinement

[17] The only exception is that global values as error norms or time measurements are evaluated by a designated processor, which also prints the control output.

block is implemented as an entity with all relevant data belonging to it: e. g. size and location, designated processor, and related edges (see details in Table 4.1.1).

An example of a block structure with two refinement levels is given in Figure 3.21 . The global block (refinement level 0) has three refinement

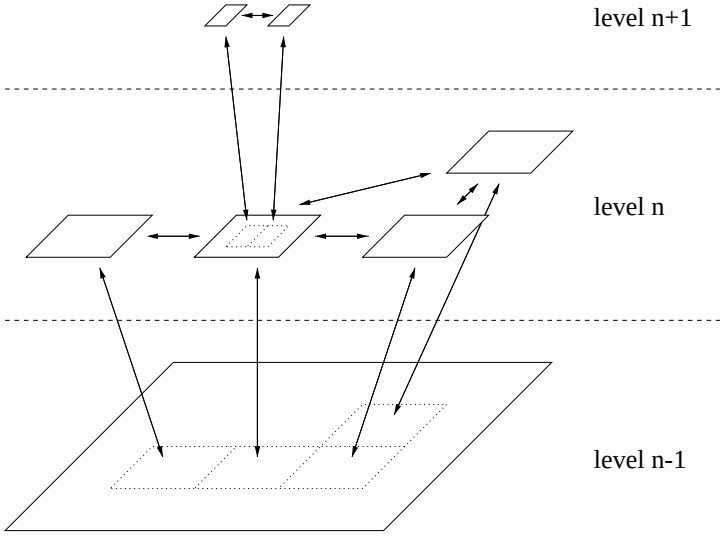

Figure 3.21: *Refinement block structure II*

blocks, the central refinement block itself has two refinements. The edges correspond to the data transfers appearing within the adaptive simulation.

The advantages of this structure are:

- The blocks and the data belonging to them are clearly arranged. All data representing a block is collected together.

- It is easy to change the block structure dynamically. When introducing new blocks and deleting obsolete ones, only the neighborhood data of the other blocks has to be updated.

The messages to be exchanged are determined on demand by evaluating the process numbers and the locations of the blocks. This is done very fast, and it is not necessary to compute and store exchange areas beforehand and adjust them each time the blocks change.

Memory for the grid functions of a block is dynamically allocated on a certain processor in case the block is distributed to this processor. This distribution of refinement blocks is computed with the load balance algorithm of the next section.

3.4.2 Load Balancing

For parallelization the refinement blocks that are created by the partitioning of the refinement areas (Section 3.3.2) have to be mapped onto the available processors of the parallel system. Here a good balance of the load is very important for parallel algorithms. An unequal distribution of the computational costs seriously limits parallel efficiency even without taking communication overhead into account (see Equation (2.25)). Moreover, the influence of load balance increases if more expensive models (three–dimensional models with more floating point operations per grid point) are simulated, or if larger grids and finer resolutions are used, in comparison to existing overhead (e. g. communications for boundary exchanges).

For the shallow water model the number of grid points is a good estimation for the computational costs of a refinement block, and we measure the work load of a block by its size.[18]

Ritzdorf and Stüben [63] developed a load balance algorithm for adaptive multigrid in a steady state context. Small blocks are mapped immediately to the processors and large blocks are further subdivided and distributed to equalize the hitherto existing distribution. However, it is important to fulfill additional conditions. Inter–level and volume data transfer were found to saturate the parallel efficiency of adaptive multigrid for increasing problem sizes and fixed numbers of processors.

In our time–dependent model two additional conditions are considered in order to reduce volume data communication.

1. The refinement structure is adapted very frequently (for the model problem of Section 5.1 in every time step) to the current weather situation. Initializations of new blocks introduce additional computational overhead and volume data communication, and their number should therefore be reduced. Moreover, unchanged blocks should stay on the same computational node and should not be re–mapped and moved to another one.

2. As there is also volume data exchange from a refinement region to its underlying coarse grid, it is preferable to map a refinement region and the coarse grid to the same node. In this way the data transfer of the adaptive algorithms can be performed locally by memory copies without interprocess communication.

To achieve optimal results these requirements are already considered when partitioning the refinement areas by hypergrids (Section 3.3.2.2). The number of initializations of new blocks is reduced, and the blocks can be

[18] The costs for the physics part of full weather models depend on the physical processes being simulated, which differ temporally and spatially. For this physically initiated load balance problem dynamic strategies are developed (e. g. Elbern [27]).

re–used in the following time steps. Furthermore, the maximal block size is limited, which is important to achieve a good balance of the work load. According to these design decisions the blocks that are created by the partitioning algorithm are not further subdivided during the load balance algorithm presented here.

Assume a set of n work loads $S = \{X^1, \ldots, X^n\}$; a mapping function π assigns the loads to the p computational nodes,

$$\pi : \quad S \to \{1, \ldots, p\} \quad ,$$

where the execution time of a parallel algorithm is determined by the maximal total load mapped to a node

$$\text{load}(\pi) := \max_{j=1}^{p} \sum_{i:\pi(i)=j} X^i \quad . \tag{3.28}$$

Minimizing $\text{load}(\pi)$ is an instance of the *makespan scheduling problem* (e. g. Coffman and Lueker [23]). Since the cost to solve this minimization problem exactly increases exponentially with n (the problem is NP–complete!), heuristic approximations are appropriate.

A simple heuristic mapping for the makespan scheduling problem is the *list scheduling* (LS) algorithm that maps the work loads $(X^i)_{i=1,\ldots,n}$ successively to the computational node according to the following rule: A load X^i is mapped to the node with the minimal sum of the previously distributed $i-1$ loads.

Denoting $\bar{X} = \frac{1}{p}\sum_{i=1}^{n} X^i$ the average load per node and assuming that X^1 is the largest work load, rough bounds can be given for the resulting mapping:

$$\bar{X} \leq \text{load}(LS) \leq \frac{1}{p}\sum_{i=2}^{n} X^i + X^1 = \bar{X} + \frac{p-1}{p}X^1 \quad ; \tag{3.29}$$

in best case the maximal total load is just the average. The worst case happens if one node starts processing the largest load, while all the others have already finished computation.

Simply put, bounds for the load balance factor λ_p that limit parallel efficiency (Equation (2.24)) are derived:

$$\frac{\bar{X}}{\bar{X} + \frac{p-1}{p}X^1} \leq \lambda_p = \frac{\bar{X}}{\text{load}(LS)} \leq 1 \quad . \tag{3.30}$$

Thus, λ_p increases if the problem size increases while the maximal block size is limited (for fixed p). Actually, this limitation is achieved by the hypergrid partitioning, where the maximal block size is defined by the mesh size of the hypergrid. Though λ_p also increases for a fixed problem size with

decreasing maximal block sizes, the introduction of additional work and communication of the block boundaries prevents unlimited reduction of the block sizes. This can be seen as *refinement agglomeration* similar to coarse grid agglomeration, where coarse grids of parallel multigrid are assembled on a reduced number of nodes (Section 2.4.3).

Since the refinement blocks are created first, their sizes are known a priori and the LS algorithm is improved easily by sorting the work loads X^i, $_{i=1,\ldots,n}$ with respect to their size, largest first, $X^1 \geq \ldots \geq X^n$. This results in the *largest processing time first* (LPT) mapping. In this way large differences that are produced by the first loads can be equalized by the later mapped smaller work units. It can be shown (e. g. Coffman and Lueker [23]) that this results in a much better asymptotic behavior for a large n. However, if n becomes large it might be more efficient to construct larger and fewer blocks.

If there are several levels of refinement, the LPT algorithms can be applied to each of the refinement blocks for each level individually. This leads to a balance of the work which has to be calculated level by level (e. g. grid operations of the MLAT for the Helmholtz equation.)

Other computations (e. g. the discretization of the explicit parts of the time scheme and the computation of the right hand side of the Helmholtz equation) can be executed independently for all refinement levels. Obviously, a balance of work for each individual level leads to a balance of the work summed over all levels. However, since the most expensive parts of the communication can be performed independently, the load balance is preferably performed for the total work of all refinement levels.

Since the refinement structure is created level by level, the local refinements of the global grid are defined first, and afterwards the refinements of the just defined refinements and so on. The resulting algorithm is *off–line* within one level, since the sizes of the blocks of each level are known before they are mapped, however *on–line* when considering all levels, because the blocks of the next higher refinement level are created after the mapping (and the migration) of the actual level has taken place.

For each process $j = 1, \ldots, p$ denote with $N(j)$ the sum of all work loads of the previous refinement levels that are already dedicated to this process and define $\bar{N} = \frac{1}{p}\left(\sum_{j=1}^{p} N(j) + \sum_{i=1}^{n} X^i\right)$, the average load over all levels including the current one. Table 3.5 shows the load balance algorithms applied on each refinement level of the adaptive method.

First, the auxilliary values are calculated and the work loads (i. e. the blocks) are sorted, largest first. The refinement blocks are distributed in the first loop preferably to the same, where the corresponding coarse block is mapped (father(X^i) denotes the coarse block of X^i), as long as the limit $\bar{N}$ for the selected process is not exceeded. In that way the balance of the work is only slightly affected.

Load Balance Algorithm

calculate $N(j)$, $_{j=1,\ldots,p}$ and $\bar{N}$

sort X^i, $_{i=1,\ldots,n}$: $X^1 \geq \ldots \geq X^n$

for i=1,...,n

(if ($N($ father$(X^i)\,) < \bar{N}$)

($\pi(X^i) := \pi($father$(X^i)\,)$
$N(\pi(X^i)\,) := N(\pi(X^i)\,) + X^i$

for i=1,...,n

(if ($\pi(X^i)$=undef and static(X^i) and $N(\pi_0(X^i)\,) < \bar{N}$)

($\pi(X^i) := \pi_0(X^i)$
$N(\pi(X^i)\,) := N(\pi(X^i)\,) + X^i$

for i=1,...,n

(if ($\pi(X^i)$=undef)

($\pi(X^i) := j_0 \quad$ *for that* $\quad N(j_0) = \min_{j=1}^{p} N(j)$
$N(\pi(X^i)\,) := N(\pi(X^i)\,) + X^i$

Table 3.5: *Load balance algorithm*

In the second loop unchanged blocks are mapped to the same processor as before, as long as they have not already been distributed in the first loop, and as long as the work load of the selected nodes is not exceeded. (static(X^i) is true, if the block X^i has existed before and false otherwise, and π_0 denotes its previous mapping.)

Finally, the remaining blocks are distributed in the third loop according to LPT one by one to the node with the current load minimum. In this way the imbalances produced in the first and second loops can be equalized with decreasing work load sizes.

The benefits of reducing the volume communication are shown in Table 5.8 , the improved load balance algorithm is compared to LPT (attained by applying the third loop of Table 3.5 only) by run time measurements.

3.4.3 Nonblocking and Asynchronous Communication

Since the terms *nonblocking* and *asynchronous communication* have various meanings in literature, their use in this work is described in the following.

- Nonblocking communication is a mode of exchanging messages of the

Message Passing Interface MPI. Nonblocking send and receive calls only solely initiate a communication operation and may return before its completion. The data transfer can be performed in the background (potentially concurrently) to the following computational operations and has to be explicitly completed by wait or test calls. The nonblocking communication mode makes it easier to avoid deadlocks and reduce synchronization.

- The communication of a parallel algorithm is called asynchronous if the message exchanges of one process are initiated at most independently of the current progress of the other processes. There are no determined communication steps between separate computational phases in a parallel algorithm with asynchronous communication. The objective is to send data as soon as it has been computed, and to perform those computations first whose required data has already arrived. In this way the synchronization of the processes is reduced, and computation and communication can be executed concurrently.

Nonblocking communication is considered a prerequisite for asynchronous data transfer.

Two examples of algorithms with asynchronous communication that have been developed and implemented for the adaptive simulation are presented and compared with their synchronous counterparts in the following. The communication pattern for a boundary exchange of the local refinements are discussed in Section 3.4.3.1 . A relaxation loop is presented in Section 3.4.3.2 .

It has to be admitted here that time measurements for asynchronous communication have not shown considerable improvement for the given model problem. Nevertheless, it is assumed that asynchronous communication can be very beneficial for problems with much higher computational costs, larger communication loads (longer messages), and also improved facilities of the parallel systems.

3.4.3.1 Synchronous and Asynchronous Boundary Exchange

In this section a synchronous communication pattern for a boundary update is compared to its asynchronous counterpart. Boundary exchanges often occur in parallel algorithms with grid partitioning. A common synchronous communication pattern (as is implemented in the communication library CLIC: Hempel and Ritzdorf [35]) is given in Table 3.6 . A two–dimensional partitioning is assumed where each processor is responsible for only one block `p` . The full boundary exchange consists of two steps, one for each dimension, in order to optimize the number of exchanged messages. This optimization is explained in Figure 3.22 (left) where a partition with four

Synchronous boundary exchange
exchange left and right boundary (p) *wait for completion, then buffer data* (p)
exchange upper and lower boundary (incl. corners) (p) *wait for completion, then buffer data* (p)

Table 3.6: Synchronous boundary exchange

blocks (and four processors) is displayed. The inner regions (the region belonging to the patch) are shaded; the overlap areas are empty. At first

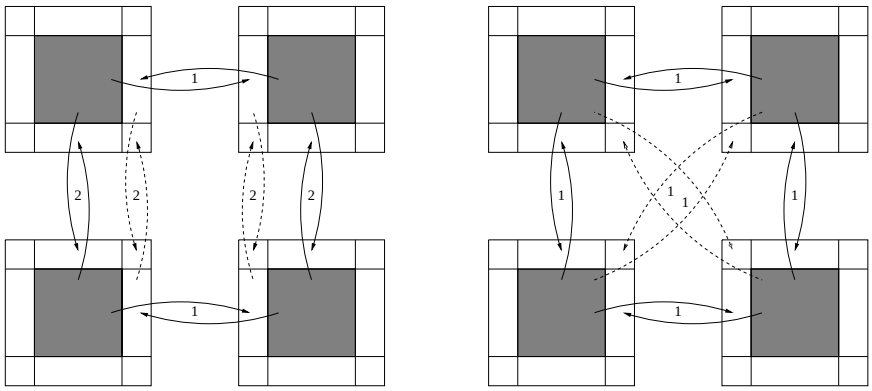

Figure 3.22: *Synchronous and asynchronous boundary exchange*

all exchanges from left to right and vice versa are performed. After the first communication step has finished (messages have arrived and are buffered) the exchanges from up to down and vice versa are carried out. The second step includes the update of the corner areas, which is indicated by dashed arrows. In this way the data belonging to the corners is passed from the diagonal patch via the upper or lower neighbor to the overlap region. The additional corner data is transmitted with the updates of the sides.

This optimization is not always possible. If one of the four patches in Figure 3.22 (left) is missing, the corner data has to be directly transferred diagonally, as in Figure 3.22 (right). Such cases often appear in general grid structures that are likely to be created in adaptive simulations.

An asynchronous boundary exchange is listed in Table 3.7 and displayed in Figure 3.22 (right). In general more than on block may be distributed to a process. The union P(lev) denotes the union of all refinement blocks of a certain refinement level lev (including the global grid with lev=0) that are distributed to the considered computational node.

Asynchronous boundary exchange	
∀p∈P(lev):	*initiate send boundaries* (p) *initiate receive boundaries* (p)
∀p∈P(lev):	*local copy of boundaries* (p)
∀p∈P(lev):	*wait for completion, then buffer data* (p)

Table 3.7: *Asynchronous boundary exchange*

The sends and receives (also sends and receives to and from diagonal blocks) are successively initiated for all local blocks. Since the sends and especially the receives are solely *initiated* by nonblocking communication operations, it becomes possible to perform computational operations during message transfer. For example, buffering of data or the local boundary exchanges between blocks that reside on the same node can be performed before the open messages are completed.

The number of messages to be exchanged is twice as much in the two–dimensional case in general; however, the degree of synchronization is reduced. In the next section asynchronous communication is applied in a relaxation step to show its full potential.

3.4.3.2 Synchronous and Asynchronous Relaxation Loop

As an example of a parallel algorithm with asynchronous communication a smoothing step that appears in parallel multigrid is considered.

To begin, a commonly used straightforward algorithm is listed in Table 3.8 . A number of nsolve grid relaxation steps are performed with subsequent boundary exchanges. These boundary exchanges are usually performed synchronously according to Table 3.6 ; however, it is also possible to apply the asynchronous boundary exchange of Table 3.7 as indicated.

First all patches distributed to the current processor are smoothed; then afterwards the boundary exchange of Table 3.7 is carried out. After the communication steps have been completed, the next grid operation can take place. In this way communication and calculation are strictly separated. This is of course an advantage for implementing and debugging, and for writing and using communication libraries. On the other hand, time spent waiting for the communication steps to be completed could be better used for subsequent computations.

The relaxation loop of Table 3.8 is still referenced synchronous, although already an asynchronous part is included. This is to differentiate the algorithm from the fully asynchronous relaxation loop of Table 3.9 with a higher degree of asynchronism. The function N(Q) denotes the number of elements

Synchronous Relaxation Loop

```
i=1,...,nsolve
  ∀p∈P(lev):   smooth (p)
  ∀p∈P(lev):   initiate send boundaries (p)
               initiate receive boundaries (p)
  ∀p∈P(lev):   local copy of boundaries (p)
  ∀p∈P(lev):   wait for completion, then buffer data (p)
```

Table 3.8: *Synchronous relaxation loop*

Asynchronous Relaxation Loop

```
i=1,...,nsolve
  Q:=P(lev)
  while N(Q)>0
    R:={p ∈ Q: for that all messages have arrived}
    ∀p∈R:   buffer data (p)
            smooth (p)
            initiate send boundaries (p)
            initiate receive boundaries (p)
    Q:=Q\R
```

Table 3.9: *Asynchronous relaxation loop*

of the union Q .

At first those blocks are selected whose messages have already been completed (especially arrived). For these blocks smoothing is performed, and the following boundary exchange is initiated. In this way the sends of data can be executed immediately when the data is available. For the first entry in the asynchronous relaxation loop it can be assumed that there are no outstanding messages and that buffering is not necessary; however, in the subsequent loop iterations it is possible that buffering and smoothing operations for some blocks are performed while other blocks still have to wait for their data.

There is no need to wait until all communication is fulfilled; the blocks whose messages have arrived can be progressed, calculation can be performed and the following messages can already be initiated. Waiting time for late messages is used and synchronization of the parallel algorithms is

reduced. In this way more advantage can be taken of a parallel system.

On the other hand it is much more difficult to implement an asynchronous algorithm like the relaxation loop, since the initiation and the completion of messages are separated in the source code as well as in the program execution.

3.4.4 Parallel Multigrid Cycle

The final section of this chapter is dedicated to the parallel adaptive multigrid cycle as the algorithmic core of this paper. Since multigrid behaves very naturally with local refinements, the algorithmic adaptations of MLAT with respect to adaptivity in Table 3.10 restrict to the distribution of multiple refinement blocks to one computational node. The sizes and locations of the refinement blocks are stored in the block structure and used only on a grid operation and message transfer level.

The algorithm is listed with a limited degree of asynchronous communication[19] for the simplicity of the presentation. In this algorithm maxlev denotes the maximum refinement level, whereas minlev is the level of the global grid (lev=0), or even a coarse grid level. The values nreld , nsolve , and nrelu denote the number of pre–smoothing steps, number of smoothing steps on the coarsest grid, and number of post–smoothing steps, respectively.

At first glance the algorithm looks like a straightforward parallelization of the sequential MLAT of Table 2.3 , whose step numbers have been inserted at the corresponding operations. However, there are two things worth mentioning:

1. The right hand side f_h of the Helmholtz equation, the Helmholtz constant c_h , and the initial values of δh_h for the iteration have to be computed in before the multigrid algorithm on all levels. Also the initial values δh_h are already interpolated at the refinement boundaries from the coarse grids. The update of the overlap areas of these variables with boundary exchanges is carried out during the first part of the multigrid cycle for optimization. The special boundary interpolation (9) was left out, since only one V–cycle is applied, and also because of the following refinement nesting procedure.

2. The grid operations (2)–(4) and (6)–(8) are carried out as operations of the refinements. In this way the load balancing of the fine grids is used also for these operations. Unnecessary imbalances would occur if these operations, which are restricted to refined areas, were carried out on the coarse grid structures. Moreover, only the smaller amount

[19] A fully asynchronous implementation according to the example of the relaxation loop in Table 3.9 was carried out; however, run time measurements have not shown considerable improvements for the given model problem.

of coarse grid data has to be transferred from fine to coarse grids and vice versa.

The smoothing rates of the applied red–black Gauss–Seidel relaxations depend on the size of the Helmholtz constant c_h. For small (positive) c_h the smoothing rates are as good as for the Laplace or Poisson–equations. With increasing c_h the rates become even better. When applying multigrid as stabilizer (Section 3.2) the multigrid cycling can be reduced to a small number of ordinary relaxations for special scenarios of the model problem of Section 5.1. Nevertheless, in the other considered scenario the full cycling algorithm (with reduced number of coarse grids) is beneficial.

Parallel Adaptive Multigrid Cycle

lev = maxlev,...,minlev

- if lev < maxlev
  - ∀p∈P(lev): *initiate receive* f_h (p)
  - ∀p∈P(lev): *local copies of* f_h (p)
  - ∀p∈P(lev): *complete receives of* f_h (p)
  - ∀p∈P(lev+1): *complete sends of* f_h (p)
- ∀p∈P(lev): *exchange boundaries of* f_h *and* δh_h (p)
- if lev > minlev
  - i=1,...,nreld
    - ∀p∈P(lev): *smooth* (p), (1)
    - ∀p∈P(lev): *exchange boundaries of* δh_h (p)
    - ∀p∈P(lev):
      - *compute* f_h (p), (2) − (4)
      - *initiate send* f_h (p)

---

i=1,...,nsolve

- ∀p∈P(lev): *smooth* (p), (5)
- ∀p∈P(lev): *exchange boundaries of* δh_h (p)

---

lev = minlev,...,maxlev

- if lev > minlev
  - ∀p∈P(lev): *initiate receive* δh_h (p)
  - ∀p∈P(lev): *local copies of* δh_h (p)
  - ∀p∈P(lev):
    - *complete receive of* δh (p)
    - *perform coarse grid correction* (p), (6) − (8)
  - ∀p∈P(lev-1): *complete sends of* δh_h (p)
  - i=1,...,nrelu
    - ∀p∈P(lev): *smooth* (p), (10)
    - ∀p∈P(lev): *exchange boundaries of* δh_h (p)
- ∀p∈P(lev): *initiate sends* δh_h (p)

Table 3.10: *Parallel adaptive multigrid cycle*

Chapter 4

Software Implementation

It has already been mentioned that the design of the dynamically adaptive method described in the previous chapter also has been implemented. This shows the practicability of the design, and is the basis of the results presented in Chapter 5 .

A full chapter is dedicated to the implementation although only a rough outline is given rather than a full documentation and reference[1]. This is justified with the great effort and difficulty implicit in implementing such an extremely complex high–end method.

The parallel adaptive method is implemented in about 10 000 lines of the computer language 'C' , which is known to be general, efficient, and very appropriate to handle dynamic structures. More than one–fourth of the lines of code is designated to the explicit communication by MPI. This is very few for an implementation with advanced communication requirements such as parallel dynamically adaptive simulation.

Section 4.1 introduces the basis data structures to administrate the information about the dynamic refinements, their neighborhood relationships, and the asynchronous communication. Based on these data structures *work units* are built, which are the primitive computational and communication parts of the parallel algorithm. The most important work units are presented in Section 4.2 .

4.1 Encapsulation of Data

The main idea of encapsulation is to gather all data together that belongs to a certain object. In this way data structures are created and easy and efficient access to this data is supported. In the following, two basic structures that are used in the parallel adaptive method are presented. The first one

[1] However, the source code is quite easy to survey and provided with many comments. Experience shows that the readability of a code is a very important issue besides efficiency.

is the central structure for adaptivity. It defines the implementation of a refinement block as data structure `spatch` and is given in Section 4.1.1 . The second data structure `smess` is the basis for the parallelization of the adaptive method. It prescribes the data belonging to a message to be exchanged (see Section 4.1.2).

4.1.1 Data Structure for Local Refinements

Each partition of the coarse grids as well as each refinement block is implemented as an instantiation of type `spatch`. All data defining a rectangular partition is gathered, e. g. the size of the block, its location, the work space for the variables, and the neighborhood relationships. Based on this object type the block structure is implemented (see Section 3.4.1 , an example is visualized in Figure 3.21).

In detail type `spatch` is declared as given in table 4.1 .[2] The element

Structure `spatch`		
type	*element*	*comment*
`int`	`level`	refinement level
`int`	`ref`	number of refinements
`int`	`nx`	number of intervals in x–direction
`int`	`ny`	number of intervals in y–direction
`sloc`	`locr`	position relative to coarse grid
`sloc`	`loca`	absolute position in intervals
`double`	`xmin`	coordinate of left boundary
`double`	`ymin`	coordinate of lower boundary
`double`	`dx`	mesh size in x–direction
`double`	`dy`	mesh size in y–direction
`double`	`*cp`	pointer to auxiliary data array
`double`	`*sp`	pointer to auxiliary data array
`double`	`*data[NARRAY]`	pointer to work space for variables
`spatch`	`*pf[MAXPATCH]`	pointer to refinements
`spatch`	`*pc`	pointer to coarse grid
`sneigp`	`np[4]`	pointer to neighbors of each side
`int`	`number`	global number of refinement
`int`	`proc`	number of processor
`smess`	`*pmess`	structure for asynchronous communication

Table 4.1: *Data structure* `spatch`

`data` is an array of pointers that are the addresses of the data arrays for the

[2] The declaration style of this and the following structures is according to the programming language 'C'. The stars (`*`) indicate pointers (variables for addresses).

model variables h, u, and v. These data arrays are dynamically allocated in case the block is mapped to the local processor. (`MAXPATCH` is the number of grid functions).

The elements `pc` and `pf` are pointers to the related coarser and finer blocks, respectively (there is only one coarse block, and the maximal number of refinements of a single block is `MAXPATCH`). The neighbors of the block of the same refinement level are stored in array `np`, which is a data structure itself and given in Table 4.2. In this way the number of adjacent blocks and the pointers to them are stored for each side of the block separately. With

Structure `sneigp`		
type	*element*	*comment*
`int`	`n`	number of neighbors
`spatch`	`*pn[MAXNEIG]`	array of pointers to neighbor patches

Table 4.2: *Data structure* `sneigp`

`pc`, `pf`, and the pointers `pn` of `sneigp` the relationships are defined. The block structure can easily be dynamically adapted during the simulation with these pointers.

The elements `locr` and `loca` are structures of type `sloc` for storing the coordinates of the block relative to its coarser block and absolute coordinates, respectively. Type `sloc` is declared as in Table 4.3.

Structure `sloc`		
type	*element*	*comment*
`int`	`imin`	first index of first coordinate
`int`	`imax`	last index of first coordinate
`int`	`jmin`	first index of second coordinate
`int`	`jmax`	last index of second coordinate

Table 4.3: *Data structure* `sloc`

Finally, element `pmess` is a pointer to the data structure, which describes the messages related with the block. Its definition is given in the following section.

4.1.2 Data Structure for Asynchronous Communication

The initiations of sends and receives are statically (with respect to the source code) and dynamically (with respect to the execution of the algorithm) separated from their corresponding completions when asynchronous communication as in Table 3.7 is applied. Information about the message such as its

content, its length, and its buffer space has to be stored when initiating the messages in order to perform the completion and buffering later.

To keep this information the data structure `smess` was declared as in Table 4.4 .

Structure `smess`		
type — *element*		*comment*
`int`	`nmess`	number of open messages
`MPI_Request`	`areq[MAXMESS]`	array of requests
`sbufinfo`	`bufinfo[MAXMESS]`	information for buffering
`int`	`(*afuncbuf[MAXMESS])` `(sbufinfo *bufinfo)`	function for buffering
`long`	`length[MAXMESS]`	length of message (for consistency check)

Table 4.4: *Data structure* `smess`

Besides the number `nmess` of initiated and not yet completed messages information for each of these message is stored. Element `areq` is an array of MPI–request handles (see Section 2.4.2), and `bufinfo` is an array of data type `sbufinfo`, which is used to store information about the contents of the message. The declaration of type `sbufinfo` is given in Table 4.5 .

Finally, element `afuncbuf` is an array of functions that are called after the message has arrived and that perform the buffering and clean up to complete the message.

The data structure `bufinfo` contains the information which is required for function `afuncbuf` to perform the buffering of the received data to the grid function the data belongs to.

Structure `sbufinfo`		
type — *element*		*comment*
`sloc`	`loc`	location of data in data array
`int`	`ny`	extension of first dimension
`int`	`istride`	every or every second point
`int`	`nvar`	number of variables to communicate
`double`	`*apbuf[NARRAY]`	pointers to data arrays
`double`	`*buffer`	communication buffer

Table 4.5: *Data structure* `sbufinfo`

Structure `sbufinfo` contains in `loc` (see Table 4.3) the location of data to be received in the data arrays. The second dimension of these arrays is

stored in `ny`, which is essential to put the received data in the array. The variable `stride` can be used to define whether the message contains data for every point or for every second point. This is very important when data is sent from a fine grid to its nesting coarse grid and back again.

Sometimes several grid functions have to be exchanged at the same time. This data is combined into one single message. The number of grid functions is `nvar`, and the pointers to the data arrays of the specified grid functions are stored in array `apbuf[NARRAY]`. Of course, a pointer to the buffer the actual message is received in is also given in `buffer`.

This data structure information is sufficient to define the contents of two–dimensional data arrays to be sent or received. It is possible to construct more general versions of `sbufinfo` to store information about general domains or three–dimensional blocks.

4.2 Work Units

Based on the declaration of the data structure `spatch` the atomar parts of computation and communication are defined as *work units*. There are two kinds: computational work units are sequences of grid operations applied on the refinement block, and communication units are initiations or completions (or local copies) of a number of messages to be transferred at a certain step of the algorithm.

The most important computational work units are:

- `solve_pre(p)`
  Calculation of the right hand side f_h and the Helmholtz constant c_h of the Helmholtz equation (Steps (1)–(3) of Table 3.1).
- `solve_post(p)`
  Complete time step (Steps (5) and (6) of Table 3.1).
- `smooth_gs(p)`
  Perform two half–steps of the Gauss–Seidel smoother (Steps (2)–(4) of Table 2.3)
- `res_ref_hw(p)`
  Computation of residuum and half–weighting and update of the right hand side of the Helmholtz equation (Steps (2)–(4) of Table 2.3)
- `int_fine_cor(p)`
  Calculation of coarse grid increments and prolongation and addition of the correction (Steps (6)–(8) of Table 2.3)

Since all data defining a refinement block is gathered, one parameter — a pointer to the related data structure — is sufficient to perform these computations. Moreover, it is possible to write interfaces to existing meteorological modules in this way.

Parallelization on this computational work unit level is coarse–grain and easy to survey.

Similar work units are implemented for communication. For example, the asynchronous boundary exchange of Table 3.7 is formulated with the following communication work units:

- `exch_boundaries_send(p)`
  initiate sends
- `exch_boundaries_recv(p)`
  initiate receives
- `exch_boundaries_local(p)`
  local copies of data
- `exch_wait(p)`
  wait and complete all open messages

The information about the message to be sent or received is stored by the related communication work units in the data structure `bufinfo` (Table 4.5), and used later for completion and buffering of the data. Of course, with asynchronous communication the computation and communication can be no longer separated by use of communication libraries. However, when applying this concept of communication work units, the details of data transfer are hidden on the computational level.

Chapter 5

Performance Results

The design and implementation described in the previous two sections must now be verified and evaluated by the results of simulations and run–time measurements. The improvements achieved by the application of multigrid as stabilizer are compared to calculations where the Helmholtz equation is solved. Dynamically adaptive simulations are compared to a highly resolved non–adaptive reference. An analytical solution exists for a reasonable model problem.

Especially with local refinements the question of a *fair* model problem arises. The reduction of the problem size with adaptive refinements clearly depends on the weather simulated. If the weather conditions force local refinements almost *everywhere* in the computational domain, clearly no advantage results from the adaptive algorithm (moreover, additional overhead is introduced). The other extreme is very calm weather conditions where no local refinements are actually necessary. In this case the acceleration due to adaptivity can be extreme.

For this background a model problem *artificial cyclone* has been developed that represents a reasonable meteorological application. The size of the refinement area can be adjusted, and an analytical solution is known. The model problem is presented in Section 5.1 . Explicit and semi–implicit time stepping with multigrid applied as solver and stabilizer are compared in Sections 5.2 . In Section 5.3 the accuracy and acceleration in computing time of the adaptive simulation is verified and evaluated and, finally, in Section 5.4 the resulting parallel speed–ups of the parallel adaptive methods are presented.

5.1 Model Problem — Artificial Cyclone

The model problem is closely related to the results presented in the following because of its strong dependence on the size of the refinement area. Its description is therefore included in this chapter.

The model problem *artificial cyclone* is a swirl that is idealized by an artificial low pressure region with a rotating flow around it. Pressure and centrifugal force are in equilibrium, so that the shape of the swirl is preserved. Dependent on a constant background flow the cyclone can be used both as stationary and as in–stationary model problem. An analytical solution exists, which is used to provide initial conditions for the simulation, and to calculate global errors in order to evaluate the adaptive model.

For a given pressure profile the equilibrating rotational flow will be derived in the following. We start from the shallow water equations in Cartesian coordinates (2.7) without orography ($h^s \equiv 0$). To derive an analytical stationary and rotational invariant solution the partial derivatives with respect to t are eliminated, the others are transformed into polar coordinates r and φ originating from the center of the swirl by

$$\begin{aligned} x &= r\cos\varphi \\ y &= r\sin\varphi \end{aligned} \quad , \quad \text{with } r > 0 \text{ and } \varphi \in [0, 2\pi[$$

and

$$\begin{aligned} \frac{\partial}{\partial x} &= \cos\varphi \frac{\partial}{\partial r} - \frac{1}{r}\sin\varphi \frac{\partial}{\partial \varphi} \\ \frac{\partial}{\partial y} &= \sin\varphi \frac{\partial}{\partial r} + \frac{1}{r}\cos\varphi \frac{\partial}{\partial \varphi} \end{aligned} \quad .$$

Introducing a rotational invariant geopotential $h(r)$ and tangential velocity $V(r)$

$$\begin{aligned} u &= -V(r)\sin\varphi \\ v &= V(r)\cos\varphi \\ h &= h(r) \end{aligned}$$

gives

$$-\frac{V^2}{r} - fV + \frac{\partial h}{\partial r} = 0 \quad , \tag{5.1}$$

hence

$$V_{1,2}(r) = -\frac{rf}{2} \overset{+}{(-)} \sqrt{\frac{r^2 f^2}{4} + r\frac{\partial h}{\partial r}} \quad . \tag{5.2}$$

A real solution exists only for $\frac{\partial h}{\partial r} > -\frac{rf^2}{4}$, and we restrict ourselves to low pressure areas ($\frac{\partial h}{\partial r} > 0$).

In case of vanishing Coriolis force ($f = 0$) there are two symmetrical solutions: a cyclonal and an anti–cyclonal flow. Otherwise ($f \neq 0$) there are also two solutions, however, only for the first one the tangential velocity vanishes far from the swirl ($V_1(r) \to 0$ if $r \to \infty$ and $\frac{\partial h}{\partial r} \to 0$)[1] and we will concentrate on it.

[1] On the northern hemisphere $f > 0$ and a cyclonal stream is counterclockwise.

The resulting stationary swirl is given by

$$\begin{aligned} u &= -\left(\sqrt{r\frac{\partial h}{\partial r}+\frac{r^2f^2}{4}}-\frac{rf}{2}\right)\sin\varphi \\ v &= \left(\sqrt{r\frac{\partial h}{\partial r}+\frac{r^2f^2}{4}}-\frac{rf}{2}\right)\cos\varphi \\ h &= h(r)>0 \end{aligned} \tag{5.3}$$

with $r=0$ being its center.

For the geopotential profile of the artificial cyclone

$$h(r)=h_0\left(1-D\,\mathrm{e}^{-\left(\frac{r}{R}\right)^2}\right) \tag{5.4}$$

is chosen, where h_0 is the main geopotential height and D and R are constant parameters. D defines the maximal depression in the center and R the horizontal extension of the cyclone. The maximum tangential velocity is reached at distance $r=R$ from the center of the cyclone, $V_{max}=V(R)=\sqrt{\frac{2\,Dh_0}{\mathrm{e}}}$.

An example for the geopotential profile (with already adapted mesh) is shown in Figure 5.1 (top). To yield an in–stationary model problem the cyclone is moved by adding a constant background flow (u_0, v_0). In case $f=0$ the cyclone is advected in a shape–preserving way. This gives the advantageous possibility of providing an analytical solution for the instationary model problem. For $f\neq 0$ the cyclone is deformed during the simulation and no analytical solution could be found. Therefore the Coriolis force was neglected in the following. Numerical tests showed little influence of the Coriolis force for the described model problem.

Two scenarios of the cyclone with different basic geopotential heights were selected to show the influence of the speed of the gravity waves on the numerical algorithm. The geopotential h_0=15 000 $\mathrm{m}^2/\mathrm{s}^2$ is related to a lower level of the atmosphere of about 1.5 km height and average speed of gravity waves, whereas for h_0=120 000 $\mathrm{m}^2/\mathrm{s}^2$ a high level of about 12 km and a high speed of the gravity waves is selected (compare Table 2.1).

The parameters of the cyclone are defined $D=\frac{1\,000}{h_0}$ and R=31.25 km for both scenarios: thus the maximal depression in both cases is 1 000 $\mathrm{m}^2/\mathrm{s}^2$ and the maximal tangential velocity is 27.12 m/s. The components of the background flow are u_0=12 m/s and v_0=9 m/s, so that the total speed is 15 m/s. In combination with the tangential velocity the maximal speed of advection is 42.12 m/s.

The computational domain is a square of 1 000 km by 1 000 km, which is discretized by 1024×1024 intervals, providing a resolution of just below

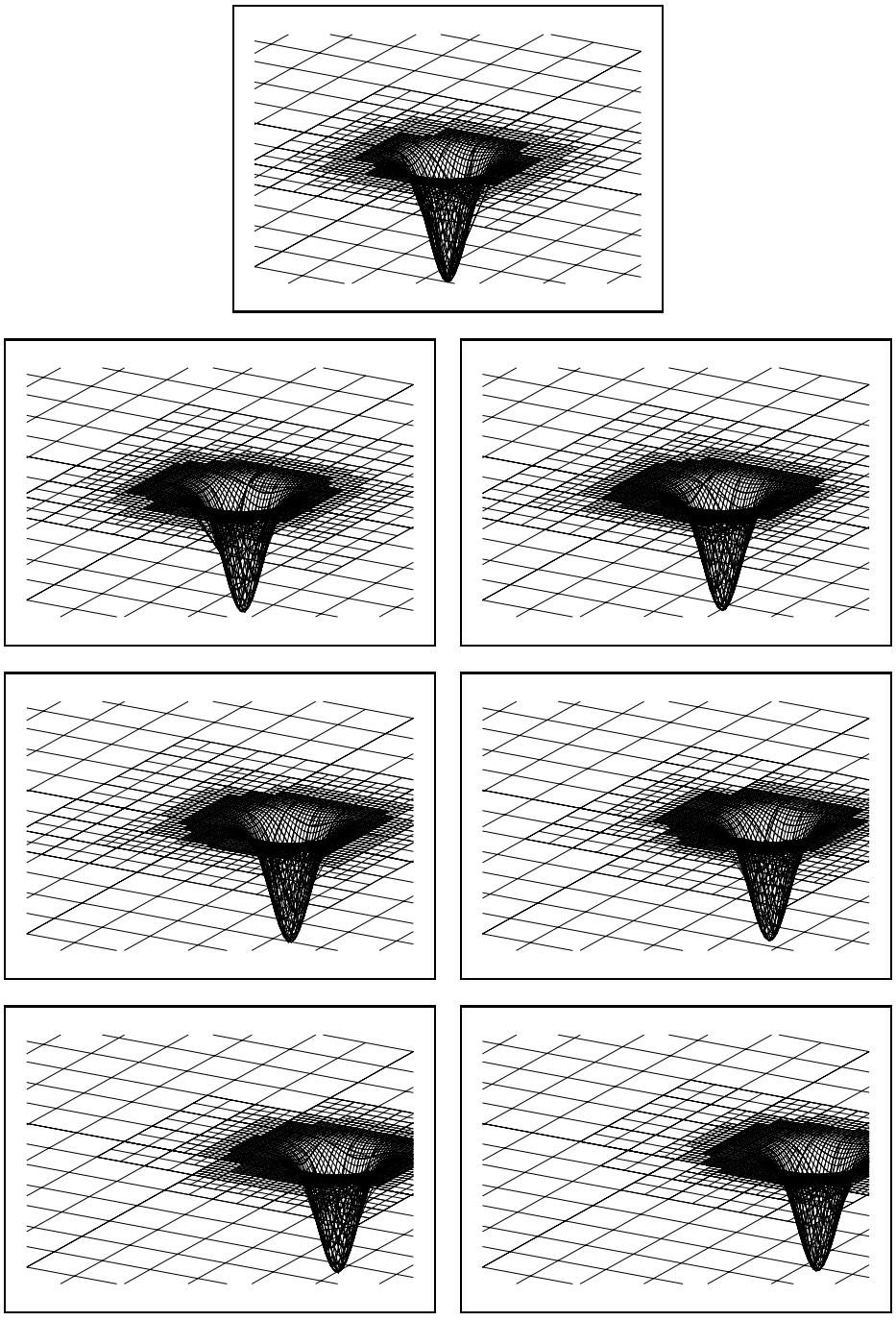

Figure 5.1: *Geopotential of instationary artificial cyclone*

1 km in non–adaptive simulations. The considered model time frame is 6 hours.

During this model time the cyclone moves 324 km. This is more than 10 times its characteristic radius of R=31.25 km, which shows the demanding requirements for the adaptive refinements. In Figure 5.1 a sequence of refined grids of a time–dependent simulation (with slightly altered parameters for visualization) is displayed. From top to bottom the artificial cyclone moves to the right upper corner, and the grid automatically adapts as shown.

The artificial cyclone is a very good model problem, because an analytical solution is available. The following simulations and run time measurement concentrate on it. Nevertheless, another scenario was simulated and visualized to experience the full capabilities of the automatic adaptation. Figure 5.2 displays the cyclone of the model problem in combination with an artificial anti–cyclone. As shown in Equation (5.1) strong anti–cyclones can not be stable. This result corresponds to the simulation where the anti–cyclone dissolves very quickly. A large circular gravity wave evolves, which is very effectively damped (as is intended with the application of the implicit Euler time scheme for the gravity terms).

5.2 Explicit and Implicit Time Stepping

In this section the explicit time stepping is compared with the semi–implicit scheme presented in Section 3.1.1 . To compare the numerical performance of the schemes, both scenarios of the cyclone with high and average geopotential and thus fast and slow gravity waves are calculated on the global non–adaptive grid with 1024×1024 intervals.

For the high basic geopotential of 120 000 m^2/s^2 the speed of the gravity waves is 346.41 m/s relative to the advection with 42.12 m/s and therefore 388.53 m/s in maximum. With spatial resolution of just below 1 km and a restriction by the CFL number 1, the maximal stable time step size is 2.5 s . With the implicit scheme and a CFL number 0.7 the maximal time step size is 15 s, which is 6 times higher.

In Table 5.1 run–time measurements for the dynamics part, the multigrid solver, and the total time (including, e. g. boundary nesting of the global grid, but excluding initializations and I/O) are listed. The first row presents the times for the explicit time scheme and the second row for the semi–implicit scheme, where the Helmholtz equation is solved by 3 multigrid V(2,1,1) cycles (with 2 pre–relaxations and 1 post–relaxation, 1 relaxation on the coarsest grid) on 10 grid levels that assure a convergence rate of below 0.1 . This provides sufficient accuracy when starting with 0 as initial values, which is a good initial condition, since the Helmholtz equation is posed for the time increment of the geopotential.

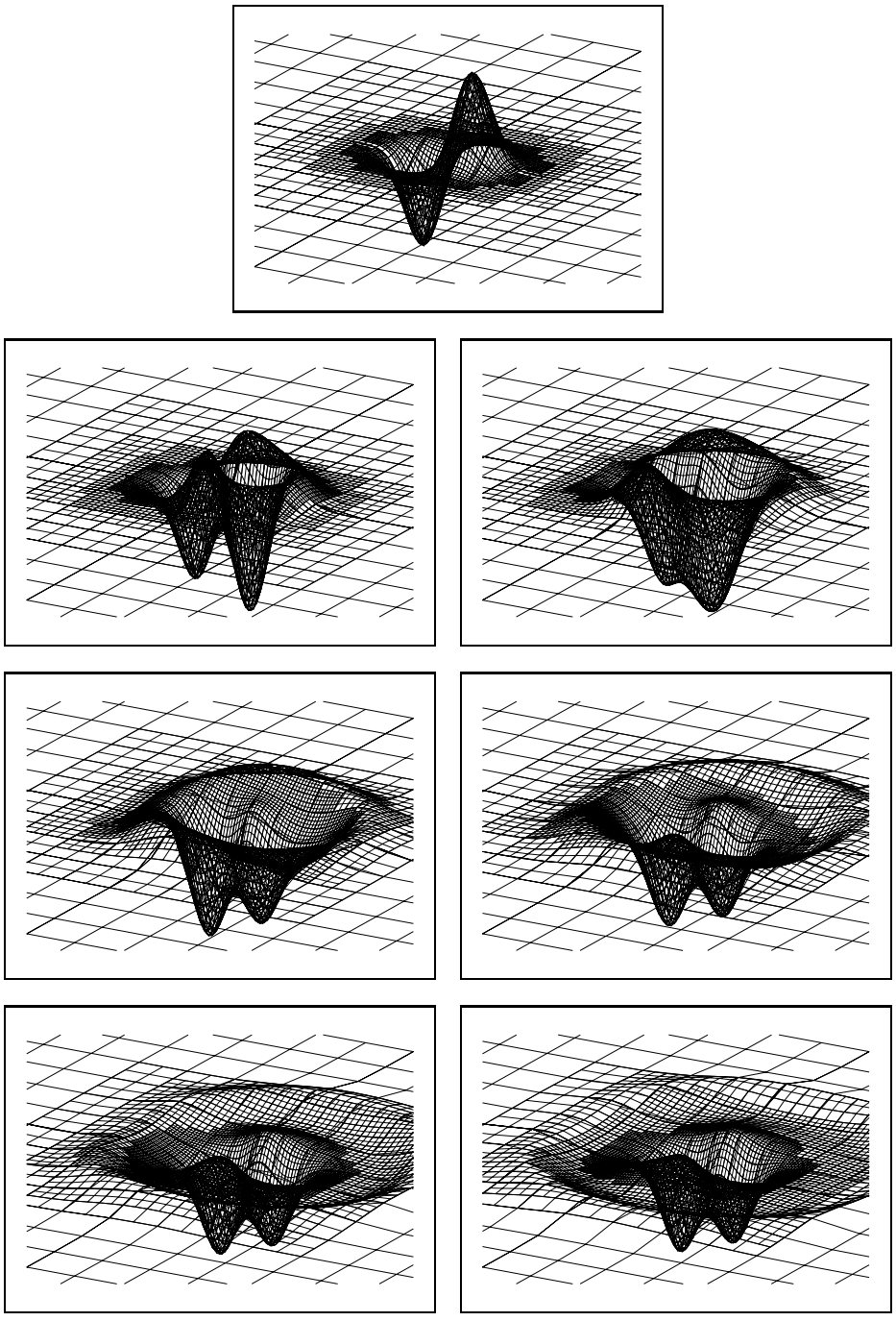

Figure 5.2: *Geopotential of artificial scenario*

Although 6 times more time steps take place with the explicit scheme, the time of the dynamics of the semi–implicit scheme is only 4.6 times smaller, because the calculation of the right hand side of the Helmholtz equation is included in the dynamics part. The multigrid cycling is very expensive in comparison to the computation of the dynamics part, nevertheless the semi–implicit scheme is faster than the purely explicit time discretization.

The given run times are measured for a parallel simulation on an IBM SP2 with 16 computational nodes. Because of memory requirements it was not possible to run this problem on a single processor. The measurements also include communication overhead, whose portion is larger for the multigrid solver in comparison to the dynamics (although agglomeration was applied for very coarse grids).

Table 5.1 also includes the maximal relative global error of the simulation in comparison to the analytical solution. Although the time steps of the explicit simulation are much smaller and the implicit time scheme is of first order only for the gravity wave and strongly damping, the error of the implicit scheme is only slightly larger.

time scheme	Δt	dynamic	multigrid	total	glob. err.
explicit	2.5	2302	—	2567	4.39e-5
semi–implicit (solve)	15	497	874	1413	5.43e-5
semi–implicit (stabilize)	15	502	319	864	5.26e-5

Table 5.1: *Explicit and semi–implicit scheme, 16 nodes — fast waves*

The third row shows the results of the application of multigrid as stabilizer (Section 3.2). In this way very cheap iterations of the Helmholtz equation are sufficient. Numerical tests show that only one V(2,2,1) cycle with 2 coarse levels is sufficient for this test case. The time for the semi–implicit scheme is essentially reduced. The global relative error is little smaller than before, showing the increased influence of the explicit initial values, and a reduced damping of the gravity waves.

The gravity waves of the low basic geopotential of $15\,000\,\mathrm{m}^2/\mathrm{s}^2$ are slower and the explicit time step size is increased to 5 s, which reduces the computational work by a factor of two. On the other hand, the computational costs of the semi–implicit scheme can also be reduced when stabilizing. The ratio between gravity wave speed and advection is smaller. As a consequence already 2 smoothing steps are sufficient to fulfill the stability requirement in this case. To be concrete: the multigrid algorithm for the solution of the Helmholtz equation degenerates into two simple smoothing steps. Run times and errors are listed in Table 5.2. Also in this case there is an essential acceleration of the simulation because of the stabilizing approach within the semi–implicit time scheme in comparison to the explicit time dis-

time scheme	Δt	dynamic	multigrid	total	glob. err.
explicit	5	1149	—	1276	3.51e-4
semi–implicit (solve)	15	500	875	1419	5.18e-4
semi–implicit (stabilize)	15	499	103	643	5.03e-4

Table 5.2: *Explicit and semi–implicit scheme, 16 nodes — slow waves*

cretization. Moreover, this algorithmic improvement is valuable especially for parallel computers, as the coarse grids, which require a large degree of communication, are avoided.

5.3 Variation of Resolution and Adaptivity

Figure 5.1 displays the geopotential of the model problem on an adapted grid. The top frame shows the initially refined grid and lower frames the situations in intervals of 1 hour. The grid is automatically adapted during the simulation, obviously the applied refinement criterion shows very reasonable results.

In order to validate the adaptive model, the loss of accuracy and the gain in computational work due to the local refinements is compared to a simulation on a global fine grid. Figure 5.3 shows the maximal relative global error of the semi–implicit calculation of the lower geopotential case for a number of consecutive global resolutions (with adjusted time step sizes) in comparison to the analytical reference. The global resolutions have 128, 256, 512, and 1024 intervals in each dimension. In a short adjustment phase the analytical initial values are adjusted to the discretization and the error increases rapidly. Afterward the global error increases very slowly due to numerical diffusion. Almost second order accuracy is achieved by the implicit time scheme, although the gravity wave terms are discretized at first order only. According to the previous section, only two relaxations were applied to solve the implicit time scheme for the "slow waves"–case. This also appears to be sufficient with local refinements.

In Figure 5.4 the accuracy of the adaptive method is compared to the global highly resolved simulations. The maximal relative errors of the high resolutions with 512 and 1024 intervals in each dimension of Figure 5.3 are displayed again. Additionally results of the dynamically adaptive simulations with a global grid with 128×128 intervals and two or three levels of refinement are shown. With the refinement ratio of 1:2, the maximum resolution of the adaptive runs and the corresponding global resolutions are equivalent.

The refinement criterion limits the local discretization error in the adaptive simulation to the maximum local error of the corresponding global fine

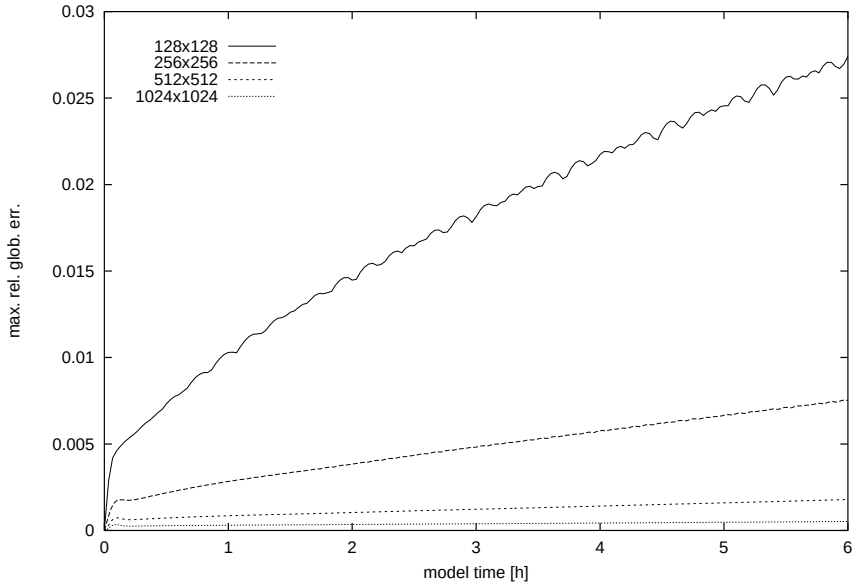

Figure 5.3: *Maximal global relative errors on global grids*

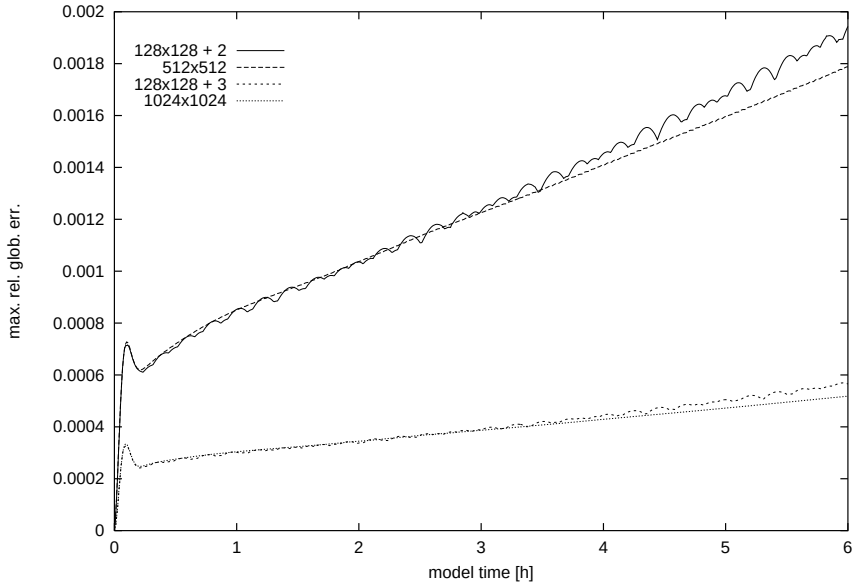

Figure 5.4: *Maximal global relative errors on adaptive grids*

grid. It is achieved that the global error of the adaptive simulation is of the same order of magnitude as the global error of the globally highly resolved computation. The errors of the refined simulation are only slightly larger, which shows the *self–consistency* of the adaptive method[2].

Not only the correct choice of refinement points, also an appropriate nesting procedure is important to achieve this result.

In the example with three refinement levels the total number of inner grid points of the dynamically adaptive method (including the global fine grid and all refinement levels) varies between 75 240 and 91 292 , which is between 13.9 and 11.5 times fewer than the grid points of the corresponding global fine grid with $1\,023^2 = 1\,046\,529$ inner points. However, the additional overhead when introducing local refinements (e. g. calculation of the refinement criterion and nesting of the refinement boundaries) decreases the total adaptive speed–up.

Table 5.3 shows the benefits of the adaptive method in measured run times. The adaptive method is 8.1 times faster with almost the same ac-

method	dynamic	multigrid	refine	total	glob. err.
non–adaptive	7444	1344	—	9184	5.03-4
adaptive	646	124	358	1128	5.51-4

Table 5.3: *Adaptive acceleration, 1 node — slow waves*

curacy. The overhead for the local refinements of three refinement levels, including the criterion and the mapping and initialization of new blocks, is still much smaller than the costs for discretizing the rather simple SWE.

The factor 8.1 shows the great potential of adaptive refinements. However, as mentioned earlier, this speed–up depends very much on the application.[3] In our case the model problem is chosen to show more or less realistic properties.

The given times in Table 5.3 denote the run time on one node (sequential run); however, the time for the non–adaptive simulation is estimated at one–forth of the time of a parallel run on 4 processors. This is reasonable when considering Table 5.4 .

[2] The wiggles in the errors of the calculation with local refinements come from their calculation for points of the global fine grid only.

[3] In this example a two–dimensional region (the low pressure area) was refined, where a higher number of refinement levels increases the degree of adaptivity only slightly further on. This is very different to lower dimensional regions as weather fronts or single points (from a coarse grid point of view, since weather fronts also become two–dimensional if calculated at very high resolution).

5.4 Parallelism

It is well known that very good parallel efficiencies can be achieved with the grid partitioning parallelization approach. This is also true for this shallow water model as shown for a non–adaptive version in the following section. It is much more difficult to obtain high parallel speed–up for adaptive simulations. Not only the problem size is drastically reduced; additional overhead (e. g. communication, sequential parts) is also introduced. The result of the design of the parallel adaptive model is shown with regard to parallel efficiencies thereafter. In the last section the results of the mapping algorithm (Section 3.4.2) are studied.

5.4.1 Parallel Non–Adaptive Simulation

In this section parallel efficiencies for a non–adaptive simulation are presented. In this case *perfect* load balance is achieved.

Table 5.4 lists run times and parallel speed–ups for the dynamic part, the multigrid solver and the total simulation for partitions with 2×2 and 4×4 nodes with the different time stepping methods for the "slow waves" case. The given parallel speed–ups are relative accelerations from 4 to 16 nodes.

The dynamics part shows a constant acceleration of 3.7 when increasing the number of nodes from 4 to 16. This is an excellent value for a discretization of the SWE on a grid with 1023^2 inner grid points computed on 16 nodes of an IBM SP2. The resulting parallel efficiency is 92.5%.

Also the multigrid algorithm scales very well in the solving as well as stabilizing mode (in the stabilizing mode only 2 smoothing steps are performed).

Consequently, the speed–ups for the total times show that an even higher number of processing nodes could be applicable.

method	part.	dynamic		multigrid		total	
explicit	2×2	4208		—		4509	
	4×4	1149	(3.7)	—		1276	(3.5)
solve	2×2	1862		2833		4794	
	4×4	500	(3.7)	875	(3.2)	1419	(3.4)
stabilize	2×2	1861		336		2296	
	4×4	499	(3.7)	103	(3.3)	643	(3.6)

Table 5.4: *Parallel speed–up, non–adaptive method — slow waves*

Similar results are obtained when calculating the model case with fast gravity waves, as seen in Table 5.5.

method	part.	dynamic	multigrid	total
explicit	2×2	8312	—	8898
	4×4	2301 (3.6)	—	2567 (3.5)
stabilize	2×2	1842	1076	3072
	4×4	502 (3.7)	319 (3.4)	864 (3.6)

Table 5.5: *Parallel speed–up, non–adaptive method — fast waves*

5.4.2 Parallel Adaptive Simulation

As before, the global grid of the adaptive simulation has 128×128 intervals; 3 refinement levels are initiated to provide locally the resolution of the non–adaptive method.

The parallel efficiencies cannot be expected to be as good as for the non–adaptive runs. Not only is the computational work essentially reduced and additional communication introduced, but load imbalances also occur. Moreover, the maximal block size is very small when the global coarse grid and the local refinements are distributed to a higher number of processors, and the additional work at the block boundaries becomes visible. With 16 nodes a partitioning with a maximum block size of 33^2 points has to be used. This leads to an additional computational overhead of almost 10% in comparison to the maximal block size of 65^2 points, which is appropriate for 4 nodes. In Figure 5.6 the total number of points involved in the adaptive run is displayed during the simulation period of 6 hours for both block sizes.

Nevertheless, the speed–ups of the dynamics listed in Table 5.6 are still good. The multigrid algorithm, however, shows difficulties for this small

method	part.	dynamic	multigrid	refine	total
explicit	1×1	1455	—	1055	2510
	2×2	408 (3.6)	—	435 (2.4)	892 (2.8)
	4×4	138 (3.0)	—	323 (1.3)	466 (1.9)
solve	1×1	644	731	352	1727
	2×2	180 (3.6)	326 (2.2)	161 (2.2)	667 (2.6)
	4×4	58 (3.1)	225 (1.4)	104 (1.5)	388 (1.7)
stabilize	1×1	649	117	358	1124
	2×2	182 (3.6)	47 (2.5)	160 (2.2)	390 (2.9)
	4×4	61 (3.0)	29 (1.6)	107 (1.5)	198 (2.0)

Table 5.6: *Parallel speed–up, adaptive method — slow waves*

problem size, when used as solver performing 3 V(2,1,1) cycles (although agglomeration is used for coarse grids). Here the advantage of the stabilizing

method becomes fully apparent. It is not the parallel efficiency which is better for the two relaxation steps in comparison to the full cycles, it is rather the smaller portion of the total work, which increases the speed–up of the total time with the stabilizing method.

It is worth mentioning that the refinement overhead is also parallelized and scales similar to the stabilizer, although much communication is performed in this part of the adaptive model. This is discussed later in Section 5.4.3 .

To summarize at this point: the simulation with dynamically local refinement has a speed–up of 5.7 on 16 nodes of the IBM SP2 with respect to a sequential run. This is considered a very good value when taking into account that the original problem size is reduced by a factor of 16 with the presented algorithmic improvements, as displayed in Figure 1.1 on Page 18 . Moreover, the speed–up 2 from 4 to 16 nodes shows that the adaptive method is scalable.[4] For larger problem sizes and computationally more demanding equations even more processors could be used in a reasonable way.

Similar results are obtained for the fast waves, where the stabilizer is a V(2,2,1) cycle and not fully degenerated as before for the slow waves. In this case inter–level communication occurs within the multigrid cycling and the grid levels are computed one after the other. The stabilization is more

method	part.	dynamic	multigrid	refine	total
stabilize	1×1	650	257	357	1264
	2×2	181 (3.6)	119 (2.2)	161 (2.2)	460 (2.7)
	4×4	59 (3.1)	76 (1.6)	108 (1.5)	245 (1.9)

Table 5.7: *Parallel speed–up, adaptive method — fast waves*

expensive than for the slow waves; however the benefits of the semi–implicit scheme are also higher, as displayed in Figure 5.5 . The fast waves restrict the time steps with fully explicit scheme to only 2.5 s . More than two magnitudes are won with the combination of the semi–implicit and adaptive method with parallelization in this case.

5.4.3 Load Balancing

The load balance is considered a very critical part of parallel and adaptive methods in general. In our adaptive shallow water model the refinement areas are partitioned into blocks with maximal size, which are then mapped onto the processors in a way that balances work and reduces communication

[4] Some parallel algorithms show sufficient parallel efficiency with up to 4 nodes, but break down very soon with higher processor numbers and then even show an increase in execution time.

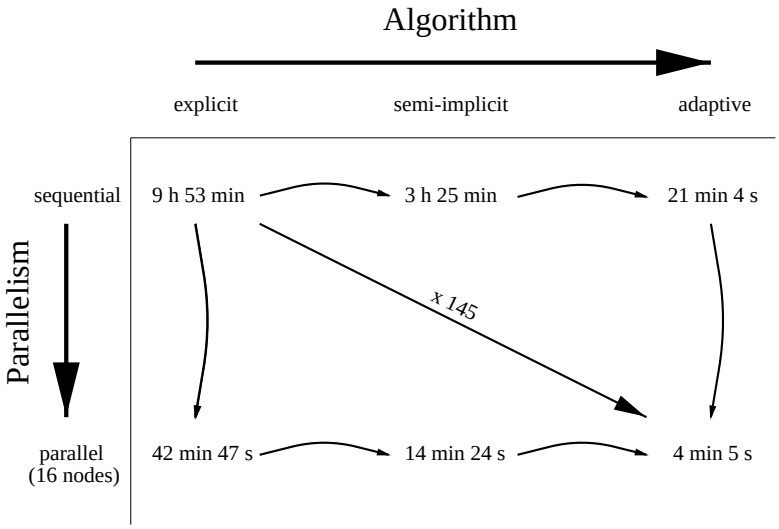

Figure 5.5: *Algorithmic and parallel acceleration — fast waves*

(see Sections 3.3.2 and 3.4.2). The maximal block size is the atomar unit for parallelization and must be adapted to the problem size and the available number of processors.

If the maximal block size is large, it is more likely that severe imbalances occur; however, if the blocks become too small, too much additional computational overhead is introduced at the block boundaries. Figure 5.6 displays the total number of points which are calculated during the dynamically adaptive simulation of a model time of 6 hours with 1440 time steps. The lower line shows the total work load for a run with a maximal block size of 65^2 points, the upper line a four times smaller size with 33^2 points. Both lines increase in the beginning, showing a growth of the refinement areas. This is because points which have already been refined are more likely to be refined again in the next adaptation step to prevent oscillations of the refinement boundaries (see Section 3.3.1.2). After this initialization phase the number of points shows fluctuations due to the dynamic adaptations; however the magnitude stays constant. On average the small blocks introduce about 10% more work.

The opposite is true for the load balance factor (Equation 2.24), which is displayed in Figure 5.7. The upper and medium lines show the load balance factor for the smaller and larger block size, respectively, when distributed onto 4 nodes in a 2×2 process grid. The excellent load balance factor of 0.98 is achieved for the small blocks, whereas the medium line for the large blocks is at about 0.9 on average.

Thus the 10% increase in total costs for the small block size is distributed optimally to the processes. In both cases the loss of parallel efficiency is about 10%, which is just the speed–up 3.6 on 4 nodes given in Tables 5.6

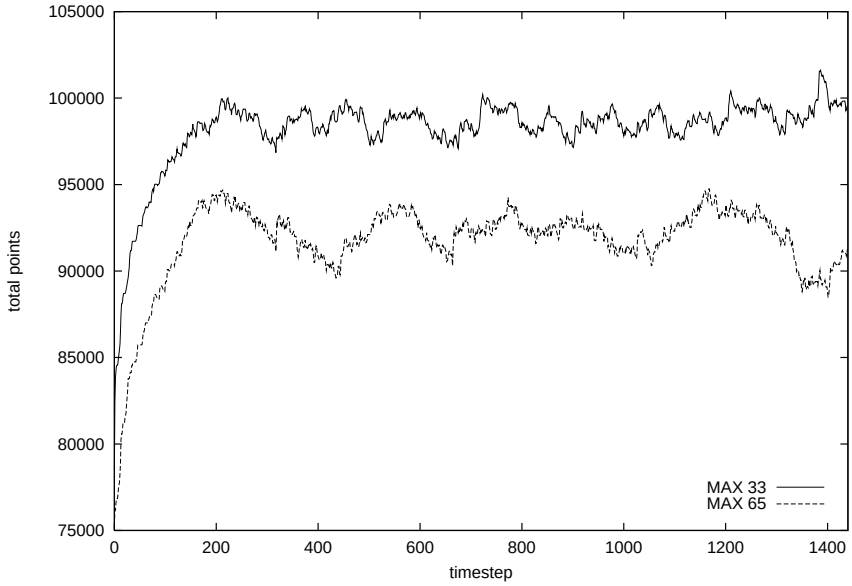

Figure 5.6: *Total number of points*

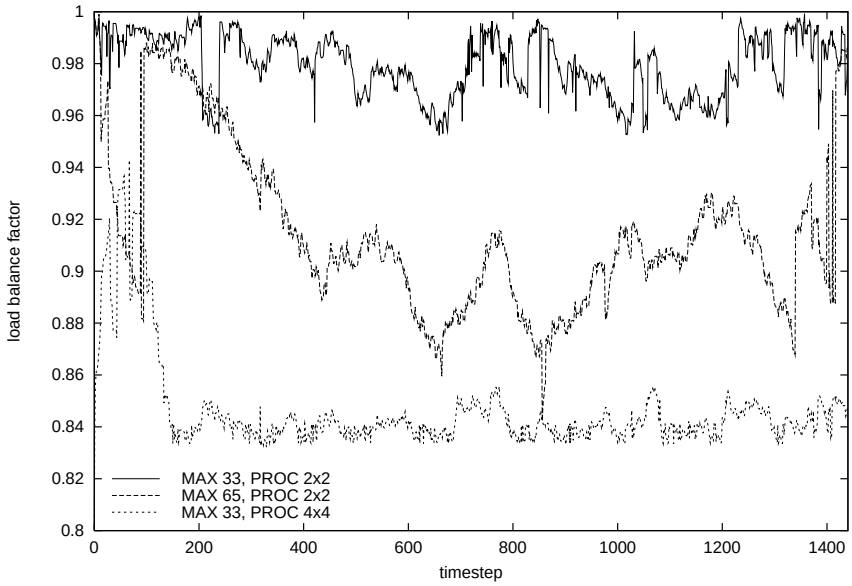

Figure 5.7: *Load balance factor*

and 5.7 for the dynamics.

The lowest line in Figure 5.7 displays the load balance factor for the small blocks distributed to 16 processes. The average value is about 0.84, which shows that the balance of the work allows parallel efficiencies of the adaptive model on 16 nodes with up to this value.

This is an important result, since when computing more expensive equations the proportion of the communication overhead is reduced and thus the parallel efficiencies automatically increases.

Moreover, it is shown that the mapping algorithm produces good balances although it also considers neighborhood relationships. The advantages of a reduced volume communication due to mapping can be seen from Table 5.8 . In the first three lines the refinement overhead given in Table 5.8 is

load bal.	part.	refine			total
		adaptation	nesting	feedback	
red. com.	1×1	188	136	33	1124
	2×2	69 (2.7)	69 (2.0)	23 (1.4)	390 (2.9)
	4×4	49 (1.4)	44 (1.6)	16 (1.4)	198 (2.0)
LPT	2×2	86 (2.2)	87 (1.6)	35 (0.9)	433 (2.6)
	4×4	56 (1.5)	47 (1.8)	20 (1.8)	205 (2.1)

Table 5.8: *Parallel speed–up of refinement overhead — slow waves*

split into the adaptation phase, where new blocks are initialized and work load is migrated; the nesting phase, where the boundaries of the refinement blocks are supported with coarse grid values; and the feedback step, where the fine grid values improve the solution on their coarser grids. All three phases have a large portion of volume data exchange.

The two lower lines are the result of the LPT mapping without consideration of the fine–coarse neighborhoods. Especially on 4 nodes the execution times are much longer. On 16 nodes the volumes to be exchanged are smaller (because of the smaller maximal block size) and less volume communication can be saved.

When comparing the total run times it is seen that the reduction in communication pays even against a probably slightly worse balance of work.

Figure 5.8 presents the portions of the artificial cyclone which are distributed to processor number one in a parallel simulation with four computational nodes. Since the cyclone moves out of the lower left corner, which is initially dedicated to processor one, more and more blocks from the upper right region are distributed to this processor during simulation in order to achieve a balance of work. Some blocks remain for a long time on the same processor (there are 30 time steps in between the pictures) and often corresponding coarse and fine grids are distributed to the same node.

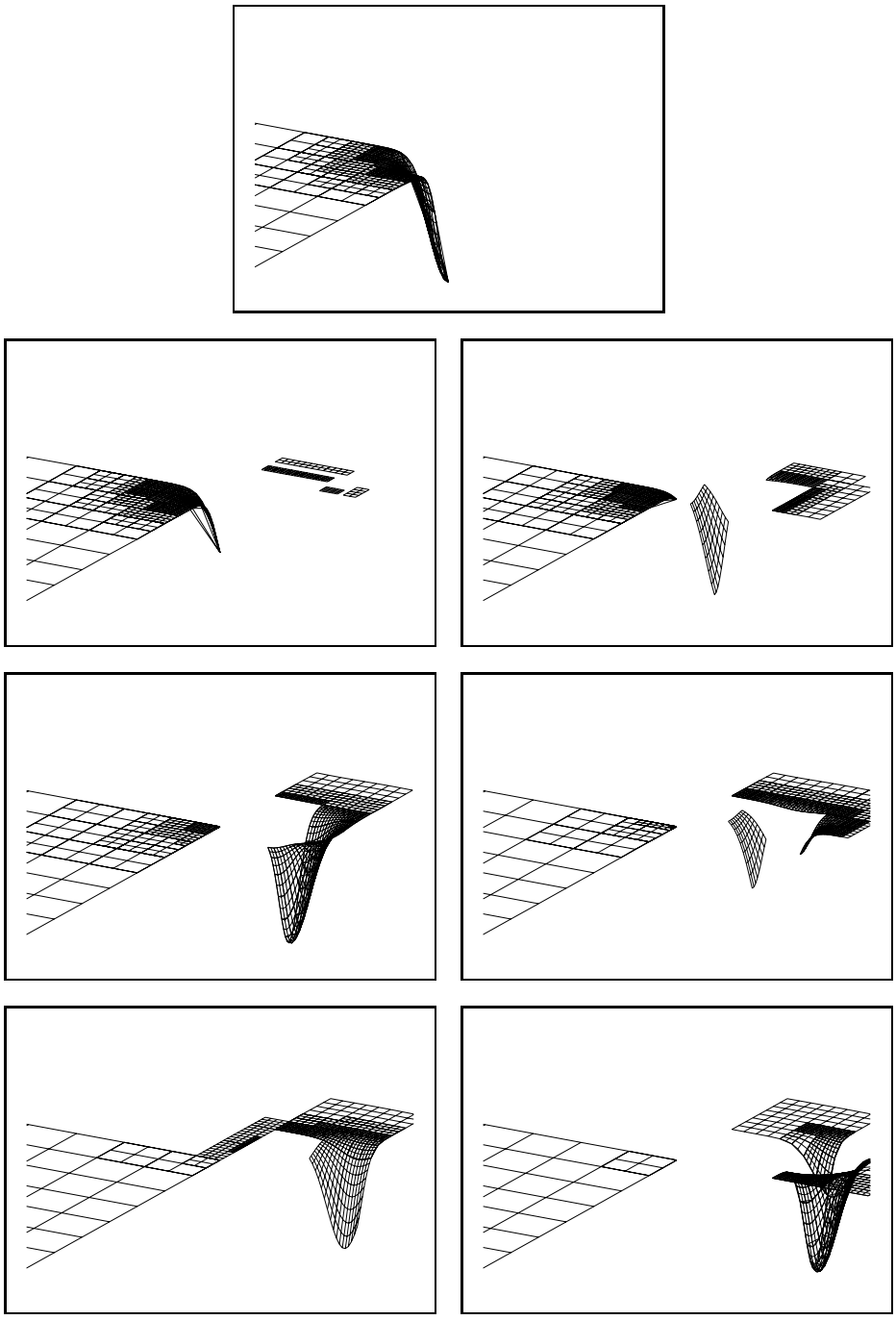

Figure 5.8: *Geopotential of instationary cyclone — 4 nodes*

Chapter 6

Conclusion and Outlook

Joppich [42] wrote in 1990:

> *In comparison with finite element methods, where local refined grids are considered as standard in numerous applications for a long time, local refinement for finite difference methods in combination with multigrid algorithms have not yet been established. Such algorithms for practical applications rarely exist hitherto, although from the theoretical point of view it is often referred to the simple facility of realizing and controlling.*

Since that time the situation has not basically changed.

In this paper dynamically adaptive multigrid is applied to a meteorological simulation. It is shown that adaptive refinements can be used even in combination with a semi–implicit time scheme on a parallel computer with distributed memory. Adaptivity is not a limitation for parallelism.

Moreover, in our application parallelism complements the adaptive simulation. Adaptivity is applied in order to provide a defined degree of accuracy at minimal costs. Consequently, the computing time depends on the simulated scenario. In a best case scenario, no refinement is necessary at all and the computing time is reduced by magnitudes. However, in a worst case scenario, high resolution is necessary everywhere in the computational domain and nothing is saved in comparison to a non–adaptive model (both extreme cases are expected to appear very rarely).

On the other hand, parallelism provides huge computing power, which is best exploited with large problem sizes. Small problems do not require large parallel systems and parallelization overhead affects parallel efficiencies, as often experienced.

Acceleration by adaptivity as well as acceleration by parallelism (as visualized in Figures 1.1 and 5.5) depend very much on the model problem. However, when combining them, the effects tend to equalize each other and the resulting acceleration becomes more stable.

Local adaptive simulations can reduce the computational costs of simulations and less computer power is required for fixed problem sizes. On the other hand larger and more accurate calculations become possible, which is an even more important advantage from the scientific point of view.

The difficulty of this paper was to selection and combine several components belonging to the fields of mathematics, computer science and meteorology. Design components were chosen that are already known to provide good results (e. g. MLAT, semi–implicit time scheme, blending nesting, MPI, LPT–load balance). In most cases these components were adapted and even sometimes improved (MLAT is applied in a time dependent frame as stabilizer, the semi–implicit scheme is based on two time levels instead of three as are commonly used, MPI is applied with nonblocking and asynchronous communication, and the LPT–load balance considers neighborhood relationships as side conditions.)

The outstanding acceleration in simulation times by approx. two magnitudes proves the quality of this design. The only disappointing component is the asynchronous communication, which has not shown the expected performance. No considerable effect of concurrent execution of computation and communication was achieved for the model problem. However, for much longer messages and more computational work (as in three–dimensional models) the situation may change.

Moreover, the dynamically adaptive method is not only designed; it has been implemented in a running model. This implementation can serve as an environment for testing and verifying new and improved components (e. g. semi–Lagrange discretization or an optimal partitioning and load balancing strategy). It is considered possible starting point for future developments in time–dependent adaptive simulations.

The developed and implemented adaptive shallow water model is currently running with very general refinement domains. Nevertheless it is designed for more general meteorological simulations.

Since for three–dimensional models the grid is only horizontally partitioned and purely horizontal refinements are appropriate in numerical weather simulations, a straightforward generalization seems possible.

However, further research is necessary to address additional problems, such as interaction with physical processes.

Appendix A

Spherical Coordinates

A.1 Transformation

The transformation of the spherical coordinates λ, φ and the radius a of the earth in Cartesian coordinates x and y on the sphere is given by

$$\begin{aligned} x &= a\,\cos\lambda\,\cos\varphi \\ y &= a\,\sin\lambda\,\cos\varphi \\ z &= a\,\sin\varphi \quad . \end{aligned} \tag{A.1}$$

Horizontal velocities transform due to

$$\begin{aligned} u &= a\,\cos\varphi\,\dot{\lambda} \\ v &= a\,\dot{\varphi} \quad . \end{aligned} \tag{A.2}$$

The Laplacian ∇^2 is expressed in spherical coordinates

$$\nabla^2\,. = \frac{1}{a^2\,\cos^2\varphi}\left(\frac{\partial^2.}{\partial\lambda^2} + \cos\varphi\,\frac{\partial}{\partial\varphi}\left(\cos\varphi\,\frac{\partial.}{\partial\varphi}\right)\right) \quad . \tag{A.3}$$

A.2 Discretization

In the dynamically adaptive shallow water model, spherical coordinates applied. Since in the previous section often Cartesian coordinates are used to facilitate the presentation, here the actually applied equations are provided. Equations (A.4)–(A.9) correspond to Equations (3.1) – (3.6).

$$\begin{aligned} u^{n+\frac{1}{2}} &= \bar{u}^n - \frac{\Delta t}{2}\left(\frac{u^n\,u^n_\lambda}{a\cos\varphi} + \frac{v^n\,u^n_\varphi}{a} - (f + \frac{u^n\tan\varphi}{a})\,v^n + \frac{h^n_\lambda + h^s_\lambda}{a\cos\varphi}\right) \\ v^{n+\frac{1}{2}} &= \bar{v}^n - \frac{\Delta t}{2}\left(\frac{u^n\,v^n_\lambda}{a\cos\varphi} + \frac{v^n\,v^n_\varphi}{a} + (f + \frac{u^n\tan\varphi}{a})\,u^n + \frac{h^n_\varphi + h^s_\varphi}{a}\right) \\ h^{n+\frac{1}{2}} &= \bar{h}^n - \frac{\Delta t}{2}\left(\frac{u^n\,h^n_\lambda}{a\cos\varphi} + \frac{v^n\,h^n_\varphi}{a} + \frac{h^n\,(u^n_\lambda + (v^n\cos\varphi)_\varphi)}{a\cos\varphi}\right) \end{aligned} \tag{A.4}$$

$$\begin{aligned} u^{n+1} &= u^n + \delta u \\ v^{n+1} &= v^n + \delta v \\ h^{n+1} &= h^n + \delta h \end{aligned} \tag{A.5}$$

$$\begin{aligned} \delta u^{adv} &= -\Delta t \left(\frac{u^{n+\frac{1}{2}}\, u_\lambda^{n+\frac{1}{2}}}{a\cos\varphi} + \frac{v^{n+\frac{1}{2}}\, u_\varphi^{n+\frac{1}{2}}}{a} - (f + \frac{u^{n+\frac{1}{2}} \tan\varphi}{a})\, v^{n+\frac{1}{2}} \right) \\ \delta v^{adv} &= -\Delta t \left(\frac{u^{n+\frac{1}{2}}\, v_\lambda^{n+\frac{1}{2}}}{a\cos\varphi} + \frac{v^{n+\frac{1}{2}}\, v_\varphi^{n+\frac{1}{2}}}{a} + (f + \frac{u^{n+\frac{1}{2}} \tan\varphi}{a})\, u^{n+\frac{1}{2}} \right) \\ \delta h^{adv} &= -\Delta t \left(\frac{u^{n+\frac{1}{2}}\, h_\lambda^{n+\frac{1}{2}}}{a\cos\varphi} + \frac{v^{n+\frac{1}{2}}\, h_\varphi^{n+\frac{1}{2}}}{a} \right) \end{aligned} \tag{A.6}$$

$$\begin{aligned} \delta u &= \delta u^{adv} - \Delta t\, \frac{h_\lambda^n + \mu\,(\delta h)_\lambda + h_\lambda^s}{a\cos\varphi} \\ \delta v &= \delta v^{adv} - \Delta t\, \frac{h_\varphi^n + \mu\,(\delta h)_\varphi + h_\varphi^s}{a} \\ \delta h &= \delta h^{adv} - \Delta t\, \frac{h^n\,(u_\lambda^n + \mu\,(\delta u)_\lambda + (v^n\cos\varphi)_\varphi + \mu\,(\delta v\,\cos\varphi)_\varphi)}{a\cos\varphi} \end{aligned} \tag{A.7}$$

$$\begin{aligned} \delta u^{exp} &= \delta u^{adv} - \Delta t\, \frac{h_\lambda^{n+\frac{1}{2}} + h_\lambda^s}{a\cos\varphi} \\ \delta v^{exp} &= \delta v^{adv} - \Delta t\, \frac{h_\varphi^{n+\frac{1}{2}} + h_\varphi^s}{a} \\ \delta h^{exp} &= \delta h^{adv} - \Delta t\, \frac{h^n\,(u_\lambda^{n+\frac{1}{2}} + (v^{n+\frac{1}{2}}\cos\varphi)_\varphi)}{a\cos\varphi} \end{aligned} \tag{A.8}$$

$$\begin{aligned} -\nabla^2\delta h + \frac{\delta h}{\Delta t^2\, h^n\, \mu^2} \quad &= \\ = \frac{\delta h^{adv}}{\Delta t^2\, h^n\, \mu^2} + \frac{\nabla^2(h^n + h^s)}{\mu} &- \\ - \frac{u_\lambda^n + \mu\,(\delta u^{adv})_\lambda + (v^n\cos\varphi)_\varphi + \mu\,(\delta v^{adv}\cos\varphi)_\varphi}{\Delta t\, \mu^2\, a\cos\varphi} & \end{aligned} \tag{A.9}$$

Appendix B

Stability Analysis

The abbreviations a_{rs} and b_{rs} , $r, s = 1, 2$ used in Equation (3.11) are

$$\begin{aligned}
a_{11} &= 1 \\
a_{12} &= \tfrac{\sigma\mu}{2}\nabla_1^1 \\
a_{21} &= 0 \\
a_{22} &= 1 - \tfrac{\sigma^2\mu^2 h_0}{4}\nabla_2^2 \qquad \text{and} \\
b_{11} &= 1 - \tfrac{\sigma u_0}{2}\nabla_1^1 + \tfrac{\sigma^2 u_0^2}{2}\nabla_1^2 \\
b_{12} &= -\tfrac{\sigma\,(1-\mu)}{2}\nabla_1^1 + \tfrac{\sigma^2 u_0}{2}\nabla_1^2 \\
b_{21} &= -\tfrac{\sigma h_0}{2}\nabla_1^1 + \tfrac{\sigma^2 u_0 h_0}{2}\nabla_1^2 + \tfrac{\sigma^2\mu\, u_0 h_0}{4}\nabla_2^2 - \tfrac{\sigma^3\mu\, u_0^2 h_0}{4}\nabla_2^3 \\
b_{22} &= 1 - \tfrac{\sigma u_0}{2}\nabla_1^1 + \tfrac{\sigma^2 u_0^2}{2}\nabla_1^2 + \tfrac{\sigma^2\mu\,(1-\mu) u_0 h_0}{4}\nabla_2^2 - \tfrac{\sigma^3\mu\, u_0 h_0}{4}\nabla_2^3 \quad ,
\end{aligned} \tag{B.1}$$

where the nabla operator is used to indicate finite differences. These can be transformed into Fourier components according to

$$\begin{aligned}
\nabla_1^1 h^n &= h_{i+1}^n - h_{i-1}^n &&= 2\,I\,H^n \sin\varphi \\
\nabla_1^2 h^n &= h_{i+1}^n - 2\,h_i^n + h_{i-1}^n &&= 2\,H^n\cos\varphi - 2 \\
\nabla_2^2 h^n &= h_{i+2}^n - 2\,h_i^n + h_{i-2}^n &&= -4\,H^n \sin^2\varphi \\
\nabla_2^3 h^n &= h_{i+2}^n - 2\,h_{i+1}^n + 2\,h_{i-1}^n - h_{i-2}^n &&= 4\,I\,H^n \sin\varphi\,(\cos\varphi - 1) \quad .
\end{aligned} \tag{B.2}$$

Literature

[1] A. Arakawa and V. R. Lamb, *Computational Design of the Basic Dynamical Processes of the UCLA General Circulation Model*, Methods in Computational Physics, 17, Academic Press, 174–265, 1977.

[2] A. Arakawa and V. R. Lamb, *A Potential Enstrophy and Energy Conserving Scheme for the Shallow Water Equations*, Mon. Wea. Rev., 109, 18–36, 1980.

[3] D. C. Arney and J. E. Flaherty, *An Adaptive Mesh–moving and Local Refinement Method for Time–dependent Partial Differential Equations*, ACM Trans. on Math. Softw., 16, 48–71, 1990.

[4] M. Ashworth, F. Foelkel, V. Gülzow, K. Kleese, D. P. Eppel, H. Kapitza, and S. Unger, *Parallelization of the GESIMA Mesoscale Atmospheric Model*, Parallel Computing, 23, 2201–2213, 1997.

[5] S. B. Baden, *Software Infrastructure for Non–Uniform Scientific Computations on Parallel Processors*, Applied Computing Review, ACM 4(1), 7–10, 1996.

[6] D. Bai and A. Brandt, *Local Mesh Refinement Multilevel Techniques*, Siam, 8, No. 2, 109–134, 1987.

[7] C. Baillie, J. Michalakes, and R. Skålin, *Regional Weather Modeling on Parallel Computers*, Parallel Computing, 23, 2135–2142, 1997.

[8] S. R. M. Barros, D. Dee, and F. Dickstein, *A Multigrid Solver for Semi–Implicit Global Shallow Water Models*, Arbeitspapiere der GMD 347, St. Augustin, 1988.

[9] P. Bastian, *Parallele adaptive Mehrgitterverfahren*, Dissertation, Univ. Heidelberg, Germany, 1994.

[10] J. R. Bates, *Finite–difference Semi–Lagrangian Techniques for Integrating the Shallow Water Equations on the Sphere*, Proceedings of the ECMWF Workshop on Horizontal Discretization Methods in NWP, Reading, 1987.

[11] J. R. Baumgardner and P. O. Frederickson, *Icosahedral Discretization of the Two–Sphere*, Siam J. Numer. Anal., 22 (6), 1107–1115, 1985.

[12] J. Behrens, *Atmospheric and Ocean Modelling with an Adaptive Finite Element Solver for the Shallow–Water Equations*, submitted to Elsevier Preprint, 1997, to appear.

[13] M. J. Berger and P. Colella, *Local Adaptive Mesh Refinement for Shock Hydrodynamics*, J. Comp. Phys., 82, 64–84, 1989.

[14] M. J. Berger and J. Oliger, *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*, J. Comp. Phys., 53, 484–512, 1984.

[15] M. J. Berger and R. J. LeVeque, *Adaptive Mesh Refinement using Wave–Propagation Algorithms for Hyperbolic Systems*, Special Issue of SIAM J. Numer. Anal., to appear.

[16] M. Böhm and E. Speckenmeyer, *Precomputation based Load Balancing*, Tech. Report 96.219, Universität zu Köln, 1996.

[17] A. Brandt, *Multigrid Techniques: 1984 Guide with Applications to Fluid Dynamics*, GMD–Studie 85, St. Augustin, 1994.

[18] A. Brandt, *Private Communication*, 1998.

[19] A. Brandt and J. Greenwald, *Parabolic Multigrid Revisited*, in: Multigrid Methods III (W. Hackbusch and U. Trottenberg, editors), International Series of Numerical Mathematics, 89, Birkhäuser, Basel, 1991.

[20] R. Calkin, R. Hempel, H.–C. Hoppe, and P. Wypior, *Portable Programming with the PARMACS Message–passing Library*, Parallel Computing, 20, 615–632, 1994.

[21] K. Cassirer, R. Hess, C. Jablonowski, and W. Joppich, *The Shallow Water Test Cases for a Global Model with Documentation of the Results*, Arbeitspapiere der GMD 999, St. Augustin, 1996.

[22] K. Cassirer, R. Hess, W. Joppich, and H. Mierendorff, *Untersuchung und Modellierung des lokalen Modells (LM) für Cluster paralleler Systeme mit gemeinsamem Speicher*, Internal report, GMD–SCAI, 1997.

[23] E. G. Coffman Jr. and G. S. Lueker, *Probabilistic Analysis of Packing and Partitioning Algorithms*, Wiley, 1991.

[24] R. Courant and K. O. Friedrichs, *Supersonic Flow and Shock Waves*, Springer, 1976.

[25] R. Daley, *Atmospheric Data Analysis*, Cambridge University Press, 1991.

[26] J. J. Dongarra, H. W. Meuer, and E. Strohmaier, *TOP500 Supercomputer Sites, 11th Edition*, published at the Supercomputer '98 Conference, Mannheim, June 18–20, 1998.

[27] H. Elbern, *On the Load Balancing Problem of Comprehensive Air Quality Models*, In I. Herlin, R. San Jose, A. Sydow, and J. Verwer, editors, Journal Systems Analysis Modelling Simulation, Gordon and Breach, 1998.

[28] Message Passing Interface Forum, *MPI: A Message–Passing Interface Standard*, University of Tennessee, 1995.

[29] S. R. Fulton, *A Comparison of Multilevel Adaptive Methods for Hurricane Track Prediction*, Electronic Transactions on Numerical Analysis, 6, 120–132, 1997.

[30] U. Gärtel, W. Joppich, and A. Schüller, *Portable Parallelization of the ECMWF's Weather Forecast Program*, Arbeitspapiere der GMD 820, St. Augustin, 1994.

[31] G. A. Grell, J. Dudhia, and D. R. Stauffer, *A Description of the Fifth–Generation Penn State/NCAR Mesoscale Model (MM5)*, National Center for Atmospheric Research, Boulder, Colorado, 1993.

[32] J. Göttelmann, *Construction of Splines and Wavelets on the Sphere and Numerical Solutions to the Shallow Water Equations of Global Atmospheric Dynamics*, Doctoral Thesis, Univ. Mainz, Germany, 1998.

[33] W. Hackbusch, *Multi–Grid Methods and Applications*, Springer, 1985.

[34] G. J. Haltiner and R. T. Williams, *Numerical Prediction and Dynamic Meteorology*, Wiley, 1980.

[35] R. Hempel and H. Ritzdorf, *The GMD Communications Subroutine Library for Grid–oriented Problems*, Arbeitspapiere der GMD 589, St. Augustin, 1991.

[36] R. Hempel and A. Schüller, *Experiments with Parallel Multigrid Algorithms, Using the SUPRENUM Communications Subroutine Library*, GMD–Studie 85, St. Augustin, 1988.

[37] R. Hess, *Numerische Berechnung linearer und nichtlinearer Überschallströmungen um Tragflächen*, Diplomarbeit, Technische Universität München, 1993.

[38] R. Hess and W. Joppich, *A Comparison of Parallel Multigrid and a Fast Fourier Transform Algorithm for the Solution of the Helmholtz Equation on the Globe*, Parallel Computing, 22, 1503–1512, 1997.

[39] C. Hirsch, *Numerical Computation of Internal and External Flows*, Wiley, Volume I and II, 1988.

[40] J. R. Holton, *An Introduction to Dynamic Meteorology*, Academic Press, 1992.

[41] W. Hundsdorfer, B. Koren, M. van Loon, and J. G. Verwer, *A Positive Finite–Difference Advection Scheme Applied on Locally Refined Grids*, Report NM-R9309, CWI, Amsterdam, 1991.

[42] W. Joppich, *Mehrgitterverfahren für Diffusionsprobleme der Prozeßsimulation*, Doktorarbeit, Universität zu Bonn, 1990.

[43] W. Joppich and S. Mijalković, *Multigrid Methods for Process Simulation*, Springer, 1993.

[44] M. Kwizak and A. J. Robert, *A Semi–Implicit Scheme for Grid Point Atmospheric Models of the Primitive Equations*, Mon. Wea. Rev., 99, 32–36, 1971.

[45] M. Lemke and D. Quinlan, *P++, a C++ Virtual Shared Grids Based Programming Environment for Architecture–Independent Development of Structured Grid Applications*, Lecture Notes in Computer Science, Parallel Processing: CONPAR 92 – VAPP V, Springer, 1992.

[46] M. Lemke, K. Witsch, and D. Quinlan, *An Object–Oriented Approach for Parallel Self Adaptive Mesh Refinement on Block Structured Grids*, In W. Hackbusch and G. Wittum, editors, Proceedings of the 9th GAMM Seminar; Adaptive Methods – Algorithms, Theory and Applications, Vieweg, 1993.

[47] M. Lemke, *Multilevelverfahren mit selbstadaptiven Gitterverfeinerungen für Parallelrechner mit verteiltem Speicher*, GMD–Bericht 227, St. Augustin, 1994.

[48] R. J. LeVeque, *Numerical Methods for Conservation Laws*, Birkhäuser, 1990.

[49] H. Lomax, *Recent Progress in Numerical Techniques for Flow Simulation*, AIAA Journal, 14, 512–518, 1976.

[50] B. Machenhauer, *The spectral method*, Numerical Methods used in Atmospheric Models, GARP Publications Series 17, 121–275, 1979.

[51] D. Majewski, *The New Global Icosahedral–hexagonal Grid Point Model GME of the Deutscher Wetterdienst*, ECMWF Seminar on Numerical Methods in Atmospheric Models, 1998.

[52] O. McBrian, *Performance of the Shallow Water Equations on the SUPRENUM–1 Parallel Supercomputer*, Leistungsmessungen für technisch–wissenschaftliche Anwendungen auf dem SUPRENUM–System, H. Mierendorff and U. Trottenberg, editors, Arbeitspapiere der GMD 624, St. Augustin, 1992.

[53] S. F. McCormick and J. Thomas, *The Fast Adaptive Composite Grid Method (FAC) for Elliptic Boundary Value Problems*, Math. Comput., 46, 439–456, 1986.

[54] S. F. McCormick, *Multilevel Adaptive Methods for Partial Differential Equations*, Siam, 1989.

[55] F. Meisgen and E. Speckenmeyer, *The PLB–Library: Dynamic Load Balancing on NOWs*, In A. Bode and A. Ganz et al.,eds., Anwendungsbezogene Lastverteilung ALV'98, SFB–342 TU München, 89–99, 1998.

[56] Meyers Lexikonredaktion, *Wie funktioniert das? Wetter und Klima*, Meyers Lexikonverlag, 1989.

[57] J. Michalakes, *A Parallel Runtime System Library for Regular Grid Finite Difference Models Using Multiple Nests*, Tech. Rep. ANL/MCS–TM–197, Mathematics and Computer Science Division, Argonne National Laboratory, 1994.

[58] J. Michalakes, *MM90: A Scalable Parallel Implementation of the Penn State/NCAR Mesoscale Model (MM5)*, submitted to Elsevier Preprint, 1997, to appear.

[59] S. Orszag, *Transform Method for Calculation of Vector Coupled Sums: Application to the Spectral Form of the Vorticity Equation*, J. Atm. Sci., 27, 890–895, 1970.

[60] N. A. Phillips, *The Equations of Motion for a Shallow Rotating Atmosphere and the "Traditional Approximation"*, J. Atm. Sci., 23, 626–628, 1966.

[61] D. A. Randall, *Geostrophic Adjustment and the Finite–Difference Shallow–Water Equations*, Amer. Met. Soc., 122, 1371–1377, 1993.

[62] H. Ritzdorf, *Lokal verfeinerte Mehrgitter–Methoden für Gebiete mit einspringenden Ecken*, Diplomarbeit, Universität Bonn, 1984.

[63] H. Ritzdorf and K. Stüben, *Adaptive Multigrid on Distributed Memory Computers*, Arbeitspapiere der GMD 781, St. Augustin, 1993.

[64] A. Sathye, M. Xue, G. Bassett, and K. Droegemeier, *Parallel Weather Modeling with the Advanced Regional Prediction System.* Parallel Computing, 23, 2243–2256, 1997.

[65] R. Sauer, *Anfangswertprobleme bei partiellen Differentialgleichungen*, Springer, 1960.

[66] R. Sauer, *Einführung in die theoretische Gasdynamik*, Springer, 1960.

[67] U. Schättler and E. Krenzien, *The Parallel 'Deutschland–Modell' — A Message–passing Version for Distributed Memory Computers*, Parallel Computing, 23, 2215–2226, 1997.

[68] U. Schättler and E. Krenzien, *Model Development for Parallel Computers at DWD*, Making its Mark — Proceedings of the Seventh ECMWF Workshop on the Use of Parallel Processors in Meteorology, G.-R. Hoffmann and N. Kreitz, eds. , World Scientific, 83–100, 1997.

[69] R. Skålin and D. Bjørge, *Implementation and Performance of a Parallel Version of the HIRLAM Limited Area Atmospheric Model*, Parallel Computing, 23, 2161–2172, 1997.

[70] W. Skamarock, J. Oliger, and R. L. Street, *Adaptive Grid Refinement for Numerical Weather Prediction*, J. Comp. Phys. , 80, 27–60, 1989.

[71] P. K. Smolarkiewicz and G. A. Grell, *A Class of Monotone Interpolation Schemes*, J. Comp. Phys. , 101, 431–440, 1992.

[72] A. Staniforth and J. Côté, *Forecast Models for Intermediate–Range Forecasting*, Adv. Space Res. ,12, (7)233–(7)242, 1992.

[73] A. Staniforth and J. Côté, *Semi–Lagrangian Integration Schemes for Atmospheric Models: A Review*, Mon. Wea. Rev. ,119, 2206–2223, 1991.

[74] J. Stoer and R. Bulirsch, *Numerische Mathematik 2*, Springer, 1990.

[75] J. J. Stoker, *The Formation of Breakers and Bores*, Comm. Appl. Math. , 1, 1–87, 1948.

[76] K. Stüben and U. Trottenberg, *Multigrid Methods: Fundamental Algorithms, Model Problem Analysis and Applications*, Lecture Notes in Mathematics, 960 1–176, 1992.

[77] K. Stüben, *Europort–D: Commercial Benefits of Using Parallel Technology*, Proceedings of ParCo'97 Conference, held in Bonn, September 16–19, 1997, to appear.

[78] V. S. Sunderam, G. A. Geist, J. Dongarra, and R. Manchek, *The PVM Concurrent Computing System: Evolution, Experiences and Trends*, Parallel Computing, 20, 531–746, 1994.

[79] C. Temperton and A. Staniforth, *An Efficient Two–time–level Semi–Lagrangian Semi–implicit Integration Scheme*, Q. J. R. Met. Soc., 113, 1987.

[80] C.–A. Thole and U. Trottenberg, *Basic Smoothing Procedures for the Multigrid Treatment of Elliptic 3D–Operators*, Arbeitspapiere der GMD 141, St. Augustin, 1985.

[81] S. J. Thomas, A. V. Malevsky, M. Desgagné, R. Benoit, P. Pellerin, and M. Valin, *Massively Parallel Implementation of the Mesoscale Compressible Community Model*, Parallel Computing 23, 2143–2160, 1997.

[82] U. Trottenberg, C. W. Oosterlee, and A. Schüller, *Multigrid Methods: Basics, Parallelism, and Adaptivity*, to appear.

[83] W. M. Washington and C. L. Parkinson, *An Introduction to Three–Dimensional Climate Modeling*, University Science Books, 1986.

[84] D. L. Williamson, J. B. Drake, J. J. Hack, R. Jakob, and P. N. Swarztrauber, *A Standard Test Set for Numerical Approximations to the Shallow Water Equations in Spherical Geometry*, J. Comp. Phys., 102, 211–224, 1992.

[85] D.–L. Zhang, H.–R. Chang, N. L. Seaman, T. T. Warner, and J. M. Fritsch, *A Two-way Interactive Nesting Procedure with Variable Terrain Resolution*, Mon. Wea. Rev., 106, 1079-1099, 1987.